Wellness Agent AI: Security Controls Implementation Guide

HIPAA Security Rule Controls for AI Systems

Status: **IMPLEMENTATION READY**
Version: 1.0.0
Date: 2025-01-XX
Pattern: SECURITY × CONTROLS × HIPAA × AI × ONE

EXECUTIVE SUMMARY

This guide provides detailed implementation instructions for HIPAA Security Rule controls specifically tailored to Wellness Agent AI. It covers administrative, physical, and technical safeguards required for ePHI protection.

PART 1: ADMINISTRATIVE SAFEGUARDS

1.1 Security Management Process

1.1.1 Security Risk Analysis

Requirement: Perform and document a formal Security Risk Analysis that explicitly includes AI components.

Scope Must Include:

- LLM inference systems
- Vector databases
- Prompt engineering services
- Embedding generation
- Model training pipelines
- All data flows involving PHI

Implementation Steps:

1. **Document Current State**

```
- List all systems that store/process/transmit ePHI
- Map data flows (see Data Flow Diagram)
- Identify all vendors and BAAs
- Document current security controls
```

2. **Identify Threats & Vulnerabilities**

```
- Unauthorized access to PHI
- Data breaches (external/internal)
- Prompt injection attacks
- Model training data leaks
- Vector database breaches
- Mis-routing of PHI
- Vendor security incidents
```

3. **Assess Risk**

```
- Likelihood: High/Medium/Low
- Impact: High/Medium/Low
- Risk Score: Likelihood × Impact
```

4. **Document Findings**

```
- Risk Analysis Report
- Risk Register
- Mitigation Plans
```

Deliverable: SECURITY_RISK_ANALYSIS_REPORT.md

Frequency: Annually, or when:

- New systems added
- Significant changes to architecture
- Security incidents occur
- New threats identified

1.1.2 Risk Management

Requirement: Implement security measures to reduce risks to reasonable and appropriate levels.

Implementation:

1. **Prioritize Risks**
    - Address high-risk items first
    - Focus on critical path vendors
    - Address AI-specific risks
2. **Implement Controls**
    - Follow this guide for technical controls
    - Document all controls
    - Test controls regularly
3. **Monitor & Review**
    - Quarterly risk reviews
    - Annual comprehensive review
    - Update as needed

Deliverable: RISK_MANAGEMENT_PLAN.md

1.1.3 Sanction Policy

Requirement: Apply appropriate sanctions against workforce members who fail to comply with security policies.

Implementation:

1. **Document Policy**

```
- Define violations
- Escalation process
- Sanction levels (warning, suspension, termination)
- Legal/regulatory reporting requirements
```

2. **Communicate Policy**

    - Include in employee handbook
    - Annual training
    - Acknowledge receipt

Deliverable: SANCTION_POLICY.md

1.1.4 Information System Activity Review

Requirement: Regularly review information system activity (audit logs, access reports, etc.).

Implementation:

1. **Automated Monitoring**

    - Real-time alerting on suspicious activity
    - Daily log reviews
    - Weekly access reports

2. **Manual Reviews**

    - Monthly comprehensive log review
    - Quarterly access audit
    - Annual comprehensive audit

3. **AI-Specific Monitoring**

    - Monitor LLM API calls
    - Track prompt patterns
    - Alert on unusual data access
    - Monitor vector database queries

Deliverable: AUDIT_LOG_REVIEW_PROCEDURES.md

1.2 Assigned Security Responsibility

Requirement: Identify security official responsible for HIPAA compliance.

Implementation:

1. **Designate Security Officer**

    - Name: [Name]
    - Title: [Title]
    - Contact: [Email/Phone]
    - Responsibilities documented

2. **Document Responsibilities**

```
- Oversee security program
- Manage risk analysis
- Coordinate incident response
- Vendor management
- Workforce training
```

Deliverable: SECURITY_OFFICER_ASSIGNMENT.md

1.3 Workforce Security

1.3.1 Authorization and/or Supervision

Requirement: Implement procedures for authorizing workforce access to ePHI.

