PII Protection Report

Enxi ERP System

Security and Compliance Analysis

June 26, 2025

Executive Summary

This report provides a comprehensive analysis of how Personally Identifiable Information (PII) is protected in the Enxi ERP system. The analysis covers authentication, encryption, data validation, access controls, audit logging, and security middleware implementations.

1. Data Encryption Practices

1.1 Password Encryption

- **Technology:** bcryptjs with salt rounds of 10
- **Implementation:** lib/services/auth.service.ts
- **Details:**
    - Passwords are hashed using bcrypt before storage
    - Salt rounds of 10 provide strong protection against rainbow table attacks
    - Plain text passwords are never stored in the database

```
async hashPassword(password: string): Promise<string> {
  return bcrypt.hash(password, 10)
}
```

1.2 JWT Token Security

- **Secret Key:** Stored in environment variable JWT_SECRET
- **Token Expiry:** 7 days
- **Storage:** HTTP-only cookies and localStorage
- **Implementation:** JWT tokens contain minimal user information (id, username, email, role)

2. Authentication and Authorization

2.1 Multi-Layer Authentication

- **Server-Side Auth:** lib/auth/server-auth.ts - Validates JWT tokens from cookies/headers
- **Client-Side Auth:** lib/hooks/use-auth.ts - Manages authentication state
- **API Auth:** lib/utils/auth.ts - Provides middleware for API route protection

2.2 Role-Based Access Control (RBAC)

- **Roles:** SUPER_ADMIN, ADMIN, MANAGER, SALES_REP, ACCOUNTANT, WAREHOUSE, VIEWER, USER
- **Middleware:** lib/middleware/rbac.middleware.ts
- **Features:**
    - Role-based route protection
    - Permission-based access control
    - Resource ownership validation
    - Hierarchical role checking

2.3 Session Management

- **Token-Based:** JWT tokens instead of server-side sessions
- **Automatic Cleanup:** Tokens expire and are validated on each request
- **Security Headers:** Set via middleware for CORS, XSS, and clickjacking protection

3. Data Sanitization and Validation

3.1 Input Validation

- **Library:** Zod for schema validation
- **Implementation:** lib/validators/ directory
- **Coverage:**
    - Email validation with format checks
    - Phone number validation with regex patterns
    - String length limits to prevent buffer overflow
    - SQL injection prevention through Prisma ORM
    - XSS prevention through input sanitization

3.2 Common Validators

```
// Email validation
emailValidator = z.string()
  .email('Invalid email format')
  .max(255, 'Email must be less than 255 characters')
  .toLowerCase()
  .trim()

// Phone validation
```

```
phoneValidator = z.string()
  .max(20, 'Phone must be less than 20 characters')
  .regex(/^[\d\s\-\+\(\)]+$/, 'Invalid phone format')
```

4. Security Middleware

4.1 Global Middleware (`middleware.ts`)

- **CORS Protection:** Configurable allowed origins
- **Security Headers:**
  - **X-Content-Type-Options:** nosniff
  - **X-Frame-Options:** DENY
  - **X-XSS-Protection:** 1; mode=block
  - **Referrer-Policy:** strict-origin-when-cross-origin

4.2 Authentication Middleware

- **Token Validation:** Automatic JWT verification
- **User Status Check:** Ensures user is active in database
- **Request Authentication:** Supports both cookie and header-based auth

4.3 Rate Limiting

- **Implementation:** In-memory rate limiting per user
- **Default:** 100 requests per 15-minute window
- **Headers:** Provides rate limit information in response headers

5. API Security

5.1 Centralized API Client

- **Location:** lib/api/client.ts
- **Features:**
    - Automatic authentication token injection
    - Standardized error handling
    - Type-safe responses
    - Automatic 401 handling with redirect to login

5.2 API Route Protection

- **Pattern:** All API routes use withAuth wrapper
- **Example:**

```
        export async function GET(request: NextRequest) {
  return withAuth(request, async ({ user }) => {
    // Route handler with authenticated user
  })
}
```

6. Audit Logging

6.1 Comprehensive Audit Trail

- **Middleware:** lib/middleware/audit.middleware.ts

- **Captured Data:**
  - User ID and action performed
  - Timestamp and IP address
  - User agent information
  - Before/after data for updates
  - Request/response size

6.2 Severity Levels

- **CRITICAL:** User deletions, permission changes, data migrations
- **HIGH:** Approvals, rejections, bulk operations
- **MEDIUM:** User/payment/invoice modifications
- **LOW:** Read operations and standard updates

6.3 PII in Audit Logs

- **Current State:** Audit logs may contain full entity data including PII
- **Recommendation:** Implement data masking for sensitive fields in audit logs

7. Database Security

7.1 ORM Protection

- **Prisma ORM:** Prevents SQL injection through parameterized queries
- **Query Building:** Type-safe query construction
- **Transaction Support:** ACID compliance for data integrity

7.2 Data Access Patterns

- **Selective Queries:** Use of select and include to limit data exposure
- **Soft Deletes:** Implemented for maintaining data integrity
- **Relationship Management:** Proper foreign key constraints

