# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **3/22/2025** | **Kim Cortez Ramirez** |  |

## 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

Kim Cortez Ramirez

**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 individualized financial plans, including savings, retirement, investments, and insurance. They deal with sensitive financial information so secure communications are crucial to maintaining confidentiality and integrity of data. A data breach could result in financial losses, reputational damage, and legal consequences.

\*It is not stated whether the company conducts international transactions. However, considering the nature of financial consulting and modern business practices, it is safe to assume that Artemis Financial may work with international clients.  
\*There are a few government regulations they need to follow to keep communications secure. Some of the main ones are the Gramm-Leach-Bliley Act to protect customer data, FISMA for securing federal information systems, and PCI DSS if they handle credit card transactions.  
\*Different external threats they could face include man-in-the-middle attacks where hackers intercept client-server communication, data breaches that expose sensitive financial info, and common web vulnerabilities like SQL injection and cross site scripting that mess with databases. There’s also the risk of DDoS attacks, which can crash the system and cause data loss.  
\*To keep the app secure and up-to-date, it’s important to update open-source libraries, switch to secure frameworks like Spring Boot, and use secure API practices. Implementing strong encryption like TLS 1.3 and following cloud security best practices are also key to protecting data and preventing 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.

\* Input Validation: Makes sure that any input is checked and cleaned up so that harmful data doesn’t break the application or cause security problems. Stopping attacks like injecting bad code or messing with functionality.  
\*Secure APIs: Keeping data safe when sent between the app and other services by using secure methods like HTTPS and verifying who’s accessing the API. Helping protect against data leaks and unauthorized access.  
\*Cryptography: Uses encryption to lock up sensitive financial information. This way, even if someone gets access to the data, they can’t read it without the key.  
\*Client/Server Security: Makes sure that the data between the user’s device and the server is safe from interception or change. It uses secure connections (like HTTPS) and authentication to keep comms protected.  
\*Code Quality and Error Handling: Writing code in a way that avoids common security risks and makes sure that errors don’t expose sensitive information keeping the app secure.

**3. Manual Review**

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

\*In CRUD.java (lines 4-5), input validation is missing for content and content2, allowing untrusted data.  
\* In Greeting.java (lines 4-5), input validation is missing for id and content, allowing untrusted data.  
\* In GreetingController.java (line 15), improper input validation for the name parameter may allow injection attacks.  
\* Format String Vulnerability - Untrusted input used in String.format().  
\* In CRUDController.java (line 13), input validation is missing for the name parameter, allowing untrusted data.  
\* In customer.java (line 12), input validation is missing for the a parameter in the deposit method, allowing untrusted data.  
\* In myDateTime.java (lines 4-6), input validation is missing for mySecond, myMinute, and myHour, allowing invalid time values.  
\* customer.java (lines 8-14): There is no validation for the deposit method, negative values could be passed and decrease the balance.

**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. BouncyCastle JDK 1.46

Code: cpe:2.3:a:bouncycastle:bouncy-castle-crypto-package:1.46

Severity: High (22 CVEs)

Description: Multiple vulnerabilities due to outdated cryptography libraries, including improper certificate validation.

Recommended Solution: Update to the latest version of BouncyCastle to mitigate cryptographic issues.

Attribution: National Vulnerability Database (NVD), CVE-2016-1000338

2. Hibernate Validator 6.0.18.Final

Code: cpe:2.3:a:redhat:hibernate\_validator:6.0.18

Severity: Medium (2 CVEs)Description: Vulnerabilities related to input validation and improper data handling that could lead to XSS or data injection.  
Recommended Solution: Upgrade to the latest version of Hibernate Validator.

Attribution: NVD, CVE-2020-10693

3. Jackson Databind 2.10.2

Code: cpe:2.3:a:fasterxml:jackson-databind:2.10.2

Severity: High (6 CVEs)

Description: Vulnerabilities related to deserialization of untrusted data.

Recommended Solution: Update to the latest version of Jackson Databind.