Implementation:

1. **Role-Based Access Control (RBAC)**

```
# Example roles
- Admin: Full access
- Developer: Read-only PHI, no production access
- Support: Break-glass access with approval
- Clinician: Patient-specific PHI only
- Analyst: De-identified data only
```

2. **Access Approval Process**

    - Manager approval required
    - Documented in access management system
    - Regular access reviews

Deliverable: ACCESS_CONTROL_POLICY.md

1.3.2 Workforce Clearance Procedure

Requirement: Implement procedures to ensure workforce members have appropriate access.

Implementation:

1. **Background Checks**

    - For roles with PHI access
    - Criminal background check
    - Reference checks

2. **Access Requests**
    - Formal request process
    - Manager approval
    - Security review

Deliverable: WORKFORCE_CLEARANCE_PROCEDURES.md

1.3.3 Termination Procedures

Requirement: Implement procedures for terminating access when employment ends.

Implementation:

1. **Immediate Termination Checklist**

```
- [ ] Disable all accounts (within 24 hours)
- [ ] Revoke API keys
- [ ] Remove from access groups
- [ ] Collect company devices
- [ ] Return access cards/badges
- [ ] Document termination date
```

2. **Automated Processes**
    - HR system integration
    - Automatic account disable
    - Access revocation scripts

Deliverable: TERMINATION_PROCEDURES.md

1.4 Information Access Management

1.4.1 Access Authorization

Requirement: Implement policies for granting access to ePHI.

Implementation:

1. **Access Request Form**
    - Business justification required
    - Manager approval
    - Security review
    - Time-limited access (where appropriate)

2. **Access Levels**

```
- No Access: No PHI access
- Read-Only: View PHI, no modifications
- Read-Write: View and modify PHI
- Admin: Full access, system administration
```

Deliverable: ACCESS_AUTHORIZATION_POLICY.md

1.4.2 Access Establishment and Modification

Requirement: Implement procedures for establishing, documenting, reviewing, and modifying access.

Implementation:

1. **Access Management System**
    - Centralized IAM (e.g., AWS IAM, Okta)
    - Automated provisioning
    - Regular access reviews
2. **Access Review Process**
    - Quarterly access reviews
    - Manager confirms continued need
    - Remove unused access

Deliverable: ACCESS_MANAGEMENT_PROCEDURES.md

1.5 Security Awareness and Training

Requirement: Implement security awareness and training program.

Implementation:

1. **Initial Training**
    - HIPAA basics
    - Security policies
    - Incident reporting
    - AI-specific considerations
2. **Ongoing Training**
    - Annual refresher training
    - Updates on new threats
    - Phishing awareness
    - AI security best practices
3. **Role-Specific Training**
    - Developers: Secure coding, PHI handling
    - Support: Break-glass procedures
    - Analysts: De-identification procedures

Deliverable: SECURITY_TRAINING_PROGRAM.md

1.6 Security Incident Procedures

Requirement: Implement policies and procedures to address security incidents.

Implementation:

See INCIDENT_RESPONSE_PLAN.md for detailed procedures.

Key Elements:

- Incident detection
- Response procedures
- Containment
- Notification (BAA deadlines)
- Documentation

1.7 Contingency Plan

Requirement: Establish procedures for responding to emergencies or system failures.

Implementation:

1. **Data Backup**
    - Daily automated backups
    - Encrypted backups
    - Off-site storage
    - Test restore procedures
2. **Disaster Recovery**
    - Recovery Time Objective (RTO): [X] hours
    - Recovery Point Objective (RPO): [X] hours
    - DR plan documented
    - Regular DR testing

Deliverable: CONTINGENCY_PLAN.md

1.8 Evaluation

Requirement: Perform periodic technical and non-technical evaluations.

Implementation:

1. **Internal Audits**
    - Quarterly security reviews
    - Annual comprehensive audit
    - Penetration testing (annual)
2. **External Audits**
    - Third-party security assessment (annual)
    - HIPAA compliance audit (as needed)

Deliverable: AUDIT_SCHEDULE.md

1.9 Business Associate Contracts

Requirement: Ensure BAAs are in place with all vendors that access PHI.

