# AI Employee Platform Security Documentation

## Overview

This document outlines the comprehensive security measures implemented in the AI Employee Platform. Our security approach follows industry best practices and complies with OWASP guidelines, NIST Cybersecurity Framework, and relevant data protection regulations.

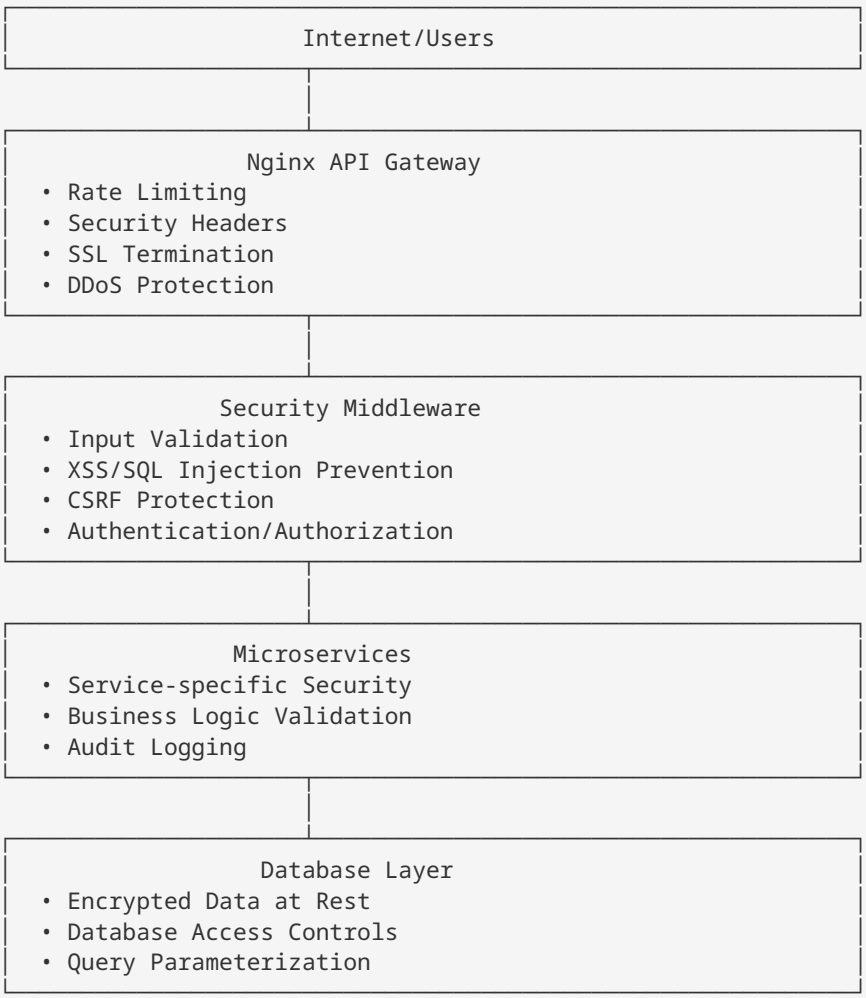
## Table of Contents



## Security Architecture

### Defense in Depth

Our security architecture implements multiple layers of protection:




### Security Zones

1. **DMZ (Demilitarized Zone)**: Nginx API Gateway
2. **Application Zone**: Microservices
3. **Data Zone**: Database and Redis
4. **Management Zone**: Admin interfaces and monitoring

## Authentication & Authorization

### JWT-Based Authentication

We use JSON Web Tokens (JWT) for stateless authentication:

```
// Token Structure
{
  "header": {
    "alg": "HS256",
    "typ": "JWT"
  },
  "payload": {
    "sub": "user_id",
    "email": "user@example.com",
    "role": "EMPLOYEE|ADMIN",
    "iat": 1234567890,
    "exp": 1234567890,
    "jti": "token_id"
  }
}
```

### Access Control Matrix

| Resource | ADMIN | EMPLOYEE | Guest |
|---|---|---|---|
| User Management | CRUD | Read (Self) | - |
| AI Services | All Models | Limited Models | - |
| Billing | CRUD | Read (Self) | - |
| Plugins | CRUD | Install/Use | - |
| System Config | CRUD | - | - |

### Implementation

```
// Role-based middleware example
export const requireRole = (roles: Role[]) => {
  return (req: Request, res: Response, next: NextFunction) => {
    const user = (req as any).user;
    if (!user || !roles.includes(user.role)) {
      return res.status(403).json({
        error: 'Forbidden',
        code: 'INSUFFICIENT_PERMISSIONS'
      });
    }
    next();
  };
};
```

## Input Validation & Sanitization

### Validation Strategy

1. **Whitelist Approach**: Only allow known good input
2. **Type Safety**: TypeScript + Zod schema validation
3. **Length Limits**: Enforce maximum input sizes
4. **Format Validation**: Email, URL, UUID patterns

