Security Configuration Guide

This document provides comprehensive security configuration guidelines for the Enhanced Two-Tiered Multi-Agent Orchestration System.

Security Architecture Overview

The system implements a multi-layered security approach:

1. **Network Security**: TLS encryption, firewall rules, network segmentation
2. **Authentication & Authorization**: API keys, SSH keys, role-based access
3. **Command Security**: Input validation, command filtering, execution sandboxing
4. **Audit & Monitoring**: Comprehensive logging, threat detection, real-time alerts
5. **Data Protection**: Encryption at rest and in transit, secure key management

Network Security

TLS/SSL Configuration

Certificate Management

Use Let's Encrypt for automatic certificate management:

```
# Install certbot
apt install certbot python3-certbot-nginx

# Obtain certificate
certbot --nginx -d mcp.xplaincrypto.ai

# Auto-renewal
echo "0 12 * * * /usr/bin/certbot renew --quiet" | crontab -
```

Nginx SSL Configuration

```
server {
    listen 443 ssl http2;
    server_name mcp.xplaincrypto.ai;

    # SSL Configuration
    ssl_certificate /etc/letsencrypt/live/mcp.xplaincrypto.ai/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/mcp.xplaincrypto.ai/privkey.pem;

    # SSL Security Settings
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_ciphers ECDHE-RSA-AES256-GCM-SHA512:DHE-RSA-AES256-GCM-SHA512:ECDHE-RSA-AES256-
GCM-SHA384:DHE-RSA-AES256-GCM-SHA384;
    ssl_prefer_server_ciphers off;
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    # HSTS
    add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

    # Security Headers
    add_header X-Frame-Options DENY;
    add_header X-Content-Type-Options nosniff;
    add_header X-XSS-Protection "1; mode=block";
    add_header Referrer-Policy "strict-origin-when-cross-origin";

    location / {
        proxy_pass http://localhost:8001;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;

        # Rate limiting
        limit_req zone=api burst=20 nodelay;
    }
}

# Rate limiting configuration
http {
    limit_req_zone $binary_remote_addr zone=api:10m rate=10r/s;
}
```

Firewall Configuration

UFW (Uncomplicated Firewall)

```
# Reset firewall
ufw --force reset

# Default policies
ufw default deny incoming
ufw default allow outgoing

# SSH access (change port if using non-standard)
ufw allow 22/tcp

# HTTP/HTTPS
ufw allow 80/tcp
ufw allow 443/tcp

# Application ports (restrict as needed)
ufw allow from 10.0.0.0/8 to any port 8001  # Internal network only
ufw allow from 172.16.0.0/12 to any port 8001
ufw allow from 192.168.0.0/16 to any port 8001

# Monitoring (optional, restrict to monitoring networks)
ufw allow from 10.0.0.0/8 to any port 9090  # Prometheus
ufw allow from 10.0.0.0/8 to any port 3000  # Grafana

# Enable firewall
ufw enable

# Check status
ufw status verbose
```

iptables Rules (Advanced)

```
#!/bin/bash
# Advanced iptables configuration

# Flush existing rules
iptables -F
iptables -X
iptables -t nat -F
iptables -t nat -X

# Default policies
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT ACCEPT

# Allow loopback
iptables -A INPUT -i lo -j ACCEPT

# Allow established connections
iptables -A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT

# SSH with rate limiting
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --set
iptables -A INPUT -p tcp --dport 22 -m state --state NEW -m recent --update --seconds 6
0 --hitcount 4 -j DROP
iptables -A INPUT -p tcp --dport 22 -j ACCEPT

# HTTP/HTTPS
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -p tcp --dport 443 -j ACCEPT

# Application ports with source restrictions
iptables -A INPUT -p tcp --dport 8001 -s 10.0.0.0/8 -j ACCEPT
iptables -A INPUT -p tcp --dport 8001 -s 172.16.0.0/12 -j ACCEPT
iptables -A INPUT -p tcp --dport 8001 -s 192.168.0.0/16 -j ACCEPT

# Log dropped packets
iptables -A INPUT -m limit --limit 5/min -j LOG --log-prefix "iptables denied: " --log-
level 7

# Save rules
iptables-save > /etc/iptables/rules.v4
```

