Title: Packet Sniffing, Firewall Configuration, and Vulnerability Scanning

1. Introduction

This task focuses on three core defensive and assessment skills: packet sniffing, host firewall configuration, and vulnerability scanning. Packet sniffing with Scapy provides visibility into protocol usage and traffic patterns, iptables rules demonstrate how to control network flows at the host level, and OpenVAS scanning highlights vulnerabilities and misconfigurations on target systems

2. Tools Required

- Operating system: Kali Linux
- Python 3
- Scapy library
- matplotlib library
- iptables
- OpenVAS / GVM

3. Packet Sniffing with Scapy

Scan Commands Used

Scapy is a Python library that can sniff, dissect, and craft packets across many protocols, making it ideal for low-level traffic analysis.

Sniffer script (packet_sniffer.py)

Use or adapt this script to capture 100 packets on a chosen interface and count protocols:

Python Code:

```
#!/usr/bin/env python3
from scapy.all import sniff
from collections import Counter
import matplotlib.pyplot as plt

# Dictionary to hold protocol counts
proto_counts = Counter()
```

```
def process_packet(pkt):
    """Callback for each captured packet: detect protocol and update counter."""
    if pkt.haslayer("TCP"):
        proto_counts["TCP"] += 1
    elif pkt.haslayer("UDP"):
        proto_counts["UDP"] += 1
    elif pkt.haslayer("ICMP"):
        proto_counts["ICMP"] += 1
    else:
        proto_counts["Other"] += 1

def main():
    iface = input("Enter interface to sniff on (e.g., eth0, wlan0, lo): ").strip()
    print(f"[+] Sniffing 100 packets on {iface} ...")
    sniff(iface=iface, count=100, prn=process_packet)
    print("[+] Capture complete.")
    print("Protocol counts:", proto_counts)

    # Bar chart of protocol distribution
    labels = list(proto_counts.keys())
    values = [proto_counts[k] for k in labels]

    plt.bar(labels, values)
    plt.title("Captured Protocol Distribution")
    plt.xlabel("Protocol")
    plt.ylabel("Packet Count")
    plt.savefig("protocol_distribution.png")
    plt.close()
    print("[+] Saved chart to protocol_distribution.png")

if __name__ == "__main__":
    main()
```

