# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/17/2025** | **Joshua Williamson** | **Revision 1** |

## 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

Joshua Williamson

**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?

The client, Artemis Financial, is a financial consulting company, ensuring that the communications between them and their clients are secured is of importance. It has been stated that the client helps customers around the world, making certain that the application secures international transactions is needed. It is important that attention be paid to the different security risks, laws, and regulations of the various countries where the client does business. There are rules and regulations in place that punish financial institutions for neglecting to secure their communications with customers. Organizations such as the FDIC have helped to create laws related to the protection of customers’ financial information and the notification of customers when a data breach occurs. It is important that the application be in accordance with all of these laws. The external threats to this application’s security come in the form of attackers trying to gain access to customers’ personal financial information. One common form of these attacks is an SQL injection. This is when an attacker inserts an SQL query into an input form in an application, which can then lead to them being able to access or even modify sensitive data from the application’s database. There are two main modernization requirements that must be considered for this application. The first is ensuring that the newest versions of open source libraries are being used to minimize vulnerabilities. The second is that https rather than http be used as the web application’s communication protocol in order to bolster the security of the client/server interactions.

**2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial software application. Justify your reasoning for why each area is relevant to the software application.

This being a RESTful API I would focus on Input Validation, Secure API Interactions, Code Errors, and Encapsulation.

Input Validation is important as you need to “sanitize” user inputs, even if from a trusted user. A malformed input, on purpose or not, could cause errant behavior. Securing the API goes one step farther as well as it ensures the connection is trustworthy. There is some code checking built into Spring but this error handling needs to be generic and not provide any signature that a hacker could use to understand more about the code structure. And finally, because we are using methods, the methods should use encapsulation to protect the sensitive data from unwanted changes such as set variables to private and use Get/Set functions.

Cryptography should also be considered as all data flows should be encrypted. RSA-2048 strength should be considered for all databases and data transmission. Encryption-type and capabilities should be checked in each operating country as laws vary on cryptography.

**3. Manual Review**

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

1. Vulnerabilities in CRUD class:
   1. public accessor and constructor methods
2. Vulnerability in CRUDController class:
   1. Vulnerable to code injections
3. Vulnerabilities in customer class:
   1. public accessor and mutator methods
4. Vulnerabilities in DocData class:
   1. public accessor and constructor methods
   2. input being handled in the URL rather than via the post method could lead to data leaking into browser history
5. Vulnerabilities in Greeting class:
   1. public accessor and constructor methods
6. Vulnerabilities of GreetingController class:
   1. No input validation
7. Lack of data encryption
8. Lack of functioning API

**4. Static Testing**

Run a dependency check on Artemis Financial 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. bcprov-jdk15on-1.46.jar**

A. CVE-2018-5382

B. BouncyCastle keystores are weak and can be brute-forced, meaning attackers could bypass integrity checks and tamper with stored keys. The recommendation is to upgrade to **1.47+**, which introduces the stronger BKS v2 format.

C. Documented by NVD.

**2. hibernate-validator-6.0.18.Final.jar**

A. CVE-2019-10219

B. Earlier versions didn’t sanitize HTML correctly, which could allow malicious HTML or JavaScript to be injected into web pages. Version 6.0.18 already contains the fix.

C. NVD release notes.

**3. jackson-core-2.10.2.jar**

A. CVE-2025-52999

B. Deeply nested JSON can overwhelm the parser, causing a crash. Attackers can exploit this by sending malicious payloads. The recommendation is to upgrade to **2.15.0+.**

C. GitHub advisory GHSA-h46c-h94j-95f3.

**4. jackson-databind-2.10.2.jar**

A. CVE-2020-36518

B. Multiple flaws allow attackers to crash the service or even trigger unsafe XML entity expansion. The recommendation is to upgrade to patched versions (**2.10.5.1+** or **2.13+**) and disable risky deserialization features when processing untrusted input.

C. All tracked in NVD and Red Hat advisories.

**5. log4j-api-2.12.1.jar**

A. CVE-2021-44228

B. When paired with log4j-core, this version is vulnerable to “Log4Shell,” which lets attackers execute remote code through JNDI lookups in log messages. The recommendation is to upgrade to **2.17.1+**.

