A Multi-Layered Technique for Fileless Malware Detection and Mitigation

By

Osama Usmani

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

January 2022

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by Mr**Osama Usmani,**Registration No. **000000317936**, of **Military College of Signals** has been vetted by undersigned, found complete in all respect as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial, fulfillment for award of MS/MPhil degree. It is further certified that necessary amendments as pointed out by GEC members of the student have been also incorporated in the said thesis.

Signature: _____

Name of Supervisor Assoc Porf Dr M Faisal Amjad

Date: _____

Signature (HoD): _____

Date: _____

Signature (Dean/Principal): _____

Date: _____

Declaration

I hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification either at this institution or elsewhere

Dedication

“In the name of Allah, the most Beneficent, the most Merciful"

I dedicate this thesis to my mother, sister, and teachers who supported me each step of the way.

Acknowledgements

I am extremely thankful to ALLAH Almighty for his bountiful blessings throughout this work. Indeed, this would not have been possible without his substantial guidance through every step, and for putting me across people who could drive me though this works in a superlative manner. Indeed, none be worthy of praise but the Almighty. In addition, my admirations be upon Prophet Hazrat Muhammad (PBUH) and his Holy Household for being source of guidance for people.

I would like to express my special thanks to my supervisor Assoc. Prof Dr. Muhammad Faisal Amjadfor hisgenerous help throughout my thesis, for being available even for the pettiest of issues. The sense of belief that he instilled in me has helped me sail through this journey. My thanks for a meticulous evaluation of the thesis, and guidance on how to improve it in the best way.

Last, but not the least, I am highly thankful to my parents. They have always stood by my dreams and aspirations and have been a great source of inspiration for me. I would like to thank them for all their care, love, and support through my times of stress and excitement.

Abstract

Anti-Viruses are programmed to detect and mitigate any suspicious program from the computer system by effective scanning of files system. Attackers are also using specially crafted techniques to breach a computer system. In fileless attack, the attacker load and execute malicious code directly into the system memory without effecting any file on the computer system. By this they also gain persistence within the computer system. Fileless malware are deadliest in nature as their detection is not quite easy. Fileless malware can be of multiple components or part. Even the first part cannot be malicious, but the reaming's can be. Traditionally signature-based analysis techniques are employed by different anti viruses to counter such threats. Fileless malware can evade antiviruses techniques this poses a serious threat to individual or organization. Thus, to detect and mitigate the fileless malware a three-layered based technique is proposed in this research along with the experiment, result, and evaluation.

Table of Content

List of Figure

List of Tables

Acronyms

Extensible Markup Language	**XML**
JavaScript Object Notation	**JSON**
Application Program Interface	**API**
Dynamic Link Library	**DLL**
Operating System	**OS**
Windows Sub Subsystem	**WOW64**
Monero	**XMR**
Uniform Resource Locator	**URL**
Antivirus	**AV**
Hypertext Markup Language	**HTML**
Windows Management Instrumentation	**WMI**

Chapter 1

Introduction

1.1 Background

Malware is any malicious program written on purpose to cause damage to the digital devices or its users. With the wide usage of internet and its expansion in terms of utilization, it begins to provide more options to adversaries with ease to propagate their malicious intentions around globe. The first malware “Creeper Worm” was created for experimental purpose that replicate itself to remotely connected systems running on TENEX Operating System [1]. Later the development of modern operating system brings more opportunities for adversaries to utilize vulnerabilities and available options to gain commercial, political, and self-merriment gains. So, to overcome the effects of malware, Anti Malware solutions are developed to detect and remove malware from compute systems.

Modern programming languages and feature-rich libraries helps in growth of more sophisticated malwares. As technologies evolve to counter malware problem, adversaries keep improving their lethal techniques from file-based malware to diversified fileless malware. Fileless technique don’t use files system to store any malicious executable on system instead they completely reside on system’s volatile memory to perform malicious execution [2]. As no file is stored by malware on the system, signature-based technique will eventually fail to detect these malwares. When there is no detection, the impact will be fatal for business and sensitive organizations [3].

When a system gets compromised with any type of malware the first thing the anti-malware solution does is to analyze the file to check the suspected file either it is marked as malicious in its signatures databases or not. A hash of file is stored in database, which is maintained by anti-malware solutions to quickly identify the malware and block them immediately before malware perform its execution. Furthermore, the forensic expert can look up the file systems for any unknown

suspected file and analyze it and mark it as malicious program for future automated detection. However, this is not the case of fileless malware as they abuse the operating system trusted software/tools aka LOLBins (Living of the Land Binaries) to perform the malicious execution. Adversaries can also use the known and known vulnerabilities within the trusted software, including Microsoft Macros and PowerShell. These software/tools help malware to directly load itself into memory without touching the system drives [4]. Fileless malware have no limitation to what type of specific attack they can perform. The attack can be of multiple type like reconnaissance, persistent and theft of data. PowerShell is used for wide number of fileless attack due to number of reasons like it is installed by default on every system, trusted by system administrators, easy to obfuscate the script, and attackers can easily exploit it to use it remotely. [5]

The usage of fileless attack is increasing day by day as its detection is not easy for standard anti-malware solution. According to WatchGuard's Security report there has been 900% increase in usage of fileless malware in 2020 as compared to 2019 [6]. This is due to the sophisticated working of fileless malware and easily to harden the generic malware detection. Adversaries leverage the fileless behavior of malware and targeting users and organization's assets behind ordinary network defenses. The fileless threats are very dangerous because of their ability to evade antimalware protection systems. The fileless attack can be crafted by using the freely available tools like Metasploit Framework that can help to easily inject the malicious code into any legitimate running process [7].

1.2 Types of Windows Malwares

Malware is classified into several forms, including viruses, worms, Trojan horses, spammers, rootkits, bloatware, scareware, spyware, ransomware, backdoors, key loggers, rogue antivirus software, and browser hijackers. Each type of malware has its own purpose and intentions which can be used for fun purposes to serious threats like leakage of private data, personal data or destruction of data or some ransom payment. Some of the types are defined below.

1.2.1 Virus

Viruses are a sort of computer programme that, when executed, reproduce itself by altering the code of other computer programmes and inserting its own code. The afflicted regions are referred to as being "infected" with a computer virus if the reproduction process is successful, a term that is taken from the term "infected" with a biological virus.

1.2.2 Worm

A computer worm is a solitary malicious computer software that copies itself in order to propagate to other systems. It often spreads over a computer network, depending on security flaws on the target machine to gain access to the network. It will utilize this computer as a host in order to scan and infect other machines.

1.2.3 Trojan Horse

A Trojan horse, often known as a Trojan, is a sort of malicious malware or software that seems to be genuine but has the capability of taking ownership of your computer network. A Trojan horse is a computer programme that is meant to injure, disturb, steal, or otherwise cause harm to your data or network in some way. Once a Trojan has been deployed, it is capable of doing the activity for which it was created.

1.2.4 Bot

Bots are computer programs that are meant to conduct specified actions autonomously and may be managed from a distance. Botnets are a unique kind of bot that may be deployed in channels to launch denial - of - service. Bots may very well be employed as spambots, which can generate adverts on websites and harm server data, as web crawlers, and in other ways. In order to prevent bots from accessing the website, CAPTCHA tests are used to confirm that the visitors are human.

1.2.5 RootKit

An example of malicious computer software is a rootkit, which is a set of computer programs built to grant unauthorized access to computer systems or a portion of its software that is not otherwise accessible. Rootkits are also used to conceal their own

presence and the presence of other application. It is a combination of the words "root" and "kit" that is used to refer to a computer virus.

1.2.6 Scareware

A kind of software scam method known as scareware use pop-up warning messages and other social engineering techniques to terrify you towards subscribing for bogus cybersecurity protection that is masquerading as legitimate cybersecurity protection. Fearware may be ineffective bloatware that is generally innocuous, or it may be malicious spyware in the worst situations.

1.2.7 Spyware

A computer program that Spies on users' activities and collects private information such as frequently visited URLs, account records (bank details), e - mail address, customer data (including fingerprints), as well as many other pieces of data. It infiltrates a computer when inexpensive and possibly hazardous software is downloaded and installed without the individual's consent or authorization.

1.2.8 Ransomware

Malicious software known as ransomware is meant to encrypt and make data and the systems that depend on them useless once they have been encrypting and making them useless Then, in return for decryption, malicious actors demand a ransom payment. When a computer is infected with ransomware, users will be unable to access their data unless a ransom is paid to the perpetrators. The existence of ransomware versions has been detected for many years, and they often seek to squeeze every penny from individuals by showing an on-screen warning.

1.2.9 Backdoor

A backdoor is a means of circumventing regular security or protection in a system, commodity, attached to a surface, and its incarnation that is generally clandestine in its operation. For the most part, backdoors are employed to protect unauthorized computer access or to get access to unencrypted in cryptosystems.

1.2.10 Keyloggers

Typically performed secretly, keystroke logging (also known as stealing personal information or keyboard recording) is the act of collecting the buttons hit on a keypad in such a way that a person that use the computer is uninformed knowing their activities are indeed being recorded. The information may then be obtained by the person in charge of the monitoring software.

1.2.11 Rogue Security Software

When users are misled into believing they have a virus on their computer, malicious code and internet scams are used to trick people into paying for a bogus antimalware tool that truly installs computer viruses on their desktop. Nefarious security software is a category of malicious programs and internet scams that is used to trick consumers into believing they have a viral infection on their computer and persuade people to charge for a bogus malware protection tool that installs malicious files on their own computer.

1.2.12 Browser Hijackers

Another example of harmful software is web hijacker, which alters the behavior, configuration, and look of a browser without any of the customer's knowledge or permission. When a compromised browser is used to generate money from advertisers for something like the hijacker, all of that may also be used to assist additional harmful actions including such information gathering and keyboard recording.

1.3 Malware Creators

Individuals or groups that generate viruses are referred to as vandals, extortionists, scam artists, computer hackers, and scammers, among other terms. In order to generate income in an unlawful way, the majority of harmful programmes are created. To keep from being bored and to get more notoriety, vigilantes in the past used to write malicious software. In subsequent years, virus was used for illicit objectives such as stealing financial data, personally identifiable information, eavesdropping, damaging secret data, and a wide range of other criminal actions, among other things. Malware authors may be found both inside and outside of companies. Employee or a trustworthy

programmer from the inside of a company who can incorporate harmful malware inside programs prior to launch to the public is referred to as an insider attack. An existential threat refers to any other individuals or organizations who may attempt to introduce harmful code into a product after it has been made available.

1.4 Malware Propagation

Malware may infiltrate a computer or smartphone in some kind of a variety of methods, including malicious e - mail attachments, document sharing, text messaging, through use of third-party applications while networking sites, through use of pirated software, as well as the usage of USB and certain other removable media. Virus may cause significant damage towards the system's system partition, application program, file systems, as well as the systems BIOS once it has gained access to the network. This results in the scheme behaving abnormally. Everyone involved in the creation of virus has one primary goal: to get their infection onto as numerous computers or mobile phones as they possibly can. Both media manipulation and attacking a computer without the patient's awareness are methods of accomplishing this goal. These techniques are frequently used in conjunction with one another, and they frequently contain operations designed to circumvent antivirus applications that have been placed just on machine being compromised.

Malware authors use camouflage tactics to prevent getting identified by anti-malware solutions. As nothing more than a consequence of such camouflage tactics, some virus changes its behavior for each dissemination and transmissions cycle. Some virus encrypts themself as well as their harmful operations, making it harder to recover unique signatures to be used in malware management and mitigation. A couple of hiding tactics are described in further detail hereunder.

1.4.1. Code Obfuscation

Obfuscation is the purposeful process of writing code or machine code that is hard to decipher for people in the field of software development. It may utilize excessively convoluted phrases to create statements, similar to how obfuscate is used in human language.Programmers are using this methodology to conceal their ransomware from fingerprint detection methods by taking certain actions such as adding extra jumps,

resurrected implantation, use of garbage commands, register gender transition, guidance replacement, subprogram reshuffling, code assimilation, and code interoperability.

1.4.2. Code Encryption

General terms, encryption refers to the process of transforming information into something like a hidden message that conceals the real meaning of the material being encrypted. When it comes to computers, plaintext refers to unencrypted data, whereas ciphertext refers to encrypted information. Security mechanisms, often known as cyphers, are formulae that are used to encrypt and decrypt communications. Essentially, this is a protective system that encodes viruses or their destructive operations by utilising an encryption method and a unique encryption key combination. It duplicates on its own and makes and builds a new encrypted connection of the virus, which contains a data encryption as well as a new password, while it is in the process of being executed. As a result, even though the encrypted key as well as the encoded code are constantly changing, they may still be recognized due to the inflexible deciphering technique.

1.4.3. Oligomorphic Strategy

Typically, a piece of malware will employ an oligomorphic program to produce a decrypt or for itself within manner very similar to the generation of a basic parametric code on its own. As little more than a protective measure, this technique encrypts virus, but it utilizes a different hacking tool for each iteration of ransomware in order to distinguish between the two. Each virus programme maintains a collection of decryptors, from which a spontaneous decryptor is picked for use in decrypting the virus's messages.

1.4.4. Polymorphic Strategy

Polymorphic malware is a computer virus which alters the distinguishing characteristics on something like a constant basis to evade suspicion. Compared to other systems approaches entail altering identifying properties such as config files and classifications, as well as encryption keys, on a regular basis to render virus unidentifiable to several different scanning tools. This method allows for the generation of millions of

decryptors by modifying the instruction in the next edition of the virus to circumvent handwriting detection. Every time the virus is executed, a new decryptor is constructed and combined with the encoded virus payload to establish a new variation of the virus that is unique to that execution. A vast number of different decryptors may be developed, however signature-based techniques still can detect infection by recognizing the initial version using an emulated version of the virus.

1.4.5. Metamorphic Strategy

Metamorphic malware and parametric malware are 2 kinds of harmful software applications (viruses) that will have the capacity to modify their programming as they spread through a computer network or the Internet. Metamorphic malware is virus that is rebuilt with each repetition, resulting for each subsequent copy of the code being distinct from of the previous one. Metamorphic virus transforms itself into a whole different entity from the initial version it was based on. Rather than establishing a new code in this case, a new sample or bodies is produced without affecting the activities of the existing one. As a result, the virus does not include a programming engine, and updates to the virus code are made on a consistent basis with each delivery.

1.5 Malware Symptoms/Sign

These methods by which virus spreads and infects a computer can change, however the indications that it causes seem to be the same. Any of the symptoms listed may appear on a computer that has been infected with malware are following

- The program's capabilities have been turned off.
- The user's computing power is too sluggish.
- Issues with the web browser's performance.
- Broadband connection issues.
- Everything seems to be entirely normal.
- Pop-up advertisements begin to appear indiscriminately.

- Your browser is constantly being diverted.
- Articles from an unknown source appear on your social media pages.
- Your will be subjected to extortion requests.
- Processor use has risen.
- A frightening message is sent by an unidentified application.
- Manifestation of unusual applications, files, or symbol appearances.
- Programmes that are now executing, quitting, or configuring themself System freezing or crashing is a common problem.
- Files are automatically modified or deleted without the user's knowledge.
- Emails and messages are delivered regularly without user's knowledge.

1.6 Preliminaries of Fileless Malware

In this section, detailed concepts related to malware and fileless based malware are presented.

