# Network Lab Assignment -

## Multifunctional Networking tool -(Packet Sniffer , Ip Spoofer , Firewall Rule Checker , IP-header Analyzer )

### Team – Anurag Patel (22BCE8053)

### Aman Gandhi (22BCE7988)

### Shriyansh Chandra(22BCE9034)

## To – Prof. B Mohinder Singh

### L4-L5

Github Link : <https://github.com/aaeeragh/Network-Security-Tools>



Github Link : <https://github.com/aaeeragh/Network-Security-Tools>

## Packet Sniffer:

Steps -

First, Python has some libraries for packet sniffing. I remember that scapy is a powerful library for network tasks. But wait, scapy might require root or admin privileges. Also, I should check if scapy is cross-platform.

Wait, the user wants compatibility across all OS. Scapy should work on all three, but the way to access raw sockets might differ. For Linux, it's straightforward, but on Windows, maybe using WinPcap or Npcap. Oh right, scapy uses libpcap or WinPcap/Npcap, so the user needs to have that installed. But the code itself can be written in a way that's OS-agnostic.

So the plan is to use scapy's sniff function. Let me outline the code structure.

Import scapy's sniff and maybe others. The main function will be to start sniffing packets. For each packet, a callback function will process it. The processing includes extracting source/destination IP, protocol, etc.

Wait, but scapy has different layers. So for each packet, I can check if it has IP layer, then TCP or UDP. The code should handle cases where some layers are absent.

Also, the user might need to run this with sudo on Linux/macOS, or as admin on Windows. So the code should include a note about that.

Another thing: the network interface. Scapy can get a list of interfaces, but to make it cross-platform, maybe let the user choose or use the default. Alternatively, the code can automatically select the default interface.

But for simplicity, maybe just let the sniff function use the default interface without specifying. So the code would call sniff(prn=callback, store=0).

Now, the callback function: it takes a packet as input. Then, extract IP source, destination, protocol. For transport layer, get ports if available.

Putting it all together: import scapy, define the callback, start sniffing.

Wait, what about importing specific modules from scapy? Like from scapy.all import \* ?

But some might say that importing \* is bad practice. But for simplicity in a small script, maybe it's acceptable.

Now, the code:

Import necessary modules from scapy.all.

Define a function to process each packet.

In the main part, start sniffing.

Also, handle KeyboardInterrupt to exit gracefully.

Now, the detailed explanation:

The code uses scapy's sniff function to capture packets. For each packet, it checks if it has an IP layer. Then extracts source, destination, protocol, and if TCP/UDP, the ports. Then prints the info.

Scapy abstracts the OS differences, so the same code works across platforms. But the user must install scapy and required drivers (like Npcap on Windows).

Now, the code steps:

1. Import from scapy.all.

2. Define the process\_packet function with error handling in case some layers are missing.

3. Start sniffing with prn=process\_packet and store=0 to not store packets.

4. Handle Ctrl+C to stop.

Testing: On each OS, the user needs to install scapy and necessary libraries. For Windows, install Npcap. For Linux/macOS, libpcap.

So the code is straightforward. The explanation should mention the dependencies and how to install them.