Authentication & Authorization

API Key Management

Strong API Key Generation

```python
import secrets
import string

def generate_api_key(length=64):
    """Generate a cryptographically secure API key"""
    alphabet = string.ascii_letters + string.digits
    return ''.join(secrets.choice(alphabet) for _ in range(length))

# Generate API key
api_key = generate_api_key()
print(f"API Key: {api_key}")
```

API Key Storage

Store API keys securely using environment variables or secret management:

```bash
# Environment variable
export API_KEY="your_secure_64_character_api_key_here"

# Docker secrets (recommended for production)
echo "your_secure_api_key" | docker secret create orchestrator_api_key -

# HashiCorp Vault (enterprise)
vault kv put secret/orchestrator api_key="your_secure_api_key"
```

API Key Rotation

```bash
#!/bin/bash
# API key rotation script

OLD_KEY="$1"
NEW_KEY="$2"

if [ -z "$OLD_KEY" ] || [ -z "$NEW_KEY" ]; then
    echo "Usage: $0 <old_key> <new_key>"
    exit 1
fi

# Update environment file
sed -i "s/API_KEY=$OLD_KEY/API_KEY=$NEW_KEY/" .env

# Restart services
docker-compose restart mcp_bridge

# Verify new key works
curl -H "X-API-Key: $NEW_KEY" http://localhost:8001/health

echo "API key rotation completed"
```

SSH Key Management

SSH Key Generation

```
# Generate Ed25519 key (recommended)
ssh-keygen -t ed25519 -C "orchestrator@mcp.xplaincrypto.ai" -f ~/.ssh/orchestrat-
or_ed25519

# Generate RSA key (if Ed25519 not supported)
ssh-keygen -t rsa -b 4096 -C "orchestrator@mcp.xplaincrypto.ai" -f ~/.ssh/orchestrat-
or_rsa

# Set proper permissions
chmod 600 ~/.ssh/orchestrator_ed25519
chmod 644 ~/.ssh/orchestrator_ed25519.pub
```

SSH Key Deployment

```
# Copy public key to target server
ssh-copy-id -i ~/.ssh/orchestrator_ed25519.pub root@mcp.xplaincrypto.ai

# Or manually add to authorized_keys
cat ~/.ssh/orchestrator_ed25519.pub | ssh root@mcp.xplaincrypto.ai "mkdir -p ~/.ssh &&
cat >> ~/.ssh/authorized_keys"
```

SSH Configuration Hardening

```
# /etc/ssh/sshd_config
cat >> /etc/ssh/sshd_config << 'EOF'

# Security hardening
Protocol 2
PermitRootLogin prohibit-password
PasswordAuthentication no
PubkeyAuthentication yes
AuthorizedKeysFile .ssh/authorized_keys
PermitEmptyPasswords no
ChallengeResponseAuthentication no
UsePAM yes
X11Forwarding no
PrintMotd no
ClientAliveInterval 300
ClientAliveCountMax 2
MaxAuthTries 3
MaxSessions 2

# Restrict users (optional)
AllowUsers orchestrator root

# Restrict ciphers and algorithms
Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-
gcm@openssh.com,aes256-ctr,aes192-ctr,aes128-ctr
MACs hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,hmac-sha2-256,hmac-
sha2-512
KexAlgorithms curve25519-sha256@libssh.org,diffie-hellman-group16-sha512,diffie-
hellman-group18-sha512

EOF

# Restart SSH service
systemctl restart sshd
```

Command Security

Security Levels

The system implements four security levels for command execution:

Low Security

- Basic command validation
- Minimal restrictions
- Suitable for trusted environments

Medium Security (Default)

- Standard security checks
- Path restrictions
- Command injection prevention
- Recommended for most use cases

High Security

- Strict command validation

- Whitelist-based command filtering
- Enhanced path restrictions
- Suitable for production environments

Critical Security

- Maximum security restrictions
- Minimal allowed commands
- Comprehensive audit logging
- Suitable for high-security environments

