Task 2 – Packet Sniffing, Firewall Configuration, and Vulnerability Scanning Report

Submitted by:

Shashanka U N

SOC Intern – Cyber Security

Submitted to:

CyArt Tech

Date:

21th August 2025

Introduction:

With the rise of cyber threats, organizations must adopt proactive measures to secure their systems and networks. Security professionals are expected to monitor network activity, enforce strict access control policies, and regularly assess vulnerabilities to ensure a strong defense posture.
This task focuses on three critical aspects of network security:

- **Packet Sniffing** – capturing and analyzing live network packets to identify communication protocols and potential anomalies.
- **Firewall Configuration** – applying iptables rules to allow only authorized traffic while blocking unnecessary or malicious connections.
- **Vulnerability Scanning** – using OpenVAS to detect weaknesses in systems, classify them by severity, and provide actionable recommendations.

Objectives:

- Perform packet sniffing and protocol identification.
- Configure firewall rules to allow only specific services (SSH & HTTP).
- Conduct vulnerability scanning on a local/test system.
- Document findings with outputs, charts, and screenshots.

Methodology

The following methodology was adopted to complete Task:

1. Environment Setup

- Ubuntu (WSL) used as Linux environment.
- Python 3 with **Scapy** and **Matplotlib** installed.
- iptables/UFW used for firewall rules.
- OpenVAS (GVM) installed for vulnerability scanning.

2. Tools Used

- **Scapy** (packet sniffing)
- **Matplotlib** (charting protocol distribution)

- **iptables / UFW** (firewall rules)
- **OpenVAS (GVM)** (vulnerability scanning)

Scan Results & Observations:

Task 1 – Packet Sniffing with Scapy

Objective

The objective of this task was to capture and analyze live network packets using Scapy, a Python-based packet manipulation tool. By collecting a sample of packets and examining their protocols, we aimed to understand the types of traffic flowing through the system and visualize their distribution.

Environment Setup

- Ubuntu WSL was used as the Linux environment.
- Python 3 was pre-installed; additional modules were installed using pip.

```
sudo apt update &&
sudo apt install python3
python3-pip -y
pip3 install scapy matplotlib
```

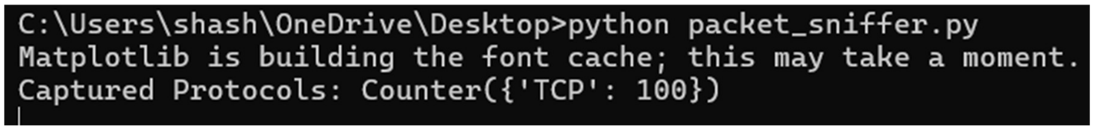
Script Development

A Python script packet_sniffer.py was written to:

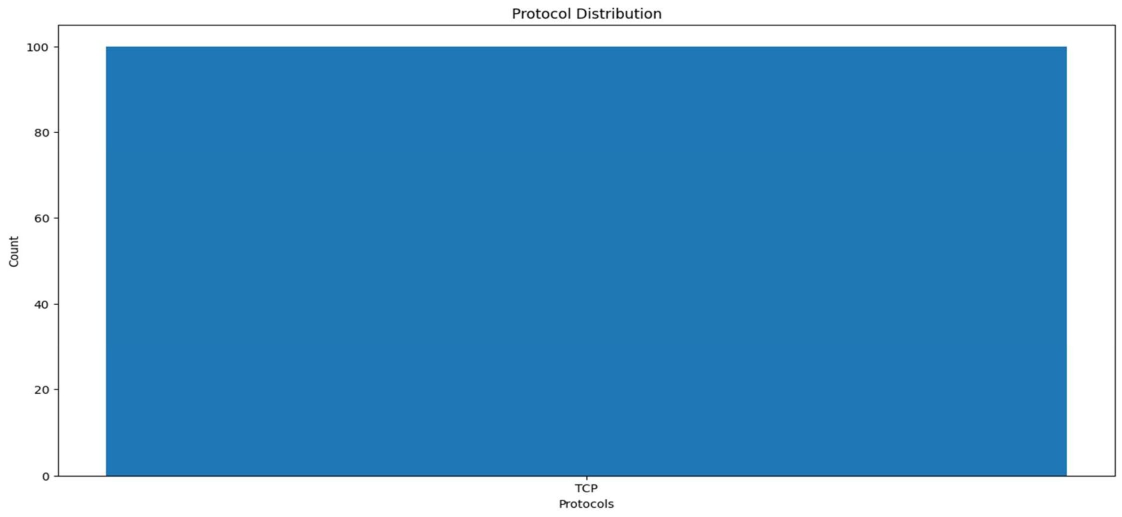
```
from scapy.all import sniff
from collections import Counter
import matplotlib.pyplot as plt

packets = sniff(count=100)
protocols = []

for pkt in packets:
    if pkt.haslayer("TCP"):
        protocols.append("TCP")
        elif pkt.haslayer("UDP"):
            protocols.append("UDP")
        elif pkt.haslayer("ICMP"):
            protocols.append("ICMP")
        else:
            protocols.append("Other")

counter = Counter(protocols)
print("Captured Protocols:", counter)
plt.bar(counter.keys(), counter.values())
plt.xlabel("Protocols")
plt.ylabel("Count")
plt.title("Protocol Distribution")
plt.savefig("protocol_distribution.png")
plt.show()
```

