Readme

MWRASP Quantum Defense System

Generated: 2025-08-24 18:15:27

TOP SECRET//SCI - HANDLE VIA SPECIAL ACCESS CHANNELS

MWRASP Quantum Defense System

Multi-Wavelength Rapid-Aging Surveillance Platform with Quantum Computer Attack Detection

A cutting-edge cybersecurity defense system designed to detect and respond to quantum computer attacks through advanced canary token detection, temporal data fragmentation, and autonomous agent coordination.

Features

Quantum Attack Detection

- **Canary Token System**: Deploy quantum-entangled canary tokens to detect unauthorized access
- **Pattern Recognition**: Identify quantum computing attack signatures including:
- Superposition-like access patterns (multiple simultaneous accesses)
- Entanglement correlation across multiple systems
- Quantum speedup detection (unnaturally fast computation patterns)

- Interference pattern analysis
- **Real-time Threat Assessment**: Continuous monitoring with configurable sensitivity thresholds

Temporal Fragmentation

- **Millisecond Data Expiration**: Fragment sensitive data with configurable lifespans (50-1000ms)
- **Quantum-Resistant Fragmentation**: Apply quantum noise and overlap patterns
- **Automatic Cleanup**: Background services ensure expired fragments are securely removed
- **Reconstruction Capabilities**: Legitimate users can reconstruct data within the validity window

Autonomous Defense Agents

- **Multi-Agent Coordination**: Distributed system with specialized agent roles:
- **Monitor Agents**: Continuous surveillance and pattern analysis
- **Defender Agents**: Active threat response and data isolation
- **Analyzer Agents**: Deep threat analysis and pattern learning
- **Coordinator Agents**: Strategic resource allocation and response planning
- **Recovery Agents**: System repair and data integrity restoration
- **Real-time Coordination**: Sub-second response times with parallel execution
- **Adaptive Response**: Escalation protocols based on threat severity

Real-time Dashboard

- **Live Monitoring**: WebSocket-powered real-time updates
- **Threat Visualization**: Interactive charts showing attack patterns and system health
- **Agent Status Display**: Visual representation of autonomous agent network
- **System Controls**: Manual coordination triggers and simulation capabilities

Architecture

```
MWRASP-Quantum-Defense/
 src/
     core/                    # Core defense systems
         quantum_detector.py  # Quantum attack detection engine
         temporal_fragmentation.py  # Data fragmentation system
         agent_system.py      # Autonomous agent coordination
     api/                     # Web API and services
         server.py            # FastAPI application
         websocket.py         # Real-time WebSocket handler
     dashboard/               # Web interface
         index.html           # Main dashboard interface
         app.js               # Dashboard JavaScript logic
         style.css            # Dashboard styling
 tests/                       # Comprehensive test suite
 requirements.txt             # Python dependencies
 demo.py                      # System demonstration script
 README.md                    # This file
```

Quick Start

Prerequisites

- Python 3.9 or higher
- Modern web browser (Chrome, Firefox, Safari, Edge)

Installation

1. **Clone the repository**: `git clone <repository-url> cd MWRASP-Quantum-Defense`
2. **Create virtual environment**: `python -m venv venv source venv/bin/activate # On Windows: venv\Scripts\activate`
3. **Install dependencies**: `pip install -r requirements.txt`

Running the System

Option 1: Full System with Dashboard

```
 # Start the complete MWRASP system
python -m uvicorn src.api.server:app --reload --host 0.0.0.0 --port
```

```
8000
```

Then open your browser to: - **Main Dashboard**: http://localhost:8000/dashboard/index.html - **API Documentation**: http://localhost:8000/docs - **System Health**: http://localhost:8000/health