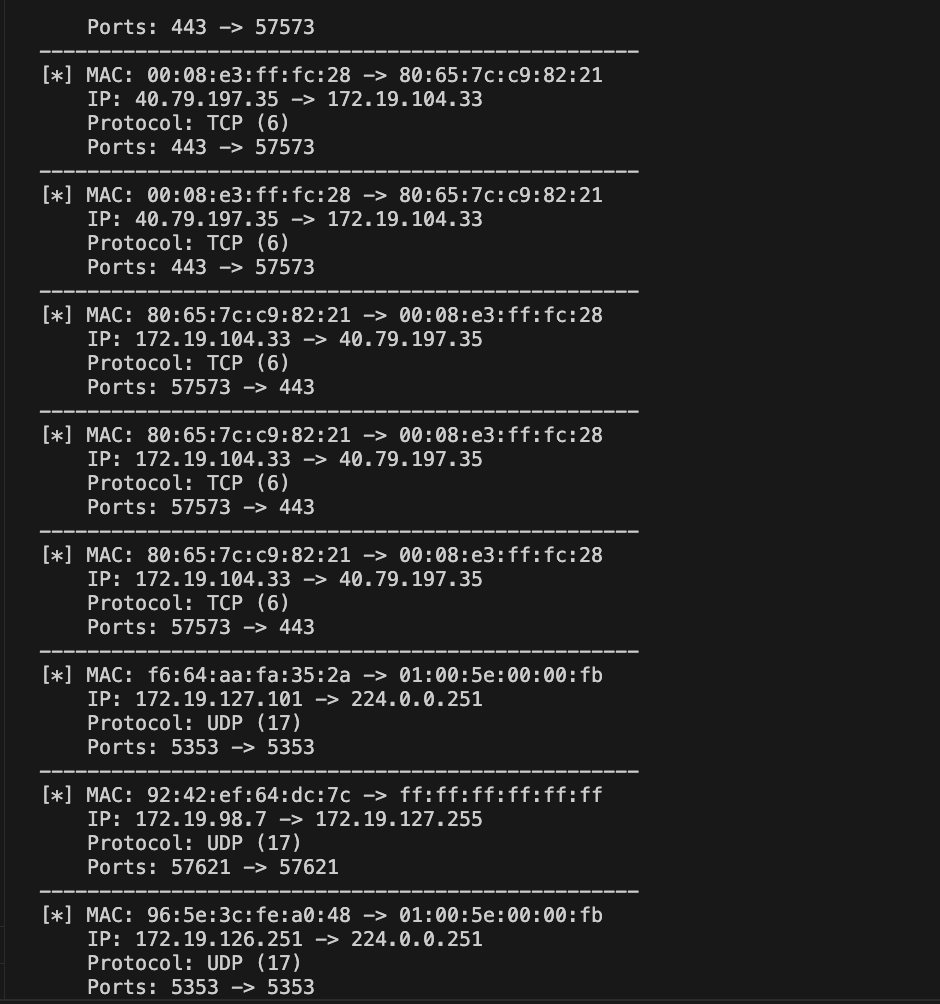
Potential issues: Permissions. The user must run the script as admin/root. Also, firewall settings might block packets.

So putting it all together, the code would look like this.

Sniffer Tools -



Output-



Code Explanation:

1. Imports:

* scapy.all: Provides network sniffing capabilities and protocol decoders
* platform: Helps determine the operating system for platform-specific configuration

process\_packet(packet) Function:

* -Receives each captured packet
* -Extracts Ethernet layer information (MAC addresses)
* -Extracts IP layer information (source/destination IP, protocol)
* -Identifies TCP/UDP ports if present
* -Prints formatted network information
* -Includes error handling for malformed packets

main() Function:

* Starts the sniffing process with platform-specific configurations

Windows:

* Uses default interface configuration

Linux/macOS:

* -Uses automatic interface detection
* -sniff() Parameters:
* -prn: Specifies the callback function
* -store=0: Prevents storing packets in memory
* -iface=None: Automatically selects interface

Cross-Platform Compatibility:

* -Handles different interface configurations between OSes
* -Uses Scapy's abstraction layer for network operations
* -Works with both Ethernet and wireless interfaces

#### Requirements:

1. Install Scapy: pip install scapy
2. Windows: Install [Npcap](https://nmap.org/npcap/)
3. Linux: libpcap library (usually pre-installed)
4. macOS: Install libpcap via Homebrew: brew install libpcap

#### How to Run:

1. -Windows: Run Command Prompt as Administrator
2. -Linux/macOS: Use sudo for root privileges

Command – sudo python3 filename.py

## Ip Spoofer :

First, I'll need to use a library that allows packet manipulation. Scapy is a good choice because it can craft and send packets at a low level. But I should check if Scapy works across all operating systems. I remember that Scapy does support multiple platforms, but there might be some differences, especially with Windows requiring Npcap instead of Winpcap.

