



# Pentest\_Report

🕒 Created by	👤 S.Kaushik
🕒 Created time	@January 22, 2026 5:57 PM
☰ Category	Pentest
🕒 Last edited by	👤 S.Kaushik
🕒 Last updated time	@January 22, 2026 6:17 PM

## Pentest Report

By:

**Kaushik Sathyamurthy(s4079307)**

## Legal Disclaimer

**This report is shared solely for educational and portfolio demonstration purposes.**

**This penetration testing report was produced as part of a university-controlled cybersecurity lab exercise .All testing was conducted on intentionally vulnerable systems owned and authorized by the institution. No real customer, organization, or production environment was targeted. This report is shared solely for educational and portfolio demonstration purposes.**

## Table of contents

Sr. No	Title	Page number
i	List of figures and tables	2
1	Introduction	3
1.1	Outline	3
1.2	Scope	3
1.3	Executive summary	3
1.4	Summary of identified issues	3
2	SQL Injection in Login and Search Functions	4
3	Directory Traversal and Exposed Sensitive Files	4
4	Weak Password Hashing and Password Modification via SQL Injection	5

5	Insecure Direct Object Reference (IDOR) in Transaction Viewing	5
6	Client-Side Security Control Bypass via Cookie Manipulation	6
7	Unrestricted File Upload with Remote Code Execution	7
8	JWT Authentication Vulnerabilities	7
9	Path Traversal in Profile Picture Retrieval	8
10	Conclusion	8
11	Appendix A	9
11.1	Penetration Testing Methodology	9
11.2	Reconnaissance Findings	10
11.3	Tools Used	10
12	Appendix B (Replication Steps)	11
12.1	SQL Injection in Login and Search Functions	11
12.2	Directory Traversal and Exposed Sensitive Files	11
12.3	Weak Password Hashing and Password Modification via SQL Injection	11
12.4	Insecure Direct Object Reference (IDOR) in Transaction Viewing	12
12.5	Client-Side Security Control Bypass via Cookie Manipulation	12
12.6	Unrestricted File Upload with Remote Code Execution	12
12.7	JWT Authentication Vulnerabilities	13
12.8	Path Traversal in Profile Picture Retrieval	13
13	Appendix C: Risk Analysis Criteria	13
14	Appendix D: Project Clean-up Instructions	14

## **List of Tables**

Table 1. Summary of identified issues.....	3
Table 2. Services identified.....	10
Table 3. Impact factors.....	13
Table 4. Likelihood factors.....	14

## **List of figures**

Fig 1. Risk assessment matrix.....	14
------------------------------------	----

## **Introduction**

**Scope:** Complete security assessment of the F-Co Bank web banking platform, including authentication mechanisms, session management, data access controls, file handling, and API security.

## **Outline**

**Testing Period:** 11th October, 2025 to 16th October, 2025.

Target Application: F-Co Bank Web Banking Platform

- URL: <https://target.lab.local>
- Hosting: lab-target
- IP Address: xxx.xx.xxx.xx
- Testing Methodology: OWASP Web Security Testing Guide v4.2
- Environment Details: - Web Server: nginx - Backend: PHP with jQuery 3.6.1 - Security: HTTPS with TLS 1.3, secure session cookies

## Executive Summary

This penetration test evaluated the security posture of F-Co Bank's web banking application and uncovered nine vulnerabilities ranging from HIGH to EXTREME severity, with three rated as EXTREME risk affecting the customer data confidentiality, account integrity, and overall platform security.

The application suffers from fundamental security flaws across multiple domains. Authentication and authorization controls are severely inadequate, allowing access to customer accounts and sensitive data by unauthorized users. Data validation is missing from the application, giving attackers access to database contents, and allowing them to execute arbitrary commands on the server. The application has access controls in place, but they can be bypassed easily. This may lead to attackers being able to view and modify accounts and transactions that they should not see or have access.

Attackers can use these vulnerabilities to steal credentials, examine any banking account, unlawfully transfer money, execute server code, and access important configuration data with little technical knowledge.

Successful exploitation could result in \$500K to \$5M+ in direct losses, with long-term reputational harm and fines potentially exceeding \$10M. Immediate remediation is strongly recommended before processing live customer data or transactions. While remediation effort will be substantial, it is necessary to prevent financial losses, regulatory fines, and catastrophic reputational damage.

