# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **09/16/2025** | **Mike Brown** |  |

## 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

Mike Brown

**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 needs secure communications to protect sensitive financial data such as customer investment plans, retirement accounts, and insurance records. Secure communication ensures confidentiality, integrity, and compliance with regulations like GDPR and HIPAA.

Since Artemis serves international clients, secure communication must follow global data protection laws and any relevant government restrictions.

External threats include:

* Man-in-the-middle attacks during API communication
* SQL injection and XSS targeting the web application
* Denial of service (DoS/DDoS) attacks aimed at financial services
* Supply chain risks through open-source libraries

Modernization requirements:

* Replacing outdated libraries with supported, secure versions
* Supporting encryption standards such as TLS 1.3
* Implementing secure RESTful APIs
* Adopting evolving web application frameworks with a security-first mindset

**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.

* Relevant Areas for Artemis Financial:
* Input Validation: Prevents malicious input, including SQL injection and cross-site scripting (XSS).
* APIs: Protect RESTful web services with authentication and rate limiting.
* Cryptography: Encrypt sensitive financial data in transit and at rest.
* Client/Server Security: Secure distributed communication to prevent eavesdropping.
* Code Errors & Code Quality: Avoid insecure coding practices, ensure exception handling.
* Encapsulation: Protect sensitive business logic and data models.

Each of these areas is directly related to safeguarding financial data integrity and stopping unauthorized access.

**3. Manual Review**

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

Sample manual findings from code inspection:

* Hardcoded credentials in DatabaseConfig.java.
* Missing input validation on user forms (LoginController.java).
* Sensitive info logged in plaintext in TransactionService.java.
* Unparameterized SQL queries present in UserDAO.java (SQL injection risk).
* No CSRF protection on account update forms.
* Use of outdated cryptographic algorithms (MD5 for hashing passwords).
* Unrestricted file upload endpoint in UploadController.java.
* Improper exception handling leaks stack traces in ErrorHandler.java.
* No rate limiting on login attempts (brute-force vulnerability).
* Missing validation for JSON payloads in PaymentAPI.java.

**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

Using the **Maven Dependency-Check plug-in**, the following vulnerabilities were found:

* hibernate-validator-6.0.18.Final.jar
  + CVE-2025-35036 (OSSINDEX)
    - Hibernate Validator before 6.2.0 and 7.0.0 may interpolate user-supplied input in constraint violation messages with Expression Language, allowing attackers to access sensitive data or execute arbitrary code. Fixed in 6.2.0 and 7.0.0. Related: CVE-2020-5245, CVE-2025-4428.
  + CVE-2020-10693
    - A flaw in Hibernate Validator 6.1.2.Final allows invalid EL expressions to bypass sanitation, potentially leading to injection attacks.
* jackson-core-2.10.2.jar
  + CVE-2020-8840
    - Denial-of-service vulnerability caused by deeply nested JSON structures in jackson-core before 2.10.3.
  + CVE-2020-11619
    - Improper handling of crafted inputs could allow attackers to exploit parser features to leak information or corrupt memory.
* jackson-databind-2.10.2.jar
  + CVE-2020-36518
    - Infinite recursion vulnerability leading to denial of service in jackson-databind before 2.12.7.1 and 2.13.4.1.
  + CVE-2022-42003
    - Crafted input could trigger uncontrolled resource consumption, crashing the application.
  + CVE-2022-42004
    - Vulnerability in polymorphic deserialization leading to potential remote code execution.
* log4j-api-2.12.1.jar
  + CVE-2021-44228 (Log4Shell)
    - Remote code execution via JNDI lookups in log messages.
* logback-classic-1.2.3.jar
  + CVE-2021-42550
    - Logback before 1.2.9 allows malicious configuration injection leading to code execution.
  + CVE-2017-5929
    - Remote code execution risk in certain logback appenders.
* logback-core-1.2.3.jar
  + CVE-2021-42550
    - Same core vulnerability as logback-classic, leading to possible RCE.
* mongo-java-driver-2.4.jar
  + CVE-2019-20933
    - Insecure deserialization issue in older MongoDB Java drivers could allow crafted payloads to execute code.
* snakeyaml-1.25.jar
  + CVE-2022-25857
    - Unsafe YAML deserialization before 1.31 allows crafted payloads to execute arbitrary code.
  + CVE-2022-38749 / CVE-2022-38750 / CVE-2022-38751 / CVE-2022-38752
    - Multiple deserialization flaws leading to denial-of-service or code execution.
* spring-aop-5.2.3.RELEASE.jar
* spring-core-5.2.3.RELEASE.jar
* spring-context-5.2.3.RELEASE.jar
* spring-expression-5.2.3.RELEASE.jar
* spring-web-5.2.3.RELEASE.jar
* spring-webmvc-5.2.3.RELEASE.jar
  + CVE-2022-22965 (Spring4Shell)
    - Remote Code Execution vulnerability in Spring MVC/WebFlux apps on JDK 9+.
    - Additional CVEs involve input validation and privilege escalation issues in Spring Framework.
* spring-boot-2.2.4.RELEASE.jar
* spring-boot-starter-web-2.2.4.RELEASE.jar
  + CVE-2023-20873
    - Spring Boot 3.0.0–3.0.5 had request mapping flaws that could expose unintended endpoints; older branches like 2.2.4 are unsupported and vulnerable.
* tomcat-embed-core-9.0.30.jar
* tomcat-embed-websocket-9.0.30.jar
  + CVE-2023-44487 (HTTP/2 Rapid Reset Attack)
    - Allows large-scale DDoS via rapidly reset HTTP/2 streams. Exploited in the wild.
* hibernate-validator-6.0.18.Final.jar
  + CVE-2025-35036 – EL injection in constraint messages (RCE).
  + CVE-2020-10693 – Invalid EL expressions bypass sanitation.
* jackson-databind-2.10.2.jar
  + CVE-2022-42003 – Resource exhaustion in data-binding.
  + CVE-2020-36518 – Infinite recursion leading to DoS.
* logback-classic-1.2.3.jar
  + CVE-2021-42550 – Malicious config files enable RCE.
* snakeyaml-1.25.jar
  + CVE-2022-25857 – Crafted YAML payload causes RCE.
* spring-core-5.2.3.RELEASE.jar
  + CVE-2022-22965 (Spring4Shell) – Remote Code Execution in Spring MVC/WebFlux.
* tomcat-embed-core-9.0.30.jar
  + CVE-2023-44487 – HTTP/2 Rapid Reset DDoS vulnerability.

**Solutions:** Upgrade all libraries to patched versions, replace weak algorithms, and enforce secure coding standards.

**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.

* Upgrade dependencies: Patch vulnerable libraries like Spring 5.3.x, Hibernate 6.2+, Logback 1.2.9+, SnakeYAML 1.33+, and Jackson 2.15+.
* Implement input validation: Sanitize and validate user inputs in forms and APIs.
* Enforce encryption: Use TLS 1.3 and replace MD5 with bcrypt or Argon2 for password hashing.
* Secure APIs: Require OAuth2 or JWT authentication and enforce rate limiting. Improve logging and error handling: Remove sensitive data from logs and disable stack trace leaks.
* Add CSRF protection: Enable CSRF tokens for form submissions.
* Harden file uploads: Restrict file types, scan uploads, and set size limits.
* Monitor and alert: Implement intrusion detection and monitor for abnormal login attempts.
* Conduct regular scans: Re-run dependency checks and static code analysis in CI/CD pipelines.
* Provide developer training: Promote secure coding practices across the Scrum team.