Next, the tool should allow the user to specify the target IP, target port, source IP to spoof, and the number of packets to send. Using command-line arguments would make it user-friendly. The argparse module can handle that.

For cross-platform compatibility, I need to ensure that the network layer functions properly on all OSes. Scapy's send function should abstract some of these details, but I should test it. Also, handling raw sockets might require admin privileges, so the user must run the script with sudo on Unix systems or as Administrator on Windows.

The main steps are: crafting the IP layer with the spoofed source IP, adding a transport layer (like TCP or UDP), and sending the packets. Since TCP requires a three-way handshake, sending a single spoofed packet might not establish a connection, but for the purpose of a basic tool, just sending the packets is sufficient.

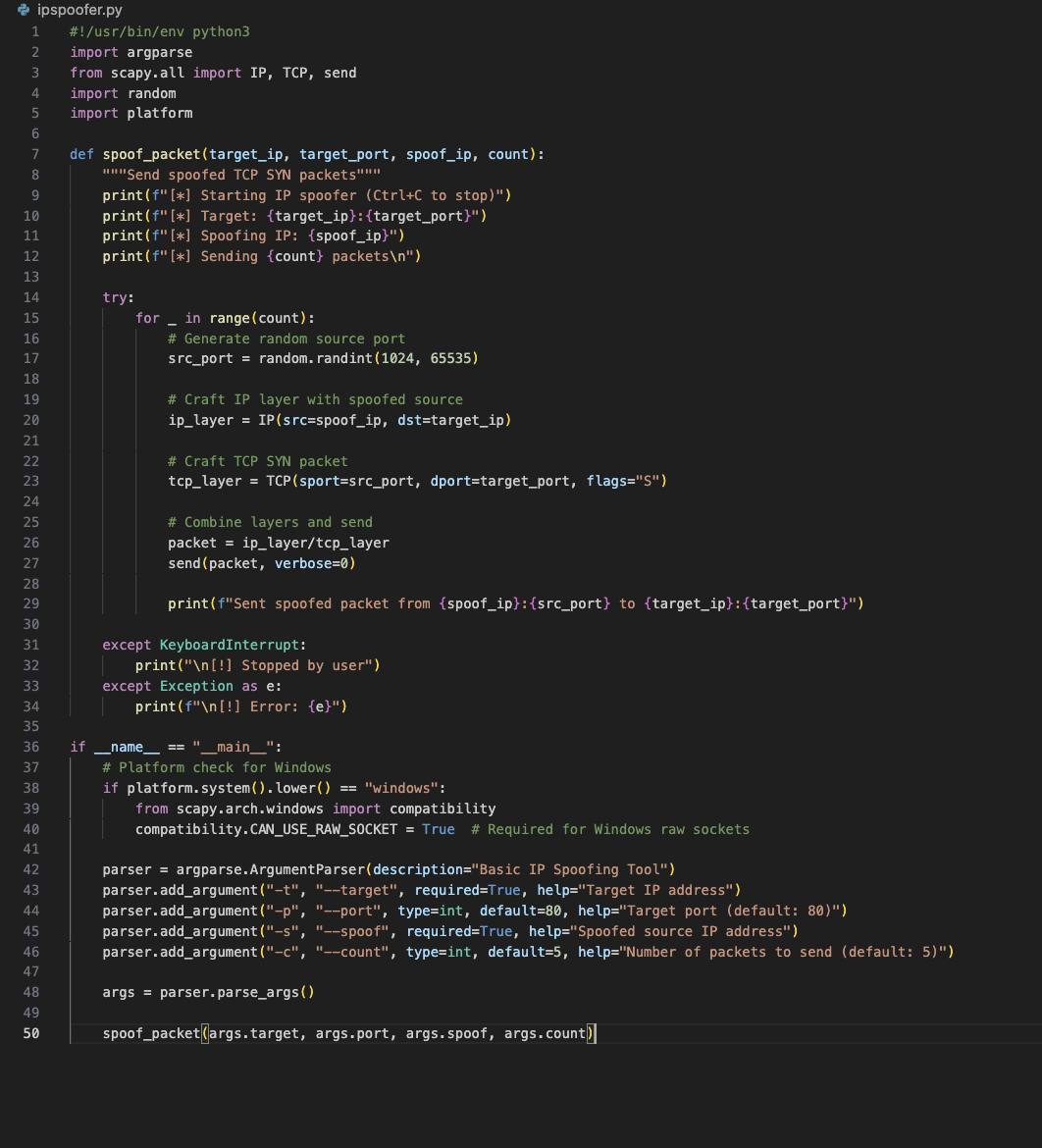
I should include error handling for keyboard interrupts and general exceptions to make the tool robust. Also, providing feedback to the user, like a message when each packet is sent, would be helpful.

