LAB: PORT SCANNING

Tools used:

Ubuntu 24.04 LTS, Python 3, Nmap, PHP, NSE (Nmap Scripting Engine), Wireshark.

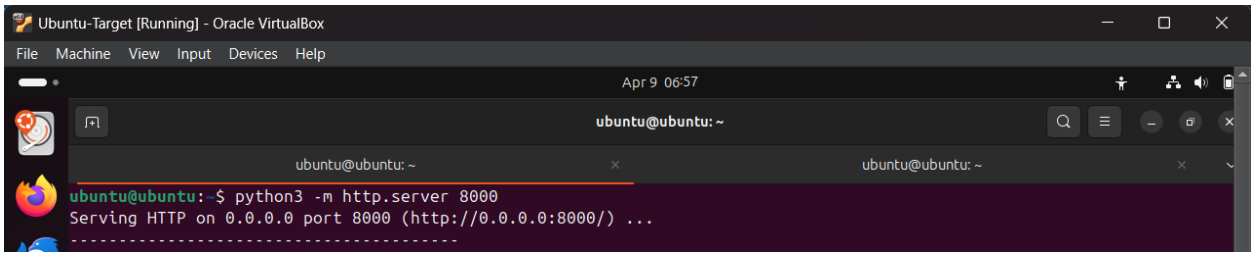
Part 1: Active and Passive Port Scanning

For this part, I used a Bridged Adapter for both the target and attacker VMs. This setup lets both machines act like separate devices on the same physical network; just like two computers connected to the same Wi-Fi. It's a pretty common setup for internal penetration testing or when an attacker already has access to a network and wants to scan for open ports or services to gain more control.

1.1 Setting Up the Target Machine

On the target VM, start a service to listen on a specific TCP port. For this lab, a simple Python HTTP server is used.

