Internship report

Cybersecurity

Task2 : Network traffic analyzer

Produced by :

ASMAE EZZINE

17/07/2024

Table des matières

Introduction :

In today's digital age, network security is of paramount importance as cyber threats become increasingly sophisticated and pervasive. Organizations must implement robust mechanisms to monitor and analyze network traffic to protect sensitive data and ensure uninterrupted services. This project, titled "Network Traffic Analyzer," aims to develop a comprehensive tool to monitor and analyze network traffic, identify potential security threats, and provide real-time alerts to network administrators.

The Network Traffic Analyzer leverages the power of packet sniffing and real-time analysis to detect anomalies such as port scans and Distributed Denial of Service (DDoS) attacks. By employing advanced techniques, this tool can provide timely alerts to administrators, allowing them to take prompt action to mitigate threats.

I. The tools used :

A. Flask:

- Purpose: A lightweight WSGI web application framework in Python.
- Usage: To create the web server and handle HTTP requests, enabling the development of the user interface and API endpoints.

B. Scapy:

- Purpose: A powerful Python library for network packet manipulation.
- Usage: To capture and analyze network packets in real-time, enabling the detection of potential security threats.

C. HTML/CSS:

- Purpose: Standard markup language and style sheet language for creating web pages.
- Usage: To design and style the web interface, providing a user-friendly environment for visualizing network traffic and alerts.

D. JavaScript:

- Purpose: A programming language for creating dynamic content on web pages.
- Usage: To periodically fetch and display alerts from the server, ensuring real-time updates on the web interface.

E. Threading (Python Standard Library):

- Purpose: A module for running multiple threads (tasks, function calls) at once.
- Usage: To handle packet sniffing and resetting counters concurrently without blocking the main application.

F. Collections (Python Standard Library):

- Purpose: A module providing specialized container datatypes.
- Usage: To utilize defaultdict for efficiently counting packets and IP addresses.

II. Code explanation :

a. Importing Libraries :

```python
from flask import Flask, render_template, jsonify
from scapy.all import sniff, IP, TCP, UDP
import threading
import time
from collections import defaultdict
```

- Flask: Used to create a web application.
- Scapy: Used to capture and analyze network packets.
- Threading: Used to run multiple threads for packet sniffing and counter resetting.
- Time: Used to implement delays in the counter reset logic.
- Collections: Used to utilize defaultdict for counting packets and IP addresses.

b. Initializing Flask and Global Variables :

```python
app = Flask(__name__)

# Dictionaries to store captured packets and alerts
packet_count = defaultdict(int)
ip_count = defaultdict(int)
alerts = []
```

- Initializes the Flask application.
- Creates dictionaries to keep track of packet counts and IP counts.
- Initializes a list to store alerts.

c. Port Scan Detection Function :

```python
# Port scan detection
def detect_port_scan(packet):
    if TCP in packet or UDP in packet:
        ip_src = packet[IP].src
        port = packet[TCP].dport if TCP in packet else packet[UDP].dport
        packet_count[(ip_src, port)] += 1

        # If more than 10 packets from the same IP to different ports within a minute
        if packet_count[(ip_src, port)] > 10:
            alert = f"Alert: Possible port scan detected from {ip_src}"
            alerts.append(alert)
            print(alert)
```

- Detects port scans by checking if more than 10 packets are sent from the same IP to different ports within a minute.

- Adds an alert if a possible port scan is detected.

d. DDoS Detection Function :

```
27  # DDoS detection
28  def detect_ddos(packet):
29      ip_src = packet[IP].src
30      ip_count[ip_src] += 1
31
32      # If more than 100 packets from the same IP within a minute
33      if ip_count[ip_src] > 100:
34          alert = f"Alert: Possible DDoS attack detected from {ip_src}"
35          alerts.append(alert)
36          print(alert)
```

- Detects DDoS attacks by checking if more than 100 packets are sent from the same IP within a minute.
- Adds an alert if a possible DDoS attack is detected.

e. Packet Callback Function :

```
38  # Callback function for packet capture
39  def packet_callback(packet):
40      if IP in packet:
41          detect_port_scan(packet)
42          detect_ddos(packet)
43          ip_src = packet[IP].src
44          ip_dst = packet[IP].dst
45          print(f"Packet: {ip_src} -> {ip_dst}")
```

