# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/24/2025** | **Andrew Chacon** | **Vulnerability Assessment** |

## Client



## Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In this report, identify your security vulnerability findings and recommend the next steps to remedy the issues you have found.

* Respond to the five steps outlined below and include your findings.
* Respond using your own words. You may also include images or supporting materials. If you include them, make certain to insert them in the relevant locations in the document.
* Refer to the Project One Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Andrew Chacon

**1. Interpreting Client Needs**

 Artemis Financial is a consulting company that develops individualized financial plans for its customers. The financial plans include savings, retirement, investments, and insurance. Due to that there are various assumptions we can make for their security requirements:

**Need Secure Communications:** Secure communications are important for a financial company since they handle account numbers, balances, and customer information.

**International Transactions:** They very likely process international transactions and must comply with different regulations.

**Governmental Restrictions:** The company will need to comply with financial regulations including SOX compliance, FFIEC cybersecurity guidelines, and other banking regulations when it comes to security.

**External Threats:** Current threats include SQL injection attacks, cross-site scripting, unauthorized file uploads, and data breaches through exposed endpoints without authentication.

**Modernization Requirements:** The application uses outdated libraries (Spring Boot 2.2.4, BouncyCastle 1.46) and lacks up to date security frameworks.

**2. Areas of Security**

**Architecture Review** - The application doesn’t have even basic security controls inclduing authentication or authorization frameworks.

**Input Validation** - All user inputs are processed without validation, this leads to injection attacks.

**APIs** - REST endpoints have no security controls.

**Cryptography** - Database credentials are stored in plain text, and the cryptographic library is outdated.

**Client/Server** - File upload allows uploads up to 215MB without them needing to be validated.

**Code Quality** – There are plenty of security violations including hardcoded credentials poorly designed access controls.

**3. Manual Review**

**Vulnerability 1: Hardcoded Database Credentials (DocData.java, lines 19-20)** Database username and password "root"/"root" are hardcoded in the source code, this would cause an attacker to get total access if discovered.

**Vulnerability 2: Missing Authentication (CRUDController.java)** The /read endpoint has no authentication, this would allow unrestricted access to sensitive data.

**Vulnerability 3: Cross-Site Scripting (GreetingController.java, line 13)** User input is directly included in responses which could lead to XSS attacks.

**Vulnerability 4: Improper Access Controls (customer.java, line 5)** The account\_balance field uses package-private access instead of just using private.

**Vulnerability 5: Unrestricted File Uploads (application.properties)** File uploads allow 200MB files with no type restrictions or validation, this could lead to malicious file uploads.

**Vulnerability 6: Format String Injection (GreetingController.java, line 13)** User input is directly inserted into String.format which could allow format string attacks.

**Vulnerability 7: Improper HTTP Methods (CRUDController.java, line 8)** Using @RequestMapping without specifying HTTP method allows all request types which is risky.

**4. Static Testing**

Running the OWASP dependency-check plugin showed us **175 vulnerabilities** across **16 vulnerable dependencies**:

**Critical Vulnerabilities (CVSS 9.8):**

* CVE-2022-1471: SnakeYAML remote code execution via deserialization
* CVE-2022-22965: Spring Framework "Spring4Shell" remote code execution
* CVE-2020-1938: Apache Tomcat "Ghostcat" AJP connector vulnerability

**High Severity Vulnerabilities:**

* CVE-2024-34447: BouncyCastle certificate validation bypass (CVSS 7.7)
* CVE-2020-25649: Jackson Databind XXE vulnerability (CVSS 7.5)
* CVE-2017-18640: SnakeYAML entity expansion DoS (CVSS 7.5)

**Component Summary:**

* **BouncyCastle 1.46**: 18 vulnerabilities (library from 2011)
* **Spring Framework 5.2.3**: Multiple critical RCE vulnerabilities across all components
* **Apache Tomcat 9.0.30**: 32+ vulnerabilities including known exploited CVEs
* **SnakeYAML 1.25**: 8 vulnerabilities including critical RCE
* **Jackson Databind 2.10.2**: 6 vulnerabilities enabling various attacks

**Java 1.8 End-of-Life:** Using Java 8 which doesn’t have modern security features and current security patches.

**5. Mitigation Plan**

**Critical Priority:**

1. Remove hardcoded database credentials and use environment variables
2. Implement Spring Security with authentication functions.
3. Update all dependencies: Spring Boot to 2.7.x+, BouncyCastle to 1.70+, Java to 11+

**High Priority:** 4. Restrict file upload types and implement size limits with validation 5. Fix access control issues by making sensitive fields private

**Medium Priority:** 6. Add proper error handling that won’t expose stack traces 7. Add HTTP method restrictions to all endpoints 8. Include output encoding to prevent XSS attacks