Command Filtering Configuration

```
# Custom security configuration
SECURITY_CONFIG = {
    "blocked_commands": [
        "rm -rf /",
        "dd if=/dev/zero",
        "mkfs",
        "fdisk",
        "format",
        "shutdown",
        "reboot",
        "halt",
        "init 0",
        "init 6",
        "killall",
        "pkill -9"
    ],

    "restricted_paths": [
        "/etc/passwd",
        "/etc/shadow",
        "/etc/sudoers",
        "/boot",
        "/sys",
        "/proc/sys"
    ],

    "allowed_commands": {
        "high": [
            "ls", "cat", "grep", "find", "ps", "top", "df", "du",
            "git", "docker", "npm", "pip", "python", "node",
            "mkdir", "touch", "cp", "mv", "chmod", "chown",
            "systemctl", "service", "curl", "wget"
        ],

        "critical": [
            "ls", "cat", "grep", "ps", "docker ps", "git status",
            "systemctl status", "curl", "wget"
        ]
    }
}
```

Input Validation

```python
import re
from typing import Tuple

def validate_command_input(command: str, security_level: str) -> Tuple[bool, str]:
    """Comprehensive command input validation"""

    # Check for null bytes
    if '\x00' in command:
        return False, "Null bytes not allowed"

    # Check command length
    if len(command) > 1000:
        return False, "Command too long"

    # Check for dangerous patterns
    dangerous_patterns = [
        r';\s*rm\s+-rf',
        r'&&\s*rm\s+-rf',
        r'\|\s*rm\s+-rf',
        r'`.*`',
        r'\$\(.*\)',
        r'>\s*/dev/',
        r'<\s*/dev/',
        r'/etc/passwd',
        r'/etc/shadow'
    ]

    for pattern in dangerous_patterns:
        if re.search(pattern, command, re.IGNORECASE):
            return False, f"Dangerous pattern detected: {pattern}"

    # Security level specific validation
    if security_level == "critical":
        # Only allow very specific commands
        allowed_prefixes = ["ls", "cat", "grep", "ps", "docker ps", "git status"]
        if not any(command.startswith(prefix) for prefix in allowed_prefixes):
            return False, "Command not allowed in critical security mode"

    return True, "Command validated"
```

Audit & Monitoring

Comprehensive Logging

Log Configuration

```python
import logging
import json
from datetime import datetime

class SecurityAuditFormatter(logging.Formatter):
    """Custom formatter for security audit logs"""

    def format(self, record):
        log_entry = {
            "timestamp": datetime.utcnow().isoformat(),
            "level": record.levelname,
            "logger": record.name,
            "message": record.getMessage(),
            "module": record.module,
            "function": record.funcName,
            "line": record.lineno
        }

        # Add extra fields if present
        if hasattr(record, 'user_id'):
            log_entry['user_id'] = record.user_id
        if hasattr(record, 'source_ip'):
            log_entry['source_ip'] = record.source_ip
        if hasattr(record, 'command'):
            log_entry['command'] = record.command

        return json.dumps(log_entry)

# Configure security logger
security_logger = logging.getLogger('security')
security_handler = logging.FileHandler('/var/log/orchestrator/security.log')
security_handler.setFormatter(SecurityAuditFormatter())
security_logger.addHandler(security_handler)
security_logger.setLevel(logging.INFO)
```

Log Rotation

```
# /etc/logrotate.d/orchestrator
/var/log/orchestrator/*.log {
    daily
    missingok
    rotate 90
    compress
    delaycompress
    notifempty
    create 644 orchestrator orchestrator
    postrotate
        systemctl reload rsyslog
    endscript
}
```

Real-time Monitoring

Fail2Ban Configuration

```
# /etc/fail2ban/jail.local
[DEFAULT]
bantime = 3600
findtime = 600
maxretry = 5

[orchestrator-auth]
enabled = true
port = 8001
protocol = tcp
filter = orchestrator-auth
logpath = /var/log/orchestrator/security.log
maxretry = 3
bantime = 7200

[orchestrator-command]
enabled = true
port = 8001
protocol = tcp
filter = orchestrator-command
logpath = /var/log/orchestrator/security.log
maxretry = 10
bantime = 3600
```

```
# /etc/fail2ban/filter.d/orchestrator-auth.conf
[Definition]
failregex = .*"event_type": "AUTHENTICATION".*"result": "failed".*"source_ip": "<HOST>"
ignoreregex =
```

```
# /etc/fail2ban/filter.d/orchestrator-command.conf
[Definition]
failregex = .*"event_type": "SECURITY_VIOLATION".*"source_ip": "<HOST>"
ignoreregex =
```