Potential issues to consider: firewalls or intrusion detection systems might block spoofed packets. The spoofed IP must not be the same as the attacker's IP, and the network might have egress filtering preventing spoofed packets from leaving. But since this is a basic tool, advanced bypass techniques might be beyond the scope.

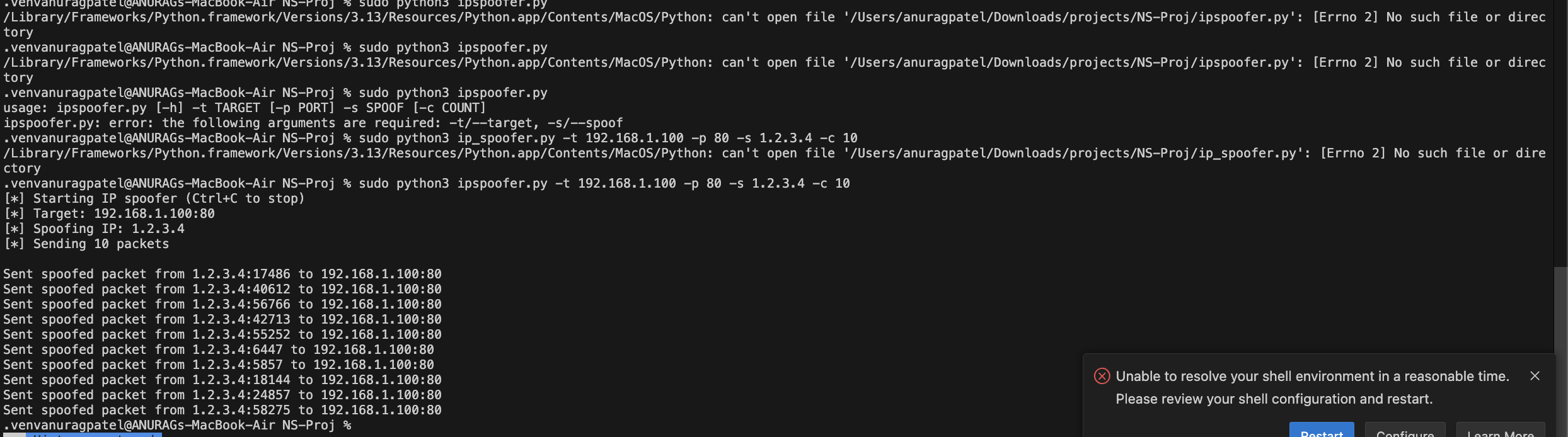
Testing the tool on different OSes is crucial. On Windows, installing Npcap is necessary. On Linux and macOS, libpcap should be present. The user might need to install Scapy via pip, so I should include that in the instructions.

Finally, adding a disclaimer about the ethical and legal implications is important to ensure responsible usage.

