# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/15/2025** | **Paul Garner** | **Starting the document.** |

## 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

Paul Garner

**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 is looking to bring their web based tools up to speed with modern tech, and they want to make sure everything’s secure understandably, since they handle pretty sensitive stuff like investment plans, savings, and retirement accounts. That means secure communication is a must. A data breach here wouldn’t just be embarrassing, it could be devastating.

They don’t mention international transactions specifically, but given they’re a financial company, it’s probably safe to assume they work with clients across borders. So we’ve got to keep international compliance in mind, like GDPR or similar regulations.

On the tech side, they’re using open source libraries and evolving web technologies. That’s great for flexibility, but it also means they’ve got to stay on top of updates and security risks. With REST APIs in play, we’re talking about common threats like injection attacks, broken access controls, and exposed data. Keeping all those risks in check will be key.

**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.

Here are the main security areas I’d focus on for Artemis Financial’s application:

* Authentication: We’ve got to make sure only verified users can access the system.
* Authorization: Even once someone’s logged in, they should only see what they’re allowed to.
* Sensitive Data Handling: Since it’s financial data, everything needs to be encrypted, no exceptions.
* Data Validation: Any user input needs to be properly checked so we’re not opening the door to attacks.
* Error Handling: Let’s avoid giving hackers any clues by keeping stack traces and debug info out of error messages.
* Configuration Management: Making sure we’re not exposing services or leaving dev settings on in production.

These areas line up with the typical weak spots for web applications, especially in finance, and cover the bases for keeping things safe.

**3. Manual Review**

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

Here’s what I found while digging through the code:

1. Hardcoded credentials in DatabaseConfig.java. Not ideal, those need to be externalized.
2. No input validation in ClientController.java. That’s risky for injections.
3. Public endpoints in FinancialPlanController.java without access checks.
4. Unrestricted file uploads in DocumentUploadService.java. Could allow harmful files.
5. XSS possibilities in UserInputHandler.java user input is being rendered directly.
6. Stack traces are exposed in GlobalExceptionHandler.java.
7. Cookies aren't secure, missing flags in session config.
8. No CSRF protection is turned on in Spring Security config.
9. Passwords are logged in AuthService.java, which is a big no no.
10. Outdated dependencies listed in pom.xml. A couple haven’t been touched in years.

**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

To check Artemis Financial’s software for any known security flaws, we ran a static dependency check using the OWASP Dependency Check plugin. The scan found several third party libraries that had reported vulnerabilities. Some of the big ones included Jackson databind (with CVE-2020-25649 and CVE-2022-42004), hibernate validator (CVE-2025-35036), and spring web (CVE-2022-22965). Other hits showed up in logback core, log4j api, snakeyaml, and mongo java driver. All of these are common libraries in Java apps, but if they’re not kept up to date, they can open the door to exploits.

Each of these vulnerabilities had a short write up in the report. For example, some let attackers manipulate serialized data, while others exposed the app to remote code execution or denial of service attacks. The tool gave straightforward solutions, mostly to update to newer versions where the issues have already been patched. In some cases, alternative libraries were suggested if the vulnerabilities were long standing or no longer actively maintained.

These vulnerabilities were flagged using CVEs (Common Vulnerabilities and Exposures), which are standardized references found in databases like the NVD (National Vulnerability Database). That means they’ve been reported, cataloged, and verified by security researchers and vendors. It’s great to have this kind of visibility because it lets us act fast before these issues cause real harm.

**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 static testing results, the first and most important step is to update any libraries that had known vulnerabilities. Most of the flagged issues have been patched in newer versions, so upgrading to the latest stable release should resolve the bulk of them. It’s especially important to deal with high risk items like spring web and Jackson databind, as those are core to the application and had multiple critical CVEs.

We should also look into replacing libraries that don’t seem to be actively maintained. If no safe version exists for a vulnerable dependency, it might be time to switch to a more secure alternative. Additionally, if any of the libraries aren’t actually used in the project, we can just remove them to reduce risk.

Another solid step is to put a routine in place for ongoing scans and reviews. Running automated dependency checks before each release can help catch issues early. Adding a manual review step now and then to double check anything the scan might’ve missed is a good backup. Between those tools and better version tracking, Artemis Financial can maintain a much stronger security posture moving forward.