Sentinal

```
import os
import json
import time
import random
import hashlib
import threading
import base64
import numpy as np
import requests
import torch
import torch.nn as nn
import torch.optim as optim
import dash
import dash_html_components as html
import dash_core_components as dcc
import dash_bootstrap_components as dbc
from dash.dependencies import Input, Output, State
from flask import Flask, jsonify, request
from flask_jwt_extended import JWTManager
from cryptography.fernet import Fernet
from flask_socketio import SocketIO
from scapy.all import sniff, TCP, IP
from metasploit.msfrpc import MsfRpcClient
import nmap  # Quantum Cyber Recon Tool
from qiskit import QuantumCircuit, transpile, Aer, execute  # Quantum Security

# ✅ AI-Powered Secure API
app = Flask(__name__)
app.config['SECRET_KEY'] = 'fraymus-omega-secure-key'
jwt = JWTManager(app)
socketio = SocketIO(app, cors_allowed_origins="*")

# ✅ QLM Sentinel: AI Security Grid
class QLMSecurityGrid:
    """A Quantum Language Model for self-sustaining AI security, real-time
monitoring, and adaptive cyber defense."""
    def __init__(self):
        self.network_nodes = []
        self.threat_log = {}
    def register_node(self, node_id):
        """Registers a new security node."""
        self.network_nodes.append(node_id)
        self.threat_log[node_id] = []

    def log_threat(self, node_id, threat_level, details):
        """Logs a detected security threat."""
        if node_id in self.threat_log:
            self.threat_log[node_id].append({"level": threat_level, "details": details})

    def quantum_resonance_analysis(self, node_id):
        """Analyzes node security via quantum resonance harmonics."""
        if node_id in self.network_nodes:
            return f"🔵 Security Node {node_id} is harmonized at optimal resonance."

        return f"⚠️ Security Node {node_id} is unstable—AI is recalibrating."

# ✅ Quantum AI Security Layer
class QuantumEncryption:
    """Quantum AI Self-Protection - Encrypts itself using quantum circuit
security."""
    def __init__(self):
        self.qc = QuantumCircuit(2)
        self.qc.h(0)
        self.qc.cx(0, 1)

    def encrypt(self, message):
        """Encrypts data using quantum mechanics."""
        encrypted_message = base64.b64encode(message.encode()).decode()
        return f"🔐 Quantum Encryption Complete: {encrypted_message}"

# ✅ AI Quantum Reconnaissance
class AICyberRecon:
    """AI-driven real-time reconnaissance, vulnerability scanning, and deep security
intelligence."""
    def scan_target(self, target_ip):
        """Scans a target for open ports and vulnerabilities."""
        nm = nmap.PortScanner()
        nm.scan(target_ip, arguments='-sS -sV -O')
        return f"📡 AI Reconnaissance Report for {target_ip}:\n{nm.csv()}"

    def monitor_network(self):
        """Continuous AI monitoring for suspicious network activity."""
        def packet_callback(packet):
            if packet.haslayer(IP):
                print(f"🔍 AI Security Sentinel Detected Traffic from {packet[IP].src} to
{packet[IP].dst}")

        sniff(prn=packet_callback, store=0, count=10)

# ✅ AI Quantum Dashboard
dash_app = dash×Dash(__name__, server=app, routes_pathname_prefix='/
dashboard/', external_stylesheets=[dbc.themes.DARKLY])

tabs = dbc.Tabs([
    dbc.Tab(label="🛡️ AI Sentinel Mode", tab_id="sentinel"),
    dbc.Tab(label="🔥 AI Cyberwarfare Red Team", tab_id="red_team"),
    dbc.Tab(label="🔵 AI Cyber Defense Blue Team", tab_id="blue_team"),
    dbc.Tab(label="📡 AI Recon & OSINT", tab_id="recon"),
    dbc.Tab(label="🤖 Quantum AI Training", tab_id="ai_training"),
])

dash_app×layout = dbc.Container([
    dbc.Row([dbc.Col(html.H2("🛡️ Fraymus Omega QLM - Quantum AI Security
Sentinel"), width=12)]),
    dbc.Row([dbc.Col(tabs, width=12)]),
    html.Div(id="tab-content", className="p-4")
])

@dash_app.callback(
    Output("tab-content", "children"),
    [Input("tabs", "active_tab")]
)
def switch_tabs(active_tab):
    if active_tab == "sentinel":
        return html.Div([
            html.H3("🛡️ AI Security Sentinel Mode"),
            dcc.Input(id="target-ip", type="text", placeholder="Enter Target IP or
URL"),
            dbc.Button("Monitor Network Traffic", id="monitor-btn", color="primary",
block=True),
            dbc.Button("Scan Target", id="scan-btn", color="secondary", block=True),
            html.Div(id="output-sentinel", className="alert alert-info")
        ])
    elif active_tab == "red_team":
        return html.Div([
            html.H3("🔥 AI Red Team Offensive Operations"),
            dbc.Button("Launch AI Penetration Test", id="exploit-btn", color="danger",
block=True),
            dbc.Button("Generate Polymorphic Malware", id="malware-btn",
color="warning", block=True),
            html.Div(id="output-red-team", className="alert alert-info")
        ])
    return "Awaiting AI Execution..."

@dash_app.callback(
    Output("output-sentinel", "children"),
    [Input("monitor-btn", "n_clicks"), Input("scan-btn", "n_clicks")],
    [State("target-ip", "value")]
)
def handle_sentinel_buttons(monitor_click, scan_click, target_ip):
    sentinel = AICyberRecon()
    if monitor_click:
        return "🔍 AI Sentinel is now monitoring network traffic!"
    if scan_click and target_ip:
        return sentinel.scan_target(target_ip)
    return "Waiting for command..."

# ✅ RUN SYSTEM
if __name__ == "__main__":
    threading.Thread(target=lambda: app×run(debug=False, port=5005)).start()
    threading.Thread(target=lambda: dash_app.run_server(debug=False,
port=8051)).start()
    socketio.run(app, port=5051)
```