Option 2: Quick Demo

```
# Run the interactive demonstration
python demo.py
```

Usage Examples

Basic Quantum Attack Detection

```
from src.core.quantum_detector import QuantumDetector

# Initialize detector
detector = QuantumDetector(sensitivity_threshold=0.7)
detector.start_monitoring()

# Create canary token
token = detector.generate_canary_token("sensitive_database")
print(f"Canary token deployed: {token.token_id}")

# Simulate access (would be triggered by actual system access)
threat_detected = detector.access_token(token.token_id,
"suspicious_user")

if threat_detected:
    threats = detector.get_active_threats()
    for threat in threats:
        print(f"QUANTUM ATTACK DETECTED: {threat.threat_level} -
{threat.confidence_score}")
```

Temporal Data Fragmentation

```
from src.core.temporal_fragmentation import TemporalFragmentation,
FragmentationPolicy

# Configure fragmentation policy
```

```
policy = FragmentationPolicy(
    max_fragment_lifetime_ms=200,  # 200ms lifetime
    min_fragments=5,
    quantum_resistance_level=4
)

# Initialize fragmenter
fragmenter = TemporalFragmentation(policy)
fragmenter.start_cleanup_service()

# Fragment sensitive data
sensitive_data = b"TOP SECRET: Nuclear launch codes"
fragments = fragmenter.fragment_data(sensitive_data, "nuclear_codes")

print(f"Data fragmented into {len(fragments)} pieces")
print(f"Expires in: {fragments[0].expires_at - time.time():.3f}
seconds")

# Reconstruct within validity window
reconstructed = fragmenter.reconstruct_data("nuclear_codes")
assert reconstructed == sensitive_data
```

Autonomous Agent Coordination

```
 import asyncio
from src.core.agent_system import AutonomousDefenseCoordinator

async def demo_coordination():
    # Initialize coordinator with detection systems
    coordinator = AutonomousDefenseCoordinator(detector, fragmenter)
    await coordinator.start_coordination()

    # Trigger coordination manually
    await coordinator.send_coordination_message({
        "type": "threat_escalation",
        "threat_id": "demo_threat",
        "level": 8,
        "source": "manual_demo"
    })

    # Check agent status
    status = coordinator.get_agent_status()
    print(f"Active agents: {status['coordination_stats']
['active_agents']}")

# Run coordination demo
asyncio.run(demo_coordination())
```

Configuration

Quantum Detection Sensitivity

```
 # High security environment
detector = QuantumDetector(sensitivity_threshold=0.9)

# Balanced detection
detector = QuantumDetector(sensitivity_threshold=0.7)  # Default

# Permissive environment (fewer false positives)
detector = QuantumDetector(sensitivity_threshold=0.5)
```

Fragmentation Policies

```
 # Maximum security (very short-lived fragments)
max_security_policy = FragmentationPolicy(
    max_fragment_lifetime_ms=50,    # 50ms lifetime
    min_fragments=10,               # Many fragments
    quantum_resistance_level=5      # Maximum quantum resistance
)

# Balanced security
balanced_policy = FragmentationPolicy(
    max_fragment_lifetime_ms=200,   # 200ms lifetime
    min_fragments=5,                # Moderate fragmentation
    quantum_resistance_level=3      # Standard quantum resistance
)

# Performance optimized
performance_policy = FragmentationPolicy(
    max_fragment_lifetime_ms=1000, # 1 second lifetime
    min_fragments=3,                # Minimal fragmentation
    quantum_resistance_level=1      # Basic quantum resistance
)
```

Testing

Run the comprehensive test suite:

```
 # Run all tests
pytest
```

```
# Run specific test categories
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

# Run performance tests
pytest -m slow -v
```

API Reference

REST API Endpoints

Quantum Detection

- `POST /quantum/token` - Create canary token
- `POST /quantum/access/{token_id}` - Access token (triggers detection)
- `GET /quantum/threats` - Get active threats
- `GET /quantum/statistics` - Get detection statistics

Temporal Fragmentation

- `POST /temporal/fragment` - Fragment data
- `POST /temporal/reconstruct/{original_id}` - Reconstruct data
- `GET /temporal/status/{original_id}` - Get fragment status
- `DELETE /temporal/expire/{original_id}` - Force expire fragments

