CYBERSECURITY COURSE

Project 1 Report: Linux + Python

Members:

Orjon Xhyra
Ergion Kopani
Indrit Shkupi

10.02.2025

Table of Contents

Executive Summary

This project aimed to analyze and exploit a targeted machine through a structured approach to cybersecurity. Initially, two virtual machines were configured within the same internal network, and the target's IP address (**10.38.1.111**) was identified through an **ARP scan**. Then, using **nmap**, open ports and active service versions were discovered, including **vsftpd, OpenSSH,** and **Apache**. Analyzing the web interface on port 80 revealed a hidden **BASE64-encoded hash**, which suggested using a brute-force wordlist for further exploration. After identifying vulnerabilities, an **exploitation technique** was applied through **GRUB** by modifying kernel parameters, allowing **root access** to the system. At this stage, the root password was changed, and directories were explored to retrieve the **flags of success**: **flag{h4ck3r}** and **flag{1337}**, confirming the successful completion of the project's objective.

1. Identifying the Device's IP Address

Action Taken:
First of all we configure two VMs in same internal network.
Using the ifconfig command on Kali Linux, the device's IP address was identified as 10.38.1.110, and the subnet mask was configured as 255.255.255.0.

Evidence

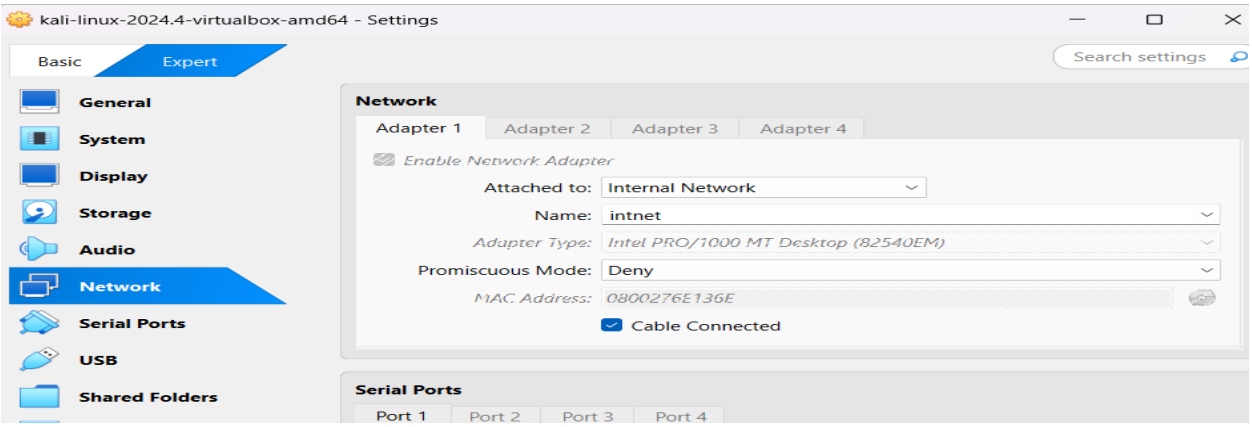

Figure 1

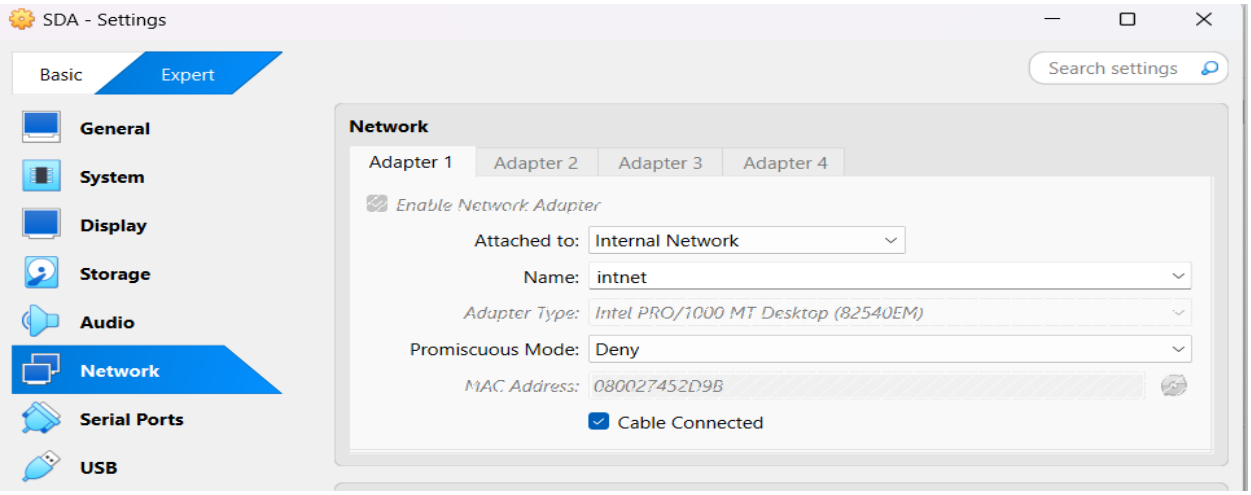

Figure 2

2. Configuring the Subnet Mask

Action Taken:
Network settings in VirtualBox were configured to use the intnet network adapter with a subnet mask of 255.255.255.0, ensuring that all devices within the subnet could communicate with each other.
By using the command below we had configure it.

Evidence

```
C:\Program Files\Oracle\VirtualBox>vboxmanage dhcpserver add --network=intnet --server-ip=10.38.1.1 --lower-ip=10.38.1.1
10 --upper-ip=10.38.1.120 --netmask=255.255.255.0 -enable
```