Attribution: NVD, CVE-2020-25649

4. Log4j API 2.12.1

Code: cpe:2.3:a:apache:log4j:2.12.1

Severity: Low (1 CVE)

Description: Log injection vulnerabilities can cause logging of malicious input.

Recommended Solution: Update to the latest version of Log4j API.

Attribution: NVD, CVE-2021-44228

5. Logback Classic 1.2.3

Code: cpe:2.3:a:qos:logback:1.2.3

Severity: High (2 CVEs)

Description: Improper handling of logging inputs can result in log injection.

Recommended Solution: Update to a more secure version of Logback.

Attribution: NVD, CVE-2021-42550

6. Logback Core 1.2.3

Code: cpe:2.3:a:qos:logback:1.2.3

Severity: High (4 CVEs)

Description: Vulnerabilities related to improper logging and data handling.

Recommended Solution: Update to the latest secure version.

Attribution: NVD, CVE-2021-42550

7. SnakeYAML 1.25

Code: cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25

Severity: Critical (8 CVEs)

Description: Vulnerabilities related to unsafe deserialization.

Recommended Solution: Update to the latest version.

Attribution: NVD, CVE-2022-1471

8. Spring Boot 2.2.4 RELEASE

Code: cpe:2.3:a:vmware:spring\_boot:2.2.4

Severity: Critical (3 CVEs)

Description: Vulnerabilities that allow attackers to bypass authentication.

Recommended Solution: Update to a secure version of Spring Boot.

Attribution: NVD, CVE-2020-5421

9. Spring Boot Starter Web 2.2.4 RELEASE

Code: cpe:2.3:a:vmware:spring\_boot:2.2.4

Severity: Critical (3 CVEs)

Description: Web framework vulnerabilities can allow for unauthorized access.

Recommended Solution: Upgrade to a secure version.

Attribution: NVD, CVE-2020-5421

10. Spring Core 5.2.3 RELEASE

Code: cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3

Severity: Critical (11 CVEs)

Description: Multiple vulnerabilities related to improper handling of HTTP requests.

Recommended Solution: Update to a patched version.

Attribution: NVD, CVE-2020-5398

11. Spring Expression 5.2.3 RELEASE

Code: cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3

Severity: Critical (12 CVEs)

Description: Expression language injection vulnerabilities.

Recommended Solution: Update to a secure version.

Attribution: NVD, CVE-2020-5398

12. Spring Web 5.2.3 RELEASE

Code: cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3

Severity: Critical (16 CVEs)

Description: Improper handling of web requests.

Recommended Solution: Update to a more secure version.

Attribution: NVD, CVE-2020-5398

13. Spring WebMVC 5.2.3 RELEASE

Code: cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3

Severity: Critical (12 CVEs)

Description: Vulnerabilities related to MVC processing.

Recommended Solution: Update to the latest version.

Attribution: NVD, CVE-2020-5398

14. Tomcat Embed Core 9.0.30

Code: cpe:2.3:a:apache:tomcat:9.0.30

Severity: Critical (28 CVEs)

Description: Server-side request forgery (SSRF) vulnerabilities.

Recommended Solution: Update to a patched version.

Attribution: NVD, CVE-2020-13934

15. Tomcat Embed WebSocket 9.0.30

Code: cpe:2.3:a:apache:tomcat:9.0.30

Severity: Critical (29 CVEs)

Description: WebSocket handling vulnerabilities.

Recommended Solution: Update to a secure version.

Attribution: NVD, CVE-2020-13934

**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.

\*To fix the vulnerabilities, Artemis Financial should update old libraries like BouncyCastle, Hibernate Validator, and Jackson Databind to their latest versions. It’s also important to add input validation to make sure data coming in is safe. Using secure API practices will help protect against unauthorized access and SQL injection. Updating logging tools like Log4j and Logback will make the system less vulnerable to attacks through logs. Finally, using stronger encryption methods and keeping everything up to date will make the application safer overall.