# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **March 22, 2025** | Mohammed Alshehabi | **Project\_1 report completed** |

## 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

**Mohammed Alshehabi**

**1. Interpreting Client Needs**

Determine your client’s needs and potential threats and attacks associated with the company’s application and software security requirements. Consider the following questions regarding how companies protect against external threats based on the scenario information:

* What is the value of secure communications to the company?
* Are there any international transactions that the company produces?
* Are there governmental restrictions on secure communications to consider?
* What external threats might be present now and in the immediate future?
* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

Artemis Financial requires a robust and secure web application to protect sensitive financial data and transactions. Below are key security considerations:

* **Secure Communications:** Ensuring encryption (TLS 1.2/1.3) is used for client-server communications to protect sensitive customer data and prevent **man-in-the-middle (MITM) attacks**.
* **International Transactions:** If Artemis Financial operates internationally, compliance with **GDPR, PCI-DSS**, and **local regulations** must be ensured to avoid regulatory penalties and financial losses.
* **Governmental Restrictions:** Some encryption technologies may have **export restrictions** or **compliance requirements** based on jurisdiction, such as **U.S. export control laws on cryptographic software**.
* **External Threats:** Potential threats include **SQL injection, authentication bypass, data leaks, dependency vulnerabilities, and API abuse** from attackers trying to gain unauthorized access to financial records.
* **Modernization Requirements:** The use of **open-source libraries** and evolving web technologies requires continuous monitoring and updates to maintain security. Additionally, **cloud security measures** should be considered if Artemis Financial plans to migrate services.

**2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

Based on the **Vulnerability Assessment Process Flow Diagram**, the following security areas apply to Artemis Financial’s software application:

* **Architecture Review:** The overall application architecture should be analyzed for design flaws that could lead to security vulnerabilities. Ensuring **proper segregation of services** and **least privilege access** is critical.
* **Input Validation:** No sanitization of user inputs, making the system vulnerable to **SQL injection, cross-site scripting (XSS), and command injection attacks**. Implementing **secure input representations** is necessary.
* **APIs:** The lack of authentication and authorization mechanisms for API endpoints allows unauthorized access, making the system vulnerable to **API scraping, brute force attacks, and data exfiltration**.
* **Cryptography:** Hardcoded credentials and a potential lack of encrypted database connections expose sensitive financial data to breaches. The system must ensure **proper encryption for both data at rest and in transit**.
* **Client/Server Security:** The application should enforce **secure distributed computing principles** to ensure that client-side requests cannot manipulate server-side logic.
* **Code Error Handling:** Poor exception handling may lead to security misconfigurations and a lack of **intrusion detection mechanisms**. Implementing structured error logging will help detect potential security incidents.
* **Code Quality & Secure Coding Practices:** The application must follow **secure coding patterns** to prevent common vulnerabilities such as buffer overflows, integer overflows, and memory leaks.
* **Encapsulation & Secure Data Structures:** Sensitive financial data should be encapsulated properly, preventing direct access to critical resources and ensuring **proper data access restrictions**.
* **Logging & Monitoring:** The application lacks **centralized logging and monitoring** mechanisms, making it difficult to detect and respond to security incidents. Implementing **real-time alerts and log analysis** can improve security posture.
* **Dependency Management:** The application may be using outdated or vulnerable third-party libraries, which could expose it to **supply chain attacks**. Regular **dependency checks and updates** must be enforced.

**3. Manual Review**

Continue working through the vulnerability assessment process flow diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