1.6.1 PowerShell

Powershell An object-oriented, strongly typed scripting language is used by PowerShell, which is an organizational computing tool. As a consequence of Microsoft's open source strategy, it is becoming more popular. As a result of the fact that PowerShell is pre-installed on the majority of operating systems [8], attackers have discovered the benefits of using it as an attack vector. In addition, since it can run/inject the process's DLL(Dynamic Link library) straight in memory, it has full access to sensitive system functions and may be managed completely by fileless malware PowerShell, on the other hand, is designed to be dynamic, and it is capable of creating script components at several degrees of sophistication. It is most likely because PowerShell is pre-installed on each and every Windows-based device that fileless

malware uses it the majority of the time. Understanding, writing, encoding, and encrypting PowerShell scripts is straightforward, but detecting and decrypting them is more challenging.

- PowerShell is a command-line tool that is mostly used for administrative tasks [9]hence it is not often utilised for everyday tasks.

- For this reason, malware attackers like PowerShell since it allows them to grant remote access to their victims

1.6.2 Fileless Malware and its Working

Fileless malware works completely in stealth mode due to which its detection is quite impossible. The malware completely residue in the memory component and perform its task from there by avoiding the AV detection. There can be various steps before fileless malware infect the system [10].

- **Case 1:** Fileless malware can be initialized into a system via spear phishing technique, then a highly encoded/encrypted file (.js, .zip etc.) can start the PowerShell process in the background without user's knowledge, which later download the actual malware form the remote address or decode any malicious code from the previous stage and then load directly into system memory without touching the file system [11].

- **Case 2:** An encoded script can be embedded along with the trusted Windows OS Software files, like .pdf, .pptx, .doc, .docx. The files do not contain any malicious code at this point thus they can bypass any AV detection. Upon opening these legitimate looks alike files, a background process of PowerShell is run in stealth mode. This PowerShell process will then download the actual malware from any remoteaddress and load the actual malware directly into the system memory where the malware executes it malicious task [12].

- **Case 3:** PowerShell can be run from chrome.exe by using one of its exploits, when a user visits a website, an exploit is called by the attacker, which then open

a PowerShell process on user machine in stealth mode, which then execute any malicious activity into the target system [13][14].

- **Case 4:** A PowerShell script file (.ps1) can be used directly for malicious purposes. A .ps1 file shared spear phishing technique. Upon opening it will be executed within a minute and perform its malicious activity e.g., ransomware attack, reverse TCP attack. In this case there might be no other child process as all activity is done using PowerShell process [15].

Fileless malware attacks are mostly employed for large scale attack for bigger organization. Though it can target single device or single user too [16]. The persistence mechanism can also be performed by attacker by combing other techniques of persistence using windows registry system. The example attack kill chain for fileless

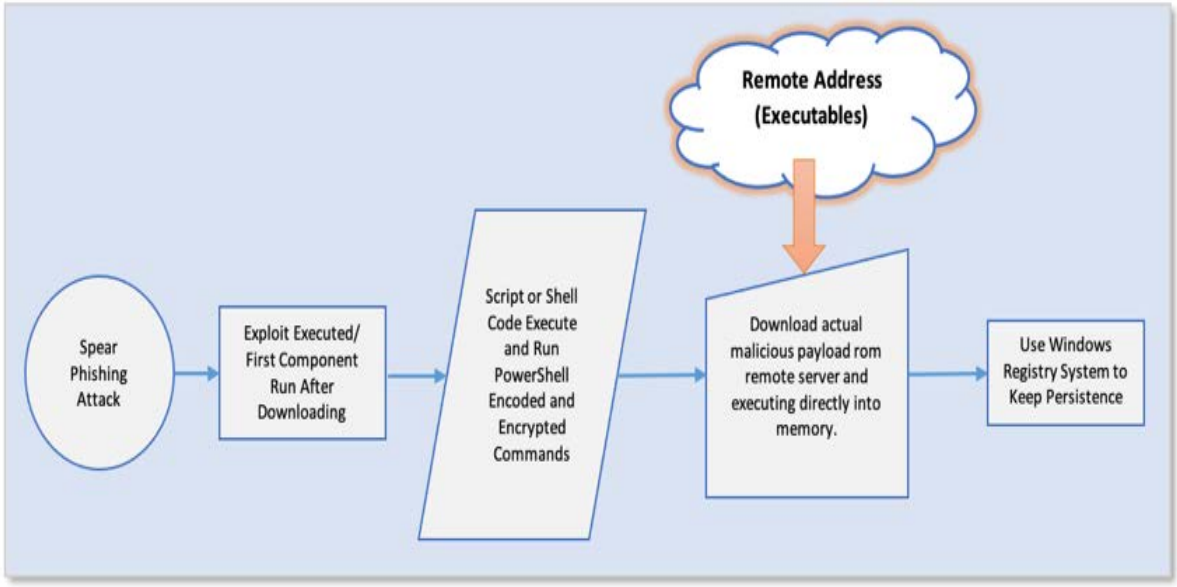


Figure 1.1: Example of Fileless Malware Attack

The process's chain for one of the known fileless malware attack (Sodinokibi: The Crown Prince of Ransomware:) is shown in Figure 1.2. This is a ransomware attack that hides its actual malicious infection using different techniques and deliver the actual malicious payload after executing the power shell process. One another Fileless malware attack known as Ramnit Banking Trojan, which is used to mine banking credential from

different resources by paring with many other file-based malware technique. Both stages of this malware are illustrated in Figure 1.3 and Figure 1.4.

Figure 1.2: Sodinokibi Fileless Malware Process Chain Before Actual Payload

One another Fileless malware attack known as Ramnit Banking Trojan, which is used to mine banking credential from different resources by paring with many other file-based malware technique. Both stages of this malware are illustrated in Figure III and Figure IV.

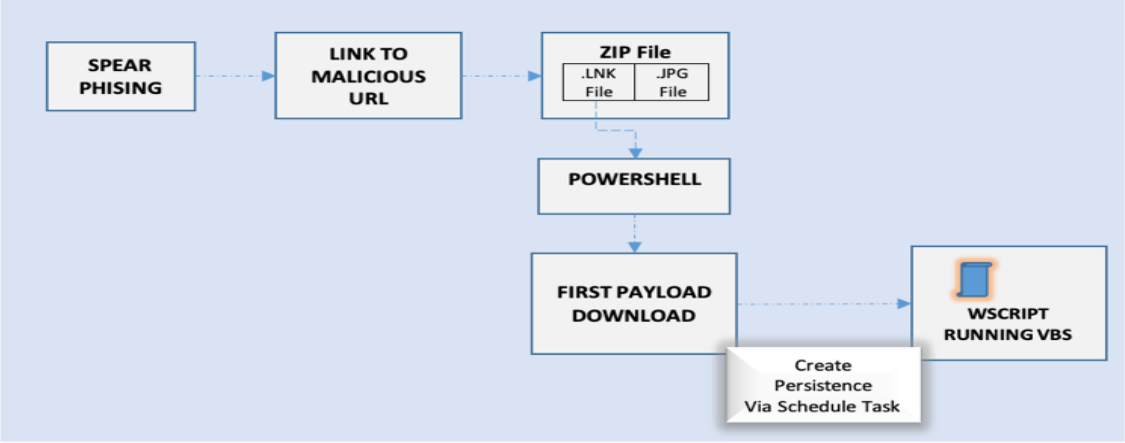
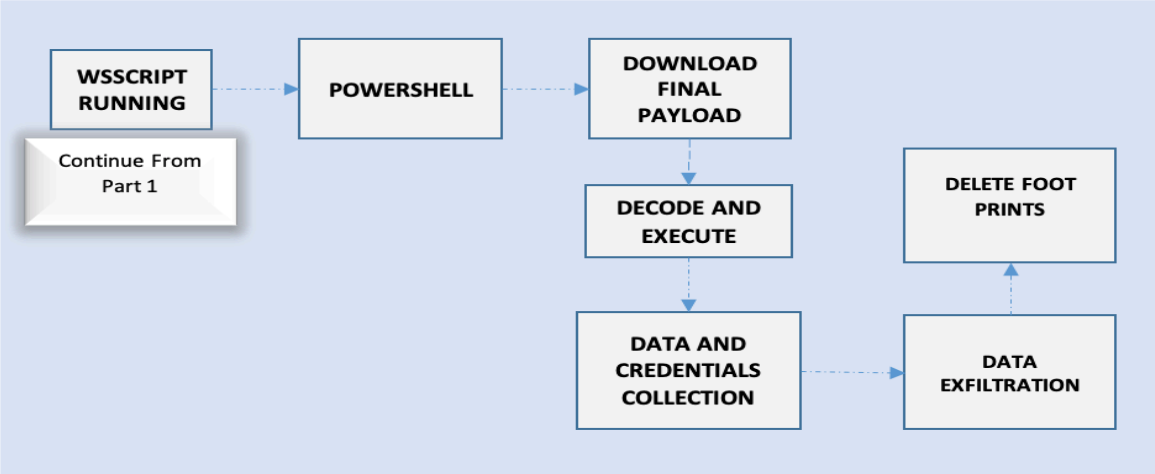


Figure 1.3: First Payload Download and Execution of Ramnit Banking Trojan

1.6.3 Fileless vs File-Based Malware

File based malware mostly lay their fingerprints on the system's storage device thus the corresponding file hash can easily find by any antivirus system (AV). If similar hash is considered as a malware on the AV signature database, it can be easily detected thus removed completely from system by AV. But on the other hand, the fileless malware utilizes the pre-installed non-malicious apps/executables to infect the system, it

becomes hard for AV to detect such malware as AV considered the malware as legitimate software running on the system [17].



Executeable	Not Available	Available
Malicious File	No	Yes
Complexity	Very High	Moderate
Detection Rate	Very Low	Moderate
Infection Loder's	Trusted windows OS executeables	All available executable files
Persistence	Yes	Possible by paring with other non-fileless technique
Signature Analysis	Not Possible	Yes (As .exe is available)

AV Detection	Not Possible	Yes
Footprints	Very Low	High
Detection Method	Dynamic Only	Hybrid (Static and Dynamic)
Category	All (Ransomware, Trojans, Keyloggers)	All (Ransomware, Trojans, Keyloggers)

1.6.4 Fileless Malware Detection

As fileless malware are memory resident malware it nearly impossible to detect them. To detect fileless malware the anti-solutions need to dive deep into the system resources. The static malware analysis is not going to work in case for files malware as it has very minimum interaction with file system thus dynamic analysis is required along with behaviors-based analysis to detect the fileless malware from its root [18]. The detailed analysis of available literature along with their critical analysis and why there is still need of some improvements to detect a fileless malware completely are discussed in Section III.

1.6.5 Static Analysis Vs Dynamic Analysis of Fileless Malware

The static analysis is analyzing any possible malware executable without running it. This may include examining of the file using different tools. These tools can perform static analysis like string analysis of the executable or any other pattern recognition of malicious code samples [19].

The Dynamic analysis is done after running the malware on the system then looking up for the possible signature to consider it as malware.[20] The dynamic malware analysis can be risky as it can immediately affect any computer system after execution. Thus, most of the dynamic analysis is performed in virtual environment so malware cannot spread to network through any slip space [21].

The fileless malware can be of multiple components, it one part will be look alike a legitimate code but other components which are not yet available on the system can be malicious [22]. For example, AV scan the available part of the fileless malware which is not malicious then after some time the first non-malicious part can down the actual payload from remote address. This behavior of fileless malware can be avoided from static analysis even dynamic analysis can be bypassed, after using literal execution mechanism using timers. Therefore, a need of dynamically behavioral analysis of system and its resources is necessary to detect fileless malware.

Table 1.2: Static and Dynamic Analysis of Fileless Malware

Factors	**Static**	**Dynamic**
Time	Minimum	More (More Clock)
Resources Consumption	Less	Medium
Detection Rate	Very Low	Moderate
Obfuscation Detection	True	False
Code Execution	Yes	No
Detection Accuracy	Least to Zero	High Detection by Deep Behavioral Analysis

1.7 Problem Statement

Fileless malware attack are increasing day by day and their detection is very limited. The previous proposed techniques lack the ability to detect and mitigate the multistage

fileless malware. Most of the previous proposed research's only work via single layered solution, which can be easily evaded via latest malware. With the digital transformation businesses, organizations and critical infrastructure are at huge risk of becoming target of such attacks, which standard solutions failed to detect and mitigate. Thus, to make defensive against such attacks it requires an evolving solution which can use a multi layered and integrated techniques that monitors entire life cycle of malwares to and behaviors of system and to detect, correlate and mitigate them.

There has been huge increase in fileless attack over the past decade. According to Trend Micro report malware attacks using fileless techniques has been increase by 265% in 2019 only. There is also significant increase in ransomware attacks by 42.98% along with data theft attacks by 319%, these all attacks use fileless techniques. Now a day's attacker performs intentional, specially crafted and targeted attacks that take advantages of the targeted system resources. With the digital transformation businesses, organizations and critical infrastructure are at huge risk of becoming target of such attacks, which standard antiviruses failed to detect and mitigate. Thus, to defend such attacks it requires an evolving solution which can use advance multi layered and integrated techniques that monitors entire life cycle of malwares to detect, correlate and mitigate them.

1.8 Objectives

Following are the objectives of this research.

1. To design a monitoring system capable of detecting malicious behaviour of running processes on machine.

2. To monitor any process suspicious behavior or activity that matches the fileless attack's techniques.

3. To design a fileless attack detection and mitigation technique.

1.9 Thesis Outline

- *Chapter 1:* Chapter 1 focusesonFileless malware basic knowledge, their stages and related issue and background study.

- *Chapter 2:* Chapter 2 focuses on detailed analysis of fileless malware and related concepts, its malicious architectures, differences with traditional malware and challenges are presented in Section II.

- *Chapter 3:* Chapter 3 focuses on the extensive study of literature review and critical analysis of available proposed techniques in the study is presented.

- *Chapter 4:* Chapter 4 focuses on proposed methodology which is an end to end and a multilayered technique and detailed modular approach for fileless malware detection and mitigation.

- *Chapter 5:* Chapter 5 focuses on the results and effectiveness of the proposed techniques which is calculated via testing the technique with various malware samples and datasets.

- *Chapter 6:* Chapter 6 focuses on the Conclusion, Limitations and Future Work

Chapter 2

Literature Review

2.1 Literature Review

Fileless Malware leaves no traces for antivirus software to detect so that making it very hard for antivirus software to detect such attacks, so discovering fileless malware attacks is a very challenging task for security analysts. Authors in [23] primarily discusses malware kinds and detection methods, as well as contemporary malware analysis methodologies. Malware detection approaches, such as signature-based and heuristic-based. It provided a comprehensive picture of malware detection. To prevent the effect of fileless malware, several detection approaches are theoretically proposed.

PSDEM is a PowerShell de-obfuscation technique proposed by the authors in [24]. To obtain original PowerShell scripts, the PSDEM technique uses two levels of de-obfuscation. Extracting PowerShell scripts from obfuscated document code is the initial stage. De-obfuscation scripts, which include encoding, text manipulation, and code logic obfuscation, make up the second layer. It also explains that PSDEM improves the speed and accuracy of detecting malicious PowerShell scripts included in word documents.

Fileless Ransomware is a new type of malware that primarily uses both ransomware and fileless malware mechanisms. Detecting and defending against these types of cyberattacks is becoming a significant challenge for IT companies. Protection against such ransomware using current security approaches is very difficult. Security professionals are currently putting a lot of effort to guard against these types of ransomware by developing various proactive and reactive procedures. Authors in [25] explains how ransomware and fileless malware act, how fileless ransomware works, what attack vectors fileless ransomware can use.He define different types of fileless ransomware and their uses, prevention measures, and recommendations for defending against fileless ransomware.

