****

# Artemis Financial Vulnerability Assessment Report

Table of Contents

[Document Revision History 3](#_Toc32574607)

[Client 3](#_Toc32574608)

[Instructions 3](#_Toc32574609)

[Developer 4](#_Toc32574610)

[1. Interpreting Client Needs 4](#_Toc32574611)

[2. Areas of Security 4](#_Toc32574612)

[3. Manual Review 4](#_Toc32574613)

[4. Static Testing 4](#_Toc32574614)

[5. Mitigation Plan 4](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **[Your name]** |  |

## Client



## Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In the report, identify your findings of security vulnerabilities and provide recommendations for 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 choose to include images or supporting materials. If you include them, make certain to insert them in all 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

[Insert your name here.]

## Interpreting Client Needs

Artemis Financial is a consulting company specializing in financial planning, and their web-based software application deals with sensitive financial data. Therefore, the following findings are identified:

* **Value of Secure Communications:** Secure communications are critically valuable to Artemis Financial. They handle financial data, and secure communications ensure data confidentiality and integrity.
* **International Transactions:** The presence of international transactions is not explicitly mentioned, but if they occur, Artemis Financial needs to consider compliance with international data protection regulations.
* **Governmental Restrictions:** We need to check for any specific governmental restrictions or industry-specific regulations that might affect secure communications.
* **External Threats:** Potential external threats include cyberattacks, data breaches, and unauthorized access to sensitive financial information. These threats are immediate and ongoing concerns.
* **Modernization Requirements:** Modernization should consider the role of open-source libraries, evolving web application technologies, and best practices in secure coding to ensure robust software security.

## Areas of Security

* **Input Validation:** This is crucial to protect against injection attacks and ensure data integrity.
* **APIs:** Security of APIs is essential, especially if they handle sensitive financial data.
* **Cryptography:** Encryption and secure key management are important for protecting financial information.
* **Code Quality:** Secure coding practices are essential to prevent vulnerabilities.
* **Encapsulation:** Protecting data structures and access control is vital.

Justifications for these areas:

* Input Validation ensures that user inputs are validated, preventing injection attacks.
* APIs handle data exchange; securing them prevents data breaches.
* Cryptography safeguards sensitive data.
* Code Quality ensures that vulnerabilities are not introduced through coding errors.
* Encapsulation restricts unauthorized access to data.

## Manual Review

* **Input Validation Review:**

The review began by inspecting the project's POM.XML file to identify any usage of Apache validator or similar input validation libraries. Unfortunately, no evidence of explicit input validation implementation was found during this initial inspection. A more detailed analysis was required to ascertain whether input validation was implemented.

Subsequently, attention was turned to the "GreetingController" class, which handles user input. Regrettably, no validation mechanisms were apparent in the code. It is essential to note that without proper input validation, the application may be susceptible to common vulnerabilities like SQL injection or Cross-Site Scripting (XSS).

* **API Security Review:**

The next focus was on the presence and security of APIs within the application. Despite diligent searching, no functional APIs were discovered within the codebase. This absence of APIs can present usability challenges for end-users, as interacting with the application becomes unclear without a well-defined API structure.

Furthermore, even though the program can access data, it does so through URLs rather than employing the more secure POST method. This approach can potentially expose sensitive data through browser history, increasing the risk of data leakage.

Additionally, allowing input via URLs without proper validation can lead to security vulnerabilities, as the program is still accepting raw user input with minimal validation or sanitization.

In the context of RESTful API design, a clear and structured means of user interaction is indispensable. This includes well-documented endpoints, request-response formats, and authentication mechanisms, all of which were conspicuously missing.

* **Cryptography Assessment:**

During the manual review, no evidence of data encryption, whether for data storage or secure transmission, was identified. Given that Artemis Financial deals with sensitive financial information, the absence of cryptographic mechanisms is a substantial security gap. Implementing robust data encryption is imperative to protect stored information and adhere to international data protection regulations, especially for international transactions.