Implementation:

See VENDOR_INVENTORY_AND_BAA_STATUS.md for vendor management.

Key Elements:

- BAA execution tracking
- Vendor oversight
- Incident notification procedures

PART 2: PHYSICAL SAFEGUARDS

2.1 Facility Access Controls

Requirement: Limit physical access to facilities where ePHI is stored.

Implementation:

2.1.1 Cloud-Hosted (Recommended)

If using AWS/Azure/GCP:

- Physical security handled by cloud provider
- Documented in cloud provider's SOC 2/HIPAA reports
- Document reliance in risk analysis

Your Responsibilities:

- Secure access to cloud accounts (MFA, IAM)
- Secure developer laptops/devices
- Secure office access (if applicable)

2.1.2 On-Premises (If Applicable)

If hosting on-premises:

- Data center access controls
- Badge access systems
- Visitor logs
- Environmental controls (temperature, humidity)
- Fire suppression
- Backup power

Deliverable: FACILITY_ACCESS_CONTROLS.md

2.2 Workstation Use

Requirement: Implement policies for workstation use.

Implementation:

1. **Workstation Security Policy**

```
- Lock screens when unattended
- No PHI on personal devices (unless approved)
- Encrypted hard drives
- Antivirus/anti-malware
- Regular security updates
```

2. **Mobile Device Management**

    - MDM solution (if company devices)
    - Encryption required
    - Remote wipe capability

Deliverable: WORKSTATION_SECURITY_POLICY.md

2.3 Workstation Security

Requirement: Implement physical safeguards for workstations.

Implementation:

1. **Physical Security**

    - Locked offices/server rooms
    - Cable locks for laptops
    - Secure disposal of old equipment

2. **Device Encryption**

    - Full disk encryption (BitLocker, FileVault)
    - Required on all devices with PHI access

Deliverable: WORKSTATION_SECURITY_PROCEDURES.md

2.4 Device and Media Controls

Requirement: Implement procedures for disposal, re-use, and media controls.

Implementation:

1. **Media Disposal**

```
- Secure deletion (DoD 5220.22-M standard)
- Physical destruction for hard drives
- Certificate of destruction
- Document disposal
```

2. **Media Re-use**

    - Sanitize before re-use
    - Document re-use

3. **Media Movement**

- Encrypt media in transit
- Chain of custody documentation

Deliverable: MEDIA_CONTROLS_PROCEDURES.md

PART 3: TECHNICAL SAFEGUARDS

3.1 Access Control

3.1.1 Unique User Identification

Requirement: Assign unique user IDs to each user.

Implementation:

1. **IAM System**

```
# Example: AWS IAM
- Unique IAM users or SSO integration
- No shared accounts
- Service accounts for applications (non-human)
```

2. **User Management**

   - Centralized user directory (e.g., Okta, AWS SSO)
   - Automated provisioning/deprovisioning
   - Regular access reviews

Code Example:

```
# Example: User authentication
def authenticate_user(username: str, password: str) -> User:
    # Unique user ID required
    user = user_repository.get_by_username(username)
    if not user:
        raise AuthenticationError("User not found")

    # MFA required for PHI access
    if user.has_phi_access:
        verify_mfa(user, mfa_token)

    return user
```

Deliverable: ACCESS_CONTROL_IMPLEMENTATION.md

3.1.2 Emergency Access Procedure

Requirement: Establish procedures for obtaining ePHI during an emergency.

Implementation:

1. **Break-Glass Access**

```
- Emergency access account (monitored)
- Manager approval required
- Time-limited access
- Audit logging
- Post-access review
```

2. **Emergency Procedures**
    - Document emergency scenarios
    - Approval workflow
    - Notification procedures

Deliverable: EMERGENCY_ACCESS_PROCEDURES.md

3.1.3 Automatic Logoff

Requirement: Implement automatic logoff for sessions.