Fileless malware can also spread by making copies of itself to a place specified in the malicious code and employing persistence techniques such as setting up an auto start function to guarantee that they continue to execute.As we know, Fileless malware only resides in memory and is written to RAM rather than being installed on the hard drive of the target machine. It is hard to identify and delete malware that does not have a file. The virus's placement, like rootkits, makesdetection and removal more difficult than aconventional malware infection. As attackers become more adept at using LOLBins, it is critical for defenders, incident responders, and forensic analysts to understand.

However, authors in [26] describes that fileless malware provides a model opportunity for an attacker to sketch out a system before dumping and running the core payload on disc. Inappropriately file-based detection techniques, like as antivirus software, have become more ineffective, detecting fewer than half of the malware that is critical to long-term system stability.

Furthermore, authors in [27] create the deobfuscation method for PowerShell scripts that is both effective and lightweight. The design is a unique subtree-based deobfuscation approach that performs obfuscation detection and emulation-based recovery at the level of subtrees to meet the key difficulty of precisely identifying the recoverable script. They develop the first semantic-aware PowerShell threat detection system and use the conventional objective-oriented association mining approach to enable semantic-based detection. They used deobfuscation on 2342 benign samples and 4141 malicious samples. On the that samples semantic-aware attack detection engine outperforms both Windows Defender and VirusTotal which result 92.3 percent true positive rate and a 0% false positive rate on average.

The rise of fileless malware and its defensive strategies can be used to mitigate it. Fileless malware may be a class of malware that runs entirely in memory and leave as small of a footprint on the target host as possible. It attacks windows applications and system administration tools such as Windows Management Instrumentation (WMI) and PowerShell to execute and spread fileless malware [28].

Fileless malware does not use traditional executables to carry-out its activities. The malware leverages the power of operating systems, trusted tools to accomplish its

malicious intent. To analyze such malware, security professionals use forensic tools to trace the attacker, whereas the attacker might use anti-forensics tools to erase their traces. Authors in [29] presented a theoretical model to detect fileless malware attacks in the incident response process. However, it lacks any practical implementation and results verifications.

Malware Analysis has always been an important topic of security threat research ever since the early days of computers. Fileless malware is purposed to be memory resident only rather than writing artifacts to the filesystem. This makes it nearly impossible for antivirus signatures to trigger a detection. Authors in [30] developed a YARA rules to detect fileless malware using the binary executable. However, it lacks any model and results verification. Also, it is quite difficult to get an executable for fileless malware to utilize the proposed system.

Authors in [31] proposed a system, which utilizes process generation mechanism to perform host-based malware detection mechanism. The proposed system collects the running processes on the system then developed a child parent relationship between them. Then they use pattern recognition for processes uniqueness to detect an anomaly. The proposed system run on multiple hosts for multiples days for total numbers of 2403203 processes from which they have only found 38 abnormal processes and 1 malicious fileless malware based upon PowerShell. The proposed system lacks any proper validation through numbers of malicious fileless datasets. Also, it doesn’t have any mechanism to detect PowerShell commands.

TrustSign is a novel, trusted automatic malware signature generation method based on high-level deep features transferred from a VGG-19 neural network model pre-trained on the ImageNet dataset. The system is proposed by authors in [32], which leverages the cloud's virtualization technology. Trust Sign analyzes the malicious process in a trusted manner since the malware is unaware and cannot interfere with the inspection procedure. By removing the dependency on the malware's executable, author’s method is capable of signing fileless malware. Authors research is focus research on crypto jacking attacks, which current antivirus solutions have difficulty to detect. The main limitation of the proposed solution is lack of run time memory analysis and PowerShell process detection.

Authors in [33], demonstrate how an attacker can leverage a feature reach programming language like JavaScript to craft an fileless attack. The authors run proposed malware through different AV solutions to validate its effectiveness. The solution to mitigate such malware is to disable JavaScript or HTML5 which is not feasible option.

Authors in [34] proposed a new method of writing and rewriting memory section to detect the exact end time of unpacking routine and extract original code from packed binary executable. The proposed method has been successfully extracted hidden code from recent malware family samples. At least 97% of the original code could be extracted from the various binary executable packed with different software packers. The main limitation for the proposed method is that the system dynamic and auto detection and mitigation. Also, the proposed solution is not validated on any runtime environment.

Monero (XMR) is by far the highest popular cryptocurrency among threat actor installing mining malware because it comes with full anonymity and resistance to an application-specific circuit mining. Authors in [35] proposes a better method for classifying conventional malware and cryptocurrency mining malware by using supervised machine learning method. The proposed approach is defining a better algorithm for enhancing accuracy and efficiency for cryptocurrency mining malware detection. However, it lacks any runtime malware detection and it still need executable images to run and process through the proposed system. In real time fileless malware takes only few minutes to halt computer operations e.g., ransomware attack.

Cybercriminals are becoming skillful day by day and crafting more sophisticated and conducting advanced malware attacks on critical infrastructures, both in the private and public sector. They leverage advanced malware techniques to bypass anti-virus software and being stealth while conducting malicious tasks. Authors in [36] proposed a technique which focuses on defining rules to monitor the binaries used by threat actors to identify malicious behaviors. The solution starts from picking a set of APIs which are considered as malicious then hooking into the any running process. Then matching the selected set of APIs with running process APIs. However, this system lacks the ability to dynamically hook into any running process and there is no way to dynamically update the collections of APIs.

Authors in [37], proposed a fileless malware detection (FMD) tool for static memory forensics. For the input authors requires an image of memory snapshot at any time, which is then passed to the proposed tool, which then utilizes the APIs of another open-source tool called as Volatility (a memory forensic tool). At the end of analysis, the tool will show up the result if there is any malicious api get called by process it will be shown to the analyst. The main drawback of the proposed tool is that it completely resides on human interaction. An analyst is required to dump the memory to create a snapshot which is then manually input to the proposed system and the analysis process in completed.

Authors in [38], developed a host-based malware detection system for Supervisory Control and Data Acquisition System (SCADA). As most of the malware detection technique developed for SCADA system rely on the network packets. They won't inspect any abnormal behavior of the running processes on the system. The authors proposed solution hook into the memory of the running process using DLL injection. After hooking it employ the pattern base decision algorithm to make output related to a specific process that it is doing some malicious activity. The author's method lacks the ability to dynamically hook the runtime process.

Authors in [39], proposed a technique to mine the api from run time process. A collection data set is provided to lookup for any malicious api call. Then making the cluster for randomly called api by any running process. Finally, the collected data from process is passed to the machine learning algorithm which remove redundancy and extract malicious api only. There is no solution provide to dynamically link any process with the proposed solution so that whole process will get analyzed.

Authors in [40], proposed a hardware base AV detector and accelerator to detect fileless malware. Th proposed technique is implemented in memory controller of the system. The authors claim to have 100% accuracy as utilizes a sperate hardware accelerator for memory to complete the detection mechanism in no time. This technique is not feasible to every system around the world.

2.2 Critical Review

Fileless Malware leaves no traces for antivirus software to detect so that making it very hard for antivirus software. This section presents the critical review of existing approaches. Different techniques related to fileless malware detection are presented. The classification of these techniques is based upon the utilization of analysis type, technique, results, and layered wise employment.

2.2.1 Used Analysis Type

Many types of fileless malware analysis are proposed to detect the malware in real time or for forensic analysis. Three types of analysis types are utilized by different authors for malware detection.

Technique based upon static malware analysis are [24] [27] [35] [37]. Technique which uses dynamic analysis are [34] [36]. Finally, the proposed techniques which utilizes both static and dynamic analysis techniques are [25] [36] [28] [29] [31]

2.2.2 Used Analysis Technique

Malware analysis technique is how the detection process in run. Either it is automated completely or required user inputs manually. Two types of malware detection techniques are employed by different researcher.

The automatic malware detection techniques are employed by [31] [32] [35] [36] while the manual methods are used by [23] [24] [25] [26] [27] [28] [30] [33] [34]

2.2.3 Proposed Technique Results

Malware This shows how many of the proposed techniques verified their model by verification and validation of the results by utilizing any samples datasets.

Techniques which verify and validated their results based upon any sample data sets are [24] [31] [32] [34] [36] [37] [38] [39] while the techniques which didn’t verified their result and only provide theoretical model are [23] [25] [26] [29] [30] [33] [35]

2.2.4 Layered Wise Technique

Different authors use multiple layered techniques to detect fileless malware. Layer I is used where only process analysis is performed [25] [29] [31]. Layer II is employed when PowerShell and its commands detections is performed. [23] [24] [25] [27] [29] [30] [38]. Layer III is used by techniques when API hooking, and calls detections are monitored and analyzed to detect fileless malware [34] [36] [38].

Table 2.1: Comparison Between Existing Techniques and Reviewed Papers

Reference #	**Type**	**Analysis Type**	**Results**	**Limitations**	**Layered Wise Approaches**		
					Layer I	**Layer II**	**Layer III**
[23]	Hybrid	Manual	No	-NA-	✖	✔	✖
[24]	Static	Manual	Yes	• Lack of automation • Analyst input is required • Signatures are not Dynamically Linked	✖	✔	✖
[25]	Hybrid	Manual	No	-NA-	✔	✔	✖
[26]	Hybrid	Manual	No	-NA-	✖	✖	✖
[27]	Static	Manual	Yes	• No way to dynamically analyze the running process • Lack of Runtime analysis	✖	✔	✖

[28]	Hybrid	Manual	No	• No result validation • Lack of single architecture	✖	✖	✖
[29]	Hybrid	-NA-	No	• No result validation • Lack of single architecture	✔	✔	✖
[30]	Hybrid	Manual	No	• No way to execute the analysis during runtime	✖	✔	✖
[31]	Hybrid	Automatic	Yes	• Single layered detection can lead to high false positive • Lack of result validation to actual fileless malware	✔	✖	✖
[32]	Static	Automatic	Yes	• Can detect specific set of malware categories only • Lack of three-layered malware detection architecture	✖	✖	✖
[33]	Static	Manual	No	• Applicable to JavaScript web-based malware only	✖	✖	✖
[34]	Dynamic	Manual	Yes	• No ability to automate the process selection for analysis	✖	✖	✔

[35]	Static	Automatic	No	• Only single set of malware category in targeted for analysis and detection	✖	✖	✖
[36]	Dynamic	Automatic	Yes	• Lack of result validation with large number of malware dataset. • Lack of the ability to select the running process automatically	✖	✖	✔
[37]	Static	Manual	Yes	• Not applicable with any real fileless malware • Can detect PowerShell command only • No dynamic way to automate the process	✖	✔	✖
[38]	Dynamic	Automatic	Yes	• Can analyze with pre-defined api signatures only. • No dynamic signature update feature is provided	✖	✖	✔
[39]	Dynamic	Manual	Yes	• Can mine api calls of running process only. • No way to dynamically select the running process.	✖	✖	✔

Chapter 3

Proposed Methodology

A three-layered technique is proposed in this research which starts from monitoring the running process on the system, along with malicious PowerShell commands detection, and then move towards the run-time process's memory inspection to find out any malicious api calls. The whole process is dynamic and perform behaviors analysis on running process. Behavioral analysis is used minimize the limitations of static signature analysis. However, we have proposed the technique with the combination of behavior and static signature analysis. The final solution is completely automatic and requires no user interaction or input during run time.

3.1 Three Layered Architecture of Proposed Technique

The complete architecture of proposed technique in illustrated in Figure VI and the details and working of each layer along with their functions is presented in upcoming paragraphs along with their respective algorithm. The notations for symbols used in algorithm are described in Table IV. All layers are dynamically connected with each other and with an online database where all behaviors are pre stored.

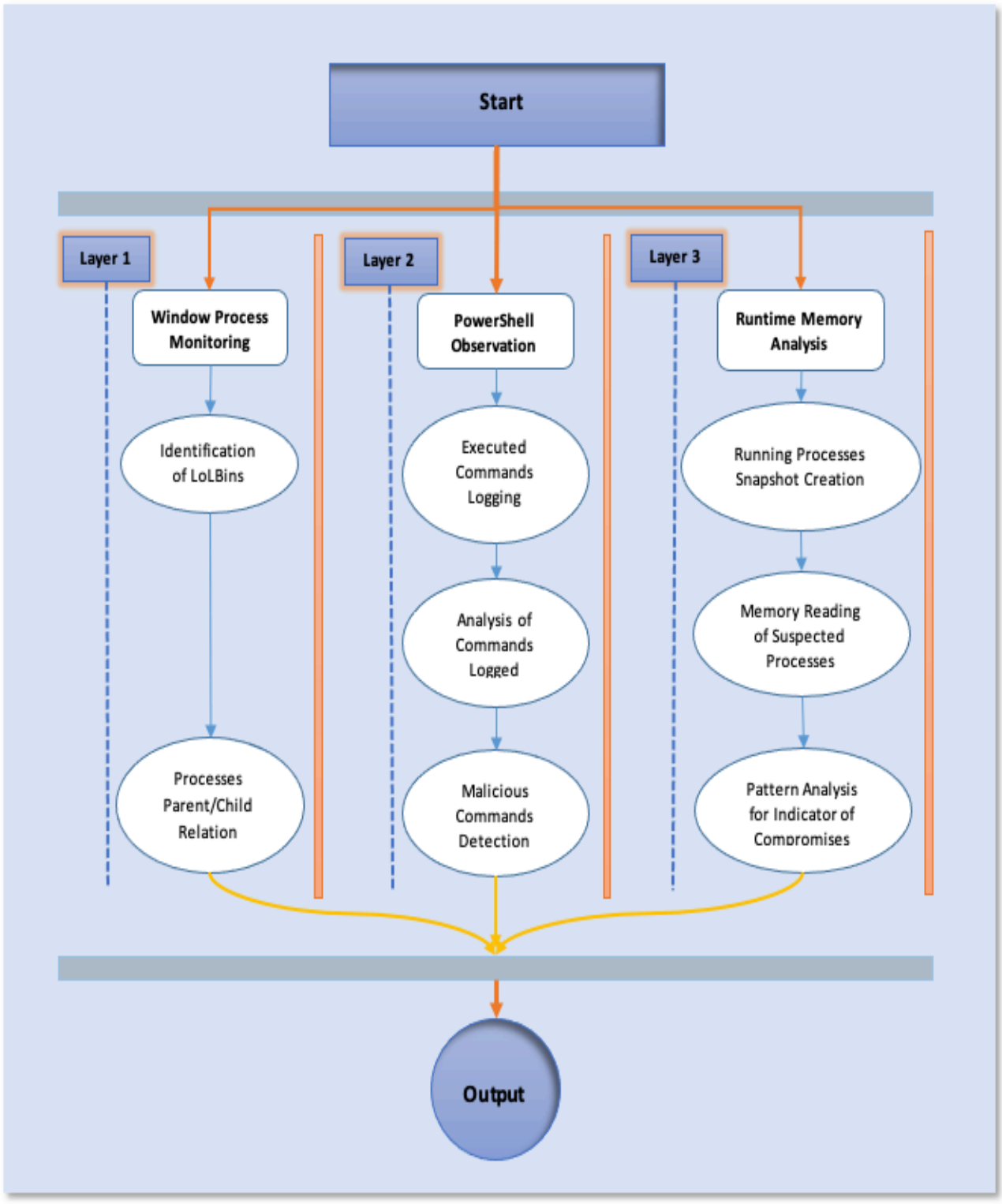
3.1.1 Layer I – Running Process's Analysis

The Layer I deal with capturing the details of running process on the system. The snapshot of running process is taken every second and the solution starts analyzing each process one by one. In the meantime, if a new process is created suddenly the solution will also capture it and starts monitoring and analyzing it. This layer will make a complete list of processes then build the child parent relationship to understand which process is parent or child of another process. After getting the relationship of processes with each other the layer will look up for any possible malicious processes chain.