* **Error Handling Review:**

The review extended to error handling within the application, focusing on the "DocData.java" class. This class exhibited error-handling practices through try-catch blocks, which is a positive aspect. However, the assessment did not encompass other sections of the codebase. A more comprehensive evaluation of error-handling practices across the application may be necessary to ensure robust fault tolerance and graceful error recovery.

* **Code Quality Evaluation:**

In terms of code quality, the application demonstrated a commendable level of craftsmanship. Code readability and organization were noteworthy. However, it's essential to reiterate that the absence of a functional API severely impacts the application's usability and user-friendliness.

## Static Testing

Static testing using the dependency-check plug-in in Maven identified the following security vulnerabilities:

* **bcprov-jdk15on-1.46.jar:** 18 CVEs with high severity.
* **hibernate-validator-6.0.18.Final.jar:** 1 CVE with medium severity.
* **jackson-databind-2.10.2.jar:** 6 CVEs with high severity.
* **log4j-api-2.12.1.jar:** 1 CVE with low severity.
* **logback-core-1.2.3.jar:** 1 CVE with medium severity.
* **snakeyaml-1.25.jar:** 8 CVEs with critical severity.
* **spring-boot-2.2.4.RELEASE.jar:** 3 CVEs with critical severity.
* **spring-boot-starter-web-2.2.4.RELEASE.jar:** 3 CVEs with critical severity.
* **spring-core-5.2.3.RELEASE.jar:** 11 CVEs with critical severity.
* **spring-web-5.2.3.RELEASE.jar:** 12 CVEs with critical severity.
* **spring-webmvc-5.2.3.RELEASE.jar:** 11 CVEs with critical severity.
* **tomcat-embed-core-9.0.30.jar:** 21 CVEs with critical severity.
* **tomcat-embed-websocket-9.0.30.jar:** 22 CVEs with critical severity.

## Mitigation Plan

* **Input Validation:**
  + **Issue**: The absence of explicit input validation mechanisms in the codebase leaves the application vulnerable to common security threats, such as SQL injection and Cross-Site Scripting (XSS).
  + **Mitigation Steps**:
    - Implement robust input validation routines using a trusted library or framework. Consider using Apache Validator or Hibernate Validator, depending on the project's requirements.
    - Ensure that all user inputs, especially those received through user interfaces, are validated and sanitized before processing.
    - Conduct regular security code reviews to verify the correctness of input validation implementations.
* **API Security:**
  + **Issue**: The application lacks a well-defined API structure, making it challenging for end-users to interact securely and predictably with the software. Data is accessed through insecure URLs.
  + **Mitigation Steps**:
    - Develop a structured and well-documented RESTful API that provides clear endpoints, request-response formats, and authentication mechanisms.
    - Implement secure API best practices, such as proper authentication and authorization checks, to ensure data security.
    - Transition from URL-based data access to secure HTTP methods like POST to prevent data exposure through browser histories.
* **Cryptography:**
  + **Issue**: The absence of data encryption mechanisms leaves sensitive financial information vulnerable during storage and transmission, potentially exposing the application to data breaches and non-compliance with international data protection regulations.
  + **Mitigation Steps**:
    - Implement strong encryption algorithms for both data at rest (database storage) and data in transit (communications).
    - Utilize industry-standard encryption libraries and frameworks to ensure compliance with international security standards.
    - Regularly review and update encryption protocols to stay aligned with evolving security best practices and regulations.
* **Error Handling:**
  + **Issue**: While error handling is present in some parts of the codebase, a more comprehensive evaluation is needed to ensure that all potential errors are addressed gracefully.
  + **Mitigation Steps**:
    - Conduct a thorough review of error-handling practices across the entire application.
    - Identify potential error scenarios and implement appropriate error-handling mechanisms, including informative error messages and proper logging.
    - Ensure that error handling does not reveal sensitive information to end-users or attackers.
* **User-Friendliness:**
  + **Issue**: The lack of a functional API, combined with the absence of input validation, impacts the user-friendliness of the application.
  + **Mitigation Steps**:
    - Prioritize the development of a user-friendly and well-documented API to facilitate secure interactions with the application.
    