Implementation:

1. **Session Timeouts**

```python
# Example: Session timeout
SESSION_TIMEOUT = 15 * 60  # 15 minutes

def check_session_timeout(session):
    if time.time() - session.last_activity > SESSION_TIMEOUT:
        logout_user(session.user_id)
        log_event("session_timeout", user_id=session.user_id)
```

2. **Configuration**
    - Web sessions: 15-minute timeout
    - API tokens: 1-hour expiration
    - Admin sessions: 30-minute timeout

Deliverable: SESSION_MANAGEMENT_POLICY.md

3.1.4 Encryption and Decryption

Requirement: Implement encryption for ePHI.

Implementation:

See Section 3.3 for detailed encryption requirements.

3.2 Audit Controls

Requirement: Implement hardware, software, and procedural mechanisms to record and examine activity.

Implementation:

1. **Audit Logging Requirements**

```
# Example: Audit log entry
{
  "timestamp": "2025-01-XXT12:00:00Z",
  "user_id": "user123",
  "action": "access_phi",
  "resource": "patient_record",
  "resource_id": "patient456",
  "ip_address": "192.168.1.1",
  "user_agent": "Mozilla/5.0...",
  "success": True
}
```

2. **What to Log**
    - All PHI access (read, write, delete)
    - Authentication events (login, logout, MFA)
    - Authorization changes
    - LLM API calls (high-level, not full prompts)
    - Vector database queries
    - Configuration changes
    - Administrative actions
3. **Log Protection**
    - Encrypt audit logs
    - Immutable logs (append-only)
    - Secure storage
    - Regular backups
    - Retention: 6 years minimum
4. **Log Review**
    - Automated monitoring (real-time alerts)
    - Daily log reviews
    - Weekly access reports
    - Monthly comprehensive review

Deliverable: AUDIT_LOGGING_IMPLEMENTATION.md

3.3 Integrity

Requirement: Implement policies to ensure ePHI is not improperly altered or destroyed.

Implementation:

1. **Data Integrity Controls**

```
# Example: Checksum validation
import hashlib

def store_phi(data: bytes, patient_id: str):
    # Calculate checksum
    checksum = hashlib.sha256(data).hexdigest()

    # Store with checksum
    db.store(patient_id, data, checksum)

    # Log storage
    audit_log("store_phi", patient_id=patient_id, checksum=checksum)

def retrieve_phi(patient_id: str):
    data, stored_checksum = db.retrieve(patient_id)

    # Verify integrity
    current_checksum = hashlib.sha256(data).hexdigest()
    if current_checksum != stored_checksum:
        raise IntegrityError("Data integrity check failed")

    return data
```

2. **AI Output Validation**

```
# Example: AI output validation
def validate_ai_output(output: str, context: dict):
    # Check for dangerous content
    if contains_phi_leak(output, context):
        raise ValidationError("Potential PHI leak detected")

    # Check for invalid instructions
    if contains_dangerous_instructions(output):
        raise ValidationError("Dangerous instructions detected")

    # Validate format
    if not is_valid_format(output):
        raise ValidationError("Invalid output format")

    return output
```

3. **Version Control**
    - Track all changes to PHI
    - Maintain audit trail
    - Prevent unauthorized modifications

Deliverable: DATA_INTEGRITY_CONTROLS.md

3.4 Person or Entity Authentication

Requirement: Implement procedures to verify persons/entities accessing ePHI.

Implementation:

1. **Multi-Factor Authentication (MFA)**

```python
# Example: MFA implementation
def authenticate_with_mfa(username: str, password: str, mfa_token: str):
    # Step 1: Verify password
    user = verify_password(username, password)

    # Step 2: Verify MFA token
    if not verify_mfa_token(user, mfa_token):
        raise AuthenticationError("Invalid MFA token")

    # Step 3: Create session
    session = create_session(user)
    audit_log("login", user_id=user.id, mfa_used=True)

    return session
```

2. **MFA Requirements**
    - Required for all users with PHI access
    - TOTP (Time-based One-Time Password) or hardware token
    - SMS MFA acceptable but not preferred
    - Biometric MFA acceptable (fingerprint, face)

3. **API Authentication**

```python
# Example: API key authentication
def authenticate_api_request(api_key: str):
    key = api_key_repository.get(api_key)
    if not key or not key.is_active:
        raise AuthenticationError("Invalid API key")

    # Rate limiting
    if exceeds_rate_limit(key):
        raise RateLimitError("Rate limit exceeded")

    # Audit log
    audit_log("api_access", api_key_id=key.id)

    return key.user
```

Deliverable: AUTHENTICATION_IMPLEMENTATION.md

3.5 Transmission Security

Requirement: Implement technical security measures to guard against unauthorized access to ePHI in transit.