C. Apache Log4j security page and NVD.

**6. logback-classic-1.2.3.jar**

A. CVE-2021-42550

B. A malicious configuration file could load remote resources through JNDI, leading to remote code execution. Recommendation is to upgrade to **1.2.9+** and restrict who can modify logging configs.

C. NVD and Logback advisory.

**7. logback-core-1.2.3.jar**

A. CVE-2021-42550

B. Same JNDI-related risk as logback-classic. An attacker with access to logging configuration could compromise the JVM. Recommendation is to upgrade to **1.2.9+**.

C. NVD and Logback advisory.

**8. snakeyaml-1.25.jar**

A. CVE-2022-25857

B. This version allows unsafe object deserialization, which can lead to arbitrary code execution if untrusted YAML is parsed. The recommendation is to upgrade to **1.31+** and avoid parsing YAML from unknown sources.

C. GitHub advisories and NVD.

**9. spring-aop-5.2.3.RELEASE.jar**

A. CVE-2020-5398

B. Expression injection in Spring AOP could expose sensitive information or allow malicious code paths. Recommendation is to upgrade to **5.2.4+** and validate user inputs.

C. Spring advisories.

**10. spring-boot-2.2.4.RELEASE.jar**

A. CVE-2020-5421

B. The whitelabel error page exposed detailed error information to users, which could help attackers gather system info. Recommendation is to upgrade to **2.2.7+** and disable detailed error output in production.

C. Spring Boot advisories.

**11. spring-boot-starter-web-2.2.4.RELEASE.jar**

A. CVE-2020-5421

B. Inherits the same vulnerability as core Spring Boot. Error pages leak sensitive stack traces. Recommendation is to upgrade to **2.2.7+** and configure generic error responses.

C. Spring Boot advisories.

**12. spring-context-5.2.3.RELEASE.jar**

A. CVE-2020-5398

B. Vulnerable to SpEL injection, which can let attackers evaluate arbitrary expressions and possibly access restricted data. Recommendation is to upgrade to **5.2.4+**.

C. Spring advisories.

**13. spring-core-5.2.3.RELEASE.jar**

A. CVE-2020-5398

B. Contains the same SpEL injection issue, making it possible to manipulate Spring internals. Recommendation is to upgrade to **5.2.4+**.

C. Spring advisories.

**14. spring-expression-5.2.3.RELEASE.jar**

A. CVE-2020-5398

B. The expression parser itself is exploitable, allowing crafted input to bypass restrictions. Recommendation is to upgrade to **5.2.4+**.

C. Spring advisories.

**15. spring-web-5.2.3.RELEASE.jar**

A. CVE-2020-5398

B. When combined with SpEL, attackers could craft malicious web requests that expose sensitive data. Recommendation is to upgrade to **5.2.4+**.

C. Spring advisories.

**16. spring-webmvc-5.2.3.RELEASE.jar**

A. CVE-2020-5398

B. Same SpEL-related risk through data binding and expression evaluation. Recommendation is to upgrade to **5.2.4+**.

C. Spring advisories.

**17. tomcat-embed-core-9.0.30.jar**

A. CVE-2020-1938

B. Vulnerable to the “Ghostcat” flaw, where the AJP connector could be abused to read or include arbitrary files. Recommendation is to upgrade to **9.0.31+** or disable AJP if not needed.

C. Apache Tomcat advisories.

**18. tomcat-embed-websocket-9.0.30.jar**

A. CVE-2020-1938

B. Inherits the same Ghostcat issue as tomcat-core. Recommendation is to upgrade to **9.0.31+** or disable AJP.

C. Apache Tomcat advisories.

**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 software application.

After interpreting your results from the manual review and static testing, identify the steps to remedy the identified security vulnerabilities for Artemis Financial software application.

The first step in mitigating the security concerns in Artemis Financial web application is to update all dependencies to their newest versions. This will implement a plethora of bug fixes that will help to protect the application from many of the attacks that it is currently vulnerable to. The next step would be to switch to HTTPS as the web application’s communication protocol. This will help to make the client/server interactions more secure. Another vulnerability that needs to be addressed is all of the public accessor, mutator, and constructor methods in the various classes. Also, input validation and error handling methods need to be implemented in a couple of areas. Finally, it is essential that the development team incorporate cryptography strategies in order to protect customer information.