# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/28/2025** | **Najjah Gray-Thompson** |  |

## 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

Najjah Gray-Thompson

**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?

As a developer at Global Rain, I have been tasked with assisting Artemis Financial in modernizing their operations. The client is a consulting company that develops individualized financial plans. Daily operations consist of handling multiple customers’ personal identifiable information as well as financial data. This company works with highly sensitive information therefore secure communication is a top priority. It is essential to ensure confidentiality as well as provide reliable protection against external threats.

Though the client did not specify international transactions within their requirements, Global Rain is prepared to remain in compliance with international data protection standards. However, in the United States we must adhere to governmental restrictions such as the Gramm-Leach-Bliley Act which "requires that financial institutions implement safeguards to protect the security and confidentiality of customer information." (SailPoint) Additionally, we intend to follow good practice by committing to the guidelines provided in the FFIEC (Federal Financial Institutions Examination Council) which specifically focuses on API security and vendor management.

Artemis Financial’s services are delivered through a web-based application that communicates using RESTful API's. This means that it is crucial that only authorized users of the web-application are able to access API data. Also, data integrity is a priority when utilizing APIs, as the information requested by the user should be the same upon retrieval. Any vulnerability leaves Artemis Financial at risk of injection attacks such as SQL injection attacks and cross-site scripting. These threats will only become more imminent in the future as attackers’ practices evolve.

Lastly, with this company relying on open-source libraries and frameworks, it is essential to put processes in place to continuously update them, as outdated versions are one of the primary causes of vulnerabilities. This can be done using tools like Maven Dependency-Check or OWASP Dependency-Track. Another part of modernization is adhering to modern security practices like encryption with Transport Layer Security (TLS). With web application technologies constantly evolving, Artemis Financial's RESTful API should be equipped to respond to change. This entails implementing secure coding practices inside the Spring framework as well as enabling secure API authentication techniques like OAuth 2.0 and JWT.

**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.

There are multiple areas of security that apply to Artemis Financial’s software application. Input valid is the first to be taken into consideration. Both CRUDController.java and GreetingController.java accept request parameters without validation or constraints. The read\_document method in DocData.java has no validation on the parameters which poses a risk for SQL injection. Next, being that the program revolves around RESTful API communication, it is essential to evaluate this component thoroughly. With there being no authentication or authorization around endpoints like /greeting or /crud, this leaves the API vulnerable to risks. Client/Server is another applicable area of security as REST services rely on client/server communication. Code Error and Code Error also needs to be considered throughout the entire program. Currently, there is no centralized exception handling to sanitize error responses. Also, there are practices throughout the program like in DocData.java, that create risk for injection or data breaches. And lastly, encapsulation should also be considered. Customer.java stores sensitive account data but does not properly protect it, which could lead to 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.

The following are vulnerabilities in the code base and where they are located as well as relevant categories:

1. Unvalidated inputs (CRUDController.java and GreetingContoller.java)
   1. CWE-20: Improper Input Validation
   2. @RequestParam with no validation
2. No parameterized queries (DocData.java)
   1. CWE-89: SQL Injection
3. Incomplete validation in business logic (customer.java)
   1. CWE-1284: Improper Validation of Specified Quantity In Input
   2. Deposit() allows negative balances
4. Error handling (DocData.java)
   1. CWE-209: Information Exposure Through an Error Message
   2. printStackTrace()
5. Encapsulation (customer.java)
   1. CWE-496: Public Data Assigned to Private Array-Typed Field
   2. Customer account information not encapsulated
6. Hardcoded database credentials (DocData.java)
   1. CWE-259: Use of Hard-coded Password
7. No authentication/authorization on any endpoints
   1. CWE-306: Missing Authentication for Critical Function

**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. CVE-2016-1000338
   1. In Bouncy Castle JCE Provider versions 1.55 and earlier, the DSA does not fully validate ASN.1 encoding. An attacker may be able to inject extra elements into the encoded signature sequence that still validate, thereby introducing “invisible” data into a signed structure.
   2. Upgrade to a version later that 1.55
   3. National Vulnerability Database (CWE-347: Improper Verification of Cryptographic Signature)
2. CVE-2025-35036
   1. Hibernate Validator before versions 6.2.0 and 7.0.0 may interpolate user-supplied input into constraint violation messages using Expression Language. This could allow attackers to inject payloads, access sensitive data, or execute arbitrary Java code.
   2. Upgrade Hibernate Validator to version 6.2.0, 7.0.0, or later. Developers should also avoid using user-supplied input in custom violation messages.
   3. National Vulnerability Database
3. CVE-2020-25649
   1. A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity.
   2. Update to a version where issue does not exist. Also, disable or securely configure XML entity expansion features when parsing XML.
   3. National Vulnerability Database (CWE-611: Improper Restriction of XML External Entity Reference)
4. CVE-2020-9488
   1. The Apache Log4j SMTP appender fails to properly validate certificate host names, which can enable a man-in-the-middle (MITM) attacker to intercept SMTPS connections and leak log messages
   2. Upgrade to a later version of Log4j
   3. National Vulnerability Database
5. CVE-2023-6378
   1. A serialization vulnerability in the Logback receiver component (version 1.4.11) allows an attacker to cause a Denial-of-Service (DoS) by sending maliciously crafted data.
   2. Upgrade to a later version of Logback and avoid using the vulnerable receiver feature
   3. National Vulnerability Database
6. CVE-2022-1471
   1. This is a YAML parsing vulnerability in SnakeYAML. Untrusted input can be parsed in a way that leads to arbitrary code execution or information exposure.
   2. Avoid unsafe YAML deserialization and upgrade to the SnakeYAML library.
   3. National Vulnerability Database
7. CVE-2023-20873
   1. In certain versions of Spring Boot (3.0.0-3.0.5, 2.7.0-2.7.10, and older unsupported releases), applications deployed to Cloud Foundry may be vulnerable to a security bypass
   2. For Spring Boot 3.0 users, upgrade to 3.0.6 or later and for Apring Boot 2.7 users, upgrade to 2.7.11 or later
8. CVE-2022-22965
   1. A remote code execution (RCE) vulnerability in the Spring Framework that allows attackers to exploit parameter binding to run arbitrary code under certain conditions.
   2. Upgrade Spring Framework or Spring Boot to a patched version
   3. National Vulnerability Database
9. CVE-2016-1000027
   1. A vulnerability in Spring Framework (older versions), potentially leading to remote code execution or similar.
   2. Upgrade Spring to a non-vulnerable version
10. CVE-2020-1938
    1. AJP connector in Apache Tomcat could allow unauthorized file reads or writes when misconfigured, exposing server contents.
    2. Disable or secure the AJP connector or upgrade to a later version where issue is fixed.

**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.

Static testing with Maven Dependency-Check identified several vulnerable dependencies. Each has public documentation in the NVD and vendor advisories, and most are mitigated through upgrading to patched versions of the affected libraries. To mitigate these risks, Artemis Financial should enforce strict server-side input validation and apply parameterized queries throughout the code. Business rules should prevent invalid transactions and sensitive fields require encapsulation. Authentication and authorization must be enforced with Spring Security, and all API communication should be encrypted using TLS. For dependency risks, all vulnerable libraries (Log4j, Jackson, Spring, SnakeYAML, Tomcat, etc.) should be updated to patched versions. Regular vulnerability scans and continuous monitoring should be integrated into development to ensure that security issues are identified and remediated quickly.