# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **3/23/2025** | **Sabrina Ozburn** |  |

## 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

Sabrina Ozburn

**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 is a financial services firm that handles both internal and external sensitive information related to its clients. Given the nature of its operations, ensuring the security of all internal and external communications is paramount to the company’s success. The firm is involved in processing financial transactions that come from various sources, both domestic and international. While there are currently no specific state or federal regulations mandating secure communication practices, it remains essential for Artemis Financial to uphold strict security protocols in its communications. This is critical to protect confidential client data and prevent any potential leaks of sensitive financial information.

Artemis Financial handles highly sensitive data, such as financial records, biometric information, Social Security numbers, and account details. As such, it is imperative to safeguard this information by ensuring it is encrypted and masked during storage and transmission. This measure is essential to maintaining clients' privacy and protecting the company from data breaches. Additionally, any proprietary information, including trade secrets, must be protected through stringent security practices.

As the financial industry and technology continue to evolve, Artemis Financial must prioritize keeping its systems and software current. This includes ensuring that the libraries and frameworks used within their applications are regularly updated to incorporate the latest bug fixes, security patches, and responses to emerging threats. The firm can effectively address vulnerabilities and enhance its overall cybersecurity posture by maintaining an ongoing focus on modernization. Keeping these systems updated is not just a precaution; it is a necessary step to mitigate risks, maintain regulatory compliance, and protect the integrity of both client and corporate data.

**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.

After reviewing the security aspects of Artemis Financial, I have identified several potential vulnerabilities that need attention:

• **Input Validation** – Ensuring proper validation of user inputs is essential to prevent security risks. Since this program allows user input, it's vital to implement robust input validation mechanisms to mitigate potential issues such as invalid data entry or security threats like SQL injection attacks. Without this, the system could be vulnerable to malicious input disrupting functionality or compromising security.

• **APIs (Application Programming Interfaces)** – Given that this application will run internally and externally (e.g., through end-user web browsers), a well-designed and secure API is crucial. The API defines how external systems and users interact with the application, specifying which methods are acceptable for accessing data. Since this software may integrate with third-party systems, ensuring that the API is secure is even more important, limiting access and data manipulation only to authorized entities and preventing potential exploitation by malicious users or applications.

• **Cryptography** – Implementing strong cryptographic measures is necessary, especially since the application will handle sensitive customer information during international transfers. This data must be encrypted in accordance with both North American regulations and the laws of the destination countries. Effective encryption protects the confidentiality of proprietary data and ensures compliance with global security and privacy standards.

• **Error Handling** – Effective error management is critical for maintaining security. This should be integrated with both the API and input validation processes. Proper error handling ensures that when errors occur, they don’t inadvertently expose sensitive information or provide attackers with valuable insights into the system. Moreover, it helps prevent unauthorized access by ensuring that any failures don’t lead to privilege escalation or unintentional access to restricted resources.

• **Code Quality** – Maintaining high code quality is essential when dealing with end users, especially concerning input handling and API interactions. Well-structured and error-free code reduces the risk of unintentional data exposure or security flaws. In addition, high-quality code helps enforce user access levels, ensuring that only authorized users can access specific features or methods, preventing potential privilege escalation or unauthorized data access.

In conclusion, ensuring the robustness of security in these areas will significantly reduce the potential for vulnerabilities and enhance the overall integrity and safety of the Artemis Financial 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.

To begin the vulnerability assessment process, I first examined the input validation. I reviewed the POM.XML file to check for any Apache validator, then moved on to the greeting controller. The input there seemed to lack any validation, and I couldn't confirm whether it was being validated as there was no output to assess. I then searched for an API but couldn't find any functional ones. However, the program was still able to access data in an unsecured manner. The data was accessed through the URL rather than via the POST method, which can expose it in browser history and make it vulnerable to exploitation. Despite not displaying any information, the program continues to accept input via the URL, which could still be exploited since raw user input is still being processed. Without an API, there is no clear way for users to understand how to interact with the program without reviewing the code. When designing a RESTful API, it should provide a clear, structured way for users to interact with it.

