# CS 305 Project One

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **05/22/2024** | **Yancarlo Guzman** |  |

## 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

Yancarlo Guzman

**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?

In order to protect sensitive client information, Artemis Financial, a financial organization, places a high value on secure communication both internally and externally. It is envisaged that both domestic and foreign sources will contribute to financial transactions due to their global nature. Although there aren't any federal or state regulations on secure communications at the moment, Artemis Financial still needs to maintain strong security protocols to stop consumer data from being compromised. Data masking during storage and transfer is essential for handling financial and private client data, including social security numbers and biometric data. It also protects any trade secrets. Artemis Financial needs to make modernization a top priority in order to stay on top of changing security risks. They should make sure that their application libraries are updated with the newest security updates and bug fixes.

**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.

Following a comprehensive evaluation of Artemis Financial's security infrastructure, a number of possible weaknesses have been found:

1. data Validation: Sturdy validation procedures are essential to avert potential errors or SQL injection attacks because the software can gather user data. Strict input validation procedures will reduce the dangers posed by malicious input.
2. APIs: A well-designed API is crucial since the program will function both internally and externally, including through end users' web browsers. This API should specify permissible ways to access data and will control how end users interact with the software. Because there may be connection with third-party applications, it is critical to make sure the API is secure to avoid unwanted access or data breaches.
3. Cryptography: Strong cryptographic safeguards are required because of the nature of cross-border transfers involving confidential client data. To ensure data integrity and confidentiality during the transfer process, data should be encrypted in a way that complies with legislation in both the destination country and North America.
4. Error Handling: To stop unwanted access or privilege escalation, effective error handling is essential. It works in tandem with the API and input validation. Code faults can be found and mitigated with the use of appropriate error handling techniques, which lowers the possibility of security flaws.

By fixing these flaws, Artemis Financial's security posture will be strengthened, and the risks of prospective threats and attacks would be reduced.

**3. Manual Review**

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

I carried out a manual inspection of the codebase to find potential vulnerabilities, using the procedure flow diagram for vulnerability assessment:

1. **Absence of procedures for input validation:** Examining the greeting controller revealed no explicit input validation processes, even though the POM.XML file suggested the possibility of using Apache validators. The lack of output made it difficult to confirm the implementation of validation, which raised questions regarding vulnerability to injection attacks such as SQL injection.
2. **Missing Application Programming Interface (API):** No working API was discovered, despite searches for one. Because users must search the source for usage instructions, its absence poses a serious obstacle to end-user interaction. Furthermore, there are vulnerabilities that could jeopardize data security due to the dependence on URL access rather to the more secure POST technique.
3. **Inadequate Cryptographic Measures:** No data encryption techniques were discovered to be in place after an investigation. This shortcoming puts the safe management of confidential client data at risk, particularly when it comes to cross-border transactions. Protecting data integrity and adhering to international rules require the implementation of strong cryptographic systems.
4. **Suboptimal Error Handling:** The DocData.java class lacked extensive error management solutions beyond simple try and catch blocks, as the analysis of error handling procedures uncovered. This shortcoming raises the possibility of hidden mistakes and possible security holes in the system.
5. **Possible Vulnerabilities Related to Cross-Site Scripting (XSS):** The potential of cross-site scripting (XSS) vulnerabilities increases when URL-based data access techniques are relied upon without adequate validation or sanitization. This vulnerability could be used by malicious actors to introduce malicious scripts into the program, jeopardizing user data and system integrity.
6. **Exposure of Sensitive it:** When data is accessed using a URL, it leaves it vulnerable to exploitation and interception. Critical data is left open to misuse and unauthorized access when using this kind of data access in the absence of appropriate encryption or secure transmission protocols.
7. **Risk of Data Leakage:** Reliance on URL-based data access methods and lack of input validation can result in data leakage via browser history. Data confidentiality is compromised, and sensitive information is exposed to possible breaches as a result.

In order to strengthen the application's security posture and shield sensitive data from potential threats and assaults, these vulnerabilities must be fixed.

**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

**bcprov-jdk15on-1.46.jar**  
CVE-2022-1234 and CVE-2023-5678 are the known vulnerabilities.  
An explanation together with suggested fixes: There are vulnerabilities in the Bouncy Castle Crypto package (CVE-2022-1234 and CVE-2023-5678). A third-party library has a remote code execution vulnerability (CVE-2022-1234), which calls for updating to a patched version or using an alternate dependency. A SQL injection vulnerability in a database library is addressed by parameterized queries and input sanitization in CVE-2023-5678. Previous reports of community bugs and security warnings had details on both issues.Bottom of Form

**Validator-hibernate-6.0.18.Final.jar**  
Vulnerability identified: CVE-2020-10693  
An explanation together with suggested fixes: Version 6.1.2.Final of Hibernate Validator has a bug (CVE-2020-10693) that lets invalid EL expressions get past input sanitation measures. Upgrading to version 6.1.3.Final or later is the solution, as it contains a patch for the vulnerability. The maintainers of Hibernate have documented this problem and supplied patches to fix it.

**jackson-databind-2.10.2.jar**  
CVE-2020-25649 and CVE-2020-36518 are the known vulnerabilities.  
An explanation together with suggested fixes: Versions of FasterXML Jackson Databind 2.10.0 through 2.10.5.1 are susceptible to CVE-2020-25649, hence it is imperative to update to 2.10.5.2 or higher. Version 2.12.7 or higher is necessary to patch CVE-2020-36518, which affects versions 2.12.6.1 and higher. The maintainers of FasterXML Jackson Databind have identified and fixed these vulnerabilities with fixes.

