# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **May 22, 2025** | **Ali Jackson** | **Based off dependency report** |

## 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

Ali Jackson

**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 reviewed Artemis Financial’s needs for their RESTful web API. Secure communications are critical because Artemis handles sensitive financial data like savings and investments, and a breach could ruin their reputation. Since they serve global customers, they likely make international transactions, so they must comply with laws like GDPR. Some countries might restrict encryption, which we need to consider. Current threats include API exploits and data breaches, and these could grow with new attack methods in the future. For modernization, Artemis should update outdated opensource libraries like Spring Boot, which often have vulnerabilities, and adopt modern web technologies like secure API authentication to stay safe.

**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.

Using the vulnerability assessment process flow diagram, I identified four key areas relevant to Artemis Financial’s RESTful API. First, Architecture Review is important to understand the API’s design and spot flaws, like weak endpoints. Next, Input Validation is critical because APIs often face injection attacks like XSS if inputs aren’t checked. APIs is directly relevant since Artemis uses a RESTful API, so we need to secure endpoints and interactions. Finally, Cryptography matters for protecting data in transit, like using HTTPS for client communications. Other areas like Client/Server or Encapsulation didn’t apply as much since the focus is on the API, not distributed systems or data structures.

**3. Manual Review**

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

I manually reviewed the rest-service codebase and identified the following specific vulnerabilities:

1. Unvalidated Input in CRUDController.java: The ‘/read‘ endpoint accepts a ‘businessname‘parameT he‘readdocument‘methodusesrawSQLwith‘DriverM anager.getConnection‘andhardcodedcred

2. Insecure HTTP in application.properties: ‘server.port=8081‘ lacks HTTPS enforcement,

exposing data. Configure ‘server.ssl.enabled=true‘ and provide a keystore.

3. Hardcoded Credentials in DocData.java: The database connection uses hardcoded ”root”/”root”

credentials, risking exposure. Move to environment variables or a secure vault (e.g.,

Spring Vault).

4. No Authentication in GreetingController.java and CRUDController.java: Endpoints like

‘/greeting‘ and ‘/read‘ lack authentication, allowing unauthorized access. Implement

Spring Security with JWT or OAuth2.

5. Missing Rate Limiting: No rate limiting is configured, risking DDoS attacks. Add Spring

Boot Actuator with a custom filter or use a library like Bucket4j.

6. Incomplete Implementation in myDateTime.java: The ‘retrieveDateTime‘ and ‘setMyDateTime‘ methods are unimplemented, potentially leading to runtime errors or security

gaps if misused. Complete with proper validation.

7. Insecure File Upload in application.properties: ‘spring.servlet.multipart.max-file-size=200MB‘

and ‘max-request-size=215MB‘ allow large uploads without antivirus checks, risking malicious files. Add file type validation and scanning.

**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

I ran the OWASP Dependency-Check plug-in (version 12.1.0) on the rest-service project in IntelliJ IDEA, generating the dependency-check-report.html file. The report identified 175 vulnerabilities across 16 dependencies. Key findings include:

• CVE-2024-34447: Improper validation of certificate with host mismatch in bcprovjdk15on-1.46.jar. Solution: Upgrade to Bouncy Castle 1.78 or later. Attribution: <https://nvd.nist.gov/vuln/detail/CVE-2024-34447>.

• CVE-2016-1000338: Improper verification of DSA signature in bcprov-jdk15on1.46.jar. Solution: Upgrade to 1.56 or later. Attribution: <https://nvd.nist.gov/vuln/detail/CVE2016-1000338>.

• CVE-2020-15522: Timing attack on ECDSA signatures in bcprov-jdk15on-1.46.jar. Solution: Upgrade to 1.66 or later. Attribution: <https://nvd.nist.gov/vuln/detail/CVE2020-15522>.

• CVE-2023-33202: Denial of Service in bcprov-jdk15on-1.46.jar due to crafted ASN.1 data. Solution: Upgrade to 1.73 or later. Attribution: <https://nvd.nist.gov/vuln/detail/CVE2023-33202>.

• CVE-2024-29857: Out-of-bounds read in bcprov-jdk15on-1.46.jar. Solution: Upgrade to 1.78 or later. Attribution: https://nvd.nist.gov/vuln/detail/CVE-2024- 29857.

• CVE-2020-1935: HTTP request smuggling in tomcat-embed-core-9.0.30.jar. Solution: Upgrade to 9.0.31 or later. Attribution: https://nvd.nist.gov/vuln/detail/CVE2020-1935.

**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.

Based on the manual review and static testing, the following steps will secure Artemis Financial’s application:

• Unvalidated Input and SQL Injection: Implement input validation and parameterized queries, testing with tools like OWASP ZAP.

• Insecure HTTP and Hardcoded Credentials: Configure HTTPS and use secure credential management, verifying with SSL labs.

• No Authentication and Rate Limiting: Add Spring Security and rate limiting, testing with Postman.

• Incomplete Implementation: Complete and validate ‘myDateTime.java‘ methods.

• Insecure File Upload: Add file validation and scanning, testing uploads.

• Dependency Vulnerabilities: Update all vulnerable libraries (e.g., Bouncy Castle to 1.78, Tomcat to 9.0.81), re-running Dependency-Check.

These steps will address code-level issues and dependency risks, ensuring compliance and security.