Intrusion Detection

```
#!/bin/bash
# Simple intrusion detection script

LOG_FILE="/var/log/orchestrator/security.log"
ALERT_EMAIL="admin@xplaincrypto.ai"

# Monitor for suspicious activities
tail -f "$LOG_FILE" | while read line; do
    # Check for multiple failed authentications
    if echo "$line" | grep -q "AUTHENTICATION.*failed"; then
        ip=$(echo "$line" | grep -o '"source_ip": "[^"]*"' | cut -d'"' -f4)
        count=$(grep -c "AUTHENTICATION.*failed.*$ip" "$LOG_FILE")

        if [ "$count" -gt 5 ]; then
            echo "ALERT: Multiple failed authentications from $ip" | mail -s "Security
Alert" "$ALERT_EMAIL"
        fi
    fi

    # Check for command injection attempts
    if echo "$line" | grep -q "SECURITY_VIOLATION.*command_injection"; then
        echo "ALERT: Command injection attempt detected" | mail -s "Critical Security
Alert" "$ALERT_EMAIL"
    fi
done
```

Security Metrics

Prometheus Metrics

```
from prometheus_client import Counter, Histogram, Gauge

# Security metrics
security_events_total = Counter(
    'orchestrator_security_events_total',
    'Total number of security events',
    ['event_type', 'security_level']
)

command_execution_duration = Histogram(
    'orchestrator_command_execution_duration_seconds',
    'Time spent executing commands',
    ['security_level', 'success']
)

active_threats = Gauge(
    'orchestrator_active_threats',
    'Number of active security threats',
    ['threat_type', 'severity']
)

# Usage in code
security_events_total.labels(
    event_type='AUTHENTICATION',
    security_level='high'
).inc()
```

Data Protection

Encryption at Rest

Database Encryption

```
# docker-compose.yml - PostgreSQL with encryption
postgres:
  image: postgres:15-alpine
  environment:
    POSTGRES_INITDB_ARGS: "--auth-host=scram-sha-256 --auth-local=scram-sha-256"
  volumes:
    - postgres_data:/var/lib/postgresql/data
  command: >
    postgres
    -c ssl=on
    -c ssl_cert_file=/etc/ssl/certs/server.crt
    -c ssl_key_file=/etc/ssl/private/server.key
    -c shared_preload_libraries=pg_stat_statements
    -c log_statement=all
    -c log_min_duration_statement=0
```

File System Encryption

```
# Encrypt sensitive directories using LUKS
cryptsetup luksFormat /dev/sdb1
cryptsetup luksOpen /dev/sdb1 encrypted_storage

# Create filesystem
mkfs.ext4 /dev/mapper/encrypted_storage

# Mount
mkdir -p /encrypted
mount /dev/mapper/encrypted_storage /encrypted

# Add to fstab
echo "/dev/mapper/encrypted_storage /encrypted ext4 defaults 0 2" >> /etc/fstab
```

Secret Management

HashiCorp Vault Integration

```python
import hvac

class VaultSecretManager:
    def __init__(self, vault_url, vault_token):
        self.client = hvac.Client(url=vault_url, token=vault_token)

    def get_secret(self, path):
        """Retrieve secret from Vault"""
        try:
            response = self.client.secrets.kv.v2.read_secret_version(path=path)
            return response['data']['data']
        except Exception as e:
            logger.error(f"Failed to retrieve secret from Vault: {e}")
            return None

    def store_secret(self, path, secret_data):
        """Store secret in Vault"""
        try:
            self.client.secrets.kv.v2.create_or_update_secret(
                path=path,
                secret=secret_data
            )
            return True
        except Exception as e:
            logger.error(f"Failed to store secret in Vault: {e}")
            return False

# Usage
vault = VaultSecretManager("https://vault.xplaincrypto.ai", vault_token)
api_key = vault.get_secret("orchestrator/api_key")
```

Docker Secrets

```yaml
# docker-compose.yml with secrets
version: '3.8'

secrets:
  api_key:
    external: true
  db_password:
    external: true
  ssh_private_key:
    external: true

services:
  mcp_bridge:
    secrets:
      - api_key
      - db_password
      - ssh_private_key
    environment:
      - API_KEY_FILE=/run/secrets/api_key
      - DB_PASSWORD_FILE=/run/secrets/db_password
      - SSH_KEY_FILE=/run/secrets/ssh_private_key
```

```
# Create Docker secrets
echo "your_api_key" | docker secret create api_key -
echo "your_db_password" | docker secret create db_password -
docker secret create ssh_private_key /path/to/private/key
```