Figure 3

```
┌──(kali㉿kali)-[~]
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.38.1.110  netmask 255.255.255.0  broadcast 10.38.1.255
        inet6 fe80::3705:6251:a85d:304e  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:6e:13:6e  txqueuelen 1000  (Ethernet)
        RX packets 9  bytes 3756 (3.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 17023  bytes 1023640 (999.6 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 8  bytes 480 (480.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 8  bytes 480 (480.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figure 4

3. Network Scanning to Identify Target IPs

Action Taken:
An ARP scan was performed, identifying the following devices in the network:
10.38.1.1 with MAC address 08:00:27:42:b0:03
10.38.1.111 with MAC address 08:00:27:45:2d:9b
Based on these results, the target was successfully identified with IP address 10.38.1.111.

Evidence

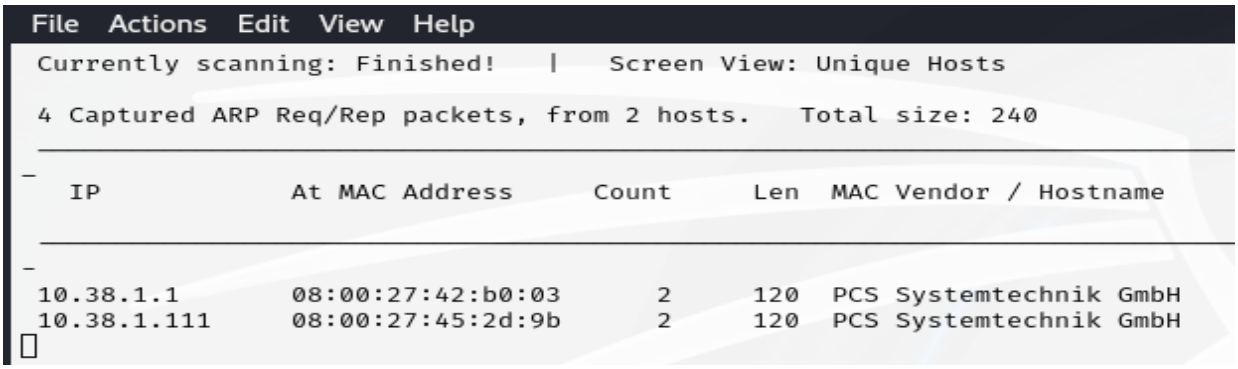

Figure 5

4. Discovering Open Ports and Service Versions on the Target Device

Action Taken:

Using the nmap command with the flags -sC and -sV, a scan was performed for the IP 10.38.1.111, revealing:

Port 21/tcp: Service vsftpd 3.0.5

Port 22/tcp: Service OpenSSH 8.9p1

Port 80/tcp: Service Apache/2.4.52

This scan provided detailed information about the active services on the target device and their versions.

Evidence

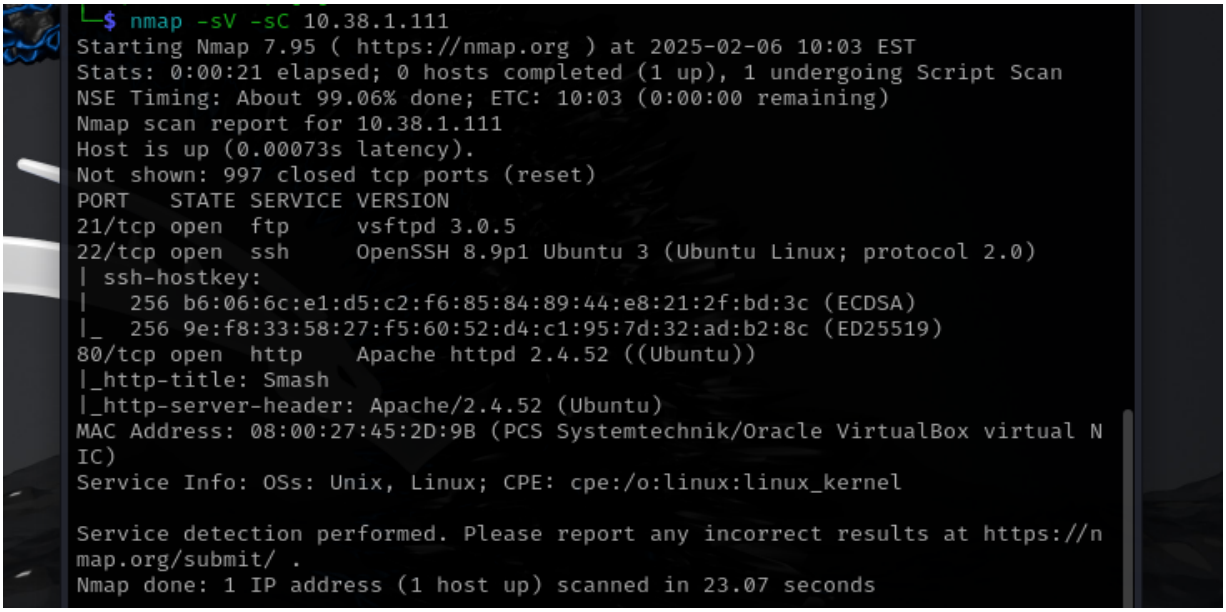

Figure 6

5. Discovering the Hash in the Web Interface

Step 1:

From analyzing the web interface in port 80 and files (HTML and JavaScript), a hidden hash or encoded message in BASE64 was identified.

The hash was decoded using a BASE64 decoder, which revealed the instruction:

"Enumerate me with directory-list-lowercase-2-3-medium.txt."

This suggested using a directory brute-forcing file for further exploration.

Evidence

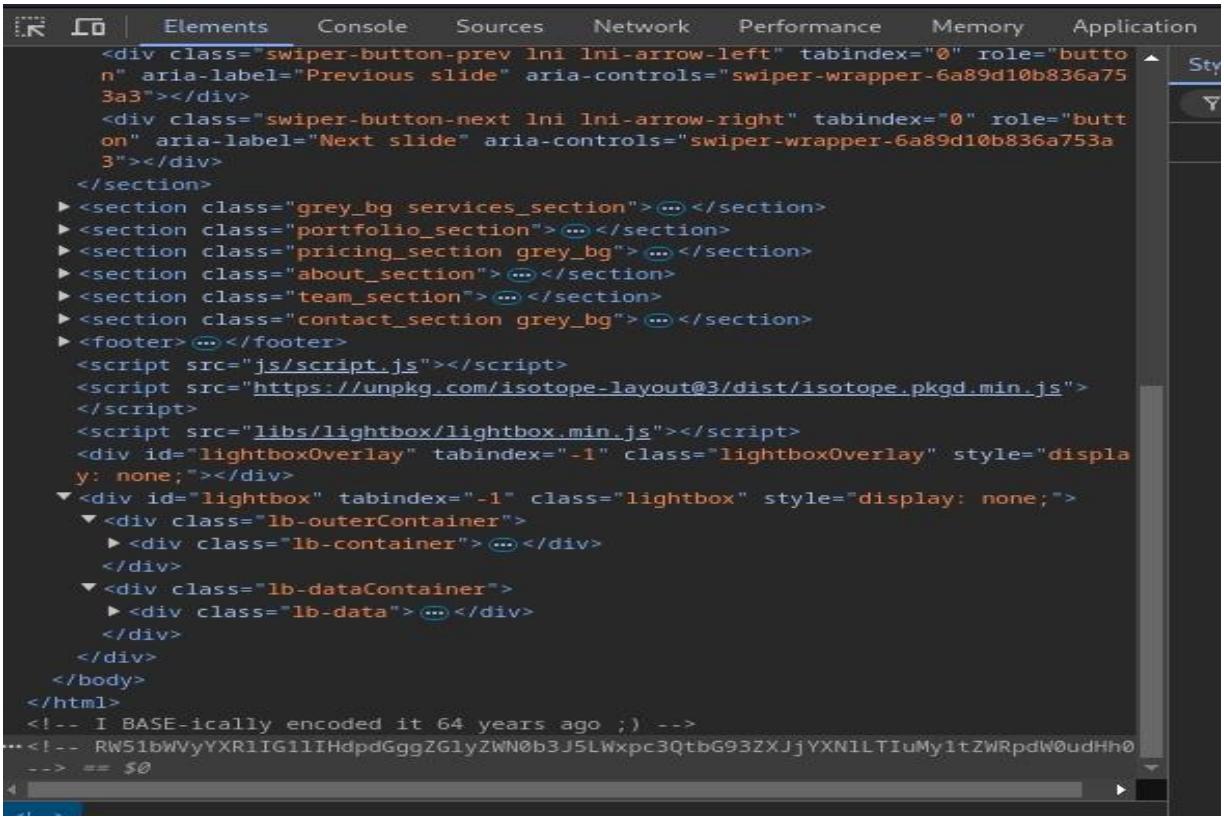

Figure 7

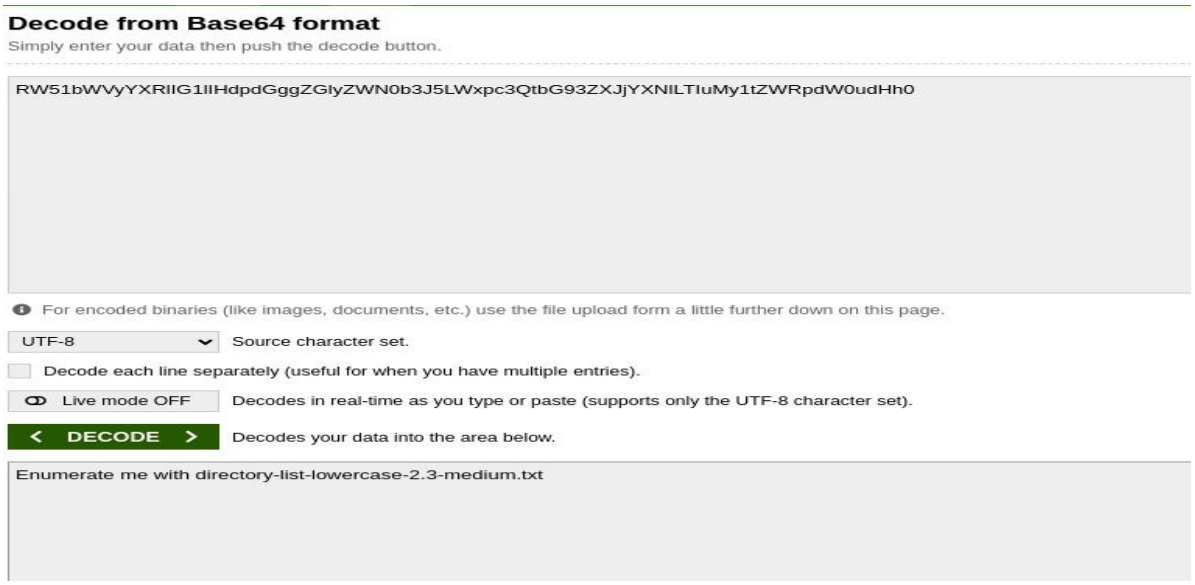

Figure 8

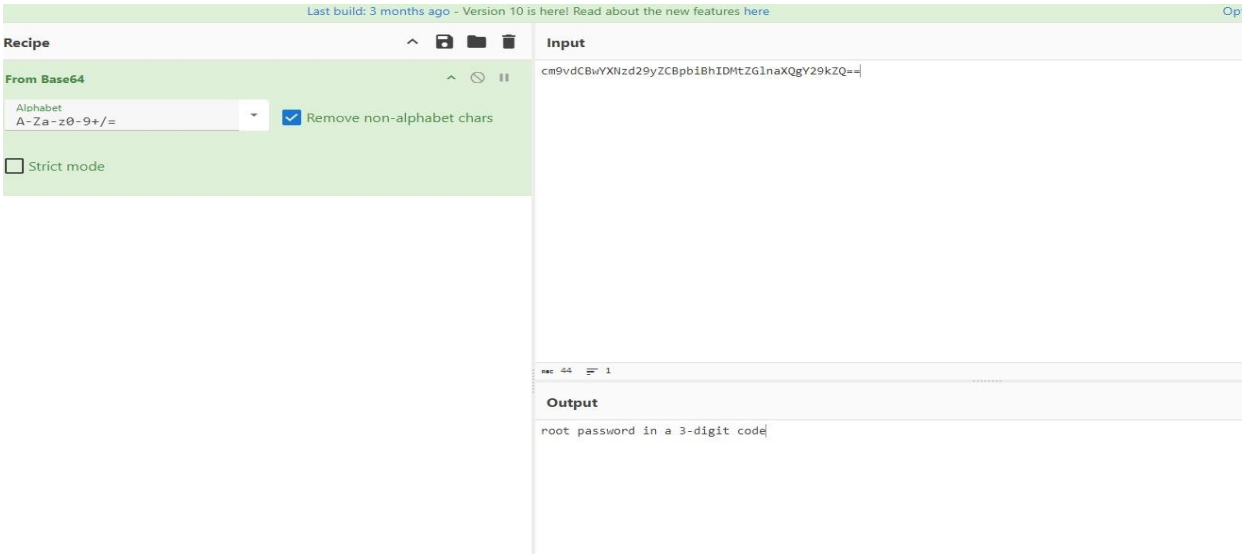

Figure 9

6. Preparing for the Attack

Step 2:

After identifying vulnerabilities, an attempt was made to exploit the virtual machine through GRUB (Grand Unified Bootloader).

Figure 10

After presing the letter “e”, we can access kernel for gaining access as root.