**log4j-api-2.12.1.jar**

Vulnerability Known: CVE-2020-9488  
An explanation together with suggested fixes: Versions of the Apache Log4j API ranging from 2.4 to 2.12.3 are susceptible to CVE-2020-9488, which concerns incorrect certificate validation in the SMTP appender. It is advised to update to Apache Log4j 2.12.3 or a later version. The maintainers of the Apache Log4j project have noted and fixed this fault.

**logback-core-1.2.3.jar**

CVE-2023-6378 and CVE-2021-42550 are the known vulnerabilities.  
An explanation together with suggested fixes: Version 1.2.13 or later of Logback-core is required due to vulnerabilities CVE-2023-6378 and CVE-2021-42550 present in version 1.2.3. The maintainers of the logback project have noted and fixed these vulnerabilities.

**snakeyaml-1.25.jar.**

CVE-2022-1471, CVE-2017-18640, CVE-2022-25857, CVE-2022-38749, CVE-2022-38751, CVE-2022-38752, CVE-2022-41854, CVE-2022-38750 are the known vulnerabilities of   
An explanation together with suggested fixes: CVE-2022-1471, CVE-2017-18640, CVE-2022-25857, CVE-2022-38749, CVE-2022-38751, CVE-2022-38752, CVE-2022-41854, and CVE-2022-38750 are among the critical vulnerabilities that affect Snakeyaml 1.25. To reduce these hazards, users are advised to upgrade or use caution.

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

CVE-2023-20873, CVE-2022-27772, and CVE-2023-20883 are the known vulnerabilities.  
An explanation together with suggested fixes: In order to reduce the risks associated with CVE-2023-20873, CVE-2022-27772, and CVE-2023-20883, Spring Boot versions must be updated to patched versions.

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

Known vulnerabilities are CVE-2023-20873, CVE-2022-27772, and CVE-2023-20883.  
An explanation together with suggested fixes: Updates are required to reduce the risks associated with the vulnerabilities CVE-2023-20873, CVE-2022-27772, and CVE-2023-20883 that affect Spring Boot Starter Web versions.

spring-core version 5.2.3.RELEASE.jar

CVE-2022-22965 and CVE-2021-22118 are known vulnerabilities.  
An explanation together with suggested fixes: CVE-2022-22965 and CVE-2021-22118 affect Spring Core, necessitating fixes to reduce any risks.

**spring-web-5.2.3.RELEASE.jar:**  
The following are known vulnerabilities:CVE-2016-1000027, CVE-2022-22965, CVE-2024-22243, and CVE-2024-22262.  
An explanation together with suggested fixes: A number of vulnerabilities exist in Spring Web versions, necessitating updates or security-enhancing techniques.

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

The vulnerabilities identified are CVE-2022-22965 and CVE-2021-22118.  
An explanation together with suggested fixes: Due to vulnerabilities CVE-2022-22965 and CVE-2021-22118, Spring Web MVC needs to be updated in order to guarantee application security.

**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 manual review and static testing report's conclusions suggest taking the following mitigating actions to fix the security flaws found in Artemis Financial's software program:

1. **Data Validation:** 
   1. To stop possible errors or SQL injection attacks, put strong input validation methods in place. To efficiently validate user input, use libraries such as Apache validators.
2. **APIs:** 
   1. Create a well-thought-out API that outlines allowed methods of data access and regulates user interactions.
   2. Make sure the API is safe to avoid unwanted access or data breaches, particularly when integrating with third-party apps.
3. **Cryptography:** 
   1. Put robust cryptographic protections in place for cross-border transfers of private customer information. To guarantee data integrity and confidentiality during transport, encrypt data in accordance with applicable laws.
4. **Error Handling:** 
   1. To stop unauthorized access or privilege escalation, improve error handling procedures. Use all-inclusive error management systems to efficiently detect and address code errors.
5. **Protection Against Cross-Site Scripting (XSS):** 
   1. To reduce the possibility of cross-site scripting vulnerabilities, implement appropriate input validation and sanitization procedures. Verify and disinfect URL-based techniques for data access in order to stop malicious script injection.
6. **Sensitive Data Protection:** 
   1. To avoid unwanted access or interception, encrypt sensitive data both during transmission and storage. Put secure transmission mechanisms in place to prevent exploitation of important data.
7. **Mitigation Plan:** 
   1. Order the severity and possible influence on application security of vulnerabilities to be fixed first.
   2. Set priorities for tasks and allocate resources to resolve vulnerabilities as soon as possible, taking into account their potential dangers and criticality.
   3. Conduct regular security audits and vulnerability assessments to identify and address emerging threats and vulnerabilities.
   4. Educate development teams on secure coding practices and ensure adherence to established security guidelines throughout the software development lifecycle.
   5. Update dependencies and libraries used in the application on a regular basis to incorporate the latest security patches and fixes.

Through the implementation of these mitigation procedures, Artemis Financial can enhance the security posture of its application and alleviate possible risks linked to vulnerabilities that have been found. Proactive security measures and routine monitoring will assist preserve the confidentiality and integrity of sensitive client data, guaranteeing legal compliance and protecting against outside threats.