# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **05/25/25** | **Donovan Taylor** |  |

## 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

Donovan Taylor

**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?

Client Needs and Threats  
Artemis Financial requires a secure, stable web-based application to handle sensitive financial data, client interactions, and business operations. Key security concerns include unauthorized access, data breaches, insecure APIs, and code vulnerabilities.

Secure Communications  
Given Artemis Financial's financial domain, secure communications (e.g., HTTPS, encryption) are essential for protecting customer data and ensuring compliance with financial regulations (like PCI DSS).

International Transactions  
If Artemis engages in international transactions, this raises the importance of encryption and data residency concerns. Secure protocols must align with regulations like GDPR, which impact data transfer across borders.

Governmental Restrictions  
Encryption algorithms and communication methods must comply with U.S. export laws and potentially international standards like FIPS 140-2.

External Threats  
Potential threats include SQL injection, insecure endpoints, broken authentication, and information disclosure. As the application expands, risks from API exposure and dependency vulnerabilities may grow.

Modernization Requirements

* Open-source libraries must be monitored for vulnerabilities (e.g., outdated hibernate-validator).
* Web technology evolution means staying current with security best practices in Spring Boot and Java, such as validating user input, avoiding hardcoded credentials, and using secure database interactions.

**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 process flow diagram, the following areas apply:

* Input Validation: User inputs are not sanitized (e.g., /read?business\_name=), risking injection attacks.
* APIs: REST endpoints (e.g., /greeting, /read) may be exposed without authentication or rate limiting.
* Cryptography: No usage detected in code; potential gap in data protection (e.g., for credentials or sensitive data).
* Client/Server: As a REST service, secure communication (HTTPS) and proper server-side input handling are vital.
* Code Error: Lack of exception handling in several files (e.g., DocData) can reveal stack traces.
* Code Quality: Duplicated constructors in CRUD, unstructured code in customer, and use of default visibility suggest weak code quality.
* Encapsulation: Several fields use default access instead of private, reducing encapsulation (e.g., in customer, myDateTime).

**3. Manual Review**

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

| # | Class/File | Vulnerability | Description |
| --- | --- | --- | --- |
| 1 | DocData.java | Hardcoded DB credentials | Database connection uses "root"/"root", making it extremely vulnerable to unauthorized access. |
| 2 | DocData.java | SQL Injection Potential | Method read\_document does not sanitize inputs, and a raw SQL connection is prepared (even if not executed). |
| 3 | CRUDController.java | Insecure API endpoint | The /read endpoint accepts user input directly and returns a string from an unsanitized object. |
| 4 | CRUD.java | Redundant constructor logic | One constructor duplicates another, which could introduce confusion and maintenance issues. |
| 5 | customer.java | Public data exposure | The account\_balance variable is default-visible; no getter/setter pattern used. |
| 6 | myDateTime.java | Incomplete implementation | Methods are stubs, and time fields are public by default, risking misuse. |
| 7 | DocData.java | Lack of exception handling | Catches SQLException but only prints stack trace, which can leak internal info. |
| 8 | GreetingController.java | Predictable ID generation | Uses an atomic counter for ID in a public endpoint; vulnerable to enumeration attacks. |
| 9 | customer.java | No authentication or access control | Anyone could call deposit() if exposed via a service layer. |
| 10 | RestServiceApplication.java | No security configuration | No filters, security policies, or authentication mechanisms are included in the app startup. |