Agent Coordination

- `GET /agents/status` - Get agent status
- `POST /agents/coordinate` - Trigger coordination
- `POST /agents/message` - Send coordination message

System Management

- `GET /health` - System health check

- `GET /stats` - Comprehensive system statistics
- `POST /simulate/quantum_attack` - Simulate quantum attack
- `POST /simulate/temporal_breach` - Simulate temporal breach

WebSocket Events

Real-time events are broadcast via WebSocket connection at `/ws`:

```
 // Connect to WebSocket
const ws = new WebSocket('ws://localhost:8000/ws');

// Subscribe to event types
ws.send(JSON.stringify({
    type: 'subscribe',
    subscriptions: ['threats', 'agents', 'fragments', 'system']
}));

// Handle incoming events
ws.onmessage = (event) => {
    const data = JSON.parse(event.data);
    console.log(`Event: ${data.type}.${data.event}`, data.data);
};
```

Security Considerations

Deployment Security

- Run the system in isolated network environments
- Use HTTPS/WSS in production deployments
- Implement proper authentication and authorization
- Regular security audits and penetration testing

Quantum Resistance

- Fragment lifetimes should be tuned based on threat model
- Higher quantum resistance levels provide better security but impact performance
- Consider hardware security modules (HSMs) for critical deployments

Data Protection

- Sensitive data is automatically fragmented and expires rapidly
- Original data should never be stored in plaintext
- Fragment cleanup ensures no data persistence beyond configured lifetimes

Troubleshooting

Common Issues

WebSocket Connection Failures

```
# Check if server is running
curl http://localhost:8000/health

# Verify WebSocket endpoint
wscat -c ws://localhost:8000/ws
```

Fragment Reconstruction Failures

- Check if fragments have expired (millisecond-level timing)
- Verify quantum resistance settings match between fragmentation and reconstruction
- Ensure cleanup service hasn't removed fragments prematurely

Agent Coordination Not Responding

- Verify all core systems are initialized properly
- Check agent status via `/agents/status` endpoint
- Look for error messages in server logs

Debug Mode

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

# Run demo with verbose output
python demo.py --verbose
```

Advanced Usage

Custom Agent Behaviors

```
 # Extend the agent system with custom behaviors
class CustomDefenseAgent(Agent):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.custom_capabilities = ["quantum_decoherence",
"timeline_manipulation"]

# Register with coordinator
coordinator.register_custom_agent(CustomDefenseAgent)
```

Integration with External Systems

```
 # Integrate with SIEM systems
class SIEMIntegration:
    async def forward_threats(self, threats):
        for threat in threats:
            await self.send_to_siem({
                "type": "quantum_attack",
                "severity": threat.threat_level.name,
                "confidence": threat.confidence_score,
                "indicators": threat.quantum_indicators
            })

# Register threat callback
detector.add_threat_callback(siem_integration.forward_threats)
```

Contributing

We welcome contributions to the MWRASP Quantum Defense System! Please see our contributing guidelines for details on:

- Code style and standards
- Testing requirements
- Security review process
- Documentation standards

Development Setup

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

# Install pre-commit hooks
pre-commit install

# Run quality checks
black src/ tests/
flake8 src/ tests/
mypy src/
```

License

This project is licensed under the MIT License - see the LICENSE file for details.

Legal Notice

This system is designed for defensive cybersecurity purposes only. Users are responsible for ensuring compliance with applicable laws and regulations. The system should not be used for unauthorized testing or attacks on systems you do not own or have explicit permission to test.

Related Projects

- Quantum-Safe Cryptography
- NIST Cybersecurity Framework
- Post-Quantum Cryptography Standardization

Support

For questions, issues, or feature requests:

1. Check the FAQ section above
2. Search existing GitHub issues

3. Create a new issue with detailed information
4. For security vulnerabilities, please use responsible disclosure

---

MWRASP Quantum Defense System - Protecting against tomorrow's threats, today.

Document: README.md | **Generated:** 2025-08-24 18:15:27