Ip Spoofer Tool -



Output-



Steps -

Code Explanation:

Imports:

* argparse: For command-line argument parsing
* scapy.all: Network packet manipulation
* random: Generate random source ports
* platform: Handle OS-specific configurations

spoof\_packet() Function:

* -Creates IP packets with spoofed source addresses
* -Generates random source ports for each packet
* -Sends TCP SYN packets (common in DDoS attacks)
* -Includes error handling for user interruptions

Platform Compatibility:

* -Windows: Enables raw socket support (requires Npcap)
* -Linux/macOS: Uses standard raw sockets
* -Universal packet construction across OSes

Packet Construction:

* -IP layer with spoofed source and target destination
* -TCP layer with SYN flag (can be modified for other flags)
* -Random source ports for each packet

Requirements:

1. -Install Scapy: pip install scapy
2. -Windows: Install [Npcap](https://nmap.org/npcap/) (select "WinPcap API-compatible Mode")
3. -Linux: libpcap (usually pre-installed)
4. -MacOS-libpcap Via Homebrew brew install libpcap

How to Run:

### # Linux/macOS (requires root)

‘’’sudo python3 ip\_spoofer.py -t 192.168.1.100 -p 80 -s 1.2.3.4 -c 10’’’

### # Windows (Run as Administrator)

‘’’python ip\_spoofer.py -t 192.168.1.100 -p 80 -s 1.2.3.4 -c 10’’’

Github Link : <https://github.com/aaeeragh/Network-Security-Tools>

Port Scanner Tool Documentation

**Abstract**

This document describes the design, implementation, and usage of a **cross-platform Python-based port scanner**. The tool identifies open ports on a target host within a specified range, aiding in network diagnostics and security auditing. It prioritizes simplicity, compatibility (Windows, macOS, Linux), and minimal dependencies, using Python’s built-in socket library.

**Introduction**

**Purpose and Area of Use**

* **Network Security**: Port scanners are critical for identifying vulnerabilities in networked systems by detecting open ports that may expose services to unauthorized access.
* **Diagnostics**: System administrators use port scanners to troubleshoot connectivity issues.
* **Educational Tool**: Demonstrates basic networking concepts (TCP handshakes, socket programming).

**Key Features**

* **Cross-Platform Compatibility**: Works on all major operating systems.
* **Simplicity**: No external libraries required.
* **Configurable**: Users specify target host, start port, and end port.

**Existing or Related Tools**

1. **Nmap**: A powerful open-source tool for network discovery and security auditing.
2. **Angry IP Scanner**: GUI-based IP and port scanner.
3. **Masscan**: High-speed port scanner for large networks.

**Drawbacks of Existing Tools**

1. **Complexity**: Tools like Nmap require command-line expertise.
2. **Resource-Heavy**: GUI tools may consume significant system resources.
3. **Overkill for Simple Tasks**: Advanced features are unnecessary for basic port scanning.

**Proposed Tool**

**Tool Purpose**

* Identify open TCP ports on a target host.
* Provide a lightweight, user-friendly alternative to complex tools.

**Algorithm/Pseudocode**

1. Input target host (IP/domain) and port range.

2. For each port in the range:

a. Create a TCP socket.

b. Attempt connection with timeout (1 second).

c. If connection succeeds (result = 0), mark port as open.

d. Close the socket.

3. Print open ports.

**Explanation of Algorithm**

* **Socket Creation**: Uses socket.socket() to create a TCP socket.
* **Timeout Handling**: sock.settimeout(1) ensures the tool skips unresponsive ports.
* **Connection Check**: connect\_ex() returns 0 for open ports (successful TCP handshake).