## Summary of identified issues

Table 1. Summary of identified issues

Vulnerability	Description	Affected components	Risk Rating
SQL Injection	Unfiltered user input allows database manipulation	Transaction search	<b>EXTREME</b>
Directory Traversal	Exposed static file directory reveals sensitive files	/static endpoint	<b>HIGH</b>

Weak Password Storage	Inadequate hashing allows credential recovery	User authentication system	<b>VERY HIGH</b>
Insecure Direct Object Reference (IDOR)	Account numbers can be manipulated to access others' data	Transaction viewing, account access	<b>EXTREME</b>
Client-Side Security Control Bypass	Client-Side Security Control Bypass	Transaction viewing restrictions	<b>VERY HIGH</b>
Unrestricted File Upload	Arbitrary code execution via uploaded files	Profile picture upload	<b>VERY HIGH</b>
Missing JWT Signature Verification	Unsigned tokens accepted for authorization	Post-transaction token validation	<b>EXTREME</b>
JWT Secret Weakness	Weak JWT signing key enables token forgery	Magic link authentication	<b>EXTREME</b>
Path Traversal in File Access	Direct file access bypasses authorization checks	Profile picture retrieval endpoint	<b>VERY HIGH</b>

## **Chapter 1: SQL Injection in Login and Search Functions**

### **Threat Description**

The transaction search functionality is vulnerable to SQL injection. User input is directly concatenated into SQL queries without sanitization or parameterization, which allows attackers to inject arbitrary SQL commands. This vulnerability exists in the **overview** page of the website; its presence allows the attacker to extract sensitive data from the database, including user credentials, account information, and transaction records. The attack could result in compromise of the database integrity and confidentiality. Attackers can read, modify, and create data.

### **Risk Level: EXTREME**

**Impact: Extreme** - Complete database compromise, unauthorized access to all customer accounts, ability to modify financial records, potential data exfiltration of all customer information.

**Likelihood: Certain** - SQL injection is trivial to exploit.

(Detailed replication steps: Appendix B.1 | Risk definitions: Appendix C)

### **Recommendations**

1. Conduct comprehensive code review to identify all SQL injection points
2. Implement parameterized queries (prepared statements) for all database interactions
3. Apply input validation and sanitization on all user inputs
4. Implement least-privilege database access (application should not use admin accounts)
5. Deploy web application firewall (WAF) rules to detect SQL injection attempts

6. Implement a secure database abstraction layer

Reference: [OWASP SQL Injection Prevention Cheat Sheet]

([https://cheatsheetseries.owasp.org/cheatsheets/SQL\\_Injection\\_Prevention\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html) )

## **Chapter 2: Directory Traversal and Exposed Sensitive Files**

### **Threat Description**

The application exposes a `/static` directory containing sensitive files including a wordlist file (`wordlist.txt`) and a user database backup file (`users.bqk`). This directory is accessible without authentication by simply modifying the URL path from `/overview.php` to `/static`. The exposed files contain critical security information: the wordlist appears to be used by the password generator feature, while the users backup file contains four user accounts with their associated password. This information disclosure enables offline password cracking attacks.

### **Risk Level: HIGH**

**Impact: Moderate**– Through SQL injection, the attacker can obtain hashed passwords, and the exposed wordlist will reduce password security for users who use the password generator. Through the backup file, the attackers can log in if user has not changed their password.

**Likelihood: Very high** - Directory traversal through URL manipulation requires no special tools and would be discovered quickly by automated scanners or manual testing.

(Detailed replication steps: Appendix B.2 | Risk definitions: Appendix C)

### **Recommendation**

1. Remove the `/static` directory from the web root or implement authentication requirements
2. Delete or relocate the `users.bqk` backup file to a secure location outside the web directory
3. Implement proper access controls on all directories containing sensitive files
4. Store sensitive data outside the web-accessible directory structure
5. Encrypt backup files and store them in secure, access-controlled locations

## **Chapter 3: Weak Password Hashing and Password Modification via SQL Injection**

### **Threat Description**

The wordlist includes common words used to create a password using the password generator. Using the extracted wordlist and hashed passwords, the attacker can crack the hash and access their account, demonstrating inadequate password hashing practices. Additionally, the SQL injection vulnerability can be used to directly modify another user's password hash in the database.

The password modification attack utilized SQL injection to execute an UPDATE statement, replacing a target user's password hash with an attacker-controlled bcrypt hash. This combination of vulnerabilities allows complete account takeover. An attacker can either crack existing passwords offline or forcibly change passwords to known values.

