rom flask import Flask, request, jsonify, send\_file

from flask\_cors import CORS

from scapy.all import sniff, ARP, IP, DNS, Raw

from collections import defaultdict

from sklearn.ensemble import IsolationForest

from flask\_socketio import SocketIO, emit

from flask\_mail import Mail, Message

from twilio.rest import Client

import csv

import io

import os

import logging

import requests

from OpenSSL import crypto

app = Flask(\_name\_)

CORS(app, resources={r"/": {"origins": ""}})

app.config['SECRET\_KEY'] = 'secret!'

socketio = SocketIO(app, cors\_allowed\_origins="\*")

# Configure Flask-Mail

app.config['MAIL\_SERVER'] = 'smtp.gmail.com'

app.config['MAIL\_PORT'] = 587

app.config['MAIL\_USERNAME'] = 'rr6022799@gmail.com'

app.config['MAIL\_PASSWORD'] = 'ugfh ubik rmzr zugb'

app.config['MAIL\_USE\_TLS'] = True

app.config['MAIL\_USE\_SSL'] = False

mail = Mail(app)

# Configure Twilio

TWILIO\_ACCOUNT\_SID = 'your\_account\_sid'

TWILIO\_AUTH\_TOKEN = 'your\_auth\_token'

twilio\_client = Client(TWILIO\_ACCOUNT\_SID, TWILIO\_AUTH\_TOKEN)

TWILIO\_PHONE\_NUMBER = '+1234567890'

ADMIN\_PHONE\_NUMBER = '+0987654321'

packet\_stats = defaultdict(int)

attack\_packets = defaultdict(list)

captured\_packets = []

historical\_data = []

known\_gateways = ["192.168.0.1"]

isolation\_forest = IsolationForest(contamination=0.1)

is\_capturing = False

known\_dns\_records = {"www.example.com": "93.184.216.34"}

known\_certificates = {} # Add your known certificates here

# Configure Logging

logging.basicConfig(level=logging.INFO, filename='nids.log', filemode='a', format='%(asctime)s - %(message)s')

# User credentials for authentication

users = {

'admin': 'admin',

'viewer': 'viewer'

}

@app.route('/')

def index():

return "Network Intrusion Detection System Backend"

@app.route('/auth', methods=['POST'])

def auth():

data = request.json

username = data.get('username')

password = data.get('password')

if users.get(username) == password:

return jsonify({'message': 'Authenticated', 'role': username})

else:

return jsonify({'message': 'Invalid credentials'}), 401

@app.route('/start\_capture', methods=['POST'])

def start\_capture():

global is\_capturing

is\_capturing = True

duration = request.json.get('duration', 60)

sniff(prn=packet\_callback, timeout=duration)

return jsonify({'message': 'Packet capturing started.'})

@app.route('/stop\_capture', methods=['POST'])

def stop\_capture():

global is\_capturing

is\_capturing = False

return jsonify({'message': 'Packet capturing stopped.'})

@app.route('/download\_packets', methods=['GET'])

def download\_packets():

try:

si = io.StringIO()

cw = csv.writer(si)

cw.writerow(["Source IP", "Destination IP", "Protocol Summary", "Attack Type"])

for packet in captured\_packets:

cw.writerow([packet['src'], packet['dst'], packet['protocol'], packet.get('attack\_type', 'N/A')])

output = io.BytesIO()

output.write(si.getvalue().encode('utf-8'))

output.seek(0)

return send\_file(output, mimetype='text/csv', download\_name='captured\_packets.csv', as\_attachment=True)

except Exception as e:

return jsonify({'error': str(e)}), 500

@app.route('/network\_attacks', methods=['GET'])

def network\_attacks():

return jsonify({

'mitm\_packets': packet\_stats['mitm\_packets'],

'spoofing\_packets': packet\_stats['spoofing\_packets'],

'dns\_spoofing\_packets': packet\_stats['dns\_spoofing\_packets'],

'https\_spoofing\_packets': packet\_stats['https\_spoofing\_packets'],

'total\_packets': packet\_stats['total\_packets']

})

@app.route('/analyze\_attacks', methods=['GET'])

def analyze\_attacks():

attack\_type = request.args.get('type')

if attack\_type not in attack\_packets:

return jsonify([])

return jsonify(attack\_packets[attack\_type])

@app.route('/logs', methods=['GET'])

def get\_logs():

with open('nids.log', 'r') as log\_file:

log\_content = log\_file.read()

return jsonify({'logs': log\_content})