Protocol distribution

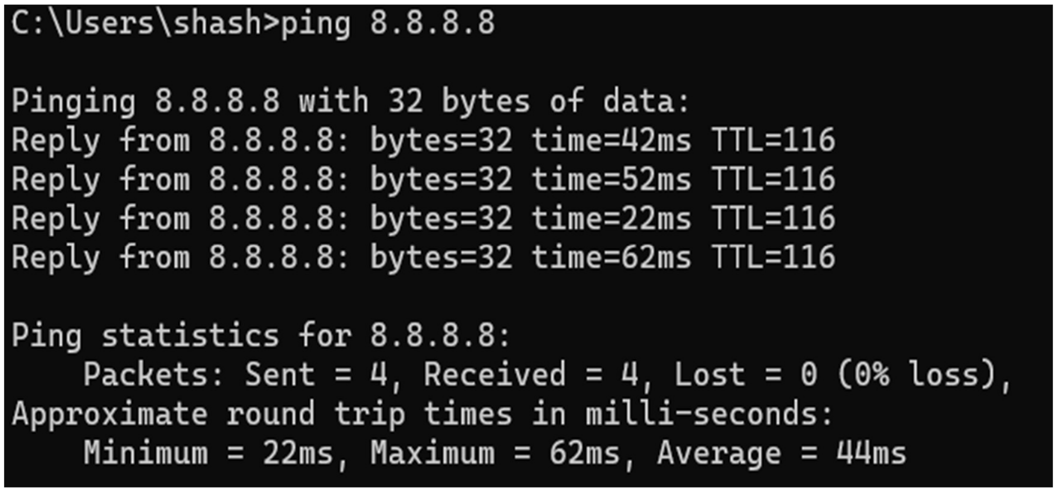
1. **TCP**



```
C:\Users\shash\OneDrive\Desktop>python packet_sniffer.py
Matplotlib is building the font cache; this may take a moment.
Captured Protocols: Counter({'TCP': 100})
```

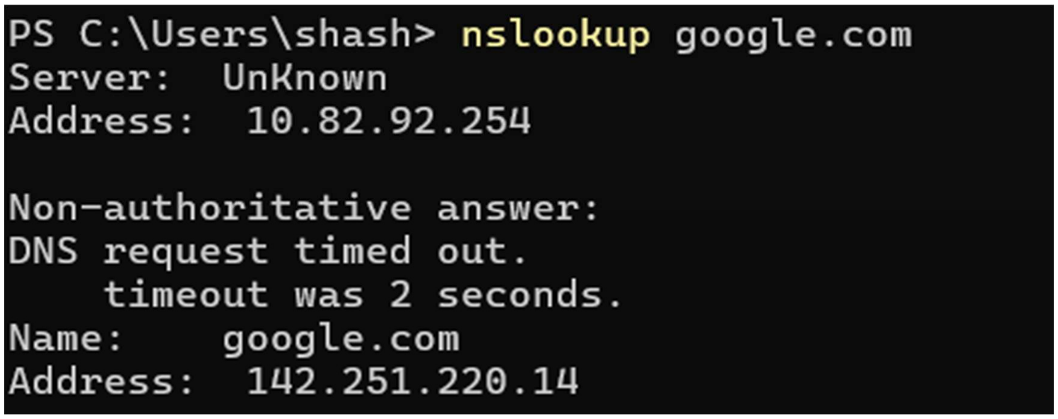


2. **ICMP**



```
C:\Users\shash>ping 8.8.8.8

Pinging 8.8.8.8 with 32 bytes of data:
Reply from 8.8.8.8: bytes=32 time=42ms TTL=116
Reply from 8.8.8.8: bytes=32 time=52ms TTL=116
Reply from 8.8.8.8: bytes=32 time=22ms TTL=116
Reply from 8.8.8.8: bytes=32 time=62ms TTL=116

Ping statistics for 8.8.8.8:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 22ms, Maximum = 62ms, Average = 44ms
```