**Risk Level: VERY HIGH**

**Impact: High** - Complete account takeover capability for any user account. Unauthorized access to financial data and ability to perform transactions as the compromised user.

**Likelihood: Very high** - Requires exploitation of previously identified SQL injection, but once access is gained, account takeover is straightforward.

(Detailed replication steps: Appendix B.3 | Risk definitions: Appendix C)

**Recommendation**

1. Deploy multi-factor authentication
2. Review and strengthen password hashing implementation
3. Implement password complexity requirements
4. Remove the "pre-hashed password" feature from useful tools page

**Chapter 4: Insecure Direct Object Reference (IDOR) in Transaction Viewing****Threat Description**

The application's transaction viewing functionality uses a URL parameter (`txn=account\_number`) to determine which account's transactions to display. The application allows users to view only certain accounts through the interface. However, by manipulating the `txn` parameter, an attacker can access hidden accounts belonging to the same user that are not displayed in the account selector, bypassing intended access restrictions. This `txn` parameter can be manipulated to view transactions for any account number of that user in spite of having a `vis\_account` cookie. By leveraging the SQL injection vulnerability to enumerate account numbers from the accounts table, an attacker identified a hidden account number. Modifying the URL parameter to this account number immediately displayed that account's transaction history without any authorization check.

This vulnerability enables unauthorized access to sensitive financial information for any customer account in the system.

**Risk Level: EXTREME**

**Impact: Extreme** - Unauthorized access to all customer financial transaction data. Privacy violation, potential for fraud, and regulatory compliance violations.

**Likelihood: Certain** - Simple URL parameter manipulation. No special tools required.

(Detailed replication steps: Appendix B.4 | Risk definitions: Appendix C)

**Recommendation**

1. Implement server-side authorization checks that verify the authenticated user owns the requested account
2. Implement comprehensive authorization framework across all application endpoints

3. Use session-stored account associations rather than URL parameters where possible
4. Implement least-privilege access controls

Reference: [OWASP Authorization Cheat Sheet]

([https://cheatsheetseries.owasp.org/cheatsheets/Authorization\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html) )

## **Chapter 5: Client-Side Security Control Bypass via Cookie Manipulation**

### **Threat Description**

The application implements an additional layer of protection for transaction viewing through a `vis\_account` cookie. However, this security control is implemented on the client side and can be trivially bypassed by manipulating the cookie value. The cookie format was determined to be: `account\_no1,account\_no2:md5\_hash`, which is then Base64 encoded. By reverse engineering this format, an attacker can generate valid cookies for any account number combination, effectively bypassing the intended access restrictions.

This demonstrates a fundamental security anti-pattern which is relying on client-side controls for authorization decisions. All client-side data should be considered untrusted and subject to manipulation.

Risk Level: **VERY HIGH**

**Impact: High** - Bypasses intended access controls, enabling unauthorized account access when combined with IDOR vulnerability.

**Likelihood: Very high** - Cookie manipulation is straightforward using browser developer tools or proxy tools like Burp Suite.

(Detailed replication steps: Appendix B.5 | Risk definitions: Appendix C)

### **Recommendation**

1. Remove client-side authorization controls and ensure all authorization checks on the server side
2. Session data should be stored server-side and referenced by session ID only

Reference: [OWASP Session management Cheat Sheet]

([https://cheatsheetseries.owasp.org/cheatsheets/Session\\_Management\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html) )

## **Chapter 6: Unrestricted File Upload with Remote Code Execution**

### **Threat Description**

The profile picture upload functionality allows for an unrestricted file upload vulnerability that allows arbitrary file execution on the server. While the application appears to validate file extensions client-side, this check can be bypassed by intercepting and modifying the upload request.

An attacker can upload a PHP shell file, with the extension `shell.php.png` then using burp suite intercept the request and modify the filename to `shell.php` before sending the request to the

server. Once uploaded, the PHP file can be accessed directly via the web server, resulting in remote code execution.

This vulnerability represents a complete compromise of the web server and potentially the entire backend infrastructure.

**Risk Level: VERY HIGH**

**Impact: Extreme** - Complete server compromise. Attacker can execute arbitrary commands with web server privileges, access database credentials, pivot to internal networks, install persistent backdoors, and exfiltrate all data.

**Likelihood: High** - Requires use of proxy tool like Burp Suite but technique is well-documented and commonly exploited. Automated tools can identify and exploit this vulnerability.