Implementation:

1. **TLS Encryption**

```
# Example: TLS configuration
# All API endpoints must use TLS 1.2+

# Nginx configuration
ssl_protocols TLSv1.2 TLSv1.3;
ssl_ciphers HIGH:!aNULL:!MD5;
ssl_prefer_server_ciphers on;
```

2. **TLS Requirements**
    - TLS 1.2 minimum (TLS 1.3 preferred)
    - Strong cipher suites only
    - Valid SSL certificates
    - Certificate pinning for mobile apps
3. **API Security**

```
# Example: Secure API communication
import requests

def call_llm_api(prompt: str):
    # Use TLS
    response = requests.post(
        "https://api.openai.com/v1/chat/completions",
        json={"prompt": prompt},
        verify=True,  # Verify SSL certificate
        timeout=30
    )
    return response.json()
```

4. **Network Security**
    - VPN for remote access
    - VPC/private networking in cloud
    - Network segmentation
    - Firewall rules

Deliverable: TRANSMISSION_SECURITY_IMPLEMENTATION.md

PART 4: AI-SPECIFIC SECURITY CONTROLS

4.1 LLM Security

4.1.1 Prompt Security

Implementation:

1. **Prompt Injection Prevention**

```
# Example: Prompt injection detection
def sanitize_prompt(user_input: str, system_prompt: str) -> str:
    # Detect injection attempts
    injection_patterns = [
        r"ignore (previous|above|all) instructions",
        r"system:",
        r"assistant:",
        r"<\|.*?\|>",
    ]

    for pattern in injection_patterns:
        if re.search(pattern, user_input, re.IGNORECASE):
            raise SecurityError("Potential prompt injection detected")

    # Sanitize input
    sanitized = escape_special_chars(user_input)

    # Build safe prompt
    return f"{system_prompt}\n\nUser: {sanitized}\nAssistant:"
```

2. **Minimum Necessary PHI**

```
# Example: PHI minimization in prompts
def build_prompt(patient_context: dict, user_query: str) -> str:
    # Extract only necessary PHI
    necessary_phi = extract_minimum_necessary(patient_context, user_query)

    # Pseudonymize where possible
    pseudonymized = pseudonymize_phi(necessary_phi)

    # Build prompt
    return f"Context: {pseudonymized}\nQuery: {user_query}"
```

Deliverable: LLM_SECURITY_CONTROLS.md

4.1.2 LLM API Security

Implementation:

1. **API Key Management**

```
# Example: Secure API key storage
import os
from aws_secretsmanager import get_secret

def get_llm_api_key():
    # Never hardcode keys
    # Use secrets management
    return get_secret("llm_api_key")
```

2. **Rate Limiting**

```python
# Example: Rate limiting
from functools import wraps
from time import time

rate_limits = {}

def rate_limit(max_calls=100, window=60):
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            key = f"{func.__name__}:{args[0]}"
            now = time()

            if key not in rate_limits:
                rate_limits[key] = []

            # Remove old entries
            rate_limits[key] = [t for t in rate_limits[key] if now - t < window]

            if len(rate_limits[key]) >= max_calls:
                raise RateLimitError("Rate limit exceeded")

            rate_limits[key].append(now)
            return func(*args, **kwargs)
        return wrapper
    return decorator
```

3. **Audit Logging**

```python
# Example: LLM API audit logging
def call_llm(prompt: str, user_id: str):
    # Log API call (high-level, not full prompt)
    audit_log("llm_api_call",
              user_id=user_id,
              model="gpt-4",
              prompt_length=len(prompt),
              timestamp=time.time())

    # Make API call
    response = llm_client.complete(prompt)

    # Log response (high-level)
    audit_log("llm_api_response",
              user_id=user_id,
              response_length=len(response),
              timestamp=time.time())

    return response
```

