# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **09/23/2025** | **Mehdi Salhi** |  |

## 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

Mehdi Salhi

**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:

* Secure communications are essential for Artemis Financial to protect sensitive financial and personal data, requiring TLS 1.2+ and modern encryption to ensure confidentiality, integrity, and client trust.
* While Artemis may not currently conduct international transactions, future operations could involve global clients, requiring compliance with international data protection laws such as GDPR.
* Governmental standards (e.g., NIST, PCI DSS) mandate the use of approved cryptographic methods and prohibit outdated algorithms, ensuring the company meets legal and compliance obligations.
* The company faces ongoing external threats such as outdated dependency vulnerabilities, injection attacks, authentication weaknesses, and API abuse, as well as evolving risks like ransomware and phishing.
* Modernization requirements include maintaining and updating open-source libraries, adopting secure and evolving web technologies, and integrating continuous security testing into the development lifecycle.

**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.

* **Input Validation:** All user input to the REST API must be validated and sanitized to prevent injection and deserialization attacks that could expose sensitive financial data.
* **APIs / Secure API Interactions:** Strong authentication, authorization, rate limiting, and strict CORS rules are required to protect Artemis Financial’s APIs from abuse and unauthorized access.
* **Cryptography:** Secure communication and data protection require TLS 1.2+ for encryption in transit, strong hashing algorithms (BCrypt/Argon2) for passwords, and modern ciphers for data at rest.
* **Secure Error Handling:** Error responses should be generic to users while detailed logs are captured internally, preventing attackers from learning about system internals through verbose error messages.
* **Secure Coding Practices:** Developers must avoid hard-coded secrets, prevent sensitive data logging, and follow OWASP secure coding practices to maintain code quality and reduce vulnerabilities.
* **Data Access:** Parameterized queries and least-privilege database roles must be enforced to prevent SQL injection and restrict unauthorized access to financial records.

**3. Manual Review**

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

1. **Input Validation (SQL Injection Risk)**
   1. **File:** DocData.java:26–27
   2. **Description:** Hard-coded database connection string with potential for SQL injection if the read\_document method were to use user input in SQL queries. The parameters key and value are not validated or sanitized.
2. **Input Validation (Cross-Site Scripting (XSS))**
   1. **File:** GreetingController.java:17
   2. **Description:** User input from the name parameter is directly inserted into the response without validation or encoding using String.format(template, name). Malicious scripts could be executed in client browsers.
3. **Error Handling (Poor Exception Management)**
   1. **File:** DocData.java:28–31
   2. **Description:** SQLException is caught but only prints the stack trace (e.printStackTrace()), potentially exposing sensitive database information to logs or console output.
4. **Cryptography (Hard-coded Database Credentials)**
   1. **File:** DocData.java:26–27
   2. **Description:** Database credentials ("root","root") are hard-coded in plain text, violating secure credential management practices.
5. **Encapsulation (Data Exposure)**
   1. **File:** customer.java:5
   2. **Description:** The account\_balance field has package-private visibility (no access modifier), allowing unauthorized access from other classes in the same package.
6. **Input Validation (Inadequate Deposit Checks)**
   1. **File:** customer.java:12–14
   2. **Description:** The deposit method accepts any integer value without validation, allowing negative deposits or extremely large values that could cause application errors.
7. **Code Quality (Incomplete Method Implementation)**
   1. **File:** myDateTime.java:9–11, 14–16
   2. **Description:** Methods retrieveDateTime() and setMyDateTime() are incomplete stubs, returning dummy values or doing nothing, creating unpredictable behavior.
8. **APIs (Missing Authentication/Authorization)**
   1. **File:** CRUDController.java:12–17 and GreetingController.java:15–18
   2. **Description:** REST endpoints /read and /greeting lack authentication mechanisms, allowing unrestricted access to potentially sensitive business data.
9. **Input Validation (Parameter Injection)**
   1. **File:** CRUDController.java:13
   2. **Description:** The business\_name parameter is accepted without validation and could be used to inject malicious data or cause application errors.
10. **Information Disclosure (Sensitive Data Exposure)**

* **File:** customer.java:7–10
* **Description:** The showInfo() method exposes sensitive account numbers directly, lacking proper data masking or access controls.

**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

### **1. Known Vulnerabilities (Codes and Names)**

* **Bouncy Castle Cryptographic Library (bcprov-jdk15on-1.46)**

CVE-2016-1000338, CVE-2020-15522, CVE-2023-33202, CVE-2015-7940

* **Apache Tomcat Embedded Server (tomcat-embed-core-9.0.30)**

Includes CVE-2020-1938 (Ghostcat), CVE-2020-9484, CVE-2021-25122, CVE-2020-17527, CVE-2023-44487, and many others.