(Detailed replication steps: Appendix B.6 | Risk definitions: Appendix C)

**Recommendation**

1. Implement server-side file type validation using MIME type checking (not file extensions)
2. Store uploaded files outside the web root directory
3. Rename uploaded files to random names, storing original name in database
4. Scan uploaded files with antivirus/malware detection
5. Disable script execution in upload directories via web server configuration
6. Implement Content Security Policy headers

Reference: [OWASP File Upload Cheat Sheet]

([https://cheatsheetseries.owasp.org/cheatsheets/File\\_Upload\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/File_Upload_Cheat_Sheet.html) )

**Chapter 7: JWT Authentication Vulnerabilities**

**Threat Description**

Two critical vulnerabilities exist in the application's JSON Web Token (JWT) implementation. First, the "magic link" authentication feature uses JWTs signed with a weak secret key that was cracked using the exposed wordlist and John the Ripper. Second, post-transaction JWTs lack signature verification entirely, allowing complete token forgery.

The magic link generator creates JWT tokens for user authentication. The signing key was successfully cracked offline, enabling attackers to forge valid tokens for any user, including privileged accounts (UID 0). After funds transfers, the application generates unsigned JWTs, which are accepted without validation, allowing attackers to forge transaction records and potentially bypass transaction logging or auditing.

These vulnerabilities enable complete authentication bypass and transaction record manipulation.

**Risk Level: EXTREME**

**Impact: Extreme** - Complete authentication bypass and ability to impersonate any user. Transaction record manipulation could enable fraud and compromise audit trails.

**Likelihood: Certain** - Weak JWT secrets can be cracked offline quickly. Missing signature verification is a fundamental implementation error that accepts any forged token.

(Detailed replication steps: Appendix B.7 | Risk definitions: Appendix C)

### **Recommendation**

1. Regenerate JWT signing keys using cryptographically secure random values (minimum 256 bits)
2. Implement mandatory signature verification for all JWT token
3. Use industry-standard JWT libraries that implement proper validation
4. Store JWT secrets in secure secret management systems
5. Regular key rotation procedures
6. Consider using asymmetric cryptography (RS256) instead of symmetric (HS256)

Reference: [OWASP JWT Cheat Sheet]

([https://cheatsheetseries.owasp.org/cheatsheets/JSON\\_Web\\_Token\\_for\\_Java\\_Cheat\\_Sheet.html](https://cheatsheetseries.owasp.org/cheatsheets/JSON_Web_Token_for_Java_Cheat_Sheet.html))

-

## **Chapter 8: Path Traversal in Profile Picture Retrieval**

### **Threat Description**

The profile picture retrieval endpoint accepts a file path parameter that is not properly validated and can be manipulated by the attacker. By manipulating this parameter with directory traversal sequences (../), an attacker can access arbitrary files on the server filesystem, including files that would normally return a 403 Forbidden error when accessed directly.

The vulnerability was exploited to access `secrets.env`, a configuration file likely containing database credentials, API keys, and other sensitive environment variables.

### **Risk Level: VERY HIGH**

**Impact: High** - Access to sensitive files containing database credentials, API keys, encryption keys, and other secrets.

**Likelihood: Very High** - Path traversal is straightforward to exploit using basic URL manipulation. The attack requires no specialized tools.

(Detailed replication steps: Appendix B.8 | Risk definitions: Appendix C)

### **Recommendations**

1. Move sensitive configuration files outside the web root directory
2. Use indirect object references (map user input to server-side lookup table)

3. Implement strict input validation: whitelist allowed filenames only

Reference: [OWASP Path traversal Cheat Sheet]

([https://owasp.org/www-community/attacks/Path\\_Traversal](https://owasp.org/www-community/attacks/Path_Traversal) )

## **Conclusion**

The penetration test identified nine vulnerabilities ranging from HIGH to EXTREME severity, with three rated as EXTREME risk vulnerabilities in the F-Co Bank web application. The findings indicate systemic security weaknesses with authentication, authorization, input validation, and secure coding practices, with serious cumulative risk exposure that requires urgent remediation.

The application should not be moved into production until all critical vulnerabilities are remediated and re-tested. The vulnerabilities we identified have the potential for a moderately skilled attacker to completely compromise the application, access all customer data, tamper with financial records, and take control of backend server infrastructure.

A comprehensive security remediation program is warranted with secure development training for developers, creation of secure coding standards, inclusion of automated security testing in the development pipeline, and regular security assessments.