4. Scan Result

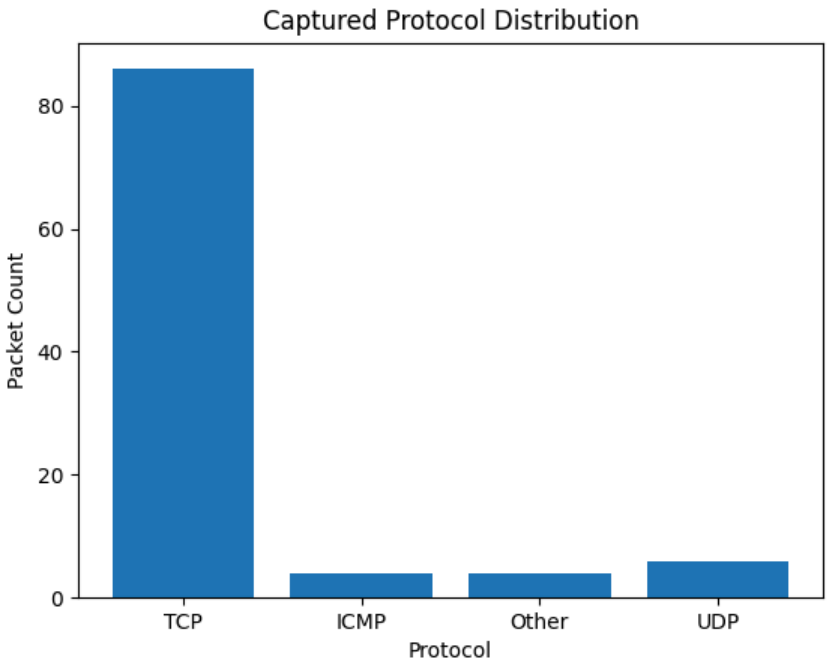
Working Structure

The script captures 100 packets, identifies TCP/UDP/ICMP/Other using Scapy's layer checks, and saves a bar chart image, which you will embed in your report.

```
┌──(venv)─(root㉿kali)-[/home/kali/py-labs]
└─# python packet_sniffer.py
Source  Index  Name  MAC                IPv4          IPv6
sys     1      lo    00:00:00:00:00:00  127.0.0.1     ::1
sys     2      eth0  PCSSystemtec:6e:13:6e  10.137.250.30  fe80::287f:8cd:871e:7c5c
Enter interface to sniff on (e.g., eth0, wlan0, lo): eth0
[+] Sniffing 100 packets on eth0 ...
[+] Capture complete.
Protocol counts: Counter({'TCP': 86, 'UDP': 6, 'ICMP': 4, 'Other': 4})
[+] Saved chart to protocol_distribution.png
```

1. Shows the available interface on the host
2. Ask user to select the interface from listed above
3. Initiate a packet capture on the selected interface
4. It capture 100 packets
5. The out saved in the protocol_distributed.png format
6. This PNG file contains a visual chart showing how many of the captured packets belonged to each protocol

5. Scan Output



- The chart shows how many of the 100 captured packets belong to each protocol.
- TCP traffic is the highest, with most of the packets being TCP.
- UDP traffic is present but much lower than TCP.
- ICMP and “Other” protocols appear only in small numbers.
- Overall, your network capture is clearly dominated by TCP packets.

6. Firewall with iptables

Goal

Configure iptables to allow incoming SSH (22) and HTTP (80), and block all other inbound traffic. This demonstrates a simple default-deny host firewall policy while preserving remote management and web access

Rules	Port
Allow	SSH(22) and HTTP(80)
Block	All other inbound

7. Configurations

1. Flush rules

- FirstClears all existing rules in all chains of the current table (filter) **sudo iptables -F**

2. Set default policies

- Sets the default policy for inbound packets to DROP; anything not explicitly allowed will be silently discarded. **sudo iptables -P INPUT DROP**
- Blocks packets being forwarded through this host **sudo iptables -P FORWARD DROP**
- Allows all outbound traffic from the host by default**. sudo iptables -P OUTPUT ACCEPT**

3. Allow loopback traffic

- Accepts all packets coming from the loopback interface lo (127.0.0.1), which is required for many local applications. **sudo iptables -A INPUT -i lo -j ACCEPT**

4. Allow established connections

- Allows inbound packets that are part of an already established connection or are related to one. **sudo iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT**

5. Allow specific services

- Accepts new incoming TCP connections to port 22, enabling remote SSH access**. sudo iptables -A INPUT -p tcp --dport 22 -j ACCEPT**
- Accepts new incoming TCP connections to port 80, allowing HTTP web traffic to a web server on this host**. sudo iptables -A INPUT -p tcp --dport 80 -j ACCEPT**

8. Testing

- Start or confirm services: sshd and a simple HTTP server
- verify the ports are running or not using : **ss -tlnp | grep -i -e ‘20’ -e ‘80’**

9. Result

```
C:\Users\rajsi>ssh kali@10.183.64.30
kali@10.183.64.30's password:
Linux kali 6.12.25-amd64 #1 SMP PREEMPT_DYNAMIC Kali 6.12.25-1kali1 (2025-04-30) x86_64

The programs included with the Kali GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Kali GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Dec 18 03:19:23 2025 from 10.183.64.68
┌──(kali㉿kali)-[~]
└─$
```

Successful connection on port 22

```
C:\Users\rajsi>curl http://10.183.64.30

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.o
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <title>Apache2 Debian Default Page: It works</title>
    <style type="text/css" media="screen">
  * {
    margin: 0px 0px 0px 0px;
    padding: 0px 0px 0px 0px;
  }

  body, html {
```

Successful connection on port 80

```
C:\Users\rajsi>curl -I http://10.183.64.30
HTTP/1.1 200 OK
Date: Thu, 18 Dec 2025 09:15:03 GMT
Server: Apache/2.4.63 (Debian)
Last-Modified: Sat, 30 Nov 2024 12:33:01 GMT
ETag: "29cf-628208420f1c0"
Accept-Ranges: bytes
Content-Length: 10703
Vary: Accept-Encoding
Content-Type: text/html


C:\Users\rajsi>curl -I http://10.183.64.30:8000
curl: (28) Failed to connect to 10.183.64.30 port 8000 after 21012 ms: Could not connect to server
```