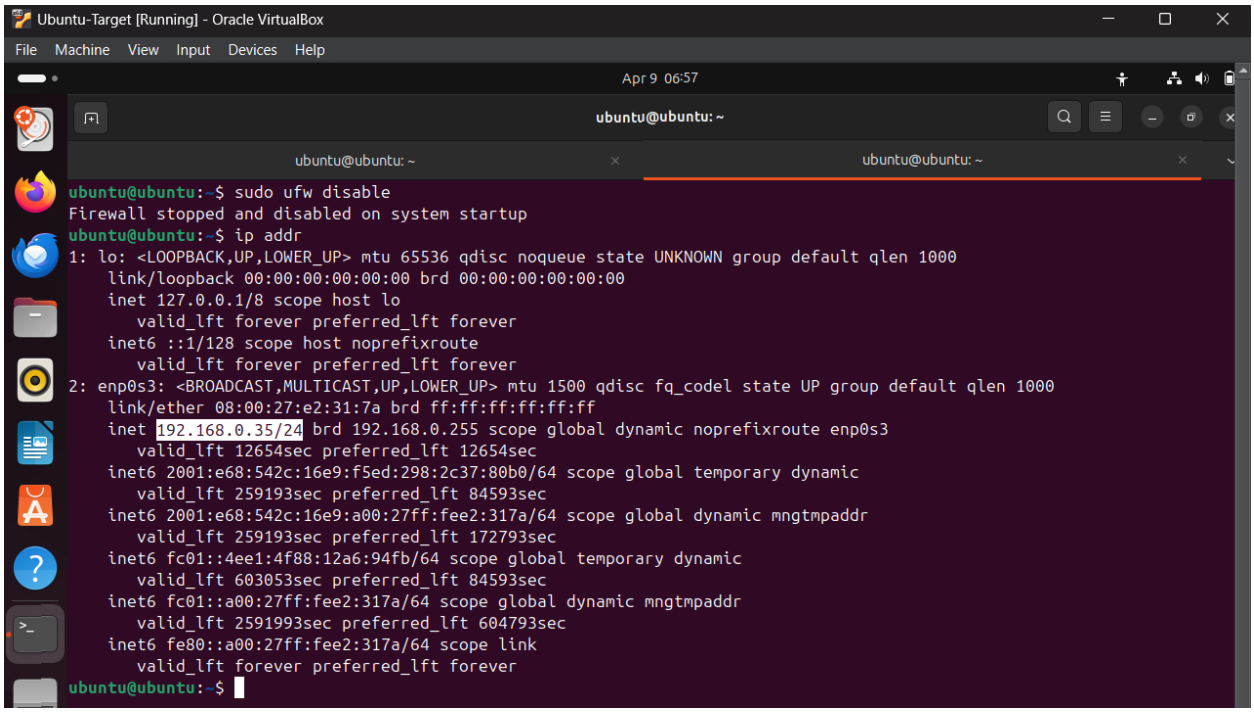
- python3 -m http.server 8000



1.1.1 Confirm the Port is Open

Ensure the firewall is disabled so the port is accessible.

- IP address of the target VM: 192.168.0.35/24



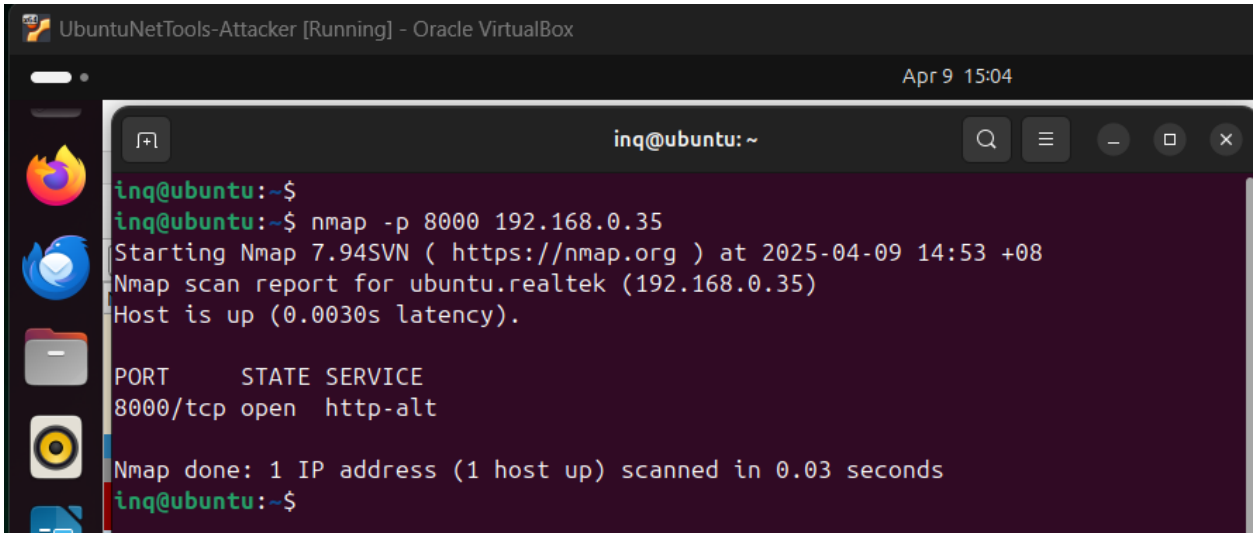
1.2. Capturing Traffic on the Attacker Machine

On the attacker VM, use Wireshark to monitor network traffic. Capture data from the network interface connected to the internet. In this case, enp0s3.

1.3. Performing a Port Scan with Nmap

From the attacker VM, run Nmap to scan the specified port on the target VM:

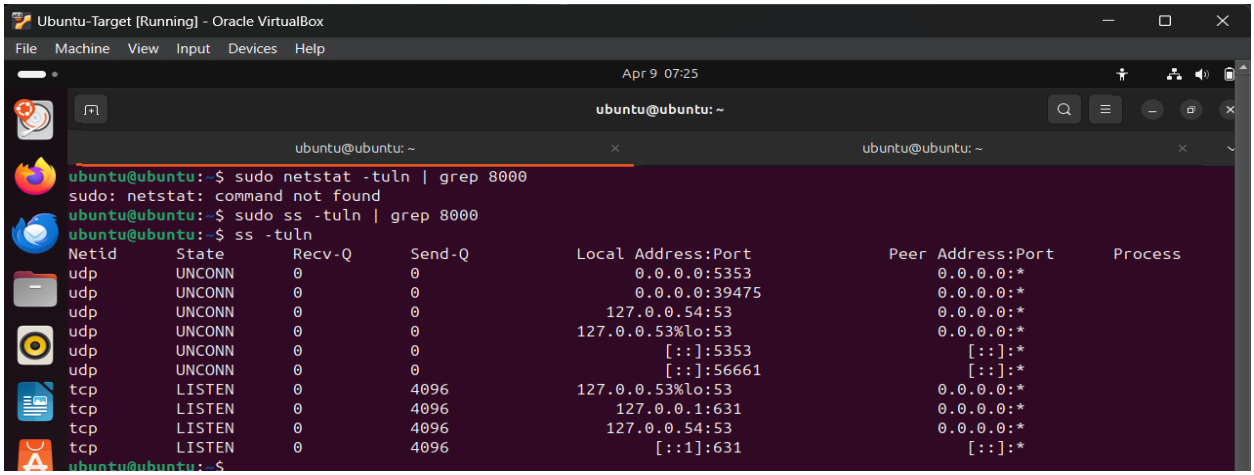
- nmap -p 8000 192.168.0.35



1.4. Scanning Without an Active Service

Stop the Python server on the target machine to simulate a scenario where no service is listening:

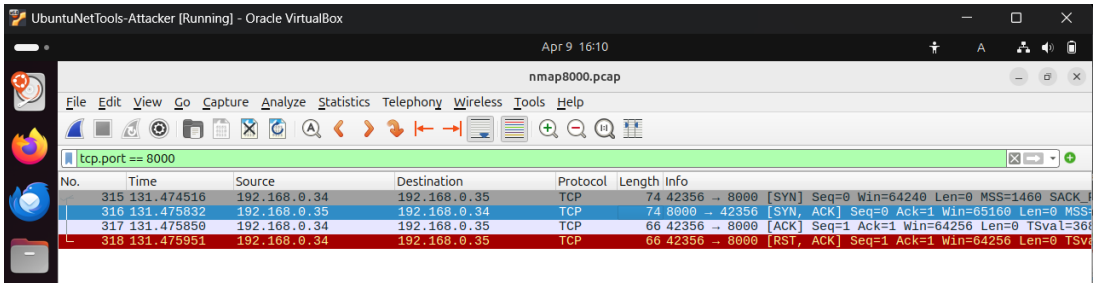
- ss -tuln



Repeat steps 2 and 3 to observe how packet behavior changes when the port is closed.

1.5. Analyzing Packet Capture Results

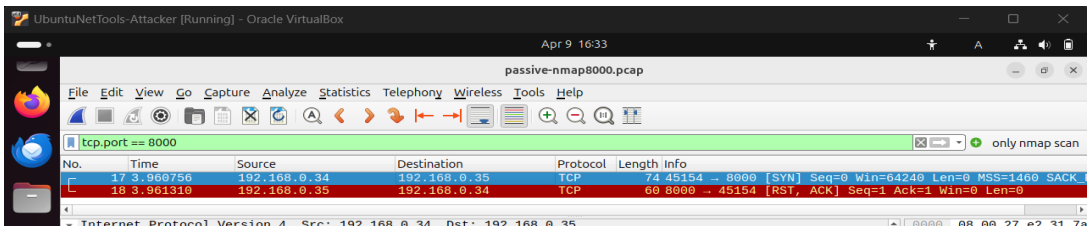
1.5.1. When a Service is Listening (Active Scanning)



Captured packets will show TCP flags like:

- SYN and ACK: Indicate that the port is open and responding.
- RST and ACK: Signal that the port is being closed/reset.

1.5.2. When No Service is Listening (Passive Scanning)



- SYN: Shows an attempt to initiate a connection (from the scanner).
- RST and ACK: Indicate that no service is available and the connection is being reset.