## **Appendix**

### **Appendix A Penetration Testing Methodology**

This penetration test followed the OWASP Web Security Testing Guide v4.2 methodology. The testing approach consisted of the following phases:

#### **1. Information Gathering**

- Application mapping and enumeration
- Technology stack identification
- User role and privilege analysis

#### **1. Vulnerability Discovery**

- Manual testing of authentication mechanisms
- Input validation testing across all forms and parameters
- Authorization testing for access control flaws
- File upload security testing
- Session management analysis
- API security assessment

#### **1. Exploitation**

- Proof-of-concept development for identified vulnerabilities

- Privilege escalation testing
- Data exfiltration capability assessment

## 1. Post-Exploitation

- Impact analysis
- Lateral movement possibilities
- Persistence mechanism identification

Reference to OWASP methodology ( [https://owasp.org/www-project-web-security-testing-guide/v42/4-Web\\_Application\\_Security\\_Testing/](https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security_Testing/) )

## A.2 Reconnaissance Findings

### About host:

**Hosting Provider:** lab-target

**Location:** Redacted

**Specific Address:** Redacted

**IP Address:** xxx.xx.xxx.xx

**Hostname:** Target lab

**Geographic Coordinates:** xxxxx,xxxxxx

### Web Technologies:

#### Server Software:

- Web Server: nginx
- Backend Language: PHP
- JavaScript Library: jQuery 3.6.1

### Security Technologies:

- HTTPS with TLS 1.3
- Secure session cookies (PHPSESSID)
- Cookie security flags: HttpOnly, Secure, SameSite=Strict

**OS: Linux** (Likely Debian 12 (Bookworm) or Ubuntu 22.10+ as OpenSSH 9.0 was found (released April 2022, commonly found on Debian 12/Ubuntu 22.10+))

### Services Identified:

Table 2. Services identified during testing

Service	Software	Version	Port
Web Server	nginx	Version not disclosed	80, 443

SSH	OpenSSH	9.0 (protocol 2.0)	22
Backend	PHP	Version not disclosed	N/A
JavaScript	jQuery	3.6.1	N/A
SSH	OpenSSH	9.0 (protocol 2.0)	22

### A.3 Tools Used

This penetration test utilized industry-standard security assessment tools and methodologies. Wappalyzer was employed for technology stack fingerprinting and web application profiling. Nmap performed port scanning and service enumeration, while OpenSSL verified SSL/TLS configurations. Whois and cURL supported reconnaissance activities and HTTP request analysis. Burp Suite served as the primary web application security testing platform for request interception, manipulation, and vulnerability exploitation. Password hash analysis utilized John the Ripper and Hashcat for credential recovery testing. All testing was conducted from a Kali Linux environment, providing a comprehensive security testing toolkit. Manual testing complemented automated tools throughout the assessment to ensure thorough coverage and accurate vulnerability identification.

## **Appendix B: Replication Steps [Sanitized payloads]**

### **B.1 SQL Injection in Login and Search Functions**

This vulnerability can be exploited through the transaction search functionality. The following steps demonstrate both data extraction capabilities.

After logging in as a user using the transaction search field, inject a payload:

```
'[INJECTION_PAYLOAD]'  
UNION SELECT [REDACTED_COLUMNS]
```

This payload allows the attacker to look into all table present in the database. Once the table names are known. The attacker can extract user account data:

1. Modify the UNION query to target the users or accounts table using:

```
'[INJECTION_PAYLOAD]'  
UNION SELECT [REDACTED_COLUMNS] FROM [REDACTED_TABLE]
```

1. Extract usernames, password hashes, and account numbers

### **B.2 Directory Traversal and Exposed Sensitive Files**

The application exposes a /static directory containing sensitive files without proper access controls. These steps demonstrate how to access the directory and retrieve the wordlist and user database backup files.

1. An authenticated session was established
2. Application endpoints were manipulated to test access controls
3. Improper authorization checks allowed access to restricted directories

4. Sensitive backup files were exposed
5. Retrieved data could be leveraged for further compromise

### **B.3 Weak Password Hashing and Password Modification via SQL Injection**

This demonstrates two attack vectors: cracking password hashes using the exposed wordlist, and directly modifying user passwords through SQL injection. Both methods result in complete account takeover.

#### **Password Cracking:**

1. Sensitive credential material was exposed through improperly protected backup files.
2. An attacker could retrieve password hashes and perform offline attacks using commonly available techniques, leading to credential compromise and unauthorized access.

