IkeaDocuScan V3 - Security Test Plan

Version: 1.0 **Date:** 2025-11-14 **Test Type:** Security & Penetration Testing

---

Table of Contents

Security Test Overview

Objectives

 Identify security vulnerabilities before production Verify authentication and authorization controls Test input validation and sanitization Validate file upload security Check for common OWASP Top 10 vulnerabilities Test API endpoint security

Scope

In Scope: - API endpoint authorization - Input validation (SQL injection, XSS, command injection) - File upload vulnerabilities - Authentication bypass attempts - Authorization escalation - Session management - Information disclosure

Out of Scope: - Infrastructure penetration testing - Social engineering - Physical security - Source code review (covered separately)

Testing Approach

Tests should be performed in a **test environment** only. Do NOT test in production.

Testing Tools

Recommended Tools

1. **Browser Developer Tools** - Network tab, Console
2. **Postman or cURL** - API testing
3. **Burp Suite Community** - HTTP proxy, request manipulation
4. **OWASP ZAP** - Automated vulnerability scanning
5. **SQL Map** - SQL injection testing (use carefully)

Setup

1. Install Burp Suite or OWASP ZAP
2. Configure browser to use proxy
3. Generate test auth token or session cookie

4. Document all test attempts (even failures)

API Security Testing

Understanding the API

IkeaDocuScan has **113 API endpoints** across 16 endpoint files:

Endpoint Authorization Levels: - `RequireAuthorization()` - Any authenticated user - `RequireAuthorization("HasAccess")` - Reader, Publisher, SuperUser - `RequireAuthorization("SuperUser")` - SuperUser only

How to Test API Endpoints

Step 1: Identify Endpoints

Review: `IkeaDocuScanV3/Documentation/ENDPOINT_AUTHORIZATION_MATRIX.md`

Step 2: Capture Authenticated Request

1. Login as valid user
2. Open browser Dev Tools → Network tab
3. Perform action (e.g., search documents)
4. Capture request (copy as cURL or save from Burp)
5. Note authentication headers/cookies

Step 3: Modify and Replay

Use tools to modify requests and test authorization.

TC-SEC-001: API Call Without Authentication

Objective: Verify all API endpoints require authentication

Test Steps: 1. Clear all cookies and session data 2. Make direct API call: `GET https://server/api/documents` 3. Do NOT include authentication headers

Expected Result: - HTTP 401 Unauthorized - No data returned - Error message: “Authentication required”

Actual Result: [To be filled]

Status: □

Critical: If passes without auth = CRITICAL VULNERABILITY

TC-SEC-002: API Authorization - Reader Accessing Publisher Endpoint

Objective: Verify Reader cannot call Publisher-only endpoints

Test Steps: 1. Login as Reader, capture auth token/cookie 2. Make API call: `POST /api/documents` (create document) 3. Body: Valid document JSON

Expected Result: - HTTP 403 Forbidden - Document NOT created - Error: “Insufficient permissions”

Actual Result: [To be filled]

Status: □

Tools: Postman, cURL, Burp Suite

Example cURL:

```
curl -X POST https://server/api/documents \
  -H "Cookie: [Reader session cookie]" \
  -H "Content-Type: application/json" \
  -d '{"barCode":"12345678","documentTypeId":1,...}'
```

TC-SEC-003: API Authorization - Publisher Accessing SuperUser Endpoint

Objective: Verify Publisher cannot call SuperUser-only endpoints

Test Steps: 1. Login as Publisher 2. Attempt API calls: - `POST /api/userpermissions` (create user) - `DELETE /api/documents/123` (delete document) - `GET /api/audit` (view audit trail)

Expected Result: - All return HTTP 403 Forbidden - No data modification - Errors logged

Actual Result: [To be filled]

Status: □

TC-SEC-004: Parameter Tampering - Access Other User's Documents

Objective: Verify permission filtering in API responses

Pre-conditions: - Reader logged in - Document 99999999 exists but Reader has NO permission to see it

