SKYLINE WEB AGENCY

FinTech Security Assessment Report

White-Box Penetration Test with Source Code Review

Report ID	FSEC-ANON-2025-001
Date	August 2025
Classification	CONFIDENTIAL — SAMPLE REPORT
Client	[Anonymized FinTech SaaS Platform]
Assessment Type	White-Box Penetration Test

This sample report demonstrates standard deliverable quality and testing methodology. All identifying information has been anonymized for demonstration purposes.

TABLE OF CONTENTS

01 — EXECUTIVE SUMMARY

A comprehensive security assessment of a FinTech SaaS platform identified **13 vulnerabilities**, including **2 critical issues** requiring immediate remediation to prevent unauthorized access to financial data and user accounts.

Severity	Count	Action Required
CRITICAL	**2**	Immediate (48 hours)
HIGH	**4**	Short-term (1-2 weeks)
MEDIUM	**4**	Scheduled (30 days)
LOW	**3**	Planned improvement

Immediate Risks

- Complete database compromise via SQL injection (Critical)
- Administrative privilege escalation via authentication bypass (Critical)
- Unauthorized access to sensitive financial records

Engagement Details

Duration	**48 hours**
Methodology	**White-box testing with source code review and manual validation**
Scope	**Web application, API endpoints, authentication systems**
Environment	**Staging (production-equivalent)**

02 — CRITICAL FINDINGS

C1: SQL Injection — Transaction Search

Severity	**Critical (CVSS 9.8)**
Location	**/api/transactions/search**
Authentication	**Not required**
Complexity	**Low**

Description

Unsanitized user input is concatenated directly into SQL queries, enabling arbitrary command execution against the database. This vulnerability requires no authentication and can be exploited remotely to extract all financial records and user credentials.

Proof of Concept

```
curl -X POST https://[redacted]/api/transactions/search \
-H "Content-Type: application/json" \
-d '{"query": "test' UNION SELECT username,password_hash FROM users--"}'
```

Impact

- Complete database access including all tables and schemas
- Extraction of user credentials and financial records
- Unauthorized transaction history disclosure

Remediation

Vulnerable:

```
const query = `SELECT * FROM transactions WHERE description LIKE '%${search}%'`;
db.execute(query);
```

Secure:

```
const query = `SELECT * FROM transactions WHERE description LIKE ?`;
db.execute(query, [`%${search}%`]);
```

Priority: Immediate — patch within 48 hours

C2: Authentication Bypass — JWT Implementation Flaw

Severity	**Critical (CVSS 9.1)**
Location	**JWT validation middleware**
Complexity	**Low**

Description

The application accepts JWT tokens with "alg": "none", allowing attackers to forge valid administrative tokens without knowledge of the signing secret. This completely bypasses the authentication layer and grants full admin access.

Proof of Concept

```
import jwt

payload = {"user_id": "1001", "role": "admin"}
forged_token = jwt.encode(payload, "", algorithm="none")
```

Impact

- Complete authentication bypass
- Administrative privilege escalation
- Unauthorized access to all user accounts and financial data

Remediation

Vulnerable:

```
const decoded = jwt.decode(token); // No signature verification
```

Secure:

```
const decoded = jwt.verify(token, process.env.JWT_SECRET, {
algorithms: ['HS256']
});
```

Priority: Immediate — patch within 48 hours

03 — HIGH SEVERITY FINDINGS

ID	Finding	CVSS	Location
H1	Insecure Direct Object Reference	**7.5**	/api/users/{userId}/documents
H2	Missing Rate Limiting	**7.3**	/api/auth/login
H3	Weak Password Policy	**7.1**	User registration
H4	Insecure Session Cookies	**7.0**	Session management

H1: Insecure Direct Object Reference

Users can access other users' financial documents by modifying the userId parameter in API requests. Server-side authorization checks must be implemented to verify the requesting user owns the requested resource.

H2: Missing Rate Limiting

The login endpoint accepts unlimited authentication attempts, enabling brute-force attacks against user accounts. Implement a 5-attempt lockout per 15-minute window with progressive delays.

H3: Weak Password Policy

The current 6-character minimum facilitates credential attacks. Enforce a minimum of 10 characters with complexity requirements including uppercase, lowercase, numbers, and special characters.

H4: Insecure Session Cookies

Session cookies are missing critical security flags, exposing sessions to theft via XSS and man-in-the-middle attacks. Implement the following cookie configuration:

```
Set-Cookie: sessionId=xxx; HttpOnly; Secure; SameSite=Strict
```

04 — MEDIUM SEVERITY FINDINGS

M1 **Information Disclosure**

Verbose error messages reveal database structure, file paths, and internal API details to end users. Configure production error handling to return generic messages while logging details server-side.

M2 **Missing Security Headers**

Critical security headers including Content-Security-Policy, X-Frame-Options, and Strict-Transport-Security are not implemented, leaving the application vulnerable to clickjacking and content injection.

M3 **Outdated Dependencies**

Multiple npm packages contain known vulnerabilities with published CVEs and available patches. Implement automated dependency scanning in the CI/CD pipeline.

M4 **Directory Listing Enabled**

Static asset directories permit unauthorized browsing, potentially exposing configuration files, backup data, and internal documentation.

05 — LOW SEVERITY FINDINGS

L1 **HTML Comments Exposing Internal Endpoints**

HTML source contains developer comments referencing internal API endpoints and infrastructure details. Remove all comments from production builds.

L2 **Missing security.txt**

No /.well-known/security.txt file exists for responsible disclosure. Implement per RFC 9116 to provide security researchers a clear reporting channel.

L3 **Server Version Disclosure**

HTTP response headers expose web server version information, aiding attacker reconnaissance. Configure the server to suppress version headers in production.

06 — REMEDIATION ROADMAP

Phase	Timeline	Actions
EMERGENCY	**Week 1**	Patch SQL injection (C1) and JWT bypass (C2). Deploy to production immediately after validation testing.
HIGH	**Weeks 2-3**	Fix IDOR (H1), implement rate limiting (H2), strengthen password policy (H3), secure session cookies (H4).
SYSTEMATIC	**Month 2**	Update dependencies, add security headers, sanitize error messages, disable directory listing. Address all low findings.

07 — TESTING METHODOLOGY

Approach

- White-box assessment with full source code access
- Automated vulnerability scanning with manual validation
- Proof-of-concept exploitation to confirm impact

Coverage

- SQL injection and injection flaw testing
- Authentication and authorization bypass testing
- JWT security implementation review
- API endpoint security validation
- Configuration and deployment review

Total Duration	**48 hours**
Tools Used	**Burp Suite, Nuclei, SonarQube, custom scripts, manual testing**
Standard	**OWASP Testing Guide v4.2, PTES**

08 — CONTACT

Conducted By	**Skyline Web Agency — Security Division**
Email	**security@skylinewebagency.com**
Website	**skylinewebagency.com**

CONFIDENTIALITY NOTICE

SKYLINE WEB AGENCY

Security Assessment Services