#### **Password Modification via SQL Injection:**

1. The application's password management functionality was vulnerable to SQL injection due to insufficient input validation and improper query construction.
2. This vulnerability allowed an attacker to modify credential data within the user database, resulting in full account takeover without prior knowledge of the original password.

### **B.4 Insecure Direct Object Reference (IDOR) in Transaction Viewing**

This vulnerability allows unauthorized access to any customer's transaction history by manipulating the account number URL parameter. No server-side authorization checks validate account ownership.

1. An authenticated user was able to access transaction data belonging to other accounts due to improper authorization checks on object references.
2. The application relied on user-supplied identifiers to retrieve transaction records without validating ownership. By manipulating these identifiers, an attacker could view sensitive financial information associated with accounts they were not authorized to access.

### **B.5 Client-Side Security Control Bypass via Cookie Manipulation**

The application implements transaction viewing restrictions through a client-side cookie `vis_account` that can be reverse-engineered and forged. These steps demonstrate how to decode, modify, and re-encode the cookie to bypass access controls.

1. The application relied on client-side state to determine account context without sufficient integrity protection or server-side validation.
2. An authenticated user was able to manipulate client-controlled data to impersonate other account contexts. Due to the use of weak cryptographic mechanisms and lack of proper verification, the server accepted modified client state and returned transaction data belonging to unauthorized accounts.

## **B.6 Unrestricted File Upload with Remote Code Execution**

The profile picture upload functionality accepts malicious PHP files that can be executed on the server. This demonstrates uploading a web shell by bypassing client-side validation using request interception.

1. The application's file upload functionality failed to properly validate uploaded content and enforce server-side file type restrictions.
2. An authenticated user was able to upload a malicious file disguised as an allowed file type. Due to insufficient validation and improper handling of uploaded files, the server accepted and stored executable content in a web-accessible directory, resulting in remote code execution.

## **B.7 JWT Authentication Vulnerabilities**

Two JWT flaws are demonstrated: cracking the weak signing secret used in magic link tokens, and exploiting missing signature verification in post-transaction tokens. Both enable authentication bypass and token forgery.

### **Magic Link JWT Forgery:**

1. JWTs issued after sensitive operations were not properly validated by the server. The application accepted modified tokens without verifying their integrity, allowing client-side manipulation of transaction-related data.
2. This flaw enabled unauthorized alteration of account context and transaction details, undermining trust in transactional integrity.

## **B.8 Path Traversal in Profile Picture Retrieval**

The profile picture endpoint accepts unvalidated file path parameters, allowing directory traversal to access sensitive files outside the intended directory. This demonstrates accessing the secrets.env configuration file containing credentials and API keys.

### **Replication Steps**

1. The application failed to properly validate user-supplied file path parameters when serving profile images.
2. By manipulating these parameters, an authenticated user could traverse the server's directory structure and access sensitive files outside the intended scope. This resulted in disclosure of configuration and environment data that should not be accessible via the application.

## **Appendix C: Risk Analysis Criteria**

Risk factors were assessed using the ISO 27001 framework, combining Impact and Likelihood to determine overall Risk Level.

## Impact Factors

Table 3. Table describing impacts used in the report

Level	Description
Extreme	Complete system compromise, massive financial loss, severe regulatory penalties, major reputational damage
Very High	Significant data breach, substantial financial loss, regulatory action likely, serious reputational harm
High	Considerable data exposure, notable financial impact, possible regulatory scrutiny, reputational damage
Moderate	Limited data exposure, moderate financial impact, some reputational concern
Low	Minor data disclosure, minimal financial impact, negligible reputational effect
Very Low	Trivial data exposure, insignificant financial impact, no reputational concern
Negligible	Insignificant impact across all dimensions

## Likelihood Factors

Table 4. Table describing impacts used in the likelihood

Level	Justification
Certain	Significant exploitation history, trivial to exploit, will definitely occur
Very High	Known exploitation patterns, highly prevalent, very easy to exploit, almost certain to occur
High	Some exploitation history and/or considered likely in the circumstances
Medium	Possible exploitation, requires some skill, may occur
Low	No known exploitation, requires significant skill, possible but unlikely
Very Low	No history, difficult to exploit, unlikely to occur
Negligible	No feasible exploitation path, extremely unlikely to ever occur

Fig 1. Risk matrix

## **Appendix D: Project Clean-up Instructions**

All testing was conducted in an isolated, resettable lab environment.  
No production systems were affected.