Part 2: Detecting a Custom Vulnerability Header Using Nmap NSE

2.1 Setting Up a Custom PHP Service on the Target VM

Create a directory and file to simulate a vulnerable web application.

- mkdir www
- nano index.php
    - <?php

      header('X-Vulnerable-Flag: BUG');

      ?>

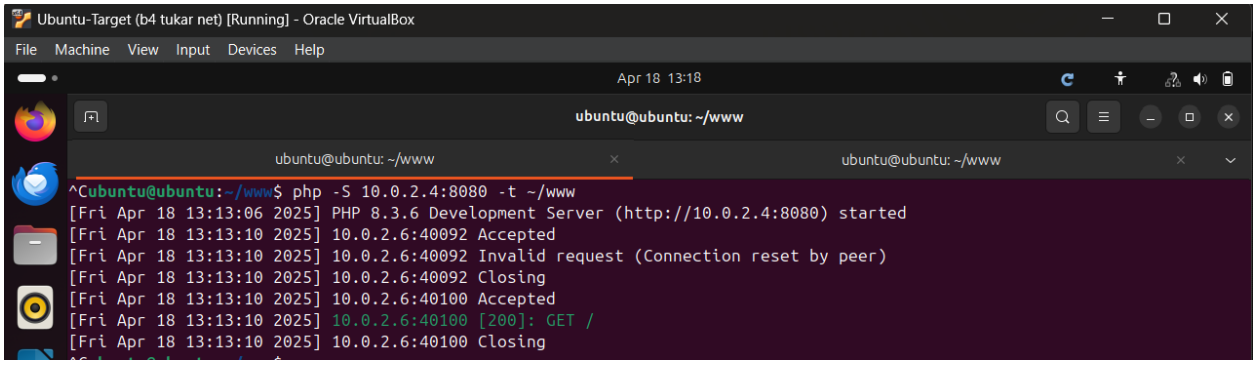
2.2. Start the PHP Server

Install PHP (if not already installed):

- sudo apt install php

Run the PHP development server (should be in the same directory as index.php):

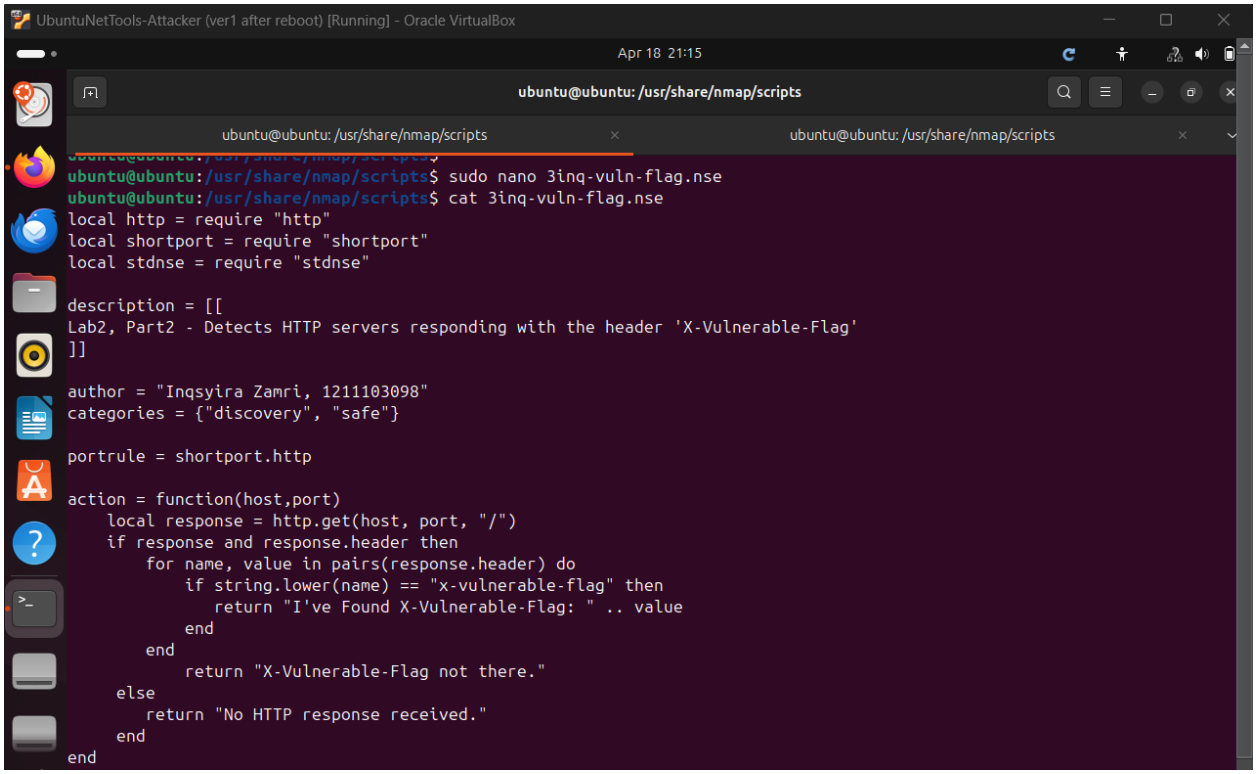
- php -S 192.168.0.35:8000 -t .



