# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/19/2025** | **Collin Laconto** |  |

## 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

Collin Laconto

**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 paramount to Artemis Financial. Because the company deals with financial transactions, we need to ensure that all communication is completely secure. Artemis Financial will be handling internation transactions. This may include confidential savings, retirement, investments, and insurance information. There are many different governmental compliance requirements to properly process these transactions. Some of these regulations are required to ensure data is securely processed while others ensure that financial transactions are not being processed to people or groups on global sanctions lists. For example, for data protection, the EU requires adherence to the GDPR regulation. The EU also requires the PSD2 regulation which concerns secure communication channels for processing electronic payments. For a company handling financial transactions, it will be a lucrative target for attackers. In the present day and future, we must look at all possible areas of attack. From the OWASP top ten, potential areas of vulnerabilities are as follows: broken access controls can cause undesired access to secure areas of information. Cryptographic failures are another large cause for vulnerabilities. That is why there are so many compliance regulations for transferring financial information. Injection is still something that needs to be considered. Unwanted commands can be sent to the database producing data that should not otherwise be accessed.

**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.

For this application, many of the vulnerabilities in the flow diagram apply to this application. Input validation is relevant because the application will be accepting user input. Cryptography is hugely important in this application because we deal with sensitive financial information. Secure distributed composing is important to ensure all aspects of the application are equally secure. We don’t want any aspects of the application to be especially vulnerable. Secure coding patterns are important for every aspect of the software. Attackers will use any weak point to exploit the system.

**3. Manual Review**

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

The first vulnerability I discovered in the code base is the lack of HTTPS data transfers. All of the requests are sent using HTTP, which is unencrypted. Additionally, all of the SpEL expressions use the user input without any input validation. The data the user enters is directly processed at runtime. In DocData.java, the program has a test database linked with a default username and password of “root”. In customer.java, there are no checks to validate the balance being deposited into the account balance. In CRUDController.java, there is no input validation to validate the “business\_name” being submitted. Throughout the program, there is no handling of a /error endpoint which can reveal unwanted information to a potential attacker. Finally, throughout the whole program, there is no verification to ensure the right person has access to the right information. If someone knew the right ID, they would have access to information that they shouldn’t.