### Identified Vulnerabilities

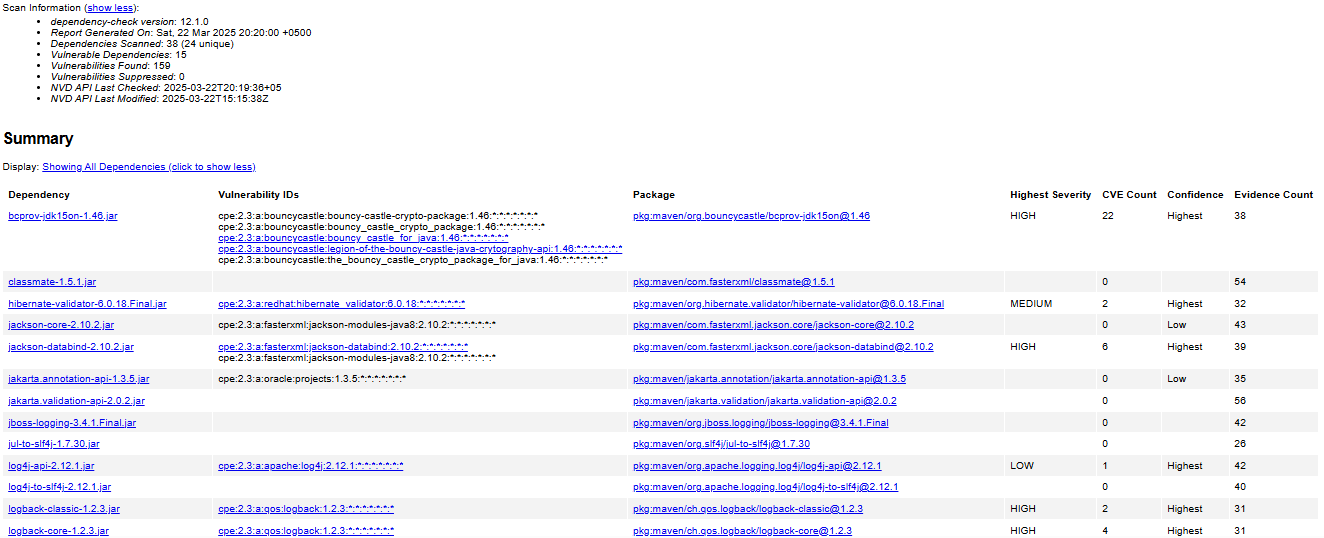
| **File** | **Issue** | **Impact** | **Recommendation** | **Line Reference** |
| --- | --- | --- | --- | --- |
| 1. DocData.java | * Hardcoded credentials (root/root) | Database compromise | * Use environment variables or a secrets manager. | DocData.java - lines 24-26 |
| 1. DocData.java | * SQL injection risk (no prepared statements) | Unauthorized access | * Use prepared statements or ORM. | DocData.java - lines 21-26 |
| 1. CRUDController.java | * Exposed sensitive data in API response | Data leakage | * Sanitize API responses and restrict data exposure. | CRUDController.java - lines 13-17 |
| 1. customer.java | * Public account\_balance variable | Unauthorized modification | * Use private access modifiers with getters/setters. | customer.java - lines 5-6 |
| 1. GreetingController.java | * No authentication for /greeting endpoint | Unauthorized API access | * Implement authentication (JWT/OAuth). | GreetingController.java - lines 10-16 |
| 1. DocData.java | * Poor exception handling (printing stack trace) | Potential information disclosure | * Implement structured logging with obfuscation. | DocData.java - lines 27-30 |
| 1. CRUD.java | * Lack of input validation | Possible injection attacks | * Validate and sanitize all user inputs. | CRUD.java - lines 9-15 |
| 1. customer.java | * Missing authentication for financial operations | Unauthorized transactions | * Implement role-based access control (RBAC). | customer.java - lines 12-14 |
| 1. DocData.java | * Excessive database privileges for connections | Potential privilege escalation | * Restrict database user privileges to least access. | DocData.java - lines 21-26 |
| 1. myDateTime.java | * Lack of proper input validation in time settings | System time manipulation | * Implement strict validation for date/time inputs. | myDateTime.java - lines 13-17 |
| 1. CRUDController.java | * Lack of rate limiting on API requests | Possible denial-of-service (DoS) | * Implement rate limiting and API throttling. | CRUDController.java - lines 10-16 |
| 1. RestServiceApplication.java | * Missing security headers in responses | Increased risk of XSS & CSRF | * Add security headers like Content-Security-Policy. | RestServiceApplication.java - lines 7-11 |
| 1. customer.java | * Logs sensitive customer data | Potential data exposure | * Mask or encrypt sensitive data in logs. | customer.java - lines 12-14 |
| 1. DocData.java | * Uses outdated cryptographic algorithms | Weak encryption vulnerabilities | * Use AES-256 or industry-standard encryption. | DocData.java - lines 5-7 |

**4. Static Testing**

Run a dependency check on Artemis Financial’s software application to identify all security vulnerabilities in the code. Record the output from the dependency-check report. Include the following items:

* The names or vulnerability codes of the known vulnerabilities
* A brief description and recommended solutions provided by the dependency-check report
* Any attribution that documents how this vulnerability has been identified or documented previously