@app.route('/fetch\_threat\_intelligence', methods=['GET'])

def fetch\_threat\_intelligence():

api\_url = "https://api.xforce.ibmcloud.com/api/alerts/urgency"

headers = {"Authorization": "Bearer YOUR\_API\_KEY"}

response = requests.get(api\_url, headers=headers)

if response.status\_code == 200:

return jsonify(response.json())

else:

return jsonify({'error': 'Failed to fetch threat intelligence'}), 500

@app.route('/block\_ip', methods=['POST'])

def block\_ip():

ip = request.json.get('ip')

automate\_response(ip)

return jsonify({'message': f'Blocked IP {ip}.'})

def automate\_response(ip):

os.system(f"sudo iptables -A INPUT -s {ip} -j DROP")

def packet\_callback(packet):

global is\_capturing

if not is\_capturing:

return

if packet.haslayer(IP):

ip\_src = packet[IP].src

ip\_dst = packet[IP].dst

packet\_stats['total\_packets'] += 1

data = {

'total\_packets': packet\_stats['total\_packets'],

'src': ip\_src,

'dst': ip\_dst,

'protocol': packet.summary()

}

captured\_packets.append(data)

store\_historical\_data(data)

# ARP Spoofing Detection

if packet.haslayer(ARP):

arp\_src = packet[ARP].psrc

arp\_dst = packet[ARP].pdst

if arp\_src in known\_gateways or arp\_dst in known\_gateways:

packet\_stats['mitm\_packets'] += 1

attack\_details = {

'ip\_src': ip\_src,

'ip\_dst': ip\_dst,

'protocol': 'ARP',

'details': packet.summary(),

'attack\_type': 'MITM (ARP Spoofing)'

}

attack\_packets['mitm\_attacks'].append(attack\_details)

log\_attack('MITM (ARP Spoofing)', attack\_details)

data['attack\_type'] = 'MITM (ARP Spoofing)'

send\_email(str(attack\_details))

send\_sms(str(attack\_details))

automate\_response(ip\_src)

# Spoofing Detection

if packet.haslayer(IP) and packet[IP].src in known\_gateways and packet[IP].dst not in known\_gateways:

packet\_stats['spoofing\_packets'] += 1

attack\_details = {

'ip\_src': ip\_src,

'ip\_dst': ip\_dst,

'protocol': 'IP',

'details': packet.summary(),

'attack\_type': 'Spoofing'

}

attack\_packets['spoofing\_attacks'].append(attack\_details)

log\_attack('Spoofing', attack\_details)

data['attack\_type'] = 'Spoofing'

automate\_response(ip\_src)

# DNS Spoofing Detection

if packet.haslayer(DNS):

dns\_qry = packet[DNS].qd.qname.decode('utf-8') if packet[DNS].qd else ""

dns\_resp = packet[DNS].an.rdata if packet[DNS].an else ""

if dns\_qry in known\_dns\_records and dns\_resp != known\_dns\_records[dns\_qry]:

packet\_stats['dns\_spoofing\_packets'] += 1

attack\_details = {

'ip\_src': ip\_src,

'ip\_dst': ip\_dst,

'protocol': 'DNS',

'details': f"DNS Spoofing detected. Query: {dns\_qry}, Response: {dns\_resp}",

'attack\_type': 'DNS Spoofing'

}

attack\_packets['dns\_spoofing\_attacks'].append(attack\_details)

log\_attack('DNS Spoofing', attack\_details)

data['attack\_type'] = 'DNS Spoofing'

automate\_response(ip\_src)

# HTTPS Spoofing Detection

if packet.haslayer(IP) and packet[IP].dport == 443:

try:

cert = crypto.load\_certificate(crypto.FILETYPE\_PEM, packet[Raw].load)

cert\_fingerprint = cert.digest("sha256").decode("utf-8")

if packet[IP].dst in known\_certificates and known\_certificates[packet[IP].dst] != cert\_fingerprint:

packet\_stats['https\_spoofing\_packets'] += 1

attack\_details = {

'ip\_src': ip\_src,

'ip\_dst': ip\_dst,

'protocol': 'HTTPS',

'details': 'HTTPS Spoofing detected: Certificate mismatch',

'attack\_type': 'HTTPS Spoofing'

}

attack\_packets['https\_spoofing\_attacks'].append(attack\_details)

log\_attack('HTTPS Spoofing', attack\_details)

data['attack\_type'] = 'HTTPS Spoofing'

automate\_response(ip\_src)

except Exception as e:

pass

socketio.emit('packet\_data', data)

def log\_attack(attack\_type, details):

logging.info(f"{attack\_type} attack detected: {details}")

def store\_historical\_data(packet):

historical\_data.append(packet)

if \_name\_ == '\_main\_':

socketio.run(app, debug=False)