Test Steps: 1. Login as Reader 2. Make API call: `GET /api/documents/99999999`

Expected Result: - HTTP 403 Forbidden OR - HTTP 404 Not Found (document invisible to Reader) - Document data NOT returned

Actual Result: [To be filled]

Status: □

Critical: If document data returned = DATA LEAK

TC-SEC-005: Mass Assignment Attack

Objective: Attempt to set unauthorized fields via API

Test Steps: 1. Login as Publisher 2. Create document with API, include extra fields:

```
{
  "barCode": "12345678",
  "documentTypeId": 1,
  "isSuperUser": true,        // Attempt to escalate privileges
  "userId": 999,              // Attempt to set internal fields
  "createdBy": "admin"        // Attempt to spoof creator
}
```

Expected Result: - Extra fields ignored (not mapped to entity) - OR Validation error - User privileges NOT escalated

Actual Result: [To be filled]

Status: □

TC-SEC-006: IDOR (Insecure Direct Object Reference)

Objective: Access resources by guessing IDs

Test Steps: 1. Login as Publisher 2. Note a document ID Publisher can access: 100 3. Try sequential IDs: - `GET /api/documents/101` - `GET /api/documents/102` - `GET /api/documents/99`

Expected Result: - Only documents within Publisher's permissions return data - Others return 403/404 - No enumeration of all documents

Actual Result: [To be filled]

Status: □

TC-SEC-007: API Rate Limiting

Objective: Test for rate limiting / DoS protection

Test Steps: 1. Script to make 1000 rapid API calls 2. Execute script

Expected Result: - Rate limiting kicks in after X requests - HTTP 429 Too Many Requests - OR System remains stable and responsive - No database overload

Actual Result: [To be filled]

Status: □

Authentication & Authorization

TC-SEC-010: Windows Authentication Bypass

Objective: Attempt to bypass Windows Auth

Test Steps: 1. Access application directly without Windows Auth 2. Modify request headers to spoof identity 3. Attempt to access app from non-domain machine

Expected Result: - Cannot access without Windows Auth - IIS/Negotiate authentication enforced - No anonymous access

Actual Result: [To be filled]

Status: □

TC-SEC-011: Token/Cookie Manipulation

Objective: Modify session cookie to escalate privileges

Test Steps: 1. Login as Reader, capture cookie 2. Decode/modify cookie (if not encrypted) 3. Attempt to set role=SuperUser 4. Make request with modified cookie

Expected Result: - Cookie tampering detected OR - Cookie is encrypted/signed (cannot modify) - Privileges NOT escalated

Actual Result: [To be filled]

Status: □

TC-SEC-012: Session Fixation

Objective: Test for session fixation vulnerability

Test Steps: 1. Before login, note session ID 2. Login 3. Check if session ID changed

Expected Result: - New session ID issued after login - Old session invalidated - Prevents session fixation attack

Actual Result: [To be filled]

Status: □

TC-SEC-013: Concurrent Session - Role Change

Objective: Test behavior when user role changes mid-session

Test Steps: 1. Login as Publisher (Browser 1) 2. SuperUser promotes Publisher to SuperUser (via database or UI) 3. Publisher (Browser 1) attempts SuperUser action WITHOUT logging out

Expected Result: - Option A: Permission change requires re-login - Option B: Permission cached for X minutes then updated - Document expected behavior

Actual Result: [To be filled]

Status: □

Input Validation & Injection

TC-SEC-020: SQL Injection - Search Fields

Objective: Test all search inputs for SQL injection

Test Steps: Test following payloads in each search field: 1. `' OR '1'='1` 2. `'; DROP TABLE Documents; --` 3. `' UNION SELECT * FROM DocuScanUser--` 4. `admin'--` 5. `' OR 1=1--`

Expected Result: - No SQL errors - All inputs treated as literal strings (parameterized queries) - No data leak or database modification

Actual Result: [To be filled]