2.3. Writing a Custom Nmap NSE Script on the Attacker VM

Create a new NSE script to detect the custom X-Vulnerable-Flag header.

- sudo nano 3inq-vuln-flag.nse



The custom NSE script above uses standard libraries such as http, shortport, and stdnse to handle HTTP requests and scripting logic. It applies the port rule shortport.http, meaning the script only targets HTTP ports. When executed, it sends a GET request to the root (/) of the target web server and checks if a response with headers is received. It then loops through the headers to detect one named X-Vulnerable-Flag, ignoring case sensitivity. If the header is found, the script prints its value; if not, it reports that the flag is not present. In cases where no HTTP response is received at all, the script returns a corresponding message.

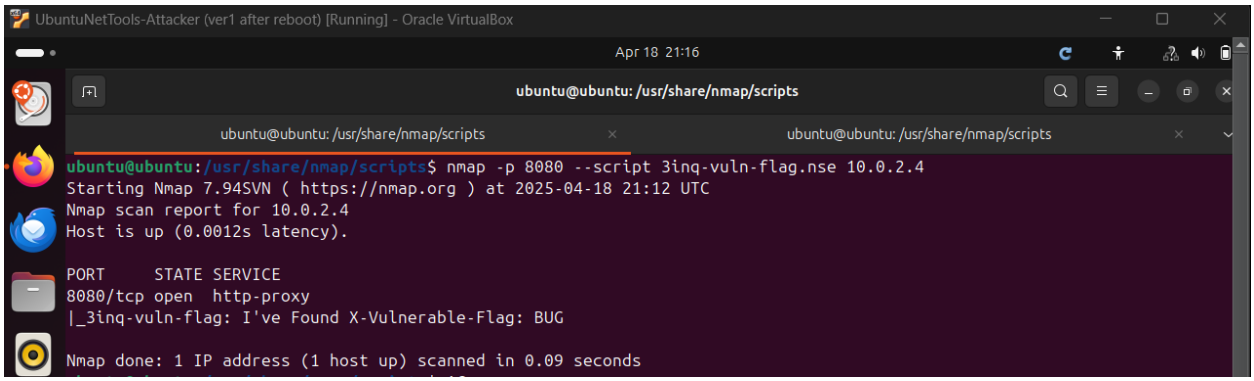
2.3.1 Update the Nmap Script Database

After creating the script, it's important to update Nmap's script database so it recognizes the new .nse file:

- sudo nmap --script-updatedb

2.4. Running the NSE Script Against the Target VM

- nmap --script vuln-flag-detect.nse -p 8000 192.168.0.35



In the screenshot above, the script successfully detected the custom header and printed out: "I've Found X-Vulnerable-Flag: BUG". This indicates that the target HTTP server responded with the X-Vulnerable-Flag header, and its value was set to "BUG". This confirms that the custom vulnerability flag was present, and the script worked as intended.