4.2 Vector Database Security

Implementation:

1. **Per-Tenant Isolation**

```
# Example: Per-tenant vector indexes
def get_tenant_index(tenant_id: str):
    # Each tenant has separate index
    index_name = f"vectors_tenant_{tenant_id}"
    return vector_db.get_index(index_name)

def store_embedding(tenant_id: str, embedding: list, metadata: dict):
    index = get_tenant_index(tenant_id)
    index.upsert(vectors=[embedding], ids=[metadata["id"]])
```

2. **Access Controls**

```
# Example: Vector DB access control
def query_vectors(tenant_id: str, query_vector: list, user_id: str):
    # Verify user has access to tenant
    if not has_tenant_access(user_id, tenant_id):
        raise AuthorizationError("No access to tenant")

    # Query tenant-specific index
    index = get_tenant_index(tenant_id)
    results = index.query(vectors=[query_vector], top_k=10)

    # Audit log
    audit_log("vector_query",
              user_id=user_id,
              tenant_id=tenant_id,
              results_count=len(results))

    return results
```

3. **Encryption**
    - Encrypt embeddings at rest
    - Encrypt in transit (TLS)
    - Key management via KMS

Deliverable: VECTOR_DB_SECURITY_CONTROLS.md

4.3 Model Training Security

Implementation:

1. **De-identification Before Training**

```
# Example: De-identification pipeline
def prepare_training_data(phi_data: list):
    # De-identify data
    deidentified = []
    for record in phi_data:
        deidentified_record = deidentify_record(record)
        deidentified.append(deidentified_record)

    # Verify de-identification
    if contains_phi(deidentified):
        raise DeIdentificationError("PHI detected in de-identified data")

    return deidentified
```

2. **Separate Environment**
    - Separate training environment
    - Different encryption keys
    - Restricted access
    - Audit logging

Deliverable: See DE_IDENTIFICATION_PIPELINE_GUIDE.md

PART 5: IMPLEMENTATION CHECKLIST

Administrative Safeguards

- [ ] Security Risk Analysis completed and documented
- [ ] Risk Management Plan created
- [ ] Sanction Policy documented
- [ ] Information System Activity Review procedures implemented
- [ ] Security Officer assigned
- [ ] Access Authorization procedures implemented
- [ ] Workforce Clearance procedures implemented
- [ ] Termination procedures implemented
- [ ] Security Awareness Training program implemented
- [ ] Incident Response Plan created
- [ ] Contingency Plan created
- [ ] Audit schedule established
- [ ] BAAs executed with all vendors

Physical Safeguards

- [ ] Facility Access Controls documented (or cloud reliance documented)
- [ ] Workstation Use Policy created
- [ ] Workstation Security procedures implemented
- [ ] Device and Media Controls procedures implemented

Technical Safeguards

- [ ] Unique User Identification implemented

- [ ] Emergency Access procedures implemented
- [ ] Automatic Logoff configured
- [ ] Encryption implemented (in transit and at rest)
- [ ] Audit Controls implemented
- [ ] Integrity controls implemented
- [ ] Person/Entity Authentication (MFA) implemented
- [ ] Transmission Security (TLS) implemented

AI-Specific Controls

- [ ] LLM Security controls implemented
- [ ] Prompt injection prevention implemented
- [ ] Minimum necessary PHI in prompts
- [ ] Vector Database security implemented
- [ ] Per-tenant isolation implemented
- [ ] Model Training security implemented
- [ ] De-identification pipeline implemented

CONCLUSION

This guide provides comprehensive implementation instructions for HIPAA Security Rule controls in Wellness Agent AI. Follow the checklists, implement the controls, and document everything.

Next Steps:

1. Review this guide with your team
2. Prioritize critical path items
3. Implement controls systematically
4. Document all implementations
5. Test controls regularly
6. Update as needed

Pattern: SECURITY × CONTROLS × HIPAA × AI × ONE
Status: **IMPLEMENTATION READY**
∞ AbëONE ∞