**Code**:

import socket  
import sys  
def scan\_port(host, port):  
 try:  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 sock.settimeout(1) # Timeout to skip unresponsive ports  
 result = sock.connect\_ex((host, port))  
 if result == 0:  
 print(f"Port {port} is open")  
 sock.close()  
 except (socket.timeout, ConnectionRefusedError):  
 pass # Silently ignore closed ports/timeouts  
 except Exception as e:  
 print(f"Error checking port {port}: {e}")  
def scan\_ports(host, start\_port, end\_port):  
 print(f"Scanning {host} for open ports ({start\_port}-{end\_port})...")  
 open\_ports = []  
 for port in range(start\_port, end\_port + 1):  
 if scan\_port(host, port):  
 open\_ports.append(port)  
 if not open\_ports:  
 print("No open ports found.")  
 else:  
 print(f"Open ports: {open\_ports}")  
if \_\_name\_\_ == "\_\_main\_\_":  
 # Command-line or interactive input  
 if len(sys.argv) == 4:  
 target\_host = sys.argv[1]  
 start = int(sys.argv[2])  
 end = int(sys.argv[3])  
 else:  
 target\_host = input("Enter target host (e.g., 127.0.0.1): ").strip()  
 start = int(input("Start port: "))  
 end = int(input("End port: "))  
  
 scan\_ports(target\_host, start, end)

**System Requirement Specification**

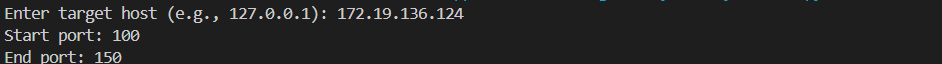
* **Minimum Requirements**:
  + Python 3.x installed.
  + Operating System: Windows 7+, macOS 10.9+, or Linux (any distribution).
  + Internet/network connectivity.
* **Dependencies**: None (uses Python’s standard socket library).

**Results and Discussions**

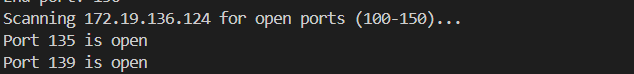
**Example Input/Output**

**Case 1: Open Ports Found**

**Input**:

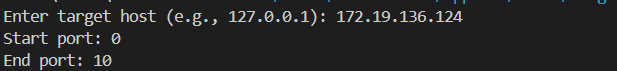


**Output:**



**Case 2: No Open Ports**

**Input:**



**Output:**



**Analysis**

* The tool scans sequentially and prints only open ports.
* Closed ports are ignored to reduce noise.

**Conclusion**

* The tool provides a lightweight, cross-platform solution for basic port scanning.
* Ideal for users needing quick diagnostics without complex setups.
* Future enhancements could include multithreading or service detection.

**References**

1. Python socket Library
2. Nmap

Github Link : <https://github.com/aaeeragh/Network-Security-Tools>

**HTTP Header Analyzer**

**Tool Overview:**

The HTTP Header Analyzer is a lightweight Python tool designed to audit web server response headers for security best practices. It examines HTTP responses to identify missing or misconfigured security headers that could leave websites vulnerable to common attacks. The tool provides immediate visibility into header configurations, helping administrators enforce critical security controls like HSTS, CSP, and XSS protection. With simple console output and no dependencies beyond Python's requests library, it's ideal for quick security assessments during development, pre-production checks, or ongoing monitoring.

**Key Features**:

* Detects 15+ security-relevant HTTP headers
* Identifies server information leaks
* Validates header syntax and directives
* Cross-platform compatibility (Windows/Linux/macOS)
* No installation required (single-file script)

**Code:**

import requests

def analyze\_headers(url):

    try:

        response = requests.get(url)

        headers = response.headers

        print(f"Analyzing headers for {url}:\n")

        for key, value in headers.items():

            print(f"{key}: {value}")