After addressing input validation and the absence of an API, I moved on to checking for cryptographic practices. I found no encryption methods in place. Artemis Financial would need to implement encryption mechanisms for securely storing information and handling international transactions in compliance with global regulations. In my review of error handling, I noticed that the DocData.java class lacked error handling, only containing try-and-catch blocks, and no further error-handling mechanisms were present. Lastly, while the code quality was generally high, the absence of an API and its incomplete functionality made the program difficult to use. Moreover, the lack of input validation and handling input through the URL instead of the POST method poses risks by potentially exposing sensitive data in browser history, creating a security vulnerability.

**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

|  |  |  |  |
| --- | --- | --- | --- |
| Dependency | Vulnerability | Description | Solution |
| bcprov-jdk15on-1.46.jar | cpe:2.3:a:bouncycastle:bouncy-castle-crypto-package:1.46:\*:\*:\*:\*:\*:\*:\*  cpe:2.3:a:bouncycastle:bouncy\_castle\_crypto\_package:1.46:\*:\*:\*:\*:\*:\*:\*  cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:\*:\*:\*:\*:\*:\*:\*  cpe:2.3:a:bouncycastle:the\_bouncy\_castle\_crypto\_package\_for\_java:1.46:\*:\*:\*:\*:\*:\*:\* | Legion of the Bouncy Castle Legion of the Bouncy Castle Java Cryptography APIs 1.58 up to but not including 1.60 contains a CWE-470: Use of Externally-Controlled Input to Select Classes or Code ('Unsafe Reflection') vulnerability in XMSS/XMSS^MT private key deserialization that can result in Deserializing an XMSS/XMSS^MT private key can result in the execution of unexpected code. This attack appears to be exploitable via A handcrafted private key that can include references to unexpected classes, which will be picked up from the classpath for the executing application. This vulnerability appears to have been fixed in 1.60 and later. | Update bouncycastle to: Version update to 1.60 |
| hibernate-validator-6.0.18.Final.jar | cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\* | A flaw was found in Hibernate Validator version 6.1.2. Final. A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages. | Upgrade to hibernate-validator-6.0.20 |
| jackson-databind-2.10.2.jar | cpe:2.3:a:fasterxml:jackson-databind:2.10.2:\*:\*:\*:\*:\*:\*:\*  cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2:\*:\*:\*:\*:\*:\*:\* | A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity. | Update to current version |
| log4j-api-2.12.1.jar | cpe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\* | Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender. | Upgrade to 2.13.2 which supports this feature. Previous versions can set the system property mail.smtp.ssl.checkserveridentity to true to globally enable hostname verification for SMTPS connections. |
| snakeyaml-1.25.jar | cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\* | The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation, a related issue to CVE-2003-1564.  Published: December 11, 2019; 10:15:10 PM -0500 | Migrate to SnakeYAML Engine. It has a configuration option to restrict aliases for collections (the aliases for scalars cannot grow and they are not restricted) |
| logback-classic-1.2.3.jar | cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\* | A serialization vulnerability in logback receiver component part of logback version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data. | Update to the current version |
| logback-core-1.2.3.jar | cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\* | A serialization vulnerability in logback receiver component part of logback version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data. | Update to the current version |
| spring-boot-2.2.4.RELEASE.jar | cpe:2.3:a:vmware:spring\_boot:2.2.4:release:\*:\*:\*:\*:\*:\* | In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+. | Update to the current version |

**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.

Most of these vulnerabilities can be addressed by updating to the latest software versions. In addition, adjusting the version of Snakeyaml and limiting the use of aliases for collections can further enhance security and protect against this specific vulnerability. You can close off many potential attack vectors by ensuring that the software is kept up to date. Additionally, restricting certain features—like aliases in Snakeyaml—can help reduce the risk of exploitation by preventing malicious configurations or code execution tied to these vulnerabilities.