**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

The software application provided has some known vulnerabilities as detected by Maven. According to the report, there are 18 vulnerable dependencies. First, the Bouncy Castle Crypto package has a vulnerability where the software does not properly validate the security certificate ([**CVE-2016-1000338**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2016-1000338)). Hibernate validator has a vulnerability where it allows unintended code injection ([**CVE-2025-35036**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-35036)). Jackson core has a vulnerability where a stack overflow error can occur if a file is parsed with deeply nested data (**CVE-2025-52999**). Jackson databind contains a vulnerability where an XXE attack can be launched allowing the attacker to interfere with XML processing ([**CVE-2020-25649**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-25649)). Apache log does not properly validate the certificate and can be intercepted ([**CVE-2020-9488**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-9488)). Logback classic contains a vulnerability where an attacker can send poisoned data to cause a denial of service attack ([**CVE-2023-6378**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-6378)). Logback core contains a very similar vulnerability ([**CVE-2023-6378**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-6378)). SnakeYaml has a known vulnerability that allows remote code execution ([**CVE-2022-1471**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-1471)). This same dependency has multiple vulnerabilities that can cause denial of service attacks ([**CVE-2022-25857**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-25857) , [**CVE-2022-38749**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-38749) , [**CVE-2022-38751**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-38751) , [**CVE-2022-38752**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-38752) , [**CVE-2022-41854**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-41854) , [**CVE-2022-38750**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-38750)). The Spring framework has a known vulnerability that can allow an attacker to remotely execute code ([**CVE-2022-22965**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22965)). This specific dependency has many known vulnerabilities associated with it resulting in a wide range of attack possibilities. ([**CVE-2022-22950**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22950) , [**CVE-2022-22971**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22971) , [**CVE-2023-20861**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-20861) , [**CVE-2023-20863**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-20863) , [**CVE-2022-22970**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22970)) can all result in denial of service attacks with this version of Spring. Spring boot also contains a few vulnerabilities. ([**CVE-2023-20873**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-20873)) can result in a security bypass if the application is deployed to Cloud Foundry. ([**CVE-2022-27772**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-27772)) is susceptible to temporary directory hijacking. ([**CVE-2023-20883**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-20883)) can cause a denial of service attack. Spring context has a known exploited vulnerability that allows an attacker to remotely execute code ([**CVE-2022-22965**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22965)). This release of Spring context contains many different vulnerabilities that attackers can use ([**CVE-2024-22259**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-22259) , [**CVE-2021-22118**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-22118) , [**CVE-2020-5421**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-5421) , [**CVE-2022-22968**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-22968) , [**CVE-2021-22060**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-22060) , [**CVE-2021-22096**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-22096)) Spring core contains many of the same vulnerabilities listed above. As a whole, this version of the Spring framework is extremely susceptible to exploit and attack. It is recommended to update the version to the latest version where these exploits are patched. Apache Tomcat core has a known vulnerability that allows many different exploits ([**CVE-2020-1938**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-1938) , [**CVE-2024-50379**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-50379) , [**CVE-2024-52316**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-52316) , [**CVE-2024-56337**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-56337) , [**CVE-2025-24813**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-24813) , [**CVE-2025-31651**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-31651) , [**CVE-2025-49124**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-49124) , [**CVE-2020-11996**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-11996) , [**CVE-2020-13934**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-13934) , [**CVE-2020-13935**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-13935) , [**CVE-2020-17527**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-17527) , [**CVE-2021-25122**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-25122) , [**CVE-2021-41079**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-41079) , [**CVE-2022-29885**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-29885) , [**CVE-2022-42252**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-42252) , [**CVE-2023-44487**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-44487) , [**CVE-2023-46589**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-46589) , [**CVE-2024-24549**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-24549) , [**CVE-2024-34750**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-34750) , [**CVE-2024-38286**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-38286) , [**CVE-2025-48988**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-48988) , [**CVE-2025-48989**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-48989) , [**CVE-2025-49125**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-49125) , [**CVE-2025-52434**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-52434) , [**CVE-2025-52520**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-52520) , [**CVE-2025-53506**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-53506) , [**CVE-2025-46701**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-46701) , [**CVE-2020-9484**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-9484) , [**CVE-2021-25329**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-25329) , [**CVE-2021-30640**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-30640) , [**CVE-2025-55668**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2025-55668) , [**CVE-2024-23672**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-23672) , [**CVE-2022-34305**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2022-34305) , [**CVE-2023-41080**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-41080) , [**CVE-2021-24122**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-24122) , [**CVE-2021-33037**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-33037) , [**CVE-2023-42795**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-42795) , [**CVE-2023-45648**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-45648) , [**CVE-2024-21733**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-21733) , [**CVE-2024-54677**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2024-54677) , [**CVE-2019-17569**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2019-17569) , [**CVE-2020-1935**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-1935) , [**CVE-2020-13943**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-13943) , [**CVE-2023-28708**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2023-28708) , [**CVE-2021-43980**](https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2021-43980)) This version of Apache Tomcat has an incredible number of known vulnerabilities and it must be updated to the latest version as soon as possible to patch these. For the remaining vulnerabilities detected during static testing, the solution is to update all the dependencies used to the latest version. Most if not all of these have already been patched, but this web application is using outdated versions of all of them. By continuing to use the outdated versions, the application is extremely vulnerable to attack.

**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.

The first step in mitigating the security vulnerabilities is to implement HTTPS processing. Without HTTPS, attackers could intercept data moving between the program and the client. We must also update all of the dependencies that are used in this program to patch all of the known exploits. Once the exploits are patched and static testing returns no vulnerabilities, we can begin to implement some form of input validation. HTTPS, updating dependencies, and input validation are the three most important things we can do to mitigate the security risks in this web application.