7. Exploiting via GRUB

Step 3:
After rebooting the target machine, GRUB was used to modify kernel parameters and access the system in recovery mode.
Commands were adjusted to ensure root access to the system.

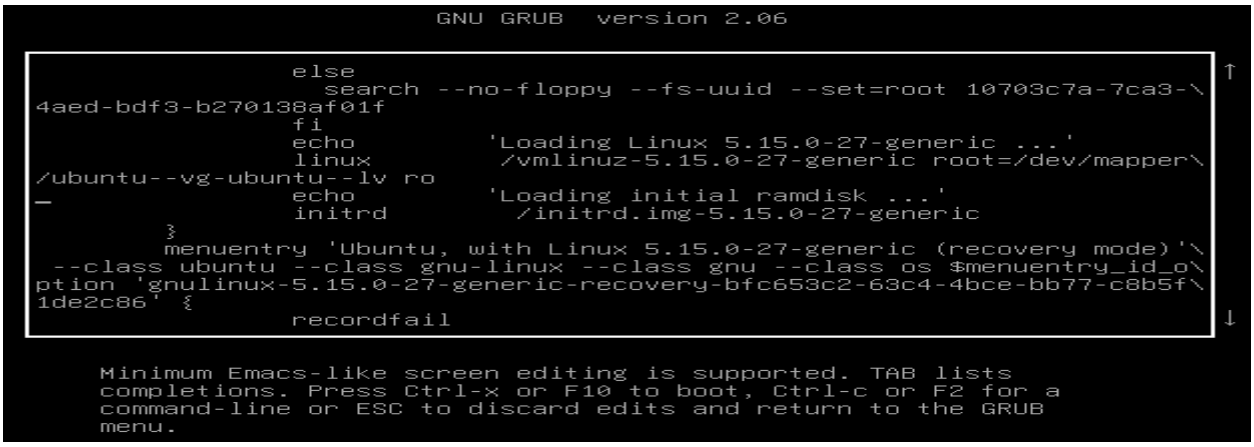

Figure 11

At this stage we change ro tp rw and append init=/bin/bash at the end of that line.
In order to save the changes we press Ctrl+X to boot into the root shell.
After the machine boots we can easily change the root password and also list the users by issuing the command "*ls /home*". To save the changes and reboot to the system we issue the command "*/sbin/reboot -f*"

8. Accessing the Machine and Retrieving the Flags

Step 4:
With administrative access, directories were explored to locate the flags:
User Flag (user.txt):
flag{h4ck3r}
Root Flag (root.txt):
flag{1337}

Evidence

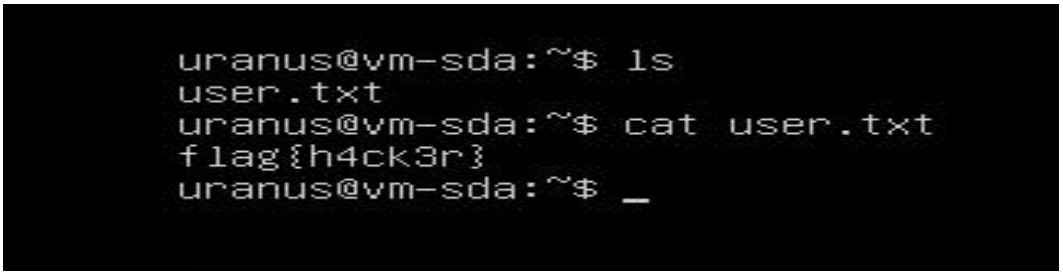

Figure 12

Figure 13

9. Python Scripts

This Python script implements above tasks workflow using Python, where it performs the following tasks on a specified target IP:

1) SYN Scan (syn_scan):

This function uses a SYN scan (TCP connection initiation) to check if a port is open on the target IP by sending a SYN packet. If the response is a SYN-ACK (flags 18), the port is open.

2) Banner Grabbing (grab_banner):

This function grabs the banner of a service running on an open port by attempting to establish a TCP connection. It then receives up to 1024 bytes of data and returns it as the banner.

3) Network Scan (network_scan):

This function scans the given network range (using ICMP) to discover hosts that are up. It checks if the target IP is present in the network.

4) Port Scan (port_scan):

Scans specific ports (21, 22, 80) on the target IP to check if they are open.

5) Banner Grabbing for Open Ports (banner_grabbing):

For each open port discovered during the port scan, it grabs the banner and displays it.

6) HTTP Service Analysis (analyze_http_service):

Specifically targets port 80 (HTTP) to grab the banner and analyze the HTTP service running on the target IP.

7) Penetration Test Workflow (penetration_test):

A comprehensive workflow that ties together the network scan, port scan, banner grabbing, and HTTP service analysis.