- Called for each captured packet.
- Calls the port scan and DDoS detection functions.
- Prints the source and destination IPs of the packet.

f. Counter Reset Function :

```
47  # Reset counters every minute
48  def reset_counters():
49      global packet_count, ip_count
50      while True:
51          time.sleep(60)
52          packet_count = defaultdict(int)
53          ip_count = defaultdict(int)
```

- Resets the packet and IP counters every minute.

g. Packet Sniffing Function :

```
55  # Start packet capture in a separate thread
56  def start_sniffing():
57      sniff(prn=packet_callback, store=0)
```

- Starts packet sniffing and calls the packet_callback function for each captured packet.

h. Starting Threads :

```
59  # Start the packet capture thread
60  sniff_thread = threading.Thread(target=start_sniffing)
61  sniff_thread.daemon = True
62  sniff_thread.start()
63
64  # Start the counter reset thread
65  reset_thread = threading.Thread(target=reset_counters)
66  reset_thread.daemon = True
67  reset_thread.start()
```

- Starts the packet sniffing and counter reset threads.

i. Flask Routes :

```
69  # Flask routes
70  @app.route('/')
71  def index():
72      return render_template('index.html')
73
74  @app.route('/alerts')
75  def get_alerts():
76      return jsonify(alerts)
77
78  if __name__ == '__main__':
79      app.run(debug=True)
```

- Defines Flask routes:
  - '/' route renders the 'index.html' template.
  - '/alerts' route returns the alerts as a JSON response.
- Runs the Flask application.

j. HTML Structure :

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Network Traffic Analyzer</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f4;
        }
        .container {
            width: 80%;
            margin: auto;
            overflow: hidden;
        }
            #alerts {
                margin: 20px 0;
                padding: 20px;
                background: #fff;
                border-radius: 8px;
                box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
            }
            .alert {
                padding: 10px;
                margin: 10px 0;
                border-left: 6px solid #ff6f61;
                background-color: #ffe6e6;
                color: #333;
            }
    </style>
</head>
<body>
    <div class="container">
        <h1>Network Traffic Analyzer</h1>
        <div id="alerts">
            <h2>Alerts</h2>
            <div id="alert-list"></div>
        </div>
    </div>
    <script>
        function fetchAlerts() {
            fetch('/alerts')
                .then(response => response.json())
                .then(data => {
                    const alertList = document.getElementById('alert-list');
                    alertList.innerHTML = '';
```

```
                    data.forEach(alert => {
                        const alertDiv = document.createElement('div');
                        alertDiv.className = 'alert';
                        alertDiv.innerText = alert;
                        alertList.appendChild(alertDiv);
                    });
                });
            }