Connection Blocked on port 8000

The above results show that connections to ports 22 and 80 complete successfully, while traffic to port 8000 fails to connect. This indicates that SSH and HTTP are allowed by the iptables rules, but other inbound ports such as 8000 are silently dropped.

ssh kali@192.183.64.30	Allowed, connection opens
curl http://10.183.64.30	Allowed, HTTP response
curl http://10.183.64.30:8000	Blocked / filtered

10. Vulnerability Scanning with OpenVAS (GVM)

Overview

Greenbone Vulnerability Management (GVM) is an open-source framework for automated vulnerability scanning and management. It uses the OpenVAS scan engine plus a management service and web interface to discover hosts, identify known vulnerabilities, and rate them by severity using a regularly updated feed of vulnerability tests.

Setup

- To initializes the Greenbone Vulnerability Management (GVM), first we need to install the GVM using : **apt install gvm**
- After installation we need to setup our user account using : **gvm-setup**
- To check and verify we need to use this command: **sudo gvm-check-setup**
- Run start the project : gvm-start
- It should give the web interface which can be accessible through : **https://127.0.0.1:9392**

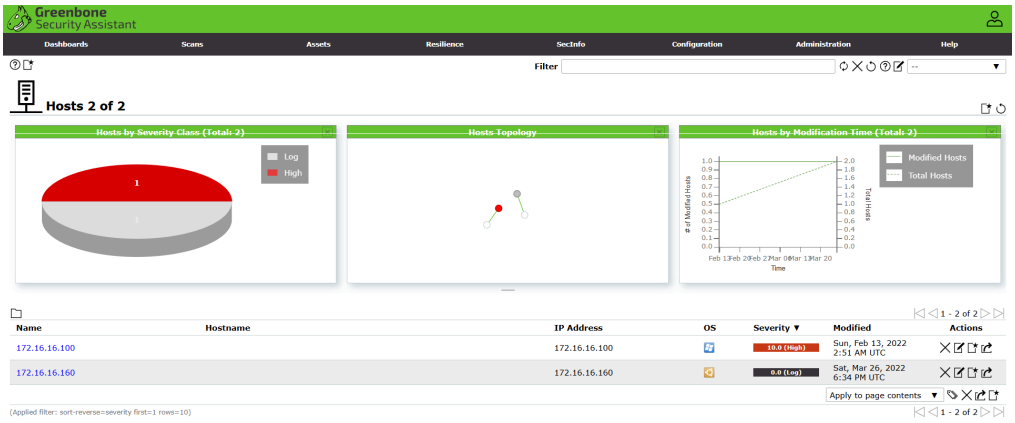
Steps to scan

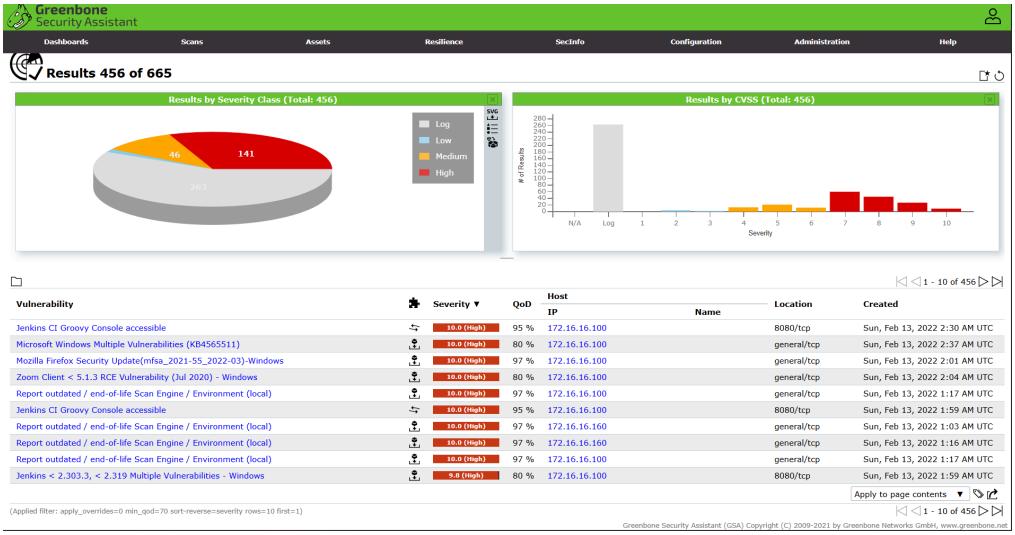
In the web UI :