* **Jackson JSON Library (jackson-databind-2.10.2)**

CVE-2020-25649, CVE-2020-36518, CVE-2022-42003, CVE-2022-42004

* **Spring Framework Core (spring-core, spring-web, spring-webmvc 5.2.3)**

CVE-2022-22965 (Spring4Shell), CVE-2024-22259, CVE-2020-5421, CVE-2022-22950

* **SnakeYAML Parser (snakeyaml-1.25)**

CVE-2022-1471, CVE-2017-18640, CVE-2022-25857, CVE-2022-38750

* **Hibernate Validator (6.0.18.Final)**

CVE-2025-35036, CVE-2023-1932, CVE-2020-10693

* **Logback Logging (logback-core-1.2.3)**

CVE-2023-6378, CVE-2021-42550

* **Apache Log4j API (log4j-api-2.12.1)**

CVE-2020-9488

### **2. Descriptions and Recommended Fixes**

* **Bouncy Castle:**

Old versions have weak cryptographic implementations that can lead to timing attacks and certificate validation bypass.

*Fix:* Update to version 1.76 or later in the Maven POM.

* **Apache Tomcat:**

Several remote code execution and request smuggling vulnerabilities exist in version 9.0.30.

*Fix:* Upgrade Spring Boot to a newer release (2.7.18+), which brings Tomcat 9.0.75 or later.

* **Jackson Databind:**

Unsafe deserialization lets attackers run malicious code through crafted JSON payloads.

*Fix:* Upgrade to Jackson 2.13.4+ and configure safe deserialization options.

* **Spring Framework:**

Issues like Spring4Shell allow attackers to exploit data binding to gain remote code execution.

*Fix:* Upgrade to Spring Boot 2.7.18 (includes Spring Framework 5.3.27) and validate input.

* **SnakeYAML:**

Can execute arbitrary Java code when loading untrusted YAML.

*Fix:* Upgrade to SnakeYAML 2.0+ and use safe constructors.

* **Hibernate Validator:**

Contains flaws that may let attackers bypass validation rules.

*Fix:* Update to the latest Hibernate Validator release.

* **Logback / Log4j:**

These older logging frameworks have vulnerabilities leading to denial-of-service or data leaks.

*Fix:* Update to current stable versions (Logback 1.5.x, Log4j 2.20+).

### **3. Attribution and Documentation**

* **Detection Tool:** OWASP Dependency-Check Maven plugin (v12.1.0) performed static analysis and matched dependencies to the National Vulnerability Database (NVD).
* **Vulnerability References:**
  + CVE system (managed by MITRE) provided official identifiers like CVE-2022-22965 and CVE-2020-1938.
  + **Vendor advisories**:
    - Spring Security Team for Spring Framework flaws (Spring4Shell).
    - Apache Software Foundation for Tomcat and Log4j issues.
    - FasterXML GitHub advisories for Jackson.
    - Bouncy Castle security notices for cryptographic issues.
* These CVEs are widely documented in NVD entries, vendor bulletins, and security research blogs.

**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.

To address the security issues found in both the manual review and the static testing scan, the first step is to update all outdated and vulnerable third-party libraries. The dependency check showed that several critical components, such as Bouncy Castle, Spring Boot/Tomcat, Jackson, SnakeYAML, Hibernate Validator, and logging frameworks like Logback and Log4j, contain known vulnerabilities. Upgrading these libraries to their latest secure versions and keeping them updated through tools like OWASP Dependency-Check or Dependabot will reduce the risk of attacks such as remote code execution and unsafe deserialization. The team should also ensure that the application always uses modern encryption standards, including TLS 1.2 or higher, and secure password hashing algorithms like BCrypt or Argon2.

In addition to updating dependencies, the application’s code should follow secure development practices. All user input should be validated and sanitized to prevent attacks such as SQL injection and cross-site scripting (XSS). Sensitive information, such as database credentials, should never be hard-coded in the code and should instead be stored securely using environment variables or a secrets manager. REST API endpoints need authentication and authorization to protect business data, along with proper error handling that avoids exposing internal details to users while keeping clear logs for developers.

Finally, Artemis Financial should integrate these security improvements into a DevSecOps approach by running static analysis and vulnerability scans automatically during development and build processes. Regular patching, secure coding standards, and ongoing monitoring of vulnerability databases like the NVD will help the team stay proactive against new threats. These actions will strengthen the company’s ability to protect sensitive financial and personal data while supporting compliance with security standards and future regulations such as GDPR or PCI DSS.