The LOLBin's processes are the actual target of this layer if any of such process found running on system then it's child and parent process will be analyzed. There are

multiple cases how the malicious process can be detected in this layer. The complete steps of the Layer I are shown in Algorithm I.

Figure 3.1: Architectural Diagram of Proposed Multilayered Fileless Malware Detection System



Case:1 A PowerShell (powershell.exe) process is found running/started on the system. Layer I will look up for its parent if any of LOLBin (wscript.exe) is its parent the process will be considered as specious and will alert the second layer for further analysis.

Case:2 A PowerShell process is a child of other processes like (Microsoft Word, Excel, PowerPoint, and Outlook etc.).

Case:3 PowerShell Process is child to any of other process in any processes chain like explorer.exe => cmd.exe => powershell.exe

Algorithm 1	**Runtime Process's Analysis**

START
Log P_{LIST}
Lookup P in P_{LIST}
    if {P_N == P_{SHELL}}
        then PS = P, and Add PS in X_{LIST}
            again, Lookup PS inP_{LIST}
                If PS_{PID} == P_{ID}
                    then PS = P
                        again, Lookup PS in MP_{LIST}
                            if PS_N == MP_N
                                then, PM = P, and *SET* P_{FLAG} = True
                            else return
                else return
            again, Lookup PS inP_{LIST}
                If PS_{ID} == P_{PID}
                    then Add P_{PID} in Y_{LIST}
                else return
    else return
END

Table 3.1: Notations List Used in Algorithms

Notation	Description
P_{LIST}	List of running processes
P	a process from running Processes List P_{LIST}
N	$_N$is process name
ID	$_{ID}$ process identifier
PID	is process parent identifier
SHELL	is a PowerShell process
PS	*PS* is suspected process
PM	is possible malicious process
MP_{LIST}	is malicious parent process list
MP_N	is name of suspected malicious parent process
X_{LIST}	is a list of P_{SHELL} processes running on system
P_{FLAG}	P_{FLAG} is the output of each layer
MC_{SIG}	*is a list of all malicious commands to be used by* P_{SHELL}
PMC_{LIST}	*is list of malicious commands used by P.*
MC	*is malicious command*
C	$_C$ *is any command run by* P_{SHELL}
MC_{SET}	set of malicious commands.
Y_{LIST}	Y_{LIST} is list of processes child to a specific P_{SHELL}
H_{LIST}	*collections of various hooks*
H	*is a hook called by process*
PH_{LIST}	*is collection of hooks called by P*
MH_{SET}	*is set of malicious hooks*

3.1.2 Layer II – PowerShell Analysis

The Layer II deals with analyzing the PowerShell process only. When PowerShell is detected by Layer I, its alert layer 2 along with process id (PID). Now Layer II will keep monitoring the PowerShell until the process is exited. The logging system is used to record all commands run or executed by PowerShell process. After logging of the

commands, each command is being analyzed using some pattern recognition technique. If any malicious command is found it will be recorded and further operations are taken.

The PowerShell commands are not actually malicious in nature but are abused by adversaries to craft their attacks. The list of such commands can be up to hundreds in numbers but here are few commands, shown in Table V.

The attackers combine multiple PowerShell commands to create a special attack which is not easy to detect. However, our technique analyzes each command executed on PowerShell even after some time. Some anti virus's solution looks for a specific process for a specific time. A malware can be sleep for some time even for a day using timer. After some time, they can download a malicious payload and execute it directly into memory. But our technique can analyze all commands whenever it is executed on any PowerShell process. The working steps of the layer II are shown in algorithm II.

Algorithm II	**PowerShell Commands Analysis**

START

For all, P in X_{LIST}

lookup MCin MC_{SIG}

if P_C == MC

then, Add P_C in PMC_{LIST}

else return

lookup PMC_{LIST} for P

if PMC_{LIST} ==MC_{SET}

then *SET* P_{FLAG} = True

else return

END

Table 3.2: PowerShell Example Commands Abused by Malware For fileless Attacks

Commands	**Description**
• Invoke-WebRequest • Invoke-RestMethod • Start-BitsTransfer	to download from remote address
• Start-Process	to start a new process
• OpenProcess	to open any running process
• WriteProcessMemory	to write into process memory
• Get-ProcAddress	to look for specific process address
• LoadLibrary	to load a module into memory address
• AdjustTokenPrivileges	to enable/disable privileges token
• Invoke-ConPtyShell	to start reverse tcp shell for windows
• mimikatz	to start post exploitation framework
• net.webclient	to send/receive data via http using .Net
• stringtobase64	to convert string to base64
• port-scan	to start port scanning on network
• reflectivepeinjection	to load .exe reflectively to process
• invoke-psinject	to load decode code logic into process
• invoke-decode	to decode the encode PowerShell script
• gzipstream	to compress/decompress exe to gzip stream
• http-backdoor	to download more PowerShell scripts
• add-persistence	to add windows script to run auto via registry

3.1.3 Layer III – Run Time Processes API Analysis

The layer III deals with the run time analysis of processes. Any process which is listed in possible suspectable malicious chain it will be auto analyzed via Layered III. APIs are built in function of windows system whose actual purposes was to build non-malicious application. But adversaries abuse the power of such api and used them for

malicious purpose. This layer starts with hooking of the target process. Hooking is a technique to introspect the api called by the target process. It will start by injecting an Inspector DLL hooking engine into the target process. The hooking engine will now handle all api calls through itself. All malicious api calls are loaded into the engine. If any similar api is get called by the process the engine will alert the system that the similar api call is found.

There are thousands of windows api which are used by begin programs as well as malicious programs. All popular malicious api are being stored in live database and then downloaded by our proposed solution. Some popular malicious api which can be used by malware for malicious purposes are shown in table VI. The step wise working of layer three is defined in Algorithm III.

Table 3.3: Example of Windows Built-in Functions or API Calls abused by malware for Fileless Attack

Commands	**Description**
• CreateToolhelp32Snapshot	used to create snapshot of running processes, thread etc on system
• CryptAcquireContext	used to start encryption system on windows mostly used by ransomware
• EnableExecuteProtectionSupport	used for modification of Data execution protection setting
• EnumProcessModules	to lookup modules loaded by process
• FindFirstFile/FindNextFile	to iterate through directory and file systems
• GetAsyncKeyState	to capture keystrokes of keyboard widely used by keyloggers
• InternetOpenUrl	to open up http, ftp based internet connection
• IsWoW64Process	to detect whether the process is running on windows 64 or not
• LdrLoadDll	low level api to perform dll injection
• MapVirtualKey	for translation of key-code into respective key value

• NetScheduleJobAdd	to add a request to run program automatically on specific condition
• OpenMutex	used by malware to make sure only one instance of its is running
• SamIGetPrivateData	to get private information of user from security account manager of windows
• ShellExecute	to execute another program
• Toolhelp32ReadProcessMemory	to read memory of any other running process
• VirtualProtectEx	to change read only portion of memory into writeable

Algorithm III	**Layer 3**

START

Load H_{LIST}

For all, P in Y_{LIST}

HOOK P

if P_H is called

Then add P_H is PH_{LIST}

else return

Lookup, PH_{LIST} for P

if MC_{SET} ==PH_{LIST}

then *SET* P_{FLAG} = True

else return

END

3.2 Interconnected Layered Wise Behavior Analysis

All three layers which we have discussed previously are interconnected with each other. Each layer has its own unique functions, which are performed completely automatic. In behavior analysis some actions are monitor accordingly to detect a suspected behavior. Behavior analysis can help to detect any abnormality in the target system.

Traditional antiviruses use the signature-based detection technique, for which they have some specific hash generated and store in a database. Whenever they are analyzing a new object on system, they calculate the sample hash or signature and then match it with the stored database. If positive signature hash found the sample will be consider as malicious. While behavioral analysis keeps analyzing different aspects of the target object. In the proposed approached all three layers are working on behavioral analysis.

The proposed solution is divided into two components.

1. **Client:** This component will be a pc or any windows-based device where the proposed solution will be run in order detect fileless malware.

2. **Cloud:** This component is server based online solution from where a malware analyst add/edit/delete behavior for the client component.

The approach begins with the download of Indicators of Behavior (IOBs) from the cloud server and the subsequent monitoring of the currently operating process. If any of the suspicions raised by Layer I are confirmed, Layer II will be invoked, and layer II procedures will be carried out with the goal of analyzing the PowerShell processes. Following the completion of the layer I and II operations, the layer III operations will be performed. Finally, the output of all three layers will be used to decide as to whether or not the target sample is malicious. If malicious behavior is discovered, the malicious process chains are terminated as soon as they are discovered. Each of these three levels is linked, and they all make use of a behavior-based detection approach for malware identification and mitigation.

Chapter 4

Proposed Technique Design and Implementation

Here we will go over all the preconditions, for both hardware and software components, malware sets of data, and what we will implement the new fileless malware detection method. We will also go over how effectively our true to its mission and how effective it is for the malware sample datasets that we have provided.

4.1 Overview

An antimalware approach is presented in this chapter, along with its formulation and construction, to limit the danger of new and atypical malicious programs. Constant behavior monitoring and the determination of both the rightness among those behaviors at program execution is among the possible approach to overcome the constraints of state-of-the-art malware detection methods when dealing with non - conventional malicious programs, so it is currently under investigation. The reference implementation methodology is used in conjunction with current malware detection systems to improve the efficiency of the detection method, which is necessary due to the limitations in known malware detection approaches. That's why we offer a technique based on the continuous behavior analysis, which would be validated against application specifications or policies, to detect future attacks that do use advanced obfuscation methods to conceal from conventional anti-malware capabilities. We believe that this approach will be effective in this situation. Based solely on the fact that advanced malware employs fileless attack techniques and makes use of legal technologies for malevolent objectives, this detecting strategy makes it hard for current antimalware methodologies to identify such assaults. There are two types of malware detection techniques included in the proposed method: reference implementation malware identification and behavior patterns for malware detection.

To begin, we must write application specs that may be stated in a document to explain the intended behavior of that application. This is the first stage in our strategy. It is this new technique that makes use of specs that explain the anticipated performance that the

application is supposed to achieve. These pre-defined requirements are then compared to the actual behavior of the programmer as it occurs throughout the run-time process. A behavior violation occurs when any one of the action events does not correspond. This causes the action to be flagged as malicious activity and alerting system administrators.

Application functionality or actions are monitored by comparing their behavior to the behavior requirements that have been created to capture the right behavior of the actions. This is known as specification-based monitoring. It is common practice to create specifications by hand, taking into consideration security policy, object functionality, and anticipated use. It is thus feasible to identify malicious changes that may modify or change the intended behaviour of a programme if the intended behaviour of the program has been clearly pre-determined. The new technique consists of a series of processes that must be followed, and significant occurrences must be studied and reviewed to ensure that they have not been tainted by evil intent. The suggested malware detection approach reaps the advantages of executing the detection process in real time and monitoring the actions of the programme while it is running.

Techniques that detect anomalies in the running behaviour of an application also perform operations by creating profiles of the application's normal behaviour, which is typically formed through automated training, and then comparing them to the actual movement of the system to detect any significant changes in the running behaviour. In certain cases, anomaly detection may identify unknown threats, but it has a significant false alarm rate, which makes it difficult to use. Furthermore, this method is not a good choice in the case of unconventional cyberattacks because of their specific characteristics, which include the ability to utilize the already current non-malicious tools and applications and the possibility of being marked as routine usage by anti-malware tools that employ anomaly-based detection techniques.

A further approach, called specification-based detection, involves manually abstracting and crafting the right behaviors of essential objects into behavior specifications, which are then compared to the actual behaviour of those objects. Reference implementation surveillance is the most effective anti-malware strategy for providing the best possible protection. When compared to anomaly-based detection methods, it offers many

benefits, including the ability to create policies with more flexibility and the ability to produce a lower number of false positives.

Table 4.1: Tools and Software Used for Development of Proposed Three Layered

Sr No.	**Name**
1	**OS Used for Development and Experiment:** • Windows 7 • Windows 8 • Windows 10
2	**Virtual Environment for Development and Experiment:** • VMWare Fusion • Virtual Box
3	**Programming Languages:** • C++ 17 • JavaScript
4	**Code Editors:** • Visual Studio Community 2020 • Visual Studio Code
5	**Other Framework/Libraries Utilized:** • ReactJS • NodeJS • NektraDeviare Hooking Engine

Fileless Malware Detection Technique

4.2 Architecture

Because we were attempting a novel malware detection method, we constructed two antimalware strategies in such a manner that they could be viewed as an efficient and productive detection mechanism. The structure of the suggested approach, seen in Figure I, is composed of two modules: a behaviour monitors component and a specifications processing component, respectively. The behavior watching module analyses the activity carried out by the application, and the specification matching module compares those actions to the anticipated behavior that has been stated before in the specification. Detailed descriptions of the layered wise component are of the suggested system are given below.

4.2.1 Layer I (Processes Watcher)

Layer I is the detection and identification of PowerShell process running within the windows-based computer and finding out relationship between child and possible malicious parent process. The first step in layer I is to take snapshot of the running process on the system, and which is done via using CreateToolhelp32Snapshot built-in function of Tool Help library of windows system.

The memory snapshot is taken every second automatically in order to actively monitored any new created process. A periodic function is created that will keep active the ProcessWatcher Function to effectively monitor and observer the processes relations ships with other process on time.

4.2.1.1. Process

It is a programme that is now in continuous running, often known as a process. More than just the programme code, it also contains the sequence number, processes stacks, caches, and other data structures that are used by the programme code. In comparison, the computer code consists simply of the text component.

During the course of its execution, a process undergoes changes in status. Depending on the present activity of a process, its condition may be influenced to some extent. New, ready, operating, blocked, and terminated are the many states that a program might be in throughout its execution.

Every one of the activities is controlled by a central processing unit (PCBC). This file includes critical information on the matter with which it is connected, such as the process status, the process number, the programme counter, a list of folders and variables, CPU data, and RAM information, among other things.

4.2.1.2. Parent Process

A process that runs the fork() system call, with the exception of the initial process, will generate all of the processes in the operating system. Typically, the process that called the forks() callback is referred to as the parent process. In other terms, a parental process is a process that initiates the creation of a child procedure. Unlike a child process, which may have many parent processes, a parent process can only have one parent process. On successful completion of a fork() function call, a PID of both the child process is passed to the parent process, and the value 0 is provided to the parent process. When the fork() system call fails, the result is sent towards the parent process as -1, and no child process is generated as a result.

4.2.1.3. Child Process

During the course of an operating system's operation, a child process is defined as a process generated by only a parent process using the fork() system function. A child process is also referred to as a subsystem or a sub - tasks in certain cases.

The child process is generated as a clone of its parent process, and it retains the majority of the properties of its parent process. The kernel produced a child process if there's no parent process for the child process to belong to.After a departure or interruption of a child process has been detected, the SIGCHLD signal is being sent to the parent process.