8. Environment Variable Security

8.1 Sensitive Configuration

- **JWT_SECRET:** For token signing
- **DATABASE_URL:** Database connection string
- **NEXTAUTH_SECRET:** NextAuth encryption key
- **ALLOWED_ORIGINS:** CORS whitelist

8.2 Best Practices

- **.env.example:** Provided for configuration reference
- **Production Notes:** Clear instructions for secure production setup

9. Session and Cookie Security

9.1 Cookie Configuration

- **auth-token:** Used for authentication
- **HttpOnly:** Not implemented (recommendation: enable HttpOnly flag)

- **Secure:** Not enforced (recommendation: enable for HTTPS)
- **SameSite:** Not configured (recommendation: set to 'strict')

10. Data Validation Examples

10.1 Lead Management

```
// Lead validation ensures PII is properly formatted
export const createLeadSchema = z.object({
  name: nameValidator,
  email: emailValidator,
  phone: phoneValidator.optional(),
  company: z.string().max(255),
  // ... other fields
})
```

10.2 User Management

- Password complexity requirements
- Email uniqueness validation
- Role assignment restrictions
- Profile data sanitization

11. Identified Vulnerabilities and Recommendations

11.1 High Priority

1. Database Encryption: Implement encryption at rest for SQLite database
2. Cookie Security: Enable HttpOnly, Secure, and SameSite flags
3. HTTPS Enforcement: No explicit HTTPS redirect in middleware
4. API Rate Limiting: Current implementation is in-memory only

11.2 Medium Priority

1. Audit Log Masking: Implement PII masking in audit logs
2. Password Policy: Add configurable password complexity requirements
3. Two-Factor Authentication: Not currently implemented
4. Session Timeout: Implement idle session timeout

11.3 Low Priority

1. Content Security Policy: Add CSP headers
2. API Versioning: Implement versioned API endpoints
3. Field-Level Encryption: For highly sensitive data fields
4. Data Retention Policy: Implement automatic PII deletion after retention period

12. Compliance Considerations

12.1 GDPR Compliance

- **Data Minimization:** Collect only necessary PII

- **Right to Erasure:** Implement hard delete functionality
- **Data Portability:** Add data export features
- **Consent Management:** Track and manage user consent

12.2 Security Standards

- **OWASP Top 10:** Address common vulnerabilities
- **PCI DSS:** If handling payment card data
- **ISO 27001:** Information security management

13. Positive Security Features

13.1 Strong Points

- Comprehensive input validation with Zod
- JWT-based stateless authentication
- Role-based access control
- Audit logging for accountability
- Parameterized queries preventing SQL injection
- Centralized error handling
- Type-safe API implementation

13.2 Defense in Depth

- Multiple layers of authentication

- Request validation at multiple points
- Automatic security headers
- Error message sanitization

14. Implementation Examples

14.1 Secure User Creation

```
async createUser(data: CreateUserDto, createdBy: string) {
  // Password hashing
  const hashedPassword = await this.hashPassword(data.password)

  // Transaction for data integrity
  const user = await prisma.$transaction(async (tx) => {
    // Create user with hashed password
    // Create profile with sanitized data
    // Set up default permissions
  })

  // Audit logging
  await this.createAuditLog({
    userId: createdBy,
    action: 'CREATE',
    entityType: 'User',
    entityId: user.id,
  })
}
```

14.2 Secure API Endpoint

```
export async function GET(request: NextRequest) {
  return withAuth(request, async ({ user }) => {
    // Validate user permissions
    if (!user.role.includes('ADMIN')) {
      return NextResponse.json({ error: 'Forbidden' }, { status: 403 })
    }

    // Fetch data with PII protection
    const users = await userService.listUsers({
      // Exclude sensitive fields
      select: { id: true, username: true, email: true, role: true }
    })

    return NextResponse.json(users)
  }, ['ADMIN', 'SUPER_ADMIN'])
}
```

15. Conclusion

The Enxi ERP system implements several good security practices for PII protection, including:

- Strong password hashing
- JWT-based authentication
- Input validation and sanitization
- Audit logging
- RBAC implementation

However, there are areas for improvement:

- Database encryption at rest
- Enhanced cookie security
- HTTPS enforcement
- PII masking in logs
- Two-factor authentication

The system provides a solid foundation for PII protection but should implement the recommended enhancements for production deployment, especially in regulated industries or when handling sensitive customer data.

Appendix: Security Checklist

- [ ] Enable database encryption at rest
- [ ] Configure secure cookie flags (HttpOnly, Secure, SameSite)
- [ ] Implement HTTPS redirect in middleware
- [ ] Add distributed rate limiting (Redis-based)
- [ ] Implement PII masking in audit logs
- [ ] Add password complexity configuration
- [ ] Implement 2FA support
- [ ] Add idle session timeout
- [ ] Configure Content Security Policy
- [ ] Implement API versioning
- [ ] Add field-level encryption for sensitive data
- [ ] Create data retention and deletion policies

- [ ] Implement GDPR compliance features
- [ ] Regular security audits and penetration testing
- [ ] Security training for development team