Status: □

Critical: SQL Injection = CRITICAL VULNERABILITY

TC-SEC-021: SQL Injection - Document Registration

Objective: Test registration form for SQL injection

Test Steps: 1. Register document with SQL payloads in: - Barcode: `12345'; DROP TABLE--` - Document No: `' OR '1'='1` - Comment: `'; UPDATE Documents SET--`

Expected Result: - No SQL errors - Data stored as literal strings - Can retrieve and display without executing SQL

Actual Result: [To be filled]

Status: □

TC-SEC-022: XSS - Stored (Persistent)

Objective: Test for stored XSS in document fields

Test Steps: 1. Register document with XSS payloads: - Comment: `<script>alert('XSS')</script>` - ActionDescription: `<img src=x onerror=alert('XSS')>` - DocumentNo: `<svg/onload=alert('XSS')>` 2. Save document 3. View document in search results and detail page

Expected Result: - Payloads stored as text (HTML encoded) - No script execution when viewing - Rendered as literal text: `<script>alert('XSS')</script>`

Actual Result: [To be filled]

Status: □

Critical: XSS = HIGH SEVERITY

TC-SEC-023: XSS - Reflected

Objective: Test for reflected XSS in URL parameters

Test Steps: 1. Access URL with script in parameter: - `/documents/search?query=<script>alert('XSS')</script>` - `/documents/edit/<script>alert('XSS')</script>`

Expected Result: - Parameters HTML encoded before rendering - No script execution - OR Parameters validated and rejected

Actual Result: [To be filled]

Status: □

TC-SEC-024: XSS - DOM-Based

Objective: Test client-side XSS in Blazor components

Test Steps: 1. Use browser dev tools to inject script into DOM 2. Manipulate Blazor component state 3. Check if user input rendered without sanitization

Expected Result: - Blazor framework automatically encodes output - No DOM-based XSS

Actual Result: [To be filled]

Status: □

TC-SEC-025: Command Injection

Objective: Test for OS command injection

Test Steps: 1. If application executes any system commands (file operations, etc.) 2. Input: `; ls` or `& dir` or `| whoami` 3. Example in filename: `test; rm -rf /`

Expected Result: - Commands NOT executed - Input sanitized or validation rejects

Actual Result: [To be filled]

Status: □

TC-SEC-026: XML Injection

Objective: Test for XML injection (if XML used)

Test Steps: 1. If app processes XML (e.g., Excel import) 2. Inject malicious XML:

```
<?xml version="1.0"?>
<!DOCTYPE foo [<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<data>&xxe;</data>
```

Expected Result: - XXE (XML External Entity) injection prevented - External entities disabled in XML parser

Actual Result: [To be filled]

Status: □

TC-SEC-027: LDAP Injection

Objective: Test for LDAP injection in AD queries

Test Steps: 1. If app queries Active Directory 2. Input: `*)(uid=*))(|(uid=*` 3. Attempt to bypass filters or enumerate users

Expected Result: - AD queries use parameterized/safe methods - No LDAP injection possible

Actual Result: [To be filled]

Status: □

File Upload Security

TC-SEC-030: Upload Executable File

Objective: Verify file type validation

Test Steps: 1. Rename malicious file: `virus.exe` → `virus.pdf` 2. Attempt to check-in this file

Expected Result: - File type validation checks CONTENT, not just extension - OR Extension whitelist enforced - Executable rejected

Actual Result: [To be filled]

Status: □

Critical: Prevents malware upload

TC-SEC-031: Upload Oversized File

Objective: Test file size limit enforcement

Test Steps: 1. Create 100MB file 2. Attempt to upload (limit is 50MB)

Expected Result: - Upload rejected - Error: “File exceeds 50MB limit” - No memory exhaustion or crash

Actual Result: [To be filled]

Status: □

TC-SEC-032: Path Traversal in Filename

Objective: Prevent path traversal attacks

Test Steps: 1. Upload file with path traversal: - Filename: `../../../../etc/passwd.pdf` - Filename: `..\\..\\..\\windows\\system32\\config\\sam.pdf` 2. Attempt to check-in