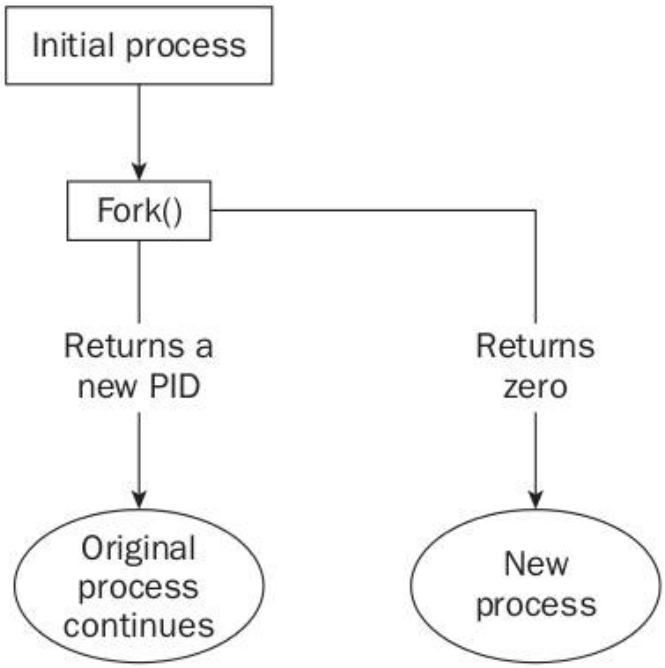


Figure 4.1: Process Parent Child Relation and New Process Creation

A memory snapshot is taken every second and from which a collection of running process is taken and transferred to other function for child parent relation identification. The metadata of processes are

- Process ID
- Process Name
- Process Parent ID
- Process Parent Name
- Process Chain ID

The parent ID is used to identify the malicious parent process which are suspectable. First PowerShell process are located, and their parent IDs are noted if any of the parent is suspectable for suspicious activity the parent ID is noted and added to a new chain. The chain will contain all the child and parent processes of PowerShell. This chain ID will be then used for furthers layers. The periodic function definition is present below.

```
template<typename TimeT = std::chrono::milliseconds>  <T>
struct periodic
{
    periodic(TimeT duration = TimeT(1))
        : start(std::chrono::system_clock::now())
        , period_duration(duration)
        , previous_duration(TimeT::zero())
    {};
    template<typename F, typename ...Args>
    TimeT execution(F func, Args&&... args)
    {
        auto duration = std::chrono::duration_cast<TimeT>
            (std::chrono::system_clock::now() - start);
        if (duration > previous_duration + period_duration)
        {
            std::forward<decltype(func)>(func)(std::forward<Args>(args)...);
            previous_duration = duration;
        }
        return duration;
    }
    std::chrono::time_point<std::chrono::system_clock> start;
    TimeT period_duration;
    TimeT previous_duration;
};
```

Figure 4.2: Periodic Function to use to take snapshot of memory

```
int GetProcessList()
{
    struct ProcessX
    {
        int processID;
        wchar_t processName[260];
        int processParentID;
    };
    HANDLE hProcessSnap;
    HANDLE hProcess;
    PROCESSENTRY32 pe32;
    DWORD dwPriorityClass;
    hProcessSnap = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    if (hProcessSnap == INVALID_HANDLE_VALUE)
    {
        _tprintf(TEXT("\nEROOR> CreateToolhelp32Snapshot  %s"));
    }
    pe32.dwSize = sizeof(PROCESSENTRY32);
    if (!Process32First(hProcessSnap, &pe32))
    {
        _tprintf(TEXT("\Process32First  %s"));
        CloseHandle(hProcessSnap);
    }
    int pSize = 0;
    struct ProcessX sArray[500];
    do
    {
        dwPriorityClass = 0;
        hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pe32.th32ProcessID);
        if (hProcess == NULL)
            _tprintf(TEXT(""));
        else
```

Figure 4.3: Function to Get Process List of the Running Process on the test system

After getting the processes list of running process on windows system, each process is analyzed for its parent process if the parent process is one of the pre-defined suspicious process the process chain is recorded and alerted by system to other layers.

For example, a powerhsell.exe process is located with PID 121 the next step which is performed is to locate it parent process with PID 114. If parent process is matched with the defined signatures of malicious processes. All their parent and child processes are added to single chain which will be then used for further analysis and identification of future child processes. The utilization of periodic function will help the layer I to actively monitor the new list of running process on windows system. The periodic functions run every seconds thus every new process is get caught by the layered system. If any process gets exited of killed intentionally by the user the process list get update automatically when the periodic functions keeps running on the system.

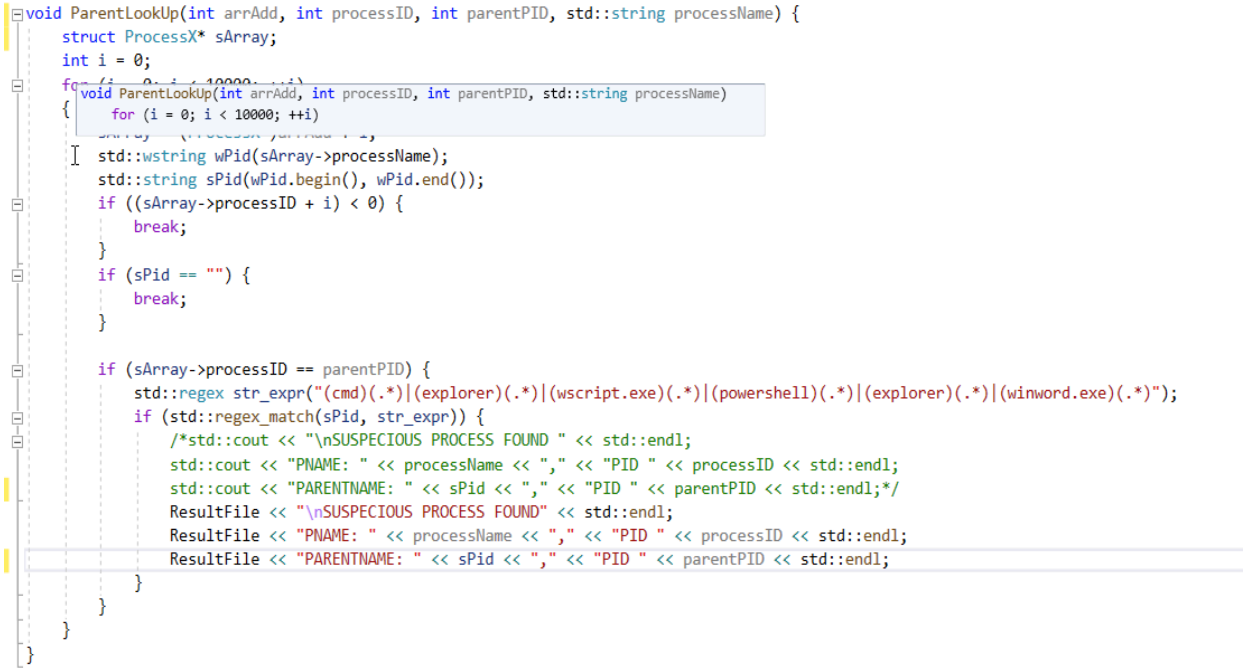


Figure 4.4: Function to Lookup for the Parent Process and find out the relationship between parent and child process.

The parent processes which are to be monitored by the layered will be downloaded in first initialization of the proposed program and then stored within the program so that it can be easy for the program to search and find for a match of suspected parent process. The downloading of signatures will be discussed with details in upcoming section.

ProcessAlerter function will alert other layers that some processes are suspected of further investigation and analysis. Other layers will check the chain and if PowerShell process is located the layer II function started working automatically and if other processes are located the layer II functions get started too for further memory analysis.

```
void ProcessAlerter(int* carrAdd)
{
    struct ProcessesChain* scArray;
    int i = 0;
    for (i = 0; i < 10; ++i)
    {
        scArray = (ProcessesChain*)carrAdd + i;
        std::cout << "SUSPECIOUS CHAIN " << scArray->chainID << std::endl;
    }

}
```

Figure 4.5: Alert Function to Alert the other layers that a suspicious chain is found

4.2.1.4. CreateToolhelp32Snapshot

To deliver their findings, some other tool assist methods evaluate the snapshot obtained by this method and compare it from their own. The snapshot can only be accessed in read-only mode. It behaves similarly to an item handling and is subject to restrictions as an object handle in terms of which tasks and threads it may be associated with. In order to iterate the stack or module contents for all activities, use the TH32CS SNAPALL option and set the th32ProcessID parameter to zero.

For each subsequent process in the snapshot, run CreateToolhelp32Snapshot again, this time supplying the process identification as well as the TH32CS SNAPHEAPLIST or TH32 SNAPMODULE value for the process.There are a multitude of reasons why the CreateToolhelp32Snapshot method may fail or produce inaccurate information when taking snapshots that contain heaps and modules for just a process besides the current process.

A function call may fail with ERROR BAD LENGTH or another error code if indeed the loader data structure inside the target database has been damaged or has not been initialized, or if the package list alters as a consequence of DLLs being imported or unloaded while the function is being called. Examine if the target program was

launched in a sort of limbo, and then attempt to invoke the function a second time. It may be necessary to call the method several times before it succeeds. If the function returns an error with the ERROR BAD LENGTH when contacted with TH32CS SNAPMODULE or TH32CS SNAPMODULE32, it may be necessary to call the method again until it fails.

Interfaces for module that have been loaded with the LOAD LIBRARY AS DATAFILE or equivalent flags are not returned by the TH32CS SNAPMODULE and TH32CS SNAPMODULE32 flags. More information may be found at LoadLibraryEx. CloseHandle is a function that may be used to destroy a snapshot.

To acquire a list of currently running programs, the following basic console programme is used: After creating a snapshot of the currently running processes in the scheme using CreateToolhelp32Snapshot, and afterwards walking through the list of processes documented in the single image using Process32First and Process32Next, the GetProcessList function returns a list of all the processes currently running in the system. GetProcessList invokes the ListProcessModules function, which is explained in more detail in Transiting the Modules Collection, and the ListProcessThreads function, which is discussed in more detail in Traversing the Thread List, for every activity in the list that is returned. Any failures, which are often caused by security limitations, may be identified by using a simple mistake function such as printError. In the case of the Idle and CSRSS processes, for instance, OpenProcess refuses to open them because their access limitations prohibit user-level code from doing so.

4.2.1.5. Process32First

In a system snapshot, this function returns information on the first process that was encountered. If the first element in the process table has been transferred to the buffers, this function returns TRUE; otherwise, it returns FALSE. The ERROR NO MORE FILES error result is provided either by GetLastError method if there are no processes running or if the snap does not include process information for running processes. The dwSize element of PROCESSENTRY32 must be set by the caller programme to the length, in bytes, of the object being processed. The Process32Next method may be used

to get details about some of the other processes that were captured in much the same snap as the current one.

4.2.1.6. Process32Next

This function returns metadata about for the next process that was captured in a computer snapshot. Unless the next entry in the process table has been transferred to the buffer, this function returns TRUE, otherwise it returns FALSE. If no process exists or if the snap does not include process information, the GetLastError method returns the error value ERROR NO MORE FILES. Process32First is a function that may be used to get metadata about first process that was captured in a snapshot.

4.2.2 Layer II (PowerShell Analyzer)

Layer II deals with steps and function utilized to log and monitors the PowerShell commands executed on any of the PowerShell process running on the system. If there are multiples PowerShell process running on system at a time the layer II can still work simultaneously and keep an eye on each PowerShell process logs their commands which are executed and further perform analysis to detect any suspicious or malicious commands.

4.2.2.1. PowerShell

PowerShell is a cross-platform task automation solution made up of a command-line shell, a scripting language, and a configuration management framework. PowerShell runs on Windows, Linux, and macOS. As a scripting language, PowerShell is commonly used for automating the management of systems. It is also used to build, test, and deploy solutions, often in CI/CD environments. PowerShell is built on the .NET Common Language Runtime (CLR). All inputs and outputs are .NET objects. No need to parse text output to extract information from output. The PowerShell scripting language includes the following features:

- Extensible through functions, classes, scripts, and modules

- Extensible formatting system for easy output
- Extensible type system for creating dynamic types
- Built-in support for common data formats like CSV, JSON, and XML

A management program in PowerShell known as Desired State Configuration (DSC) allows you to manage your company infrastructure by writing configuration as code. You can do the following using DSC:

- Create declarative setups and bespoke scripts for repeated deployments using the Scripting Language.
- Configuration settings should be enforced, and configuration drift should be reported.
- Configuration may be deployed using either the push or pull models.

4.2.2.2. PowerShell For Malicious Purposes

PowerShell is a sophisticated Windows scripting language that is utilised by both IT professionals and their opponents, according to a recent study. PowerShell is favoured by attackers for a variety of reasons:

- It is a command-line utility that comes pre-installed.
- It has the ability to download and run code from another system.
- In the case of Windows systems, it gives unrivalled access.
- Sys admins utilize PowerShell to manage a variety of operations (for example, shutting down your PCs regularly at Twelve a.m.—do this by using task scheduler).
- PowerShell is enabled on the majority of systems.

Its malicious usage is often undetected or undetected even by typical endpoint defenses since files and instructions are not saved to disc when they are executed. As a result, there will be fewer artefacts to retrieve for forensic examination.

Numerous malicious applications that are based on or utilize PowerShell are available, including the following:

- Metasploit
- Mimikatz
- Empire
- PowerSploit

4.2.2.3. Mimikatz

Mimikatz, which the author describes as "a small tool to experiment with Windows security," is a highly successful offensive security tool created by Benjamin Delpy that is characterised as "a little tool to toy with Windows security." Both ethical hackers and malware producers make use of this tool on a regular basis. The devastating NotPetya virus of 2017 combined revealed NSA vulnerabilities such as EternalBlue with Mimikatz to do the most amount of harm possible.

Mimikatz, which was originally developed by Delpy as a research project to better understand Microsoft security, now contains a component that dumps Minecraft from memory and shows you where all the mines are hidden.

Mimikatz is a simple tool of using, and Mimikatz v1 is included as a meterpreter program as part of the Metasploit security framework. As of this writing, the latest Mimikatz v2 update has still not been included into Metasploit's core framework.

The term "mimikatz" is derived from a French slang term "mimi," which means "cute," and so "cute cats." The author (Delpy), who is French, writes in his own tongue for the Mimikatz site.Among the most well-known Mimikatz assaults were the Petya and NotPetya attacks, which infected thousands of computers throughout the globe in 2016 and 2017. A virus called NotPetya, which is similar to the Petya virus, infects a computer system, encodes any data stored on the device, and shows a message to the

victim instructing him or her on how to transfer bitcoins in attempt to restore the encrypted information.