3. **UDP**



```
PS C:\Users\shash> nslookup google.com
Server:  UnKnown
Address:  10.82.92.254

Non-authoritative answer:
DNS request timed out.
    timeout was 2 seconds.
Name:    google.com
Address:  142.251.220.14
```

Task 2 – Firewall Configuration using UFW

Objective

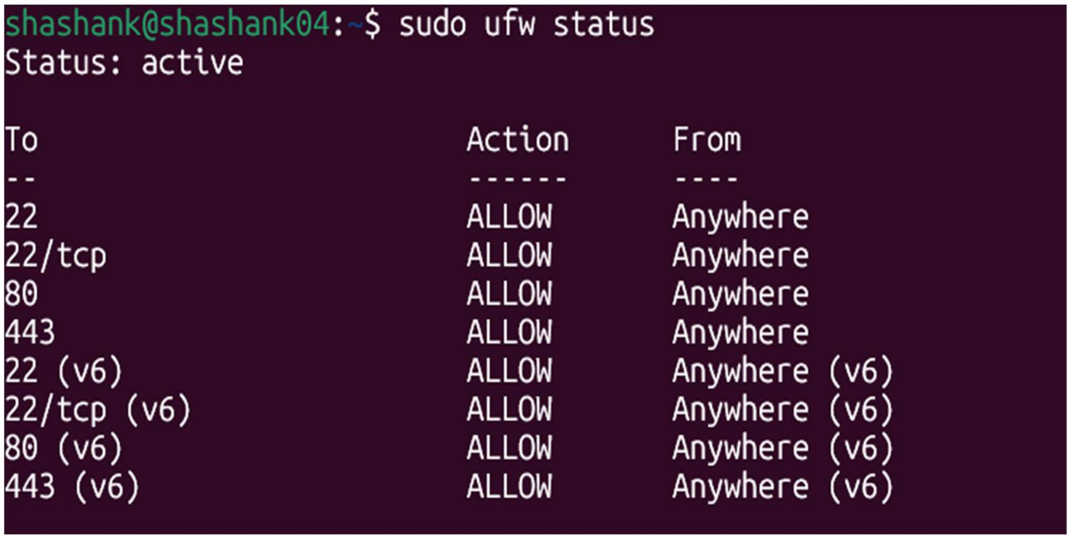
The objective of this task was to configure a basic firewall on a Linux system using UFW (Uncomplicated Firewall). The goal was to allow only essential services such as SSH (port 22) and HTTP/HTTPS (ports 80/443), while blocking all other inbound traffic, thereby reducing the system's attack surface.

Methodology

1. **Firewall Installation & Status Check**
   - Verified whether UFW was installed: *sudo ufw status*

- Installed UFW (if not available): *sudo apt install ufw -y*

2. **Enabling UFW**
   - Enabled firewall protection: *sudo ufw enable*
3. **Configuring Rules**
   - Allowed **SSH** (port 22): *sudo ufw allow 22*
   - Allowed **HTTP** (port 80) and **HTTPS** (port 443): *sudo ufw allow 80*
     sudo ufw allow 443
4. **Verifying Firewall Rules**
   - Checked status in verbose mode: *sudo ufw status*



```
shashank@shashank04:~$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
22                         ALLOW       Anywhere
22/tcp                     ALLOW       Anywhere
80                         ALLOW       Anywhere
443                        ALLOW       Anywhere
22 (v6)                    ALLOW       Anywhere (v6)
22/tcp (v6)                ALLOW       Anywhere (v6)
80 (v6)                    ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)
```

Task 3 – Vulnerability Scanning with OpenVAS

Objective

The objective of this task was to perform a vulnerability scan using **OpenVAS (Greenbone Vulnerability Manager)** to identify security weaknesses, assess their severity, and understand potential risks to the system.

Methodology

1. Attempted to install OpenVAS on Ubuntu environment:
2. sudo apt install gvm -y
3. sudo gvm-setup
4. sudo gvm-start

5. Feed update process was initiated to download vulnerability definitions.
6. Intended steps after setup:
   - Access Greenbone Security Assistant via browser at https://127.0.0.1:9392.
   - Log in with admin credentials created during setup.
   - Run a **Full and Fast Scan** on the local/test machine.
   - Review the scan results in the web interface.

Analysis

- Vulnerability scanning is an essential part of security operations, as it helps detect outdated software, misconfigurations, and exploitable services.
- Even though the scan could not be executed fully in this environment, understanding the **process and expected outputs** is valuable.
- In a proper setup (dedicated VM or server), OpenVAS would provide detailed reports to guide patching and mitigation.