Write a Port Scanner using Python in 10 Minutes

July 20, 2023 • 5 min read
Tags: Python

Port scanning is a technique that allows you to discover which ports are open or closed on a target host or network. Port scanning can be useful for network security, penetration testing, or ethical hacking.

In this post, we will explore 3 possible ways to create a port scanner in Python using the socket, python-nmap, and scapy libraries.

Using the socket module

The built-in socket module provides low-level access to network interfaces and protocols. The main advantage of using the socket module is it doesn't need any dependencies.

Here is the code:

```
import socket

def scan_ports(host, ports):
    for port in ports:
        try:
            s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            s.connect((host, port))
            print(f'Port {port} is open.')
        except socket.error as e:
            pass

# Usage example
ports = [21, 22, 80, 443, 3306, 5432]
scan_ports('localhost', ports)
```

Code explanation:

1. The `scan_ports` function takes two arguments, `host` and `ports`. `host` represents the target host to scan, and `ports` is a list of port numbers that need to be checked.

2. It uses a `for` loop to iterate over each port number in the ports list.

3. Inside the loop, it tries to create a TCP socket and connect to the target host and port. If the connection is successful, it means the port is open and accessible.

4. The code silently handle any `socket.error` exceptions.

Using the python-nmap library

The python-nmap library is a wrapper for the nmap tool, which is a powerful and popular port scanner and network mapper. To use this library, you need to have nmap installed on your machine.

Install the python-nmap library:

```
pip install python-nmap
```

The code to scan the ports:

```
import nmap

def scan_ports(host, start_port, end_port):
    nm = nmap.PortScanner()
    nm.scan(host, f'{start_port}-{end_port}')

    for host in nm.all_hosts():
        print(f"Scanning ports on {host}.")
        for port in nm[host]['tcp'].keys():
            state = nm[host]['tcp'][port]['state']
            print(f"Port {port}: {state}")

# Usage example
scan_ports('localhost', 3000, 4001)
```

Code explanation:

1. The `scan_ports` function takes three parameters: `host`, `start_port`, and `end_port`. `host` represents the target host to scan, `start_port` and `end_port` define the range of ports to be scanned.

2. The function creates an instance of the `PortScanner` and perform the scan.

3. For each of the scanned hosts, it iterates through the TCP ports and print the state for each port.

Using the scapy library

The scapy library is a powerful packet manipulation tool that allows you to create, send, receive, and analyze network packets. You can use the `scapy` library to craft custom packets and perform various types of scans, such as SYN scan, ACK scan, XMAS scan, and more.

Install the scapy library:

```
pip install scapy
```

The code to scan ports:

```
from scapy.all import *

def scan_ports(host, ports):
    for port in ports:
        packet = IP(dst=host)/TCP(dport=port, flags='S')
        response = sr1(packet, timeout=1, verbose=0)
        if response and response.haslayer(TCP) and response[TCP].flags ==
'SA':
            print(f'Port {port}: open')

# Usage example
ports = [21, 22, 80, 443, 3306, 5432]
scan_ports('localhost', ports)
```

Code explanation:

1. The `scan_ports` function takes two parameters: `host` and `ports`. `host` represents the target host to scan, and `ports` is a list of port numbers that need to be checked.

2. It uses a `for` loop to iterate over each port number in the ports list.

3. For each port, a SYN packet (TCP SYN) is created using Scapy. The packet is constructed with the `IP()` and `TCP()` functions. The `IP()` function specifies the destination IP address (`dst=host`), and the `TCP()` function specifies the destination port (`dport=port`) and sets the TCP flags to "S" (SYN).

4. The constructed SYN packet is sent using the `sr1()` function, which sends the packet and captures the response. The `timeout=1` parameter sets the timeout for the response to 1 second, and `verbose=0` suppresses Scapy's output during packet sending.

5. After sending the packet, the function checks if a response was received. It also verifies if the response contains a TCP layer and if the TCP flags of the response indicate a SYN-ACK response (SA).

6. Print the results.

Summary

In this blog post, we have shown you how to write a port scanner using the socket, python-nmap, and scapy libraries.

Each of these methods has its own advantages and disadvantages. The best method to use will depend on your specific needs. If you need a simple port scanner, the socket module is a good option. If you need a more efficient port scanner, python-nmap and scapy are good options.