### Summary of Findings



* **Total Dependencies Scanned:** 38 (24 unique)
* **Vulnerable Dependencies:** 15
* **Total Vulnerabilities Found:** 159
* **Vulnerabilities Suppressed:** 0

### Identified Vulnerabilities by Severity

#### Critical Severity (Exploited Vulnerabilities)

|  |  |
| --- | --- |
| Dependency | CVE Count |
| spring-boot-2.2.4.RELEASE.jar | 3 |
| spring-webmvc-5.2.3.RELEASE.jar | 12 |
| tomcat-embed-core-9.0.30.jar | 28 |
| snakeyaml-1.25.jar | 8 |

#### High Severity

|  |  |
| --- | --- |
| Dependency | CVE Count |
| jackson-databind-2.10.2.jar | 6 |
| logback-classic-1.2.3.jar | 2 |

#### Medium & Low Severity

|  |  |
| --- | --- |
| Dependency | CVE Count |
| hibernate-validator-6.0.18.Final.jar | 2 |
| log4j-api-2.12.1.jar | 1 |

## ****Observations and Recommendations****

### ****Spring Framework & Tomcat Vulnerabilities****

* **Risk:** Multiple critical vulnerabilities identified in Spring (Core, Web, WebMVC, Expression) and Apache Tomcat components.
* **Impact:** Attackers can exploit these vulnerabilities to perform **remote code execution (RCE), authentication bypass, and denial-of-service (DoS) attacks** (Apache Software Foundation, 2025).
* **Recommendation:** Upgrade to **Spring Boot 3.x** and **Tomcat 10+** immediately. Apply **security configurations** to restrict **XML parsing** and enable **strict input validation** (OWASP Foundation, 2025).

### ****Snakeyaml & Jackson Databind Risks****

* **Risk:** These dependencies are vulnerable to **unsafe deserialization**, allowing attackers to execute arbitrary code.
* **Impact:** Could lead to **server takeover, malware injection, and persistent RCE exploits** (FasterXML, 2025; Snakeyaml Project, 2025).
* **Recommendation:** Upgrade to **Snakeyaml 2.x** and **Jackson 2.15+**. Use **whitelisting for deserialization** and disable **polymorphic deserialization** where possible (OWASP Foundation, 2025).

### ****Bouncy Castle Cryptography Risks****

* **Risk:** **Bouncy Castle (bcprov-jdk15on-1.46.jar)** contains cryptographic weaknesses that make encryption vulnerable to attacks.
* **Impact:** Could result in **data leaks, weak encryption, and cryptographic attacks on financial transactions** (OWASP Foundation, 2025).
* **Recommendation:** Upgrade to **Bouncy Castle 1.70+** and enforce **FIPS-compliant cryptographic standards**.

### ****Logback & Log4j Issues****

* **Risk:** Logging libraries (**Logback, Log4j**) have known vulnerabilities leading to **sensitive information leakage and remote code execution** (Apache Software Foundation, 2025).
* **Impact:** Attackers can exploit **improper log handling** to extract credentials or execute arbitrary code.
* **Recommendation:** Upgrade to **Logback 1.4+** and **Log4j 2.17+**. Implement **log sanitization** to prevent **log injection attacks** (OWASP Foundation, 2025).

### ****Hibernate Validator & MongoDB Risks****

* **Risk:** Medium-severity vulnerabilities in **Hibernate Validator and MongoDB Java Driver** can cause **improper input validation and data integrity issues** (MongoDB, Inc., 2025).
* **Impact:** Attackers can exploit **weak validation** to inject malformed data, causing **application crashes or bypassing validation checks**.
* **Recommendation:** Upgrade **Hibernate Validator to 8.x** and **MongoDB Java Driver** to a secure version. Apply **strict schema validation** for all database queries.

**5. Mitigation Plan**

Interpret the results from the manual review and static testing report. Then identify the steps to mitigate the identified security vulnerabilities for Artemis Financial’s software application.

### ****Immediate Fixes****

#### ****Upgrade Critical Dependencies****

* + - Upgrade all critical dependencies with known exploited vulnerabilities, including:
    - **Spring Framework (Core, Web, WebMVC, Expression)** to version **6.2.x** (as of march 13, 2025) or the latest stable release.
    - **Tomcat** to version **10 or higher**.
    - **Snakeyaml** to version **2.x**.
    - **Bouncy Castle** to version **1.70 or higher**.