Incident Response

Security Incident Playbook

1. Detection and Analysis

```
#!/bin/bash
# Incident detection script

INCIDENT_LOG="/var/log/orchestrator/incidents.log"

detect_incident() {
    local incident_type="$1"
    local details="$2"

    timestamp=$(date -u +"%Y-%m-%dT%H:%M:%S.%3NZ")
    echo "[$timestamp] INCIDENT: $incident_type - $details" >> "$INCIDENT_LOG"

    # Send alert
    curl -X POST "https://hooks.slack.com/services/YOUR/SLACK/WEBHOOK" \
        -H "Content-Type: application/json" \
        -d "{\"text\": \"🚨 Security Incident: $incident_type - $details\"}"
}

# Example usage
detect_incident "UNAUTHORIZED_ACCESS" "Multiple failed login attempts from
192.168.1.100"
```

2. Containment

```
#!/bin/bash
# Incident containment script

contain_threat() {
    local threat_ip="$1"
    local threat_type="$2"

    # Block IP immediately
    ufw insert 1 deny from "$threat_ip"

    # Add to fail2ban
    fail2ban-client set orchestrator-auth banip "$threat_ip"

    # Log containment action
    logger "CONTAINMENT: Blocked $threat_ip for $threat_type"

    # Notify administrators
    echo "Threat contained: $threat_ip ($threat_type)" | \
        mail -s "Threat Containment" admin@xplaincrypto.ai
}
```

3. Eradication and Recovery

```
#!/bin/bash
# System recovery script

recover_system() {
    # Stop all services
    docker-compose down

    # Backup current state
    tar -czf "/backup/incident_backup_$(date +%Y%m%d_%H%M%S).tar.gz" \
        /opt/enhanced_orchestrator_v2

    # Reset to known good state
    git checkout main
    git pull origin main

    # Rebuild and restart
    docker-compose build --no-cache
    docker-compose up -d

    # Verify system health
    sleep 30
    curl -f http://localhost:8001/health || exit 1

    logger "RECOVERY: System restored to clean state"
}
```

Backup and Recovery

Automated Backup Script

```
#!/bin/bash
# Comprehensive backup script

BACKUP_DIR="/backup/orchestrator"
DATE=$(date +%Y%m%d_%H%M%S)
RETENTION_DAYS=30

# Create backup directory
mkdir -p "$BACKUP_DIR"

# Database backup
docker-compose exec -T postgres pg_dump -U postgres orchestrator_db | \
    gzip > "$BACKUP_DIR/database_$DATE.sql.gz"

# Configuration backup
tar -czf "$BACKUP_DIR/config_$DATE.tar.gz" \
    /opt/enhanced_orchestrator_v2/.env \
    /opt/enhanced_orchestrator_v2/docker-compose.yml \
    /opt/enhanced_orchestrator_v2/keys/

# Logs backup
tar -czf "$BACKUP_DIR/logs_$DATE.tar.gz" \
    /opt/enhanced_orchestrator_v2/logs/

# Vector stores backup
tar -czf "$BACKUP_DIR/vector_stores_$DATE.tar.gz" \
    /opt/enhanced_orchestrator_v2/vector_stores/

# Clean old backups
find "$BACKUP_DIR" -name "*.gz" -mtime +$RETENTION_DAYS -delete

# Upload to remote storage (optional)
# aws s3 sync "$BACKUP_DIR" s3://your-backup-bucket/orchestrator/

logger "BACKUP: Completed backup for $DATE"
```

Compliance and Auditing

Compliance Frameworks

SOC 2 Type II Compliance

- **Security**: Multi-factor authentication, encryption, access controls
- **Availability**: High availability, disaster recovery, monitoring
- **Processing Integrity**: Input validation, error handling, data integrity
- **Confidentiality**: Data encryption, access controls, secure transmission
- **Privacy**: Data minimization, consent management, data retention

GDPR Compliance

- **Data Protection**: Encryption, access controls, data minimization
- **Right to be Forgotten**: Data deletion capabilities
- **Data Portability**: Export functionality
- **Breach Notification**: Automated incident reporting