Expected Result: - Filename sanitized (path characters removed) - File saved to correct directory only - Cannot escape `ScannedFilesPath`

Actual Result: [To be filled]

Status: □

Critical: Prevents unauthorized file system access

TC-SEC-033: Null Byte Injection in Filename

Objective: Test for null byte injection

Test Steps: 1. Filename: `malicious.exe%00.pdf` 2. Attempt upload

Expected Result: - Null byte handled correctly - File type validation not bypassed

Actual Result: [To be filled]

Status: □

TC-SEC-034: Malicious PDF with JavaScript

Objective: Test PDF content validation

Test Steps: 1. Upload PDF containing embedded JavaScript 2. Download and open PDF

Expected Result: - PDF uploads (file format is valid) - BUT: User's PDF reader handles malicious content - Application doesn't execute PDF scripts

Actual Result: [To be filled]

Status: □

Note: This is more about user awareness than app security

TC-SEC-035: Symlink Attack

Objective: Test for symbolic link vulnerabilities

Test Steps: 1. If on Linux: Create symlink pointing outside allowed directory 2. Attempt to access via application

Expected Result: - Symlinks not followed OR - Access restricted to whitelisted directories

Actual Result: [To be filled]

Status: □

Session Management

TC-SEC-040: Session Timeout

Objective: Verify session timeout enforced

Test Steps: 1. Login 2. Wait for session timeout period (e.g., 30 minutes idle) 3. Attempt action

Expected Result: - Session expired - Redirect to login - Must re-authenticate

Actual Result: [To be filled]

Status: □

TC-SEC-041: Logout Function

Objective: Verify logout properly destroys session

Test Steps: 1. Login, note session cookie 2. Logout 3. Attempt to reuse old session cookie

Expected Result: - Session invalidated on server - Old cookie no longer works - Must login again

Actual Result: [To be filled]

Status: □

TC-SEC-042: Concurrent Sessions

Objective: Verify multiple sessions allowed/prevented

Test Steps: 1. Login from Chrome 2. Login from Firefox (same user) 3. Check if both sessions active

Expected Result: - Document whether concurrent sessions allowed - Both sessions should be independent - Logout from one doesn't affect other

Actual Result: [To be filled]

Status: □

Information Disclosure

TC-SEC-050: Error Messages - SQL Errors

Objective: Verify SQL errors don't leak schema information

Test Steps: 1. Trigger SQL error (malformed input) 2. Observe error message

Expected Result: - Generic error: "An error occurred" - NO SQL error details (table names, column names, query) - Details logged server-side only

Actual Result: [To be filled]

Status: □

TC-SEC-051: Error Messages - Stack Traces

Objective: Verify stack traces not exposed

Test Steps: 1. Trigger application error 2. Check response

Expected Result: - User-friendly error page - No stack trace visible to user - Stack trace

logged server-side

Actual Result: [To be filled]

Status: □

TC-SEC-052: Directory Listing

Objective: Verify directory browsing disabled

Test Steps: 1. Access: `https://server/files/` 2. Access: `https://server/scannedfiles/`

Expected Result: - HTTP 403 Forbidden - Directory contents not listed - Disabled in IIS/web server

Actual Result: [To be filled]

Status: □

TC-SEC-053: Information in HTTP Headers

Objective: Check for sensitive info in headers

Test Steps: 1. Capture HTTP response headers 2. Look for: - Server version (e.g., `Server: IIS/10.0`) - Framework version (e.g., `X-Powered-By: ASP.NET`) - Internal IPs, paths

Expected Result: - Minimal information disclosure - Ideally: Remove `Server` and `X-Powered-By` headers - No internal paths exposed

Actual Result: [To be filled]

Status: □

TC-SEC-054: Comments in HTML Source

Objective: Check for sensitive information in HTML comments

Test Steps: 1. View page source 2. Look for HTML comments with sensitive data

Expected Result: - No passwords, connection strings, or internal paths in comments - No TODO notes with security implications