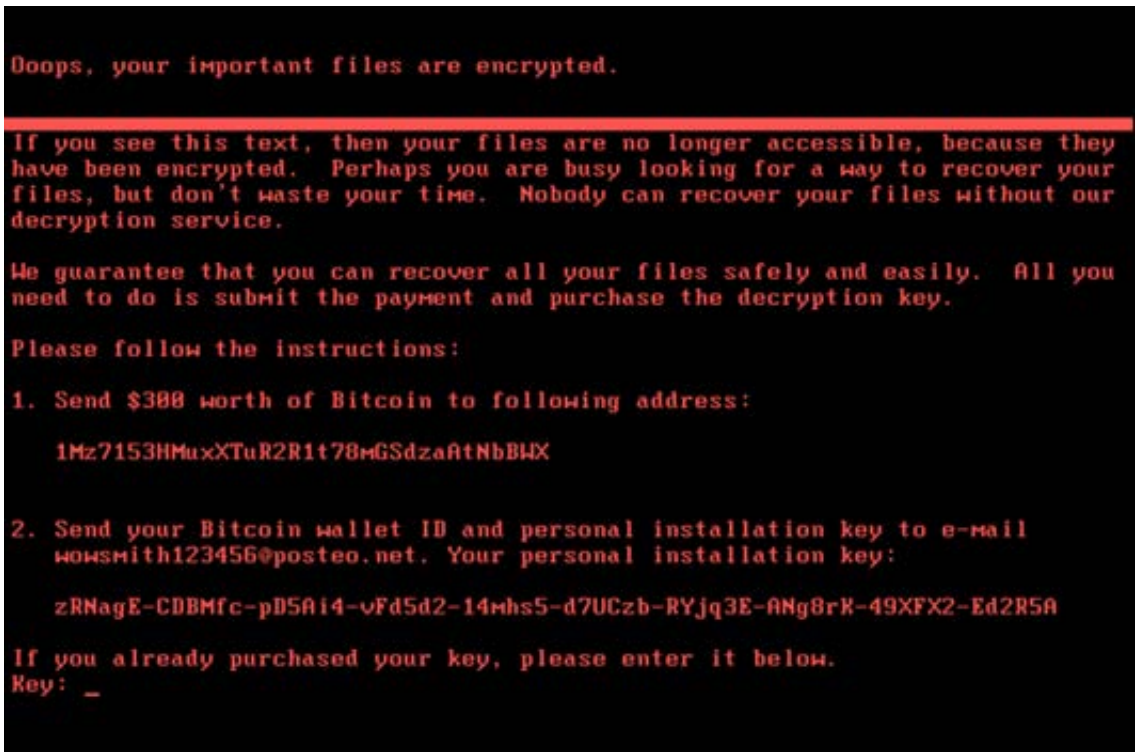


Figure 4.6: NotPetya Ransomware note which displayed on victim machine

Check out this PowerShell script that was written to get access to a fileless network and to run malicious code from the command line. Through the commandline, this command sends the received data located at the following URL to PowerShell, which then executes the file "in ram" on the target computer: This command passes all file located at the following URL to PowerShell:

```
1 powershell.exe "IEX (New-Object
2 Net.WebClient).DownloadString('https://raw.githubusercontent.com/mattifestation/
3 PowerSploit/master/Exfiltration/Invoke-Mimikatz.ps1'); Invoke-Mimikatz -DumpCreds"
```

Figure 4.7: PowerShell Command to Invoke (Download) Mimikatz to Dump OS credentials

- The IEX or Invoke-Expression function is responsible for running the function specified on the local computer.

- A new object of a.NET Framework download string is created by using the New-Object cmdlet. download the contents of GitHub into a program memory, which will be used by IEX to execute the code.
- The DumpCreds option informs Mimikatz to extract passwords from the LSASS database.

In order to detect similar type of attacks the layer II will take runtime actions during malicious PowerShell commands executions. The process starts from logging of each command executed on all running process. For each PowerShell process running on the system a log files are created automatically where the commands are logs that are executed on the PowerShell process. Even if the process is under sleep mode to avoid early detection and woke up after few times and start executing malicious commands the proposed technique will still logs the new commands in the similar file where if any old commands executed by same PowerShell process. The FileWatcher function will keep an eye on all created logs file of PowerShell process and then pass the individual process log's one by one to FiltersCommands function.

```
void FileWatcherMain(void*) {
    GetCurrentDate();
    cout << ">> WATCHER ACTIVE FOR POWERSHELL" << " C:/Plogs/" + current_date << endl;
    LogFileWatcher fw{
            "C:/Plogs/" + current_date , std::chrono::milliseconds(4000)
            //"C:/Plogs/20210128" , std::chrono::milliseconds(4000)
    };
    fw.start([](std::string path_to_watch, FileStatus status) -> void {
        if (!std::filesystem::is_regular_file(std::filesystem::path(path_to_watch)) && status != FileStatus::erased) {
            return;
        }
        switch (status) {
        case FileStatus::created:
            //cout << "Log Created " << endl;
            FilterCommands(path_to_watch);
            break;
        case FileStatus::modified:
            //cout << "Log Updated " << endl;
            FilterCommands(path_to_watch);
            break;
        case FileStatus::erased:
            break;
        default:
            std::cout << "Error! Unknown file status.\n";
        }
        });
}
```

Figure 4.8: Function to Generate Logs file of each running PowerShell process

An example of created logs file of individual process with ID 3944 is show in fig belowthe meta data of logs file

- StartTime
- Windows User Account
- PowerShell Version
- Process ID
- PowerShell Version
- PowerShell Build Version
- PowerShell Edition Name
- All Commands Executed

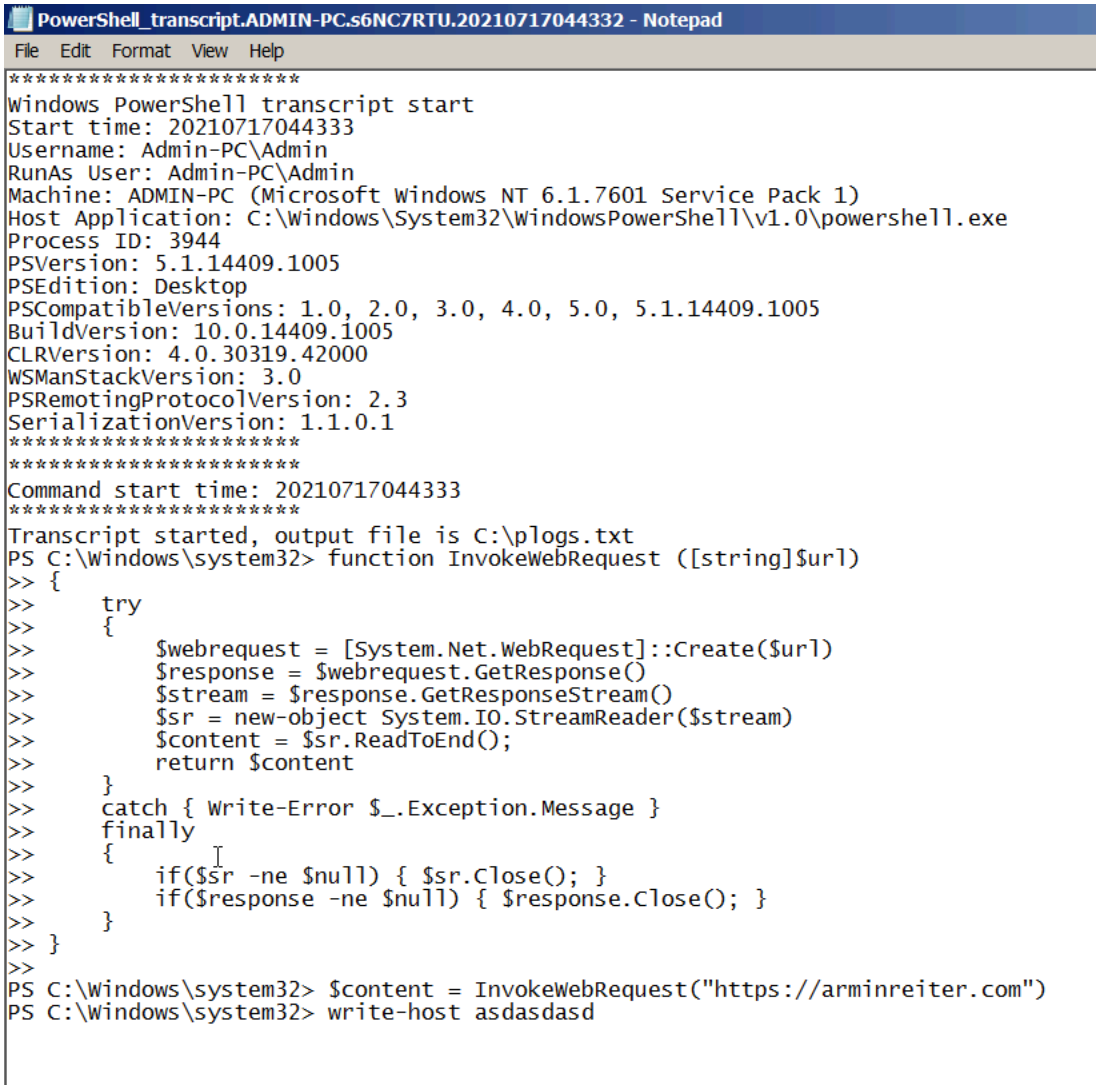


Figure 4.9: Preview of Generated Logs File of Individual PowerShell Process

The malicious PowerShell commands are early recorded into the system upon which the string analysis is performed using regex. The logs files are compared line by line and string by string and character by characters with the malicious regex.

The string iterator function will iterate into the logs file and compared all commands executed by the processes with the malicious regex If any of similar or suspected malicious commands found in the logs file the system will be alerted.

```
static std::string regexString() {
    std::string rxString = "Process ID|write-host|System.Net|InvokeWebRequest|System.IO.StreamReader|StreamReader|WebRequest|Invoke-
      Mimikatz|DownloadString|MapVirtualKey|GetKeyboardState|DownloadFile|write-host|Start-Dnscat2|-Recurse|Compress-Archive|Invoke-
      DllInjection|Invoke-DllInjection|Invoke-Shellcode|Invoke-WmiCommand|Get-GPPPassword|Get-GPPPassword|Get-Keystrokes|Get-
      TimedScreenshot|Invoke-CredentialInjection|Invoke-CredentialInjection|Invoke-Mimikatz|Invoke-NinjaCopy|Invoke-TokenManipulation| Out-
      Minidump|Out-Minidump|VolumeShadowCopyTools|Invoke-ReflectivePEInjection|Invoke-UserHunter|Find-GPOLocation|Invoke-DowngradeAccount|
      Invoke-DowngradeAccount|Get-ServiceUnquoted|Get-ServicePermission|Get-ServiceFilePermission|Invoke-ServiceAbuse|Install-
      ServiceBinary|Get-RegAutoLogon|Get-VulnAutoRun|Get-VulnSchTask|Get-UnattendedInstallFile|Get-ApplicationHost|Get-
      RegAlwaysInstallElevated|Get-Unconstrained|Add-RegBackdoor|Add-ScrnSaveBackdoor|Gupt-Backdoor|Invoke-ADSBackdoor|Enabled-
      DuplicateToken|Invoke-PsUaCme|Remove-Update|Check-VM|Get-LSASecret|Get-PassHashes|Show-TargetScreen|Port-Scan|Invoke-PoshRatHttp|
      Invoke-PowerShellTCP|Invoke-PowerShellWMI|Add-Exfiltration|Add-Persistence|Do-Exfiltration|Start-CaptureServer|Get-ChromeDump|Get-
      ClipboardContents|Get-FoxDump|Get-IndexedItem|Get-Screenshot|Invoke-Inveigh|Invoke-NetRipper|Invoke-EgressCheck|Invoke-PostExfil|
      Invoke-PSInject|Invoke-RunAs|MailRaider|New-HoneyHash|Set-MacAttribute|Invoke-DCSync|Invoke-PowerDump|Exploit-Jboss|Invoke-
      ThunderStruck|Invoke-VoiceTroll|Set-Wallpaper|Invoke-InveighRelay|Invoke-PsExec|Invoke-SSHCommand|Get-SecurityPackages|Install-SSP|
      Invoke-BackdoorLNK|PowerBreach|Get-SiteListPassword|Get-System|Invoke-BypassUAC|Invoke-Tater|Invoke-WScriptBypassUAC|PowerUp|
      PowerView|Get-RickAstley|Find-Fruit|HTTP-Login|Find-TrustedDocuments|Invoke-Paranoia|Invoke-WinEnum|Invoke-ARPScan|Invoke-PortScan|
      Invoke-ReverseDNSLookup|Invoke-SMBScanner|Invoke-Mimikittenz|Invoke-AllChecks|Add-ConstrainedDelegationBackdoor|Set-
      DCShadowPermissions|DNS_TXT_Pwnage|Execute-OnTime|HTTP-Backdoor|Set-RemotePSRemoting|Set-RemoteWMI|Invoke-AmsiBypass|Out-CHM|Out-HTA|
      Out-SCF|Out-SCT|Out-Shortcut|Out-WebQuery|Out-Word|Enable-Duplication|Remove-Update|Download-Execute-PS|Download_Execute|Execute-
      Command-MSSQL|Execute-DNSTXT-Code|Out-RundllCommand|Copy-VSS|FireBuster|FireListener|Get-Information|Get-PassHints|Get-WLAN-Keys|Get-
      Web-Credentials|Invoke-CredentialsPhish|Invoke-MimikatzWDigestDowngrade|Invoke-SSIDExfil|Invoke-SessionGopher|Keylogger|Invoke-
      Interceptor|Create-MultipleSessions|Invoke-NetworkRelay|Run-EXEonRemote|Invoke-Prasadhak|Invoke-BruteForce|Password-List|Invoke-
      JSRatRegsvr|Invoke-JSRatRundll|Invoke-PoshRatHttps|Invoke-PowerShellIcmp|Invoke-PowerShellUdp|Invoke-PSGcat|Invoke-PsGcatAgent|
      Remove-PoshRat|Add-Persistance|ExetoText|Invoke-Decode|Invoke-Encode|Parse_Keys|Remove-Persistence|StringtoBase64|TexttoExe|
      Powerpreter|Nishang|DataToEncode|LoggedKeys|OUT-DNSTXT|Jitter|ExfilOption|Tamper|DumpCerts|DumpCreds|Shellcode32|Shellcode64|
      NotAllNameSpaces|exfill|FakeDC|Exploit|set-content|set-content|PromptForCredential|PS ATTACK|ServerRemoteHost|ServerRemoteHost|
      DownloadString|System.IO.File|Replace|ReadAllText";
```

Figure 4.10: Regex Example of Malicious PowerShell Commands

Regular expression pattern matching is performed on an input string by using the Regex.Match method, which returns the very first appearance of the matching substring in the form of a single Match object.This function returns the first substring in an input string that fits a regular expression profile specified by the RegexOptions and TimeSpan parameters.

When a Regex object is constructed using the Regex (String, RegexOptions, TimeSpan) function Object () { [native code] } and the instance Match(String) method is used, it is identical to calling the public Match(String, String, RegexOptions, TimeSpan).

Using regular expression language components, the pattern parameter describes the string to match in a symbolically descriptive manner. The.NET Framework Regular Phrases and Frequent Expression Language-Quick Reference documents include further information on regular expressions.

By inspecting the Success property of the returned Match object, you can verify if the regular expression match included in the input string has been detected. The Value property of the return Match object holds the substring from the input that fits the

standard expression pattern. If a match is detected, the value of the returned Match object is returned. String is returned as the value if no match was found. Empty.

This function returns the first substring discovered in the input that fits the regular expression pattern specified in the input parameter list. You may get further matches by periodically using the NextMatch method on the Match object that has been returned to you. In addition, by invoking the Regex.Matches(String, String, RegexOptions) function, you may receive all the matches in a single call.

```
void FilterCommands(string path) {
    regex regexp(regexString());
    smatch m;
    ofstream fout;
    string line;
    ifstream fin;
    string pID;
    fin.open(path);
    while (fin) {
        getline(fin, line);
        line = RemoveWhiteSpaces(line, { '\0' }, "");
        regex_match(line, m, regexp);
        sregex_iterator next(line.begin(), line.end(), regexp);
        sregex_iterator end;
        //cout << line << "\n";
        while (next != end) {
            std::smatch match = *next;
            if (match.str() == "Process ID") {
                pID = line;
            }
            else {
                if (pID != "") {
                    cout << pID << " > Found : " << match.str() << "\n";
                    PResultFile << "\MALICIOUS PSHELL COMMANDS DETECTED" << std::endl;
                    PResultFile << pID << " > Found : " << match.str() << "\n";
                }
            }
            next++;
        }
    }
```

Figure 4.11: FilterCommands Function to Parse and Locate any malicious Commands if executed by any of the PowerShell process

4.2.3 Layer III (Memory Hooking/APIs Calls Monitoring)

Layer III will be activated if any new process is get created in the process chain created during layer I. All process is get hooked by the proposed technique one by one and monitor will be activated by using hooking into the main process.