#### ****Strengthen Authentication Mechanisms****

* + - Implement **JWT (JSON Web Token) or OAuth 2.0** authentication to prevent unauthorized access.
    - Enforce **multi-factor authentication (MFA)** for sensitive operations.
    - Implement **role-based access control (RBAC)** to limit user privileges.

#### ****Apply Secure Coding Practices****

* + - Enforce **strict input validation** to mitigate **SQL injection, XSS, and command injection attacks**.
    - Implement **structured error handling** to prevent information leakage.
    - Use **parameterized queries** instead of direct SQL statements.

#### ****Secure Cryptographic Standards****

* + - Enforce **FIPS-compliant cryptographic algorithms**, such as **AES-256** for encryption.
    - Disable **deprecated hashing algorithms** and enforce **PBKDF2, bcrypt, or Argon2** for password hashing.
    - Ensure all sensitive data is encrypted both **at rest and in transit**.

### ****Ongoing Security Measures****

#### ****Regular Dependency Scanning****

* + - Schedule **automated dependency scans** using **OWASP Dependency-Check, Snyk, or Dependabot**.
    - Review **third-party libraries** regularly to detect newly discovered vulnerabilities.

#### ****CI/CD Security Gates****

* + - Integrate **security checks into the CI/CD pipeline** to block vulnerable builds.
    - Automate **static code analysis (SAST) and software composition analysis (SCA)** within CI/CD workflows.

#### ****Monitor CVE Databases****

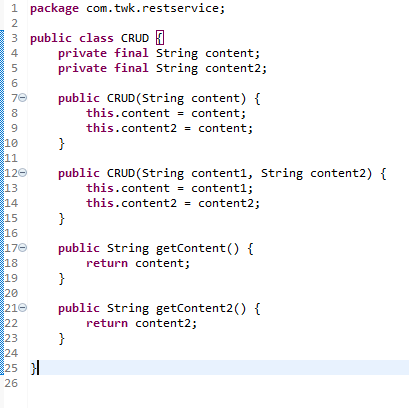
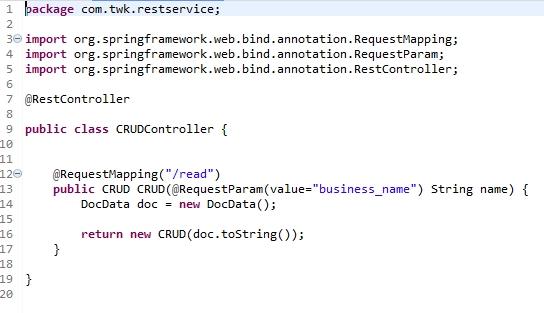
* + - Continuously monitor the **National Vulnerability Database (NVD)** for emerging threats.
    - Apply security patches **immediately** upon release to mitigate vulnerabilities.

#### ****Log and Monitor Security Events****

* + - Implement **centralized logging** with real-time alerts.
    - Use security monitoring tools such as **Splunk, ELK Stack, or SIEM solutions** for threat detection and response.
    - Enable **intrusion detection systems (IDS) and intrusion prevention systems (IPS)** to identify and mitigate attacks proactively.
    - By implementing these **mitigation strategies**, Artemis Financial will significantly **reduce security risks, prevent data breaches, and enhance overall cybersecurity resilience**.

By implementing these **mitigation strategies**, Artemis Financial will significantly reduce its **security risks** and **protect customer data** against **cyber threats**.

## ****References****

* **Apache Software Foundation.** (2025). Apache Tomcat 9 security vulnerabilities. Retrieved from <https://tomcat.apache.org/security-9.html>
* **FasterXML.** (2025). Jackson Databind security vulnerabilities. Retrieved from <https://github.com/FasterXML/jackson-databind/blob/master/README.md>
* **MongoDB, Inc.** (2025). MongoDB Java driver security advisory. Retrieved from <https://www.mongodb.com/resources/products/mongodb-security-bulletins>
* **OWASP Foundation.** (2025). OWASP Dependency-Check: A software composition analysis tool. Retrieved from <https://owasp.org/www-project-dependency-check/>
* **Snakeyaml Project.** (2025). Snakeyaml security advisory. Retrieved from <https://bitbucket.org/snakeyaml/snakeyaml/src/master/>