**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

| # | Dependency | CVEs (Sample) | Severity | Description |
| --- | --- | --- | --- | --- |
| 1 | bcprov-jdk15on-1.46 | CVE-2024-34447, CVE-2016-1000338, etc. | HIGH | Cryptographic weaknesses including improper certificate validation, ASN.1 injection, and timing attacks. |
| 2 | hibernate-validator-6.0.18.Final | CVE-2023-1932, CVE-2020-10693 | MEDIUM | Allows XSS via improper HTML tag filtering and unsafe EL evaluation in error messages. |
| 3 | jackson-databind-2.10.2 | CVE-2020-25649, CVE-2020-36518, etc. | HIGH | Vulnerable to XXE, deserialization attacks, and resource exhaustion (DoS). |
| 4 | log4j-api-2.12.1 | CVE-2020-9488 | LOW | Weak certificate validation in SMTP appender; fixed in 2.12.3+. |
| 5 | logback-classic/core-1.2.3 | CVE-2023-6378, CVE-2021-42550, etc. | HIGH | Arbitrary code execution via poisoned config files, deserialization vulnerabilities, SSRF, and EL injection. |
| 6 | spring-boot-2.2.4.RELEASE | Multiple (3) | CRITICAL | Outdated version with vulnerabilities leading to potential remote code execution and other critical issues. |
| 7 | spring-boot-starter-web-2.2.4.RELEASE | Multiple (3) | CRITICAL | Includes vulnerable components like spring-web and spring-core; subject to RCE and injection risks. |
| 8 | spring-context-5.2.3.RELEASE | Multiple (12) | CRITICAL | Known to contain RCE vulnerabilities; outdated framework version with multiple exposed weaknesses. |
| 9 | spring-core-5.2.3.RELEASE | Multiple (11) | CRITICAL | Includes serious vulnerabilities such as those enabling RCE and DoS; upgrade required. |
| 10 | spring-expression-5.2.3.RELEASE | Multiple (12) | CRITICAL | Vulnerable to expression language injection and unsafe expression evaluation, leading to potential RCE. |
| 11 | spring-web-5.2.3.RELEASE | Multiple (16) | CRITICAL | Includes vulnerabilities enabling remote code execution, SSRF, and path traversal. |
| 12 | spring-webmvc-5.2.3.RELEASE | Multiple (12) | CRITICAL | Susceptible to RCE and insecure input handling; mitigation requires full framework update. |
| 13 | snakeyaml-1.25 | CVE-2022-1471, CVE-2022-25857, etc. | CRITICAL | Multiple RCE and DoS vulnerabilities due to unsafe deserialization and deep recursion in YAML structures. |
| 14 | tomcat-embed-core-9.0.30 | CVE-2020-1938 | CRITICAL | AJP connector vulnerability allows arbitrary file access and JSP execution, enabling remote code execution. |

**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 vulnerabilities identified in the static testing and manual review, the following mitigation steps are recommended:

First, upgrade all outdated and vulnerable dependencies to their latest secure versions. The bcprov-jdk15on library should be updated to resolve cryptographic weaknesses like improper certificate validation and ASN.1 injection. The hibernate-validator library must be upgraded to eliminate cross-site scripting risks due to improper HTML tag filtering. For jackson-databind, update to a patched version (at least 2.12.7.1 or later) to mitigate deserialization and denial-of-service vulnerabilities.

The log4j-api should be upgraded to version 2.12.3 or newer to fix the SMTP appender issue. Likewise, logback-core and logback-classic must be updated to resolve vulnerabilities related to arbitrary code execution and expression language injection.

All Spring framework components, including spring-boot, spring-context, spring-core, spring-web, spring-webmvc, and spring-expression, are critically outdated and must be upgraded to the latest supported versions (Spring 5.3+ or 6.x). These updates address multiple critical issues, including remote code execution, SSRF, and unsafe input handling.

The snakeyaml library must be upgraded to version 2.0 or higher to avoid multiple remote code execution and denial-of-service vulnerabilities stemming from unsafe deserialization and deep recursion. Ensure that SafeConstructor is used when parsing any untrusted YAML input.

The tomcat-embed-core library should be updated to at least version 9.0.31 to mitigate an Apache JServ Protocol (AJP) vulnerability. Additionally, the AJP connector should be disabled entirely if not required, or limited to internal network access only.

From the manual code review, hardcoded database credentials must be removed and replaced with secure environment-based storage. Any dynamic SQL statements should be refactored to use parameterized queries to prevent SQL injection. Unsecured API endpoints must implement input validation and access control. Redundant constructor logic should be consolidated, and sensitive fields must be marked private with proper encapsulation. Exception handling must be implemented to avoid stack trace exposure, and predictable ID generation should be replaced with UUIDs or secure random values. Lastly, the application should enforce authentication, authorization, and a clear security configuration using Spring Security or equivalent frameworks.