4.2.3.1. API Hooking

API hooking is indeed a method that allows us to instrument and manipulate the behaviour and execution of API requests in a controlled manner. API hooking may be accomplished on Windows using a variety of approaches. Memory cutoff point and other techniques are examples of this. Instructions for DEP and JMP insertion were included. We shall cover the trampoline insertion procedures in a short manner. It is possible to utilize hooking to pause and reflect calls inside a Windows programme, or it is possible to collect certain information related to API calls.For most Windows developers, monitoring Win32 API calls was always a difficult subject to master; nonetheless, it has always been one of my personal favorites. Gaining command over a specific section of code execution is represented by the word Hooking, which is a basic method. While it does not give access to the source code of third-party programs, it does provide a basic interface for altering the behaviour of the operating system and its components. Spying methods are used by many current systems to call attention to the fact that they are able to make use of already-existing Windows apps. Hooking is used not just to add to enhanced functionality, but also to introduce consumer code for debugging reasons, which is a major motivator for using hooks in programming.

Windows Hooks are certainly a highly common way for inserting DLL into a specific process. This technique depends on the functionality given by Windows Hooks. According to the Microsoft Developer Network, a hook is a pitfall in the system's message-handling mechanism. A customized filtering function may be installed by a program to watch the message flow in the network and filter certain sorts of messages as they reach its target window procedure, if one is available. To address the fundamental need for system-wide hooks, a hook is often implemented in a dynamic link library (DLL). That kind of hooks operate on the fundamental principle that the hook return method is run in the memory addresses of every program that has been hooked up to them in the system. You must use SetWindowsHookEx() with the right arguments in order to install a hook. Whenever an application installs a scheme hook,

the os translates the DLL into to the namespace of each of the client processes running on the system. The global variables included inside the DLL will be "each" as a result and will thus not be shareable across programs which have loaded same hook DLL. The shared data section must include all parameters that contain data that is shared with other variables. Example of a hooks registered by Hook Servers and inserted into to the memory addresses "App one" and "Implementation two" as seen in the following figure.

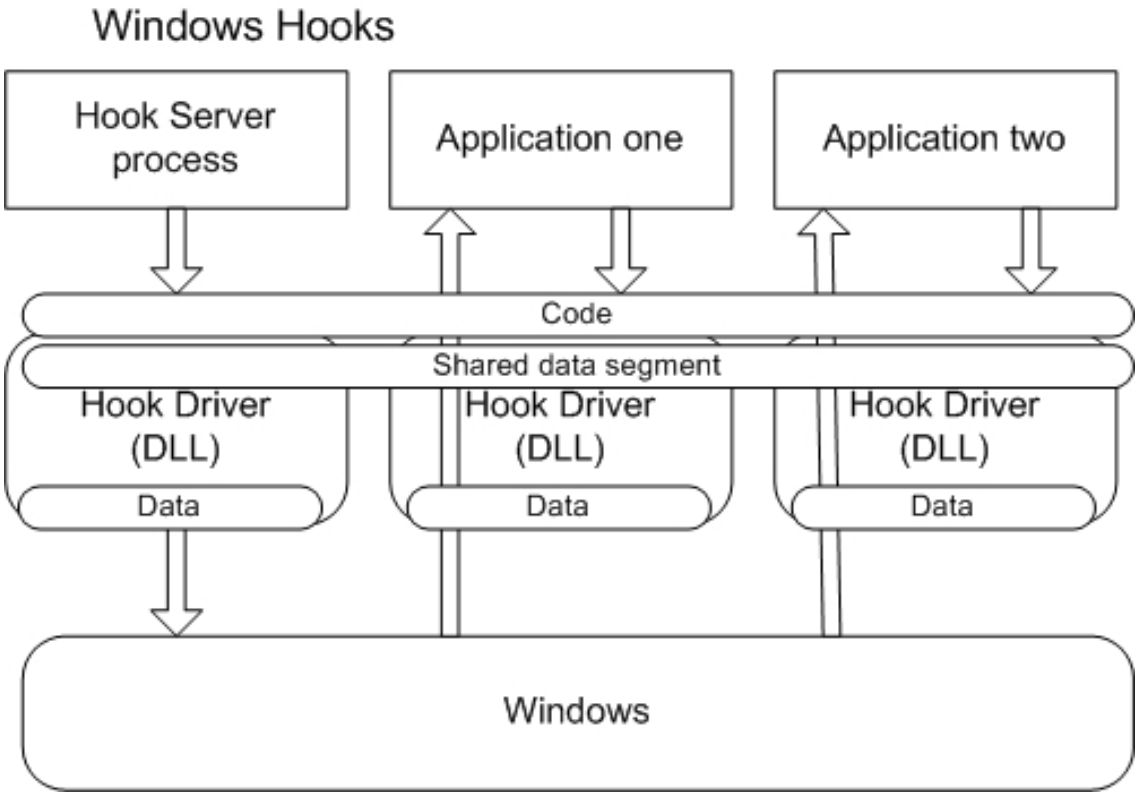


Figure 4.12: Hooking API calls using DLL Injection

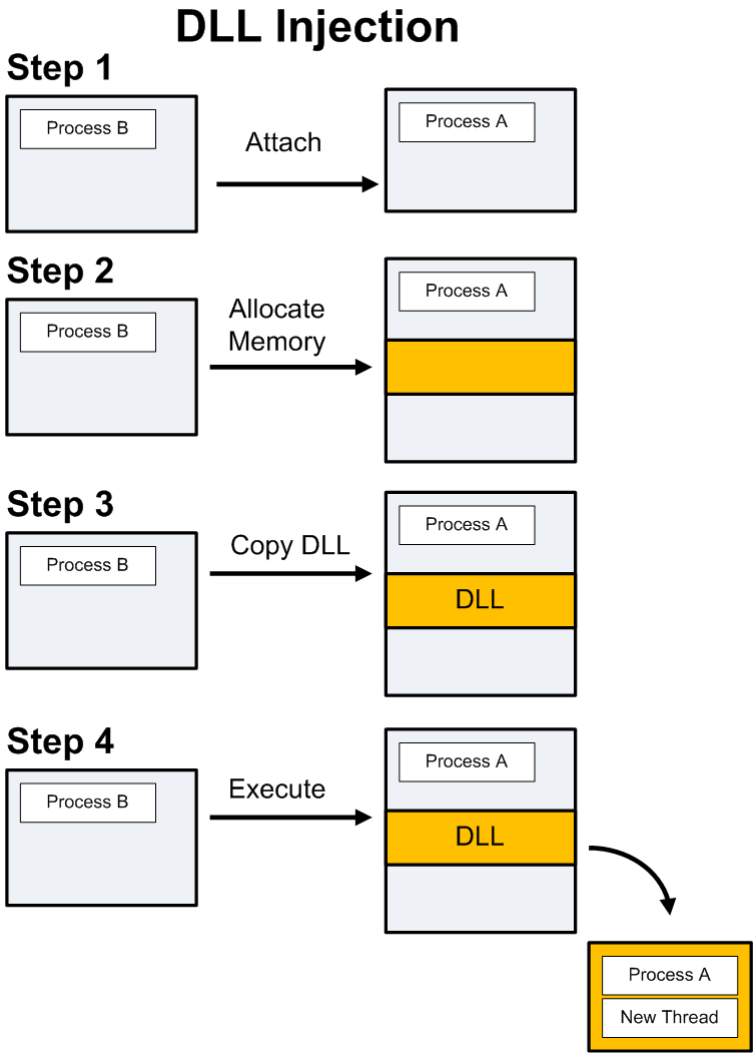
The injection of a DLL through into main memory of an independent process is a critical component of a monitoring system. It gives a fantastic chance to exert control on the thread activities of a process. If you wish to monitor API call calls inside the process, it is not enough to just have the DLL injected.It is the goal of this section of the paper to provide a quick overview of a number of real-world hooking characteristics that are now accessible. It concentrates on the fundamental framework of each of them, outlining their merits and downsides in detail as well.Throughout relation to the level at which the hook is deployed, there are two types of API spying mechanisms: kernel

level espionage and user surveillance. In order to have a better grasp of these two levels, one should be aware of the link between both the Win32 component API and the Local API.

4.2.3.2. DLL Injection

DLL injection is a method in computer science that allows you to execute code into the main memory of some other process by compelling the process to acquire a dynamic-link library (or DLL). When external programs attempt to alter the behaviour of another programme in a manner that the creators did not expect or plan, DLL injection is frequently utilized. An example of DLL Injection is illustrated below in which Process B will inject a .dll binary into the private memory space of the process A.

Figure 4.12: Simple DLL Injection into Memory of Another Process



In layer III a hooking engine called as Deviare is used to inject a monitoring DLL into the suspected process and by using the prerecorded Windows based API calls. After a successful DLL injection into the target process a loaded code will be activated and will actively monitored for the possible api calls.The hooked is called on main function of the running process and whenever any malicious api is called from the target process the hooking engine will inform the main solution via COM object that the specific api function is being called at suspected process. Hooking Process function is defined as below figure.

```
void  HookProcess(__in Deviare2::INktSpyMgr* lpSpyMgr, int pid) {
    //attach to running processes
    cProc.Release();
    hRes = lpSpyMgr->ProcessFromPID((long)(pid), &cProc);
    if (SUCCEEDED(hRes))
    {
        cVt = cProc;
        hRes = cHooksEnum->Attach(cVt, VARIANT_TRUE);
        cVt.Clear();
        if (SUCCEEDED(hRes)) {
            LogPrintNoTick(L"MemRadar::Hooking Process::[%lu]\n", pid);
        }
        else {
            LogPrintNoTick(L"MemRadar::Hooking Failed::[%lu]::Process Not Found\n", pid);
        }
        //activate hooks
        if (SUCCEEDED(hRes))
        {
            hRes = cHooksEnum->Hook(VARIANT_TRUE);
            //LogPrintNoTick(L"CNktDvHooksEnumerator2::Hook => %08X\n", hRes);
        }
    }
}
```

Figure 4.13: Hooking Process into Suspected Running Process on Windows

One of these approaches is to examine API calls, which are directives in the program that instruct computers to conduct certain actions. Rather than attempting to decipher the contents of a tenderly packed file, we undertake a dynamic assessment based on the API calls that have been made to determine what a particular file is intended to achieve. Using the API calls made by a file, we can tell whether or not it is harmful. Some API calls are indicative of malware of a particular sort. For example, URLDownloadToFile is a common downloader API that may be used. The Method GetWindowDC is often used by screen-grabbers, which may be found in malware and keyloggers on a regular basis. The collections of hook file are illustrated presented below. The hooks are the

API functions that can be possibly executed by malicious process. The API calls are built-in for windows and not malicious in nature but can be used by adversaries for malicious purposes. There can be multiple API functions calls at a time executed by a malicious program.

```
<hook name="InternetWriteFile">wininet.dll!InternetWriteFile</hook>
<hook name="PrivacyGetZonePreference">wininet.dll!PrivacyGetZonePreferenceW</hook>
<hook name="PrivacySetZonePreference">wininet.dll!PrivacySetZonePreferenceW</hook>
<hook name="ReadUrlCacheEntryStream">wininet.dll!ReadUrlCacheEntryStream</hook>
<hook name="ResumeSuspendedDownload">wininet.dll!ResumeSuspendedDownload</hook>
<hook name="RetrieveUrlCacheEntryFile">wininet.dll!RetrieveUrlCacheEntryFileA</hook>
<hook name="RetrieveUrlCacheEntryFile">wininet.dll!RetrieveUrlCacheEntryFileW</hook>
<hook name="RetrieveUrlCacheEntryStream">wininet.dll!RetrieveUrlCacheEntryStreamA</hook>
<hook name="RetrieveUrlCacheEntryStream">wininet.dll!RetrieveUrlCacheEntryStreamW</hook>
<hook name="SetUrlCacheEntryGroup">wininet.dll!SetUrlCacheEntryGroup</hook>
<hook name="SetUrlCacheEntryGroup">wininet.dll!SetUrlCacheEntryGroupA</hook>
<hook name="SetUrlCacheEntryGroup">wininet.dll!SetUrlCacheEntryGroupW</hook>
<hook name="SetUrlCacheEntryInfo">wininet.dll!SetUrlCacheEntryInfoA</hook>
<hook name="SetUrlCacheEntryInfo">wininet.dll!SetUrlCacheEntryInfoW</hook>
<hook name="SetUrlCacheGroupAttribute">wininet.dll!SetUrlCacheGroupAttributeA</hook>
<hook name="SetUrlCacheGroupAttribute">wininet.dll!SetUrlCacheGroupAttributeW</hook>
<hook name="UnlockUrlCacheEntryFile">wininet.dll!UnlockUrlCacheEntryFile</hook>
<hook name="UnlockUrlCacheEntryFile">wininet.dll!UnlockUrlCacheEntryFileA</hook>
<hook name="UnlockUrlCacheEntryFile">wininet.dll!UnlockUrlCacheEntryFileW</hook>
<hook name="UnlockUrlCacheEntryStream">wininet.dll!UnlockUrlCacheEntryStream</hook>
<hook name="ReadFile">kernel32.dll!ReadFile</hook>
<hook name="ReadFileEx">kernel32.dll!ReadFileEx</hook>
<hook name="CreateWindowExW">user32.dll!CreateWindowExW</hook>
<hook name="RegisterClassW">user32.dll!RegisterClassW</hook>
<hook name="RegisterClassExW">user32.dll!RegisterClassExW</hook>
```

Figure 4.14: List of Possible API calls in XML format which can be abused by any malicious Program

4.2.4 Interconnected Layer Automation

All the three layers are interconnected with each other via central communication system. The whole process starts from layer I and follow up to the layer III. The whole process is automated and doesn't require any user input during or before start of analysis. All three layers are active all the time when ever they receive any input, they start their operations and generate the output upon which the final decision is made, either to kill the processes chain or not. The output of all three layers is recorded in central database in cloud for further analysis if required.

4.2.5 GUI Presentation

A GUI based solution is created to add/edit/delete records of behavioral signatures for the proposed solution. The proposed solution will download the data from cloud and

then utilize the downloaded data for all there layered wise analysis. Furthermore, the output of all three layers is also obtained from the client-side application and uploaded to the cloud server.

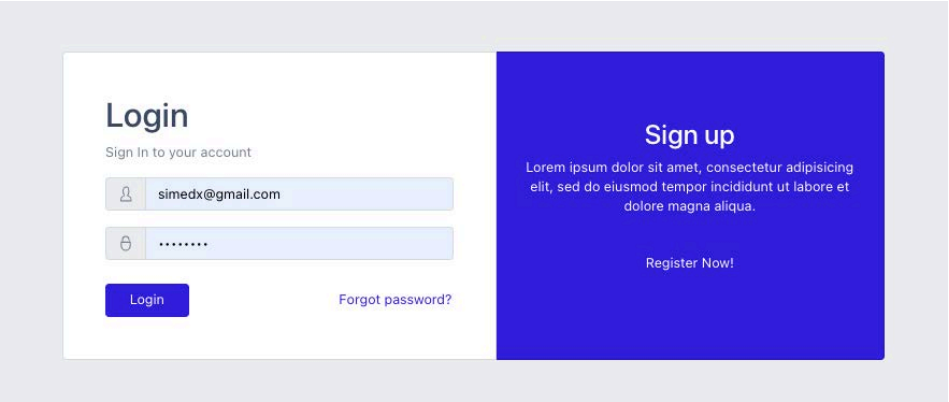


Figure 4.15: Authentication Screen Before Logging into the Dashboard

The GUI system have authentication system as illustrated in figure above before going to the dashboard area where the statistic of the solution is shown. Furthermore, in the GUI panel the user has to add all behaviors which is wanting to be analyzed on the client-side application on windows-based systems.