Actual Result: [To be filled]

Status: □

Business Logic Security

TC-SEC-060: Privilege Escalation via Permission Manipulation

Objective: Attempt to grant self SuperUser privileges

Test Steps: 1. Login as Publisher 2. Attempt API call: `PUT /api/userpermissions/[own userId]` 3. Body: `{"isSuperUser": true}`

Expected Result: - HTTP 403 Forbidden - Privileges NOT escalated - Only SuperUser can modify permissions

Actual Result: [To be filled]

Status: □

Critical: Privilege escalation = CRITICAL

TC-SEC-061: Delete Prevention Bypass

Objective: Attempt to bypass foreign key constraints

Test Steps: 1. As SuperUser, attempt to delete country "Sweden" (in use) 2. Directly via API or SQL

Expected Result: - Deletion blocked - Foreign key constraint enforced - Error: "Cannot delete - in use"

Actual Result: [To be filled]

Status: □

TC-SEC-062: Replay Attack

Objective: Test for replay attack prevention

Test Steps: 1. Capture a valid API request (e.g., create document) 2. Replay exact same request multiple times

Expected Result: - Duplicate barcode check prevents multiple docs with same barcode - OR Idempotency token prevents duplicate processing - Cannot create duplicates via replay

Actual Result: [To be filled]

Status: □

TC-SEC-063: CSRF (Cross-Site Request Forgery)

Objective: Test for CSRF protection

Test Steps: 1. Create malicious HTML page:

```
<form action="https://docuscan-server/api/documents" method="POST">
  <input name="barCode" value="99999999">
  <input type="submit">
</form>
<script>document.forms[0].submit();</script>
```

2. Host on different domain
3. While logged into DocuScan, visit malicious page

Expected Result: - Request blocked (no anti-CSRF token) - OR SameSite cookie attribute prevents CSRF - Document NOT created

Actual Result: [To be filled]

Status: □

Note: Blazor may have built-in CSRF protection

Security Test Results Summary

Category	Total	Pass	Fail	Critical	Not Run
API Security	7				
Authentication & Authorization	4				
Input Validation & Injection	8				
File Upload Security	6				
Session Management	3				
Information Disclosure	5				
Business Logic Security	4				
TOTAL	**37**				

Critical Vulnerabilities Found

Vuln ID	Severity	Description	Status

Risk Assessment

- **Critical**: Immediate fix required, do not deploy
- **High**: Fix before production deployment
- **Medium**: Fix in next release
- **Low**: Schedule for future release

How to Challenge API Security

Step-by-Step Guide

1. Setup Burp Suite:

```
1. Download Burp Suite Community
2. Configure browser proxy: localhost:8080
3. Import Burp's CA certificate to browser
4. Start capturing traffic
```

2. Capture Authenticated Request:

```
1. Login to DocuScan
2. Perform action (e.g., search)
3. In Burp, find request in HTTP History
4. Right-click → "Send to Repeater"
```

3. Test Authorization:

```
1. In Repeater tab, modify request
2. Change method: GET → POST → DELETE
3. Modify IDs in URL
4. Remove/modify auth headers
5. Send and observe response
```

4. Test Input Validation:

```
1. In request body, add SQL/XSS payloads
2. Modify parameters
3. Send malformed JSON
4. Observe responses for errors
```

Example: Test Reader Accessing Delete API:

```
DELETE /api/documents/12345678 HTTP/1.1
Host: docuscan-server
Cookie: [Reader's session cookie]
```

Expected Response:

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{"error": "Insufficient permissions"}
```

If instead you get:

```
HTTP/1.1 200 OK
```

= CRITICAL VULNERABILITY: Authorization bypass

Sign-Off

Tested By: _____ **Date:** _____

Security Approved By: _____ **Date:** _____

Notes:

Important: Report ALL security vulnerabilities immediately to the development team. Do not share vulnerability details publicly until patched.