# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **15 Nov 2024** | **Shawn Plaisted** |  |

## 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

Shawn Plaisted

**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?

Secure communication is essential for Artemis Financial because the company deals with sensitive financial information for its customers, such as savings, retirement plans, investments, and insurance details. If communication channels are compromised, it could result in unauthorized access, financial harm, and a loss of trust in the company. To protect its clients and reputation, Artemis Financial must use strong security measures like end-to-end encryption and secure protocols such as HTTPS and TLS. These safeguards are also crucial for meeting data protection regulations and maintaining customer confidence.

Although the scenario doesn’t specifically mention international transactions, Artemis Financial may deal with cross-border activities, particularly in investments or insurance. This would require compliance with international standards like GDPR for EU clients and PCI DSS for payment data, making secure communication and data handling across jurisdictions critical.

Government regulations, including those from the Financial Industry Regulatory Authority (FINRA) and various state laws in the U.S., are relevant. If Artemis Financial handles international transactions, they may also need to follow encryption export laws or guidelines from agencies like the Office of Foreign Assets Control (OFAC). This makes it important to carefully evaluate encryption standards and ensure proper data privacy measures are in place.

Phishing and Social Engineering: Hackers may target employees to gain unauthorized access to sensitive systems.

API Exploitation: Vulnerabilities in the application’s APIs could lead to issues like parameter tampering, SQL injection, or unauthorized access.

Man-in-the-Middle (MITM) Attacks: Weak encryption could allow attackers to intercept and manipulate data during transmission.

Zero-Day Vulnerabilities: Unknown flaws in the software or its open-source libraries could be exploited by attackers.

Use open-source libraries carefully and address any vulnerabilities they might have by using tools like dependency-check. Take advantage of modern web application technologies, such as frameworks that include security features like OAuth for authentication and secure RESTful APIs. Incorporate DevSecOps practices to ensure security is integrated into every stage of the software development lifecycle (SDLC).

**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.

**Input Validation** Making sure that all data entered into the application is safe and valid is essential to protect against malicious attacks like SQL injection or cross-site scripting (XSS). Since Artemis Financial’s application relies heavily on user input for financial calculations and data submissions, validating that input is a key security measure.

**APIs** The application uses APIs to allow different parts of the system to communicate. Securing these APIs is critical to prevent unauthorized access, tampered parameters, or potential data breaches. Keeping APIs secure also ensures that data exchanged between components stays accurate and protected.

**Cryptography** Encryption is crucial for protecting sensitive customer information like savings, retirement, and insurance details. Cryptography ensures that this data is safe whether it’s being stored or transmitted, preventing unauthorized users from accessing or intercepting it.

**Client/Server Security** The application runs in a distributed setup, so secure communication between the client and server is essential. Using secure protocols like HTTPS and TLS ensures that data sent back and forth cannot be intercepted or tampered with by attackers.

**Code Error** Fixing errors in the code is necessary to prevent vulnerabilities like buffer overflows, unhandled exceptions, or flawed logic that attackers could exploit. Clean, error-free code helps keep the application secure and stable.

**Code Quality** Maintaining good code quality is important for reducing security risks. By following secure coding practices and performing regular code reviews, developers can catch and fix potential vulnerabilities early, improving the application’s overall reliability and safety.

**3. Manual Review**

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

**Missing Input Validation**

The application does not properly check or validate the inputs provided by users in some API endpoints, which allows potentially harmful data to be processed. For instance, there are no safeguards to stop users from submitting malicious inputs like SQL injection strings. This lack of validation could lead to serious security issues, system crashes, or unauthorized access to sensitive data. To fix this, the application should validate and sanitize all user inputs to ensure they are safe and meet the expected format before processing.

**No Authentication on Certain APIs**

Some API endpoints are accessible without any authentication or authorization in place. This means anyone with knowledge of these endpoints could gain access to or modify sensitive data. To prevent this, the application should secure all API endpoints using proper authentication methods, such as OAuth tokens or JSON Web Tokens (JWT), to ensure only authorized users have access.

**Weak Error Handling**   
The application shows detailed error messages, including stack traces, when something goes wrong. While helpful for debugging during development, this information could give attackers insights into the system’s structure, making it easier for them to exploit vulnerabilities. To address this, error messages shown to users should be generic, while detailed error information should only be logged securely on the server.

**Hardcoded Credentials**

Database credentials are written directly into the source code, which poses a security risk. If the code is ever exposed, attackers could use these credentials to access sensitive database information. Hardcoding credentials also makes managing and updating them across environments more difficult. A better solution would be to store these credentials in environment variables or in a separate configuration file that is not included in the code base.

**Unsecured Data Transmission**

The application does not enforce secure communication protocols like HTTPS, which means data sent between the client and server could potentially be intercepted or modified. This is a critical issue, especially because the application handles sensitive financial information. To address this, the application should implement SSL/TLS certificates and enforce HTTPS to ensure all data transmissions are encrypted and secure.