1. Log in with the admin credentials created during gvm-setup.
2. Create a Target: give it a name and the IP.
3. Create a Task using that target and a scan configuration.
4. Start the task and wait for the scan to complete, then open the report.

The report will list hosts, open ports, and vulnerabilities categorized by severity (e.g., Low/Medium/High/Critical with CVSS scores). Screenshots of the summary and vulnerability list are ideal for your PDF

11. Scan Result





Result Analysis

Target - 172.16.16.100 and 172.16.16.160 (**Vulnearble VMs from HackTheBox**)

- Host 172.16.16.100 has a High severity score of 10.0, while 172.16.16.160 only has Log-level findings (0.0), indicating it is much less exposed.
- The dashboard pie chart “Tasks by Severity Class” shows most completed tasks contain High severity results, with only one in the Log category.
- The “Results 456 of 665” page shows 456 findings for the selected task, most of them High or Log severity.
- High-impact issues include items like “Jenkins CI Groovy Console accessible”, “Microsoft Windows Multiple Vulnerabilities (KB4556511)”, Firefox and Zoom update vulnerabilities, and several “Report outdated / end-of-life scan engine / environment” entries, all mainly affecting host 172.16.16.100 on ports such as 8080/tcp and general TCP

Vulnerability Research

The “Jenkins CI Groovy Console accessible” finding means your Jenkins Script Console can be reached in a way that lets an attacker run Groovy code on the server, which is extremely dangerous. The Script Console lives at the /script endpoint and is intended only for fully trusted admins, because any Groovy script executed there runs inside the Jenkins controller JVM with high privileges and can call Java APIs, read files, or spawn system commands. If access control on Jenkins is weak or misconfigured, an attacker who reaches this console (sometimes even unauthenticated) can dump stored secrets, execute OS commands to gain a reverse shell, and tamper with build pipelines or artifacts across the CI environment.

12. Key Learnings

- Learned how to capture live network traffic with Scapy, classify packets by protocol, and visualize the distribution using a bar chart image for clearer analysis.
- Configured a host firewall with iptables using a default-deny policy that allows only specific ports (SSH 22 and HTTP 80) while silently dropping other inbound connections.
- Observed the difference between dropped traffic (timeouts) and actively rejected traffic (“connection refused”), deepening understanding of firewall behavior and TCP/IP responses.
- Set up and used GVM/OpenVAS to run a vulnerability scan against a test target, then interpreted the findings in terms of severity and potential impact on the system.
- Combined packet sniffing, firewall rules, and vulnerability scanning into a small end-to-end workflow similar to what is used in real VAPT and defensive security operations.

13. Conclusion

This task provided hands-on experience across three essential layers of network security: traffic observation, traffic control, and vulnerability assessment. By scripting packet sniffing with Scapy, enforcing iptables rules, and running GVM/OpenVAS scans, it became clear how attackers are detected or blocked and how defenders can proactively uncover and remediate weaknesses. The practical exercises reinforced theoretical networking concepts and showed how automation and proper configuration significantly improve both offensive testing and defensive hardening in a lab environment and in real-world networks.

14. References

- Scapy - https://scapy.readthedocs.io/en/latest/usage.html
- Scapy packet sniffing - https://www.geeksforgeeks.org/python/packet-sniffing-using-scapy/
- iptables firewall guides - https://fideloper.com/iptables-tutorial
- OpenVAS vulnerability scanner - https://www.greenbone.net/en/
- OpenVAS scanning tips - https://hackertarget.com/openvas-tutorial-tips/