Audit Trail

Comprehensive Audit Logging

```python
class ComplianceAuditor:
    def __init__(self):
        self.audit_logger = logging.getLogger('compliance')

    def log_data_access(self, user_id, resource, action, result):
        """Log data access for compliance"""
        audit_entry = {
            "event_type": "DATA_ACCESS",
            "timestamp": datetime.utcnow().isoformat(),
            "user_id": user_id,
            "resource": resource,
            "action": action,
            "result": result,
            "compliance_relevant": True
        }
        self.audit_logger.info(json.dumps(audit_entry))

    def log_configuration_change(self, user_id, component, old_value, new_value):
        """Log configuration changes"""
        audit_entry = {
            "event_type": "CONFIGURATION_CHANGE",
            "timestamp": datetime.utcnow().isoformat(),
            "user_id": user_id,
            "component": component,
            "old_value": old_value,
            "new_value": new_value,
            "compliance_relevant": True
        }
        self.audit_logger.info(json.dumps(audit_entry))
```

Security Testing

Penetration Testing

Automated Security Testing

```bash
#!/bin/bash
# Automated security testing script

# Network scanning
nmap -sS -O -A mcp.xplaincrypto.ai

# SSL/TLS testing
testssl.sh mcp.xplaincrypto.ai

# Web application testing
nikto -h https://mcp.xplaincrypto.ai

# API security testing
python3 -m pytest security_tests/

# Docker security scanning
docker run --rm -v /var/run/docker.sock:/var/run/docker.sock \
    aquasec/trivy image enhanced_orchestrator_v2:latest
```

Security Test Cases

```python
import pytest
import requests

class TestAPISecurity:
    def test_api_requires_authentication(self):
        """Test that API endpoints require authentication"""
        response = requests.get("https://mcp.xplaincrypto.ai/workflows")
        assert response.status_code == 401

    def test_invalid_api_key_rejected(self):
        """Test that invalid API keys are rejected"""
        headers = {"X-API-Key": "invalid_key"}
        response = requests.get("https://mcp.xplaincrypto.ai/health", headers=headers)
        assert response.status_code == 401

    def test_sql_injection_protection(self):
        """Test protection against SQL injection"""
        headers = {"X-API-Key": "valid_key"}
        malicious_payload = {"command": "ls; DROP TABLE users; --"}
        response = requests.post(
            "https://mcp.xplaincrypto.ai/execute",
            json=malicious_payload,
            headers=headers
        )
        assert "SQL injection" not in response.text.lower()

    def test_command_injection_protection(self):
        """Test protection against command injection"""
        headers = {"X-API-Key": "valid_key"}
        malicious_payload = {"command": "ls; rm -rf /"}
        response = requests.post(
            "https://mcp.xplaincrypto.ai/execute",
            json=malicious_payload,
            headers=headers
        )
        assert response.status_code == 400
```

Security Maintenance

Regular Security Tasks

Daily Tasks

- Review security logs
- Check for failed authentication attempts
- Monitor system resource usage
- Verify backup completion

Weekly Tasks

- Update security signatures
- Review access logs
- Test backup restoration
- Security patch assessment

Monthly Tasks

- Rotate API keys
- Review user access permissions
- Security configuration audit
- Penetration testing

Quarterly Tasks

- Comprehensive security assessment
- Update security policies
- Security training for team
- Disaster recovery testing

Security Monitoring Dashboard

```
# Grafana dashboard configuration for security metrics
SECURITY_DASHBOARD = {
    "dashboard": {
        "title": "Security Monitoring",
        "panels": [
            {
                "title": "Authentication Failures",
                "type": "graph",
                "targets": [
                    {
                        "expr": "rate(orchestrator_security_events_total{event_type='AU
THENTICATION',result='failed'}[5m])"
                    }
                ]
            },
            {
                "title": "Active Threats",
                "type": "singlestat",
                "targets": [
                    {
                        "expr": "orchestrator_active_threats"
                    }
                ]
            },
            {
                "title": "Command Execution by Security Level",
                "type": "pie",
                "targets": [
                    {
                        "expr": "orchestrator_command_execution_duration_seconds_count"
                    }
                ]
            }
        ]
    }
}
```

This comprehensive security configuration guide provides the foundation for a secure deployment of the Enhanced Two-Tiered Multi-Agent Orchestration System. Regular review and updates of these security measures are essential to maintain a strong security posture.