            setInterval(fetchAlerts, 5000);
        </script>
    </body>
    </html>
```

HTML Head Section

- Sets the character encoding to UTF-8.
- Sets the viewport to ensure the page is responsive on different devices.
- Includes a title for the web page.
- Provides styles to make the page visually appealing, such as setting the font, margins, padding, background color, and styling the container and alert elements

HTML Body Section

- Contains a header with the title "Network Traffic Analyzer".
- Contains a section for alerts, with a header and an empty div to display the alerts dynamically.

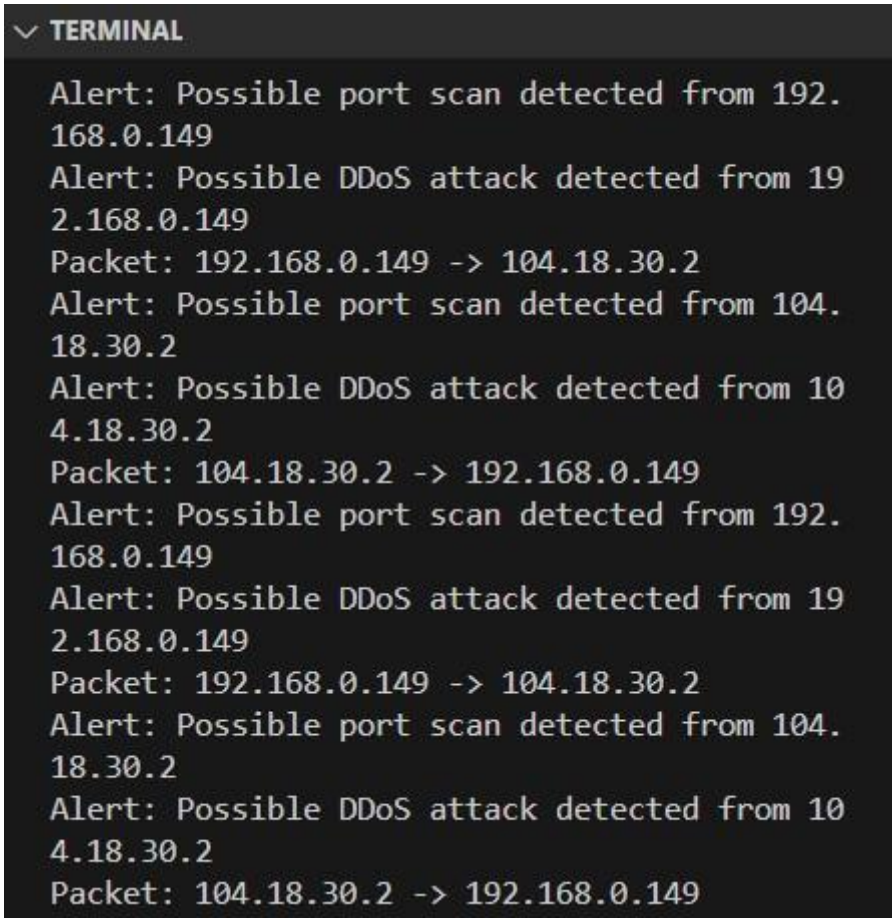
JavaScript Section

- Fetches alerts from the '/alerts' endpoint.
- Parses the JSON response and updates the 'alert-list' 'div' with the fetched alerts.
- Creates new 'div' elements for each alert and appends them to the alert list.
- Calls the fetchAlerts function every 5 seconds to update the alert list in real-time.

III. Demonstration :

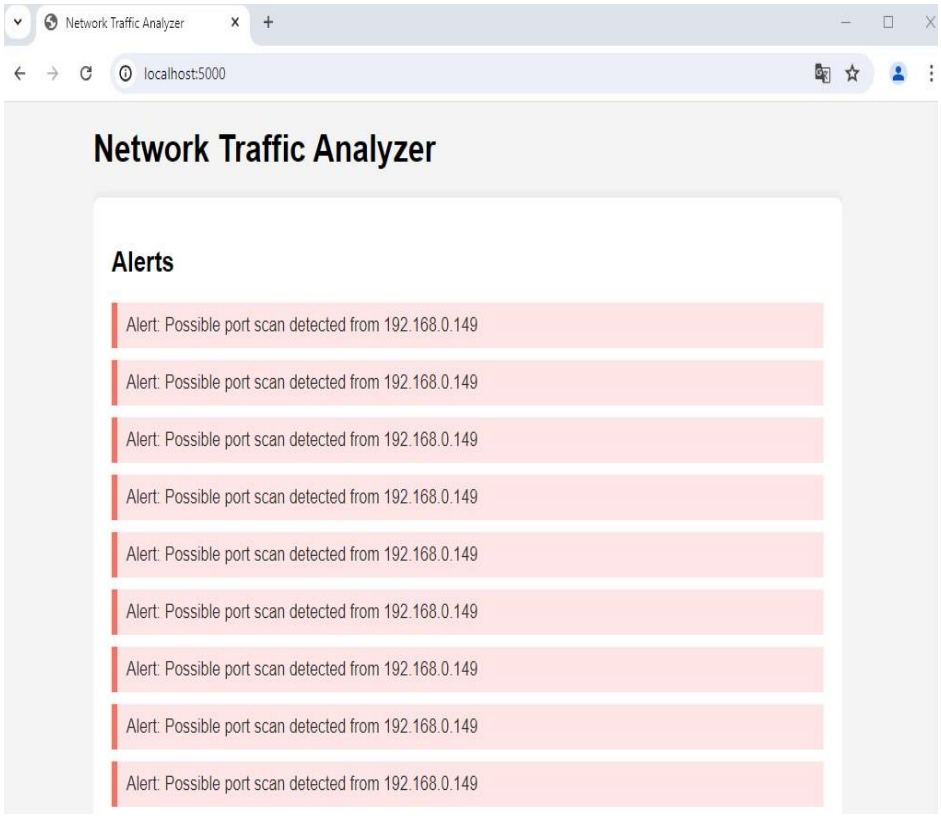
Terminal Output

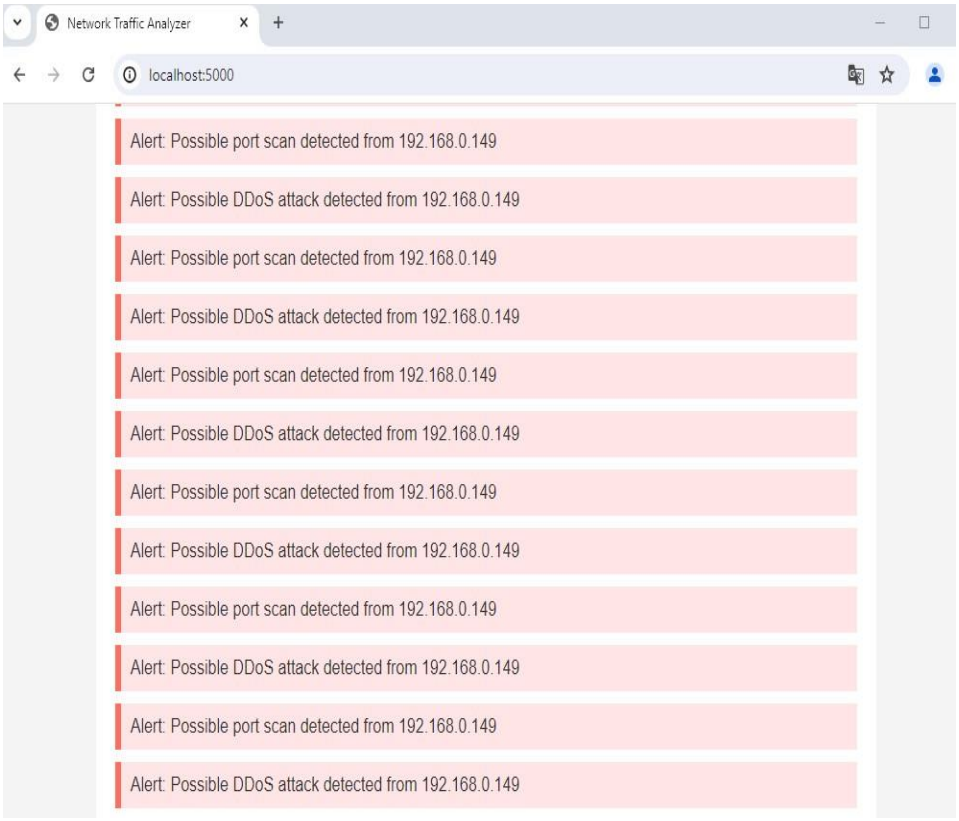
This screenshot shows the terminal output generated by the Python script. The script is actively monitoring network traffic and detecting potential security threats. The terminal logs indicate that the system has detected multiple possible port scans and DDoS attacks from specific IP addresses. The alerts highlight suspicious activity, demonstrating the system's real-time threat detection capabilities.



Web Interface: Alerts Display

These screenshots capture the web interface of the Network Traffic Analyzer. The interface effectively displays a list of alerts detected by the system. Each alert is highlighted in a distinct box, making it easy for users to identify and review potential security threats quickly. The interface updates dynamically to reflect new alerts as they are detected by the monitoring script. The extensive list of alerts showcases the system's ability to continuously monitor and report suspicious network activity over time, indicating persistent threats that may require further investigation.


Network Traffic Analyzer
localhost:5000
Network Traffic Analyzer
Alerts
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149


Network Traffic Analyzer
localhost:5000
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible DDoS attack detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible DDoS attack detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible DDoS attack detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible DDoS attack detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible DDoS attack detected from 192.168.0.149
Alert: Possible port scan detected from 192.168.0.149
Alert: Possible DDoS attack detected from 192.168.0.149

Conclusion :

The Network Traffic Analyzer project successfully demonstrates a robust system for monitoring and detecting network security threats in real-time. By leveraging Python's Scapy library for packet sniffing and Flask for the web interface, the application provides an efficient solution for identifying potential port scans and DDoS attacks. The implementation of multithreading ensures that the packet capturing and alerting mechanisms operate seamlessly, without disrupting the system's performance.