```
from scapy.all import *
import socket
import threading
import time

# Function to perform a SYN scan to discover open ports
def syn_scan(target_ip, port):
    ip = IP(dst=target_ip)
    syn = TCP(dport=port, flags="S")
    pkt = ip/syn
    resp = sr1(pkt, timeout=1, verbose=0)
    if resp is None:
        return False
    elif resp.haslayer(TCP) and resp.getlayer(TCP).flags == 18:  # SYN-ACK
        return True
    return False

# Function for banner grabbing
def grab_banner(target_ip, port):
    try:
        s = socket.socket()
        s.settimeout(2)
        s.connect((target_ip, port))
        banner = s.recv(1024)
        return banner.decode().strip()
    except Exception as e:
```

```
        return None

# Task II: Scan the network for the target IP
def network_scan():
    print("Scanning network range:", network_range)
    ans, unans = sr(IP(dst=network_range)/ICMP(), timeout=2, verbose=0)
    for snd, rcv in ans:
        print(f"Host {rcv.src} is up")
        if rcv.src == target_ip:
            print(f"Target IP found: {target_ip}")
            return
    print("Target IP not found in the network.")

# Task III: Scan the open ports on the target
def port_scan():
    open_ports = []
    print(f"Scanning open ports on {target_ip}...")
    for port in [21, 22, 80]:
        if syn_scan(target_ip, port):
            print(f"Port {port} is open.")
            open_ports.append(port)
        else:
            print(f"Port {port} is closed.")
    return open_ports

# Task IV: Banner Grabbing for the discovered open ports
def banner_grabbing():
    open_ports = port_scan()
    print("\nGrabbing banners for open ports...")
```

```
    for port in open_ports:
        banner = grab_banner(target_ip, port)
        if banner:
            print(f"Banner for port {port}: {banner}")
        else:
            print(f"No banner found for port {port}")

# Extra Task I: Network Scan and HTTP Service Analysis
def analyze_http_service():
    print("\nPerforming HTTP Service Analysis...")
    banner = grab_banner(target_ip, 80)  # Port 80 for HTTP
    if banner:
        print(f"HTTP Banner: {banner}")
    else:
        print("No banner received from HTTP service.")

# Function to handle the entire pentest workflow
def penetration_test():
    print("Starting penetration test...\n")
    network_scan()  # Scan for the target in the network
    banner_grabbing()  # Banner grabbing for open ports
    analyze_http_service()  # Analyze HTTP service and banners

if __name__ == "__main__":
    # Run the penetration test
    penetration_test()
```