Claude

MWRASP Quantum Defense System

Generated: 2025-08-24 18:15:09

TOP SECRET//SCI - HANDLE VIA SPECIAL ACCESS CHANNELS

CLAUDE.md

This file provides guidance to Claude Code (claude.ai/code) when working with code in this repository.

Project Overview

MWRASP-Quantum-Defense - Multi-Wavelength Rapid-Aging Surveillance Platform with Quantum Computer Attack Detection. A complete cybersecurity defense system designed to detect and respond to quantum computer attacks through advanced canary token detection, temporal data fragmentation, and autonomous agent coordination.

Architecture

The system consists of three core components:

Core Systems (`src/core/`)

- **quantum_detector.py**: Quantum attack detection engine with canary tokens
- **temporal_fragmentation.py**: Data fragmentation system with millisecond expiration
- **agent_system.py**: Autonomous defense agent coordination

API Layer (`src/api/`)

- **server.py**: FastAPI application with REST endpoints
- **websocket.py**: Real-time WebSocket communication

Dashboard (`src/dashboard/`)

- **index.html**: Web-based monitoring dashboard
- **app.js**: Real-time dashboard JavaScript
- **style.css**: Dashboard styling

Development Setup

Prerequisites

- Python 3.9 or higher
- Modern web browser for dashboard access

Installation

```
 # Create virtual environment
python -m venv venv
source venv/bin/activate  # On Windows: venv\Scripts\activate

# Install dependencies
pip install -r requirements.txt
```

Common Commands

Running the System

```
# Start complete system with web dashboard
python -m uvicorn src.api.server:app --reload --host 0.0.0.0 --port
8000

# Run interactive demo
python demo.py

# Quick demo mode
python demo.py --quick

# Verbose demo output
python demo.py --verbose
```

Testing

```
# Run all tests
pytest

# Run specific test modules
pytest src/tests/test_quantum_detector.py -v
pytest src/tests/test_fragmentation.py -v
pytest src/tests/test_integration.py -v

# Run with coverage
pytest --cov=src --cov-report=html

# Performance tests
pytest -m slow -v
```

Development Tools

```
# Code formatting
black src/ tests/

# Linting
flake8 src/ tests/

# Type checking
mypy src/
```

System Access Points

- **Main Dashboard**: http://localhost:8000/dashboard/index.html
- **API Documentation**: http://localhost:8000/docs
- **System Health Check**: http://localhost:8000/health
- **WebSocket Endpoint**: ws://localhost:8000/ws

Key Implementation Details

Quantum Detection System

- Uses canary tokens with quantum-entangled signatures
- Detects superposition, entanglement, and speedup attack patterns
- Configurable sensitivity thresholds (0.5-0.9)
- Real-time threat monitoring with sub-second response

Temporal Fragmentation

- Fragments data into 3-10 pieces with configurable overlap
- Fragment lifetimes: 50-1000ms (default 100ms)
- Quantum noise application for additional security
- Automatic cleanup of expired fragments

Agent Coordination

- 5 agent roles: Monitor, Defender, Analyzer, Coordinator, Recovery
- Autonomous coordination with escalation protocols
- Real-time message passing and resource allocation
- Performance metrics and success rate tracking

Security Considerations

- All data is automatically fragmented and expires rapidly
- Quantum-resistant noise patterns protect against reconstruction attacks
- Multi-layer defense with detection, fragmentation, and agent response
- No persistent storage of sensitive data

Testing Strategy

Unit Tests

- Individual component testing for each core system
- Mock-based testing for external dependencies
- Performance benchmarks for critical operations

Integration Tests

- Full system integration scenarios
- Real-time coordination testing
- Error handling and recovery testing
- Long-running stability tests

Performance Tests

- High-volume canary token creation
- Rapid fragmentation/reconstruction cycles
- Agent coordination under load
- Memory usage and cleanup verification

Troubleshooting

Common Issues

- **WebSocket connection failures**: Check server status at `/health`
- **Fragment reconstruction failures**: Verify timing (millisecond precision required)
- **Agent coordination not responding**: Check `/agents/status` for agent health

Debug Mode

```
# Run server with debug logging
python -m uvicorn src.api.server:app --reload --log-level debug
```

Extension Points

The system is designed for extensibility: - Custom quantum detection patterns can be added to `quantum_detector.py` - New fragmentation policies can be implemented via `FragmentationPolicy` class - Additional agent roles can be created by extending the `Agent` class - WebSocket message types can be extended in `websocket.py`

Document: CLAUDE.md | **Generated:** 2025-08-24 18:15:09