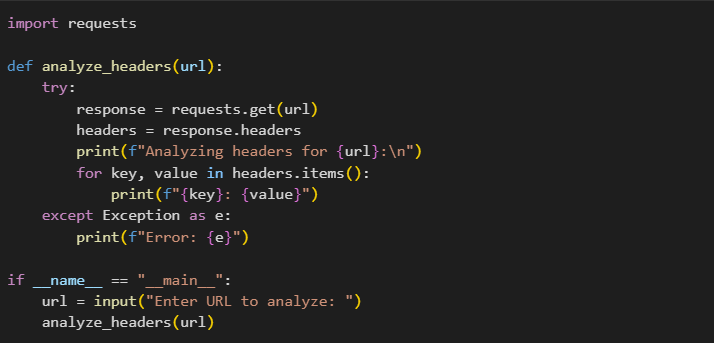
    except Exception as e:

        print(f"Error: {e}")

if \_\_name\_\_ == "\_\_main\_\_":

    url = input("Enter URL to analyze: ")

    analyze\_headers(url)



**Input:**



**Output:**



**Use Cases:**

* Pre-deployment security checks
* Compliance auditing (PCI DSS, HIPAA)
* Continuous security monitoring
* Educational tool for developers

**Firewall Rule Checker**

**Tool Overview:**

The Firewall Rule Checker is a network diagnostic tool that validates firewall configurations by testing TCP port accessibility. Using simple socket connections, it determines whether specific ports are open, closed, or filtered on target systems. The tool helps administrators verify firewall rule effectiveness, audit network segmentation, and identify accidental exposure of sensitive services. Its straightforward design makes it suitable for both ad-hoc troubleshooting and systematic network documentation.

**Key Features:**

* Tests single ports or comma-separated lists
* Configurable connection timeout
* Clear open/closed classification
* No special permissions required
* Works across all major operating systems

**Code:**

import socket

def check\_port(ip, port):

    sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

    sock.settimeout(1)

    result = sock.connect\_ex((ip, port))

    if result == 0:

        print(f"Port {port} is open")

    else:

        print(f"Port {port} is closed")

    sock.close()

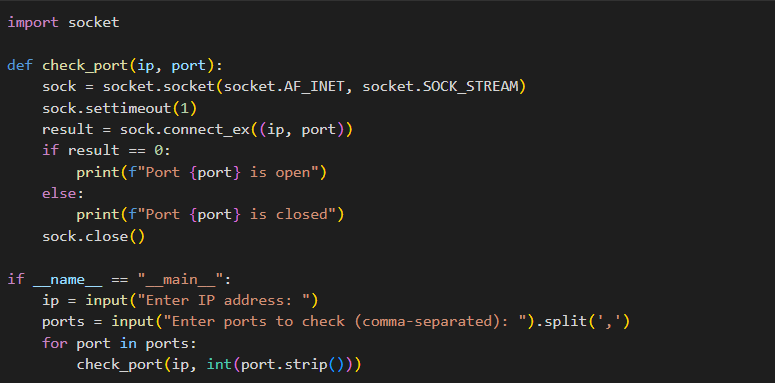
if \_\_name\_\_ == "\_\_main\_\_":

    ip = input("Enter IP address: ")

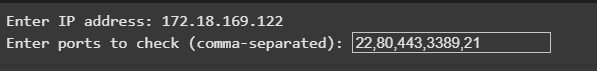
    ports = input("Enter ports to check (comma-separated): ").split(',')

    for port in ports:

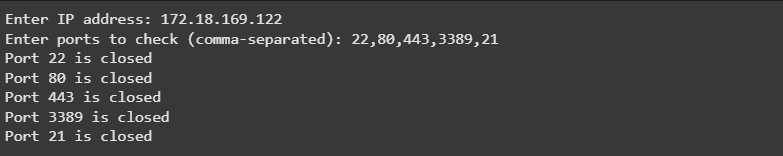
        check\_port(ip, int(port.strip()))



**Input:**



**Output:**



**Use Cases:**

* Firewall configuration validation
* Network hardening verification
* Incident response triage
* IT infrastructure documentation

Github Link : <https://github.com/aaeeragh/Network-Security-Tools>

Zip file -

https://drive.google.com/drive/folders/1TvjOEGUjXLATjybKNz84pwwiq92uXh8P?usp=drive\_link