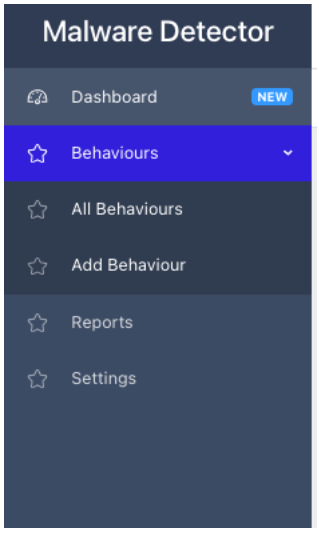


Figure 4.16: Options Available in Dashboard Menu

The view of the dashboard is illustrated below where user can see the statistic of the malware samples detection and other data related to the process analysis and PowerShell detection.

Figure 4.17: Statistics for Dashboard

Chapter 5

Results Analysisand Finding

5.1 Experiment Design

To test the proposed technique, we have run various experiments on multiples malware dataset obtain from virustotal.com and other source. The test environment is set up on virtual machine with following specifications:

- **OS**: Windows 7 and Windows 10
- **Software**: VMware Virtual Machine
- **RAM**: 8GB
- **Storage**: 200GB

The solution of the proposed technique is installed on the selected system and malwares are run one by one. There is also non malicious program in each dataset, which use same processes chain, to test the false positive rate of the proposed technique.

Table 5.1: Dataset of Experimental Evaluation

Name	**Malicious Samples**	**Non-Malicious Samples**
Dataset I	20	5
Dataset II	20	5
Dataset III	20	5
Dataset IV	20	5

Each of the malware dataset used for the evaluation contain twenty malware sample and five non-malicious programs. The database of results and report are cleaned before starting the evaluation. Each sample program is run on the experimental machine on different time one by one.

The output of the experiment stores the reports related to each sample on the cloud from where we can evaluate effectiveness of our proposed technique. The resources utilization of the machine and the proposed technique on machine are also calculated to make sure that, the technique is lightweight and don't cause an overhead to system resources. Whenever a sample is run, the technique performs analysis layer by layer and usage system resource according to it. Once layer I job completed it will free its resource and vice versa for layer II and III. The resource utilization of the experimental machine is shown in Table 5.2.

Table 5.2: Performance Evaluation and Resource Utilization on Test Machine

Factor	**Layer I**	**Layer II**	**Layer III**	**Total**
RAM	5 MB	20-30 MB	80-100 MB	130-150 MB
Storage	5 MB	5 MB	40 MB	50 MB
CPU	1-5%	5-20%	20-30%	1-30%

Table 5.3: Results of Tested Datasets

Dataset No	Total Samples	Samples Detected	Effectiveness	False Positive	False Negative
I	25	20	0.8	0.04	0.16
II	25	17	0.68	0.16	0.16
III	25	19	0.76	0.04	0.20
IV	25	21	0.84	0.12	0.04
Total	100	77	0.77	0.09	0.14

The effectiveness of the proposed technique for each sample date set are shown in Table 5.3 on the scale of 0 to 1. The technique shows most effectiveness for dataset no IV with 84% accuracy along with least no false negative alerts. The least detected sample dataset is no II with 68% detection rate with 52% accuracy.

5.2 False Positive Result Analysis

There are quite few false positive alerts for each of the sample datasets as presented in Table IX. The reason behind this is the non-malicious app are not completely are three layered their behavior looks like malicious app on first layer or second layered only. For such samples the detection is done most of the time from first layer only along with second layer. Like the non-malicious program word.exe can also open a powershell.exe process and download something. But in real world a user will not download anything from word.exe => powershell.exe => download file. A user can directly download a file using browser.

5.3 False Negative Result Analysis

The experimental result indicates from the Table IX that there are also some false negative results produced by the proposed technique. Which mean some samples are not marked as malicious by proposed solution. The reasons behind this are multiple like it can be the missing behaviors of specific malware from the database. If any of Layer I behavior is missing for a specific malware sample, then layer II and III will be not alerted for further analysis thus missed by the proposed solution. This issue can be done by adding the missing behaviors in the database using cloud server component.

5.4 Detected Samples Classification

From the experimental data set we have tried to classify the malware samples into different classifications. This classification is based upon the numbers of various APIs calls and malicious PowerShell commands executed by different processes on the test environment.

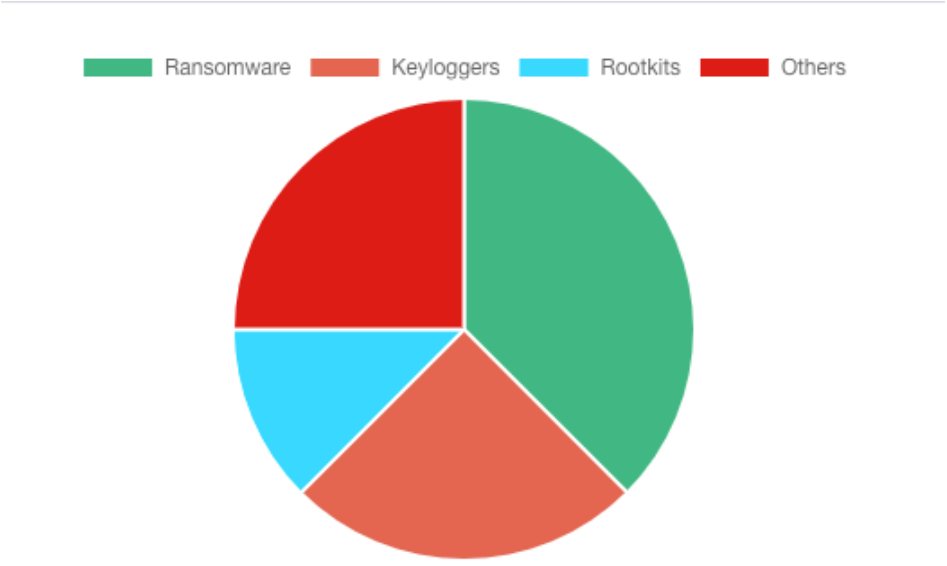


Figure 5.1: Classification of Samples Dataset into Malware Categories

Chapter 6

Conclusion and Future Directions

Although there are numbers of research performed for malware detection, fileless malware remains a serious threat and challenging task for anti-software solutions. are at huge risk of becoming target of such attacks, which standard antiviruses failed to detect and mitigate. The general malware detection techniques mostly relay on signatures for malware. If new malware spread and their signatures are not yet recorded by anti-solution software, the effects can be catastrophic for individuals or organizations. Thus, a new evolving solution is required to counter such fileless attack which not only perform signature analysis but also deeply observe and monitor the whole process chains for identification of malicious behaviors to thwart the effect in real time and quickly.

In this thesis, a three-layered base technique for detection of fileless malware, not only it will detect the malware but also try to mitigate its effects from the victim machine. The technique is based on the behavioral analysis of different aspects of system. The proposed system will use indicators of behaviors for each layer to make decision again a specific sample. Each layer relates to each other thus making a plus point in deciding to make as earliest as possible. The complete analysis is done on any machine with very minimum resource utilization. The complete three-layered analysis is done via single program, and everything is automatic without a user input.

There is a need of few improvements in the proposed solution. Firstly, before starting an analysis on any malware sample the behaviors databases should be up to date. All possible malicious behaviors should be recorded and stored in database. Whenever a new malware starts using similar behaviors the solution can easily detect them. Another suggestion for future work is employment of some machine learning or artificial intelligence techniques for each layer to increase the effectiveness of the proposed technique and to lower down the false negative and false positive results.

Bibliography

[1] Skoudis, E., & Zeltser, L. (2004). *Malware: Fighting malicious code*. Prentice Hall Professional.

[2] Mansfield-Devine, S. (2017). Fileless attacks: compromising targets without malware. *Network Security*, *2017*(4), 7-11.

[3] Sanjay, B. N., Rakshith, D. C., Akash, R. B., & Hegde, V. V. (2018, December). An approach to detect fileless malware and defend its evasive mechanisms. In *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)* (pp. 234-239). IEEE.

[4] Tian, Z., Shi, W., Wang, Y., Zhu, C., Du, X., Su, S., ... & Guizani, N. (2019). Real-time lateral movement detection based on evidence reasoning network for edge computing environment. *IEEE Transactions on Industrial Informatics*, *15*(7), 4285-4294.

[5] Bulazel, A., & Yener, B. (2017, November). A survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium* (pp. 1-21).

[6] WatchGuard Internet Security Report for Q4 2020. Available from: https://www.watchguard.com/wgrd-resource-center/security-report-q4-2020

[7] Tian, Z., Cui, Y., An, L., Su, S., Yin, X., Yin, L., & Cui, X. (2018). A real-time correlation of host-level events in cyber range service for smart campus. *IEEE Access*, *6*, 35355-35364.

[8] Hendler, D., Kels, S., & Rubin, A. (2018, May). Detecting malicious PowerShell commands using deep neural networks. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security* (pp. 187-197).

[9] Fang, Y., Zhou, X., & Huang, C. (2021). Effective method for detecting malicious PowerShell scripts based on hybrid features☆. *Neurocomputing*, *448*, 30-39.

[10] Ucci, D., Aniello, L., & Baldoni, R. (2019). Survey of machine learning techniques for malware analysis. *Computers & Security*, *81*, 123-147.

[11] Sihwail, R., Omar, K., & Ariffin, K. A. Z. (2018). A survey on malware analysis techniques: Static, dynamic, hybrid and memory analysis. *International Journal on Advanced Science, Engineering and Information Technology*, *8*(4-2), 1662.

[12] O,Murchu L, Gutierrez FP. “The evolution of the fileless click-fraud malware poweliks”. Symantec Corp Report. (2015) Avialable:https://www.slideshare.net/symantec/the-evolution-of-the-fileless-clickfraud-malware-poweliks

[13] Adas, H., Shetty, S., & Tayib, W. (2015, May). Scalable detection of web malware on smartphones. In *2015 international conference on information and communication technology research (ICTRC)* (pp. 198-201). IEEE.

[14] Al-Taharwa, I. A., Lee, H. M., Jeng, A. B., Wu, K. P., Mao, C. H., Wei, T. E., & Chen, S. M. (2012, June). Redjsod: A readable javascript obfuscation detector using semantic-based analysis. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications* (pp. 1370-1375). IEEE.

[15] Zeltser L., "The history of Fileless malware-looking beyond the buzzword", Available : https://zeltser.com/fileless-malware-beyond-buzzword/ , 2018

[16] Ajay Ohri, "Fileless Malware: A Comprehensive Guide In 2021", Jigsaw Academy, From: https://www.jigsawacademy.com/blogs/cyber-security/fileless-malware/,

[17] Fred O'Conner, "Fileless Malware 101: Understanding Non-Malware Attacks", Avialable: https://www.cybereason.com/blog/fileless-malware, 2019

[18] Nick Ismail, "Defending against fileless malware" https://www.information-age.com/defending-fileless-malware-123466835/, 2017

[19] Moser, A., Kruegel, C., & Kirda, E. (2007, December). Limits of static analysis for malware detection. In *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)* (pp. 421-430). IEEE.

[20] Egele, M., Scholte, T., Kirda, E., & Kruegel, C. (2008). A survey on automated dynamic malware-analysis techniques and tools. *ACM computing surveys (CSUR)*, *44*(2), 1-42

[21] Afianian, A., Niksefat, S., Sadeghiyan, B., & Baptiste, D. (2019). Malware dynamic analysis evasion techniques: A survey. *ACM Computing Surveys (CSUR)*, *52*(6), 1-28.

[22] Lee, G., Shim, S., Cho, B., Kim, T., & Kim, K. (2021). Fileless cyberattacks: Analysis and classification. *ETRI Journal*, *43*(2), 332-343.

[23] Shalaginov, A., Banin, S., Dehghantanha, A., & Franke, K. (2018). Machine learning aided static malware analysis: A survey and tutorial. In *Cyber threat intelligence* (pp. 7-45). Springer, Cham.

[24] Liu, C., Xia, B., Yu, M., & Liu, Y. (2018, June). PSDEM: A Feasible De-Obfuscation Method for Malicious PowerShell Detection. In *2018 IEEE Symposium on Computers and Communications (ISCC)* (pp. 825-831). IEEE

[25] Krishna, B. L. (2020). Comparative Study of Fileless Ransomware. In International Journal of Trend in Scientific Research and Development (IJTSRD), Vol. 4

[26] M. Amin, P. Sharma, "A Survey on Fileless Malware Attacks: Malware Hiding in Memory", IJSRD - International Journal for Scientific Research & Development|,Vol. 5, Issue 03, 2017

[27] Li, Z., Chen, Q. A., Xiong, C., Chen, Y., Zhu, T., & Yang, H. (2019, November). Effective and light-weight deobfuscation and semantic-aware attack detection for powershell scripts. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1831-1847).

[28] Vala Khushali, "A Review on Fileless Malware Analysis Techniques", International Journal of Engineering Research & Technology Vol. 9 Issue 05, 2020

[29] Kumar,Sushil (2020). An emerging threat Fileless malware: a survey and research challenges. *Cybersecurity*, *3*(1), 1-12.

[30] Sanjay, B. N., Rakshith, D. C., Akash, R. B., & Hegde, V. V. (2018, December). An approach to detect fileless malware and defend its evasive mechanisms. In *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)* (pp. 234-239). IEEE.

[31] Tsuda, Y., Nakazato, J., Takagi, Y., Inoue, D., Nakao, K., & Terada, K. (2018, August). A lightweight host-based intrusion detection based on process generation patterns. In *2018 13th Asia Joint Conference on Information Security (AsiaJCIS)* (pp. 102-108). IEEE.

[32] Nahmias, D., Cohen, A., Nissim, N., & Elovici, Y. (2019, July). Trustsign: Trusted malware signature generation in private clouds using deep feature transfer learning. In *2019 International Joint Conference on Neural Networks (IJCNN)* (pp. 1-8). IEEE.

[33] Saad, S., Mahmood, F., Briguglio, W., & Elmiligi, H. (2019, November). Jsless: A tale of a fileless javascript memory-resident malware. In *International Conference on Information Security Practice and Experience* (pp. 113-131). Springer, Cham.

[34] Lim, C., Kotualubun, Y. S., & Ramli, K. (2017). Mal-Xtract: Hidden Code Extraction using Memory Analysis. In *Journal of Physics: Conference Series* (Vol. 801, No. 1, p. 012058). IOP Publishing.

[35] Handaya, W. B. T., Yusoff, M. N., & Jantan, A. (2020, February). Machine learning approach for detection of fileless cryptocurrency mining malware. In *Journal of Physics: Conference Series* (Vol. 1450, No. 1, p. 012075). IOP Publishing.

[36] Tarek, R., Chaimae, S., & Habiba, C. (2020, March). Runtime api signature for fileless malware detection. In *Future of information and communication conference* (pp. 645-654). Springer, Cham.

[37] Gadgil, P., & Nagpure, S. (2019, March). Analysis Of Advanced Volatile Threats Using Memory Forensics. In Proceedings 2019: Conference on Technologies for Future Cities (CTFC).

[38] Lee, J. M., & Hong, S. (2021, June). Host-Oriented Approach to Cyber Security for the SCADA Systems. In *2020 6th IEEE Congress on Information Science and Technology (CiSt)* (pp. 151-155). IEEE.

[39] Obeis, N. T., & Bhaya, W. (2018). Malware analysis using APIs pattern mining. *International Journal of Engineering & Technology*, *7*(3.20), 502-506.

[40] Botacin, M., Grégio, A., & Alves, M. A. Z. (2020, September). Near-Memory & In-Memory Detection of Fileless Malware. In The International Symposium on Memory Systems (pp. 23-38).