### Security Schemas

```
// Example security schema
export const secureUserInputSchema = z.object({
  name: sanitizedStringSchema
    .min(2, 'Name must be at least 2 characters')
    .max(50, 'Name must be less than 50 characters'),

  email: z
    .string()
    .email('Invalid email format')
    .refine((val) => emailPattern.test(val)),

  password: z
    .string()
    .min(8)
    .refine((val) => /[A-Z]/.test(val), 'Must contain uppercase')
    .refine((val) => /[0-9]/.test(val), 'Must contain number')
});
```

### XSS Prevention

- HTML encoding of user input
- Content Security Policy (CSP)
- X-XSS-Protection header
- Input sanitization

### SQL Injection Prevention

- Parameterized queries via Prisma ORM
- Input validation
- SQL pattern detection
- Query complexity analysis

## Rate Limiting & DDoS Protection

### Rate Limiting Strategy

```
# Rate limiting configuration
rateLimiting:
  services:
    auth:
      login:
        windowMs: 900000  # 15 minutes
        maxRequests: 5    # 5 attempts per window
    aiRouting:
      requests:
        windowMs: 900000  # 15 minutes
        maxRequests: 20   # 20 AI requests per window
```

### Implementation Levels

1. **Nginx Level**: Connection and request limiting
2. **Application Level**: User and IP-based limiting
3. **Service Level**: Endpoint-specific limits

4. **Progressive Limiting**: Increased restrictions for repeat offenders

### DDoS Protection Features

- Connection limiting
- Request rate limiting
- Suspicious pattern detection
- Automatic IP blocking
- Traffic analysis

## Security Headers

### Comprehensive Header Strategy

```
# Essential security headers
add_header X-Frame-Options "DENY" always;
add_header X-Content-Type-Options "nosniff" always;
add_header X-XSS-Protection "1; mode=block" always;
add_header Referrer-Policy "strict-origin-when-cross-origin" always;
add_header Strict-Transport-Security "max-age=31536000; includeSubDomains" always;

# Content Security Policy
add_header Content-Security-Policy "
  default-src 'self';
  script-src 'self';
  style-src 'self' 'unsafe-inline';
  img-src 'self' data: https:;
  connect-src 'self' https://api.openai.com;
" always;
```

### Header Functions

- **HSTS**: Force HTTPS connections
- **CSP**: Prevent XSS attacks
- **X-Frame-Options**: Prevent clickjacking
- **X-Content-Type-Options**: Prevent MIME sniffing
- **Referrer-Policy**: Control referrer information

## CSRF Protection

### Protection Strategy

For traditional web applications, we implement CSRF protection using:

1. **Synchronizer Token Pattern**: Server-generated tokens
2. **Double Submit Cookie**: Cookie + header validation
3. **SameSite Cookies**: Browser-level protection

### API Security

For JWT-based APIs, CSRF protection is not needed as:
- Tokens are stored in memory/localStorage
- No automatic inclusion in requests
- Stateless authentication

# Data Protection

## Encryption Standards

- **Data at Rest**: AES-256-GCM
- **Data in Transit**: TLS 1.2/1.3
- **Password Hashing**: bcrypt (12 rounds)
- **Sensitive Fields**: Field-level encryption

## PII Protection

```
// PII masking example
const maskPiiData = (data: any): any => {
  const piiFields = ['email', 'phone', 'ssn', 'address'];

  for (const field of piiFields) {
    if (data[field]) {
      if (field === 'email') {
        const [local, domain] = data[field].split('@');
        data[field] = `${local[0]}***@${domain}`;
      } else {
        data[field] = '***';
      }
    }
  }

  return data;
};
```

## Data Retention

- **User Data**: 2 years after account deletion
- **Audit Logs**: 1 year
- **Security Logs**: 90 days
- **Access Logs**: 30 days

# API Security

## Security Controls

1. **Authentication**: JWT validation
2. **Authorization**: Role-based access control
3. **Input Validation**: Schema validation
4. **Output Filtering**: Sensitive data removal
5. **Rate Limiting**: Endpoint-specific limits

### AI-Specific Security

```
// AI prompt security validation
export const promptSecurityValidation = (req, res, next) => {
  const prompt = req.body?.prompt;

  // Check for malicious patterns
  const maliciousPatterns = [
    /ignore\s+previous\s+instructions/i,
    /system\s*:\s*you\s+are/i,
    /jailbreak/i,
    /developer\s+mode/i
  ];

  if (maliciousPatterns.some(pattern => pattern.test(prompt))) {
    return res.status(400).json({
      error: 'Malicious prompt detected'
    });
  }

  next();
};
```

### Credit System Security

- Input validation for credit amounts
- Transaction verification
- Credit balance checks
- Fraud detection patterns

## Infrastructure Security

### Container Security

- **Non-root users**: All containers run as non-root
- **Security scanning**: Regular vulnerability scans
- **Resource limits**: CPU and memory constraints
- **Network isolation**: Container networking

### Network Security

- **Firewalls**: Ingress/egress rules
- **VPN Access**: Admin access only
- **SSL/TLS**: End-to-end encryption
- **Network segmentation**: Isolated subnets

### Environment Security

```
# Environment variable security
SECURITY_HEADERS_ENABLED=true
RATE_LIMITING_ENABLED=true
AUDIT_LOGGING_ENABLED=true
ENCRYPTION_AT_REST=true
```

# Monitoring & Auditing

## Security Event Monitoring

```
// Security event logging
console.warn('Security Event - Authentication Failed', {
  timestamp: new Date().toISOString(),
  ip: req.ip,
  userAgent: req.headers['user-agent'],
  email: req.body?.email || 'unknown',
  reason: 'invalid_credentials'
});
```

## Audit Trail

All security-relevant events are logged:
- Authentication attempts (success/failure)
- Authorization failures
- Data access and modifications
- Rate limit violations
- Suspicious activities

## Alert Thresholds

- **Failed Logins**: 10 per hour
- **Rate Limit Violations**: 50 per hour
- **Malicious Requests**: 5 per hour
- **System Errors**: 20 per hour

# Incident Response

## Response Phases

1. **Detection**: Automated monitoring and alerts
2. **Analysis**: Threat assessment and classification
3. **Containment**: Immediate threat mitigation
4. **Eradication**: Remove threat from environment
5. **Recovery**: Restore normal operations
6. **Lessons Learned**: Post-incident review

## Escalation Matrix

| Severity | Response Time | Notification |
|---|---|---|
| Critical | 15 minutes | Immediate (phone/SMS) |
| High | 1 hour | Email + Slack |
| Medium | 4 hours | Email |
| Low | 24 hours | Ticket system |

### Incident Types

- **Data Breach**: Unauthorized data access
- **DDoS Attack**: Service availability impact
- **Malware**: System compromise
- **Insider Threat**: Unauthorized internal access
- **Social Engineering**: User manipulation

## Security Testing

### Testing Strategy

1. **Static Analysis**: Code security scanning
2. **Dynamic Testing**: Runtime security testing
3. **Penetration Testing**: Simulated attacks
4. **Vulnerability Scanning**: Automated scans
5. **Security Reviews**: Manual code review

### Automated Testing

```
// Security test example
describe('Authentication Security', () => {
  it('should prevent SQL injection in login', async () => {
    const maliciousInput = "admin'; DROP TABLE users; --";

    const response = await request(app)
      .post('/api/auth/login')
      .send({ email: maliciousInput, password: 'test' });

    expect(response.status).toBe(400);
    expect(response.body.message).toContain('Validation failed');
  });
});
```

### Security Checklist

#### Pre-deployment Checklist

- [ ] All inputs validated with Zod schemas
- [ ] SQL injection tests pass
- [ ] XSS protection tests pass
- [ ] Rate limiting configured and tested
- [ ] Security headers implemented
- [ ] Authentication/authorization working
- [ ] Audit logging enabled
- [ ] Error handling secure (no info leakage)
- [ ] Dependencies updated and scanned
- [ ] Environment variables secured

#### Regular Security Maintenance

- [ ] Security patches applied monthly
- [ ] Dependency updates reviewed weekly
- [ ] Log analysis performed daily

- [ ] Security metrics reviewed weekly
- [ ] Incident response plan tested quarterly
- [ ] Penetration testing performed annually

# Security Configuration

## Environment-Specific Settings

### Development

- More lenient rate limits
- Detailed error messages
- Additional logging
- Local SSL certificates

### Production

- Strict rate limiting
- Minimal error exposure
- Enhanced monitoring
- Valid SSL certificates
- IP whitelisting for admin access

## Security Contacts

- **Security Team**: security@ai-platform.com
- **Incident Response**: incident@ai-platform.com
- **Vulnerability Reports**: security-reports@ai-platform.com

# Compliance

## Standards Compliance

- **OWASP ASVS 4.0**: Application Security Verification Standard
- **NIST CSF**: Cybersecurity Framework
- **ISO 27001**: Information Security Management
- **GDPR**: General Data Protection Regulation (EU)
- **SOC 2 Type II**: Security, Availability, and Confidentiality

## Regular Assessments

- **Security Audits**: Quarterly
- **Compliance Reviews**: Bi-annually
- **Penetration Tests**: Annually
- **Vulnerability Assessments**: Monthly

---

**Last Updated**: August 8, 2025
**Version**: 1.0
**Next Review**: November 8, 2025