import scapy.all as scapy

import sqlite3

import re

from datetime import datetime

import threading

import tkinter as tk

from tkinter import messagebox, scrolledtext

def setup\_database():

conn = sqlite3.connect('ids.db', check\_same\_thread=False)

cursor = conn.cursor()

cursor.execute('''CREATE TABLE IF NOT EXISTS rules (

id INTEGER PRIMARY KEY,

rule\_name TEXT,

pattern TEXT,

action TEXT

)''')

cursor.execute('''CREATE TABLE IF NOT EXISTS logs (

id INTEGER PRIMARY KEY,

timestamp TEXT,

source\_ip TEXT,

destination\_ip TEXT,

protocol TEXT,

message TEXT

)''')

conn.commit()

return conn

def add\_rule(conn, rule\_name, pattern, action):

cursor = conn.cursor()

cursor.execute("INSERT INTO rules (rule\_name, pattern, action) VALUES (?, ?, ?)",

(rule\_name, pattern, action))

conn.commit()

messagebox.showinfo("Success", f"Rule '{rule\_name}' added successfully.")

def log\_event(conn, source\_ip, destination\_ip, protocol, message, app\_instance):

cursor = conn.cursor()

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

cursor.execute("INSERT INTO logs (timestamp, source\_ip, destination\_ip, protocol, message) VALUES (?, ?, ?, ?, ?)",

(timestamp, source\_ip, destination\_ip, protocol, message))

conn.commit()

app\_instance.update\_logs\_display()

def analyze\_packet(packet, conn, app\_instance):

if packet.haslayer(scapy.IP):

source\_ip = packet[scapy.IP].src

destination\_ip = packet[scapy.IP].dst

protocol = packet[scapy.IP].proto

if packet.haslayer(scapy.ICMP):

cursor = conn.cursor()

cursor.execute("SELECT rule\_name, pattern, action FROM rules WHERE rule\_name = 'Block Ping Requests'")

rule = cursor.fetchone()

if rule:

rule\_name, pattern, action = rule

message = f"Rule '{rule\_name}' triggered: ICMP packet detected from {source\_ip}"

log\_event(conn, source\_ip, destination\_ip, "ICMP", message, app\_instance)

if action == "block":

block\_ip(source\_ip)

elif action == "alert":

send\_alert(source\_ip, destination\_ip, "ICMP", message)

if packet.haslayer(scapy.Raw):

payload = packet[scapy.Raw].load.decode('utf-8', errors='ignore')

cursor = conn.cursor()

cursor.execute("SELECT rule\_name, pattern, action FROM rules")

rules = cursor.fetchall()

for rule in rules:

rule\_name, pattern, action = rule

if re.search(pattern, payload, re.IGNORECASE):

message = f"Rule '{rule\_name}' triggered: {pattern}"

log\_event(conn, source\_ip, destination\_ip, protocol, message, app\_instance)

if action == "block":

block\_ip(source\_ip)

elif action == "alert":

send\_alert(source\_ip, destination\_ip, protocol, message)

def block\_ip(ip):

print(f"Blocking IP: {ip}")

def send\_alert(source\_ip, destination\_ip, protocol, message):

print(f"ALERT: {message}")

def start\_sniffing(app\_instance):

conn = app\_instance.conn

print("Starting packet sniffing...")

app\_instance.sniffing = True

scapy.sniff(prn=lambda packet: analyze\_packet(packet, conn, app\_instance), store=False, stop\_filter=lambda p: not app\_instance.sniffing)

class IDPSApp:

def \_init\_(self, root):

self.root = root

self.root.title("Intrusion Detection and Prevention System")

self.conn = setup\_database()

self.sniffing = False

self.start\_button = tk.Button(root, text="Start Sniffing", command=self.start\_sniffing\_thread)

self.start\_button.pack(pady=10)

self.stop\_button = tk.Button(root, text="Stop Sniffing", command=self.stop\_sniffing, state=tk.DISABLED)

self.stop\_button.pack(pady=10)

self.rule\_name\_label = tk.Label(root, text="Rule Name:")

self.rule\_name\_label.pack()

self.rule\_name\_entry = tk.Entry(root)

self.rule\_name\_entry.pack()

self.pattern\_label = tk.Label(root, text="Pattern:")

self.pattern\_label.pack()

self.pattern\_entry = tk.Entry(root)

self.pattern\_entry.pack()

self.action\_label = tk.Label(root, text="Action (block/alert):")

self.action\_label.pack()

self.action\_entry = tk.Entry(root)

self.action\_entry.pack()

self.add\_rule\_button = tk.Button(root, text="Add Rule", command=self.add\_rule)

self.add\_rule\_button.pack(pady=10)

self.logs\_label = tk.Label(root, text="Detected Events:")

self.logs\_label.pack()

self.logs\_text = scrolledtext.ScrolledText(root, height=15, width=80)

self.logs\_text.pack()

self.clear\_logs\_button = tk.Button(root, text="Clear Logs", command=self.clear\_logs)

self.clear\_logs\_button.pack(pady=10)

self.update\_logs\_display()

def start\_sniffing\_thread(self):

self.start\_button.config(state=tk.DISABLED)

self.stop\_button.config(state=tk.NORMAL)

self.sniffing\_thread = threading.Thread(target=start\_sniffing, args=(self,))

self.sniffing\_thread.daemon = True

self.sniffing\_thread.start()

def stop\_sniffing(self):

self.sniffing = False

self.start\_button.config(state=tk.NORMAL)

self.stop\_button.config(state=tk.DISABLED)

print("Stopping packet sniffing...")

def add\_rule(self):

rule\_name = self.rule\_name\_entry.get()

pattern = self.pattern\_entry.get()

action = self.action\_entry.get()

if rule\_name and pattern and action:

add\_rule(self.conn, rule\_name, pattern, action)

self.rule\_name\_entry.delete(0, tk.END)

self.pattern\_entry.delete(0, tk.END)

self.action\_entry.delete(0, tk.END)

else:

messagebox.showwarning("Input Error", "Please fill all fields.")

def update\_logs\_display(self):

self.logs\_text.delete(1.0, tk.END)

cursor = self.conn.cursor()

cursor.execute("SELECT \* FROM logs ORDER BY timestamp DESC")

logs = cursor.fetchall()

for log in logs:

self.logs\_text.insert(tk.END, f"{log[1]} | {log[2]} -> {log[3]} | {log[4]} | {log[5]}\n")

def clear\_logs(self):

cursor = self.conn.cursor()

cursor.execute("DELETE FROM logs")

self.conn.commit()

self.update\_logs\_display()

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

root = tk.Tk()

app = IDPSApp(root)

root.mainloop()