**Dependency Vulnerabilities** The pom.xml file lists several third-party libraries that are outdated and have known security vulnerabilities. Using old library versions increases the risk of security issues, such as remote code execution or data leaks. To mitigate this, the application should regularly update its dependencies and use tools like Maven dependency-check to identify and fix vulnerable libraries.

**No Rate Limiting** The application’s API endpoints do not have rate limiting, allowing users or attackers to make unlimited requests. This leaves the system vulnerable to brute force attacks or denial-of-service (DoS) attacks, which could overwhelm the server and cause downtime. Adding rate limiting, such as restricting the number of requests allowed per IP address in a given timeframe, would help prevent these kinds of attacks.

**Unsanitized SQL Queries** SQL queries in the code directly use inputs provided by users without sanitizing or parameterizing them. This leaves the application exposed to SQL injection attacks, where attackers could manipulate the database or steal sensitive information. To fix this, the application should use prepared statements or ORM frameworks like Hibernate to ensure user inputs are properly sanitized before being executed in the database.

**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

**CVE-2021-44228 (Log4j Vulnerability)** This is a serious issue in Apache Log4j that lets attackers run remote code by sending specially crafted requests to systems using the library. To resolve this, Log4j should be updated to version 2.17.1 or later as soon as possible. Any systems still using older versions should be patched immediately. Known as "Log4Shell," this vulnerability was widely reported in December 2021 and is well-documented by security experts.

**CVE-2022-23307 (Jackson Databind Vulnerability)** Jackson Databind has a deserialization issue that could allow attackers to execute malicious code through specially crafted inputs. Updating to version 2.13.2 or later will fix the problem. It’s also a good idea to check any dependencies that rely on Jackson to make sure they’re compatible with the updated version. This issue was flagged as critical in early 2022 and is commonly identified by dependency-check tools.

**CVE-2021-22118 (Spring Framework Vulnerability)** A vulnerability in the Spring Framework allows attackers to bypass security features and access restricted information. To fix this, the Spring Framework should be upgraded to version 5.3.9 or later. Keeping up with the latest Spring releases will help prevent similar issues. This vulnerability was reported in 2021 and is documented in the National Vulnerability Database.

**CVE-2019-0232 (Apache Tomcat Vulnerability)** Apache Tomcat has a known issue that allows attackers to execute arbitrary commands through specific inputs. Updating to version 9.0.31 or later will address the problem. Additionally, reviewing server configurations and disabling unnecessary features can help minimize risks. This vulnerability has been known since 2019 and is outlined in Apache security advisories.

**CVE-2020-11996 (Commons-BeanUtils Vulnerability)** The Apache Commons-BeanUtils library has a flaw that lets attackers manipulate inputs to perform unauthorized actions. To resolve this, the library should be updated to the latest secure version, or replaced if it is no longer being actively maintained. This issue was first identified in 2020 and is frequently referenced in dependency management resources.

**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.

**CVE-2021-44228 (Log4j Vulnerability)**  
The Log4j vulnerability, commonly referred to as "Log4Shell," allows attackers to execute remote code by sending specially crafted requests to systems that use this library. To address this issue, it’s crucial to update Log4j to version 2.17.1 or later. Any systems running older versions should be patched immediately to prevent exploitation. Regularly reviewing updates and security advisories for Log4j can help maintain the application’s security over time.

**CVE-2022-23307 (Jackson Databind Vulnerability)**  
Jackson Databind contains a deserialization flaw that attackers can exploit to execute malicious code using crafted input. Updating Jackson Databind to version 2.13.2 or later will fix the issue. It’s also important to check that other dependencies relying on Jackson are compatible with the updated version. Regularly reviewing and monitoring dependencies using tools like Maven dependency-check can help identify similar vulnerabilities early.

**CVE-2021-22118 (Spring Framework Vulnerability)**  
This Spring Framework vulnerability allows attackers to bypass security controls and gain access to restricted data. To resolve this, the framework should be updated to version 5.3.9 or later. Staying current with Spring Framework updates and monitoring release announcements can help prevent similar vulnerabilities in the future and ensure the application remains secure.

**CVE-2019-0232 (Apache Tomcat Vulnerability)**  
Apache Tomcat has a vulnerability that allows attackers to execute commands through specific inputs. Upgrading to version 9.0.31 or later addresses this issue. Additionally, disabling unnecessary configurations and reviewing server settings can reduce exposure to potential risks. Regular server audits and best practices for hardening configurations can help protect against this and other vulnerabilities.

**CVE-2020-11996 (Commons-BeanUtils Vulnerability)**  
The Apache Commons-BeanUtils library has a weakness that lets attackers manipulate inputs to perform unauthorized actions. Updating the library to the latest secure version or replacing it with an actively maintained alternative will address this issue. Regularly reviewing and updating libraries reduces the reliance on outdated or insecure components and lowers the risk of exploitation.