# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/28/2025** | **Reice Morgan** | **Initial draft created** |

## 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

Reice Morgan

**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 develops personalized financial plans for its clients, which means the company handles very sensitive data like account balances, retirement goals, and investment information. Because of the nature of the business, protecting this data is not just important for compliance reasons but also to maintain the trust of customers who rely on them.**

* **Secure communications: For Artemis, secure communications are essential. All customer data must be transmitted using strong encryption so no one can intercept or manipulate it. Without it, client trust and the company’s reputation would be at serious risk.**
* **International transactions: Even though they’re not stated outright, financial firms often work across borders. That means Artemis may need to think about how regulations like GDPR or other data privacy laws come into play if they have international clients.**
* **Government restrictions: As a financial services provider, Artemis has to comply with data protection laws like PCI DSS or GLBA. These regulations set the standard for how sensitive data should be secured.**
* **Threat landscape: The company faces modern threats such as phishing, SQL injection, ransomware, or supply-chain attacks from vulnerable libraries. These are common in today’s financial sector and need to be actively defended against.**
* **Modernization requirements: Since Artemis wants to modernize its operations, it will be relying more heavily on open-source tools and frameworks. This is positive for innovation but also brings the challenge of managing software updates and avoiding known vulnerabilities.**

**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.

**Looking at the vulnerability assessment process, several areas clearly apply to Artemis Financial:**

1. **Authentication & Authorization — Clients need to know that only authorized people have access to their financial information. Strong role-based access controls will help enforce this.**
2. **Data Protection & Cryptography — Financial information must be encrypted both in storage and in transit. Outdated or weak cryptography is not acceptable.**
3. **Input Validation — Web applications that take user input are at risk of attacks like SQL injection or XSS. Proper input handling is critical.**
4. **Session Management — Secure session cookies and short expiration times reduce the risk of session hijacking.**
5. **Dependency Management — Open-source tools are very useful, but they bring risks. Vulnerabilities in third-party libraries could be exploited if not monitored.**
6. **Logging & Monitoring — Logs should help detect attacks but not expose sensitive information. This balance is important for both security and compliance.**
7. **Configuration & Deployment — Secure defaults, strong TLS, and hardened environments will help keep the application safe once deployed.**

**These areas matter the most because they directly address how a web application interacts with its users and protects their sensitive financial data.**

**3. Manual Review**

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

**After reviewing the codebase structure and considering common risks in Java REST applications, here are several vulnerabilities that stood out:**

1. **SQL Injection Risk — In AccountRepository.java, user inputs appear to be concatenated directly into SQL statements rather than using prepared statements.**
2. **Hardcoded Credentials — Sensitive information, like database passwords, is stored directly in application.properties. This is dangerous if the file is exposed.**
3. **Cross-Site Scripting (XSS) — Some user input is being returned in views without sanitization, which opens the door for malicious scripts.**
4. **Missing CSRF Protection — In AccountController.java, state-changing actions like password updates do not have protections against CSRF.**
5. **Insecure File Upload Handling — File uploads do not validate type or size, leaving room for malicious files to be stored on the server.**
6. **Sensitive Data in Logs — Logging statements currently output entire objects, which may include personal or financial data.**
7. **Weak Session Cookie Settings — Session cookies are missing secure flags like HttpOnly and SameSite.**
8. **Missing Input Validation — Some numeric and enum fields don’t have constraints, which could allow invalid or malicious values.**
9. **Unrestricted CORS Configuration — The application appears to allow requests from all origins, which is risky.**
10. **Weak Cryptography — Passwords are stored using weak or outdated hashing methods instead of bcrypt or Argon2.**

**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

**A dependency-check was run using Maven to scan for vulnerable libraries. Here are examples of what was found:**

* **CVE-2022-22965 (Spring4Shell): A vulnerability in Spring Core that could allow remote code execution. Fix: Upgrade to the patched version of Spring Core.**
* **CVE-2021-44228 (Log4Shell): A critical flaw in Log4j allowing remote code execution. Fix: Update to Log4j 2.17.1 or higher.**
* **CVE-2020-36518 (Jackson Databind): Certain versions of Jackson Databind allow unsafe deserialization. Fix: Upgrade to a safe version and disable unsafe typing.**

**These vulnerabilities have been documented by the National Vulnerability Database (NVD) and the OWASP community, making them widely recognized issues with clear mitigation paths.**

**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 both the manual and static reviews, here’s what needs to be done:

**High Priority:**

* Replace concatenated SQL with prepared statements.
* Remove hardcoded secrets from the codebase and use a secure secrets manager.
* Patch or upgrade all libraries flagged in the dependency-check scan.
* Transition password storage to bcrypt or Argon2.

**Medium Priority:**

* Add CSRF tokens to protect critical POST endpoints.
* Harden cookie attributes (Secure, HttpOnly, SameSite).
* Restrict CORS to trust domains only.
* Sanitize and validate all inputs and outputs.

**Lower Priority:**

* Adjust logging sensitive information is redacted.
* Enforce file validation checks (size, type, content) for uploads.
* Double-check deployment settings to ensure strong TLS and secure configurations.

**Ongoing Steps:**

* Automate dependency-checks in the CI/CD pipeline.
* Perform regular vulnerability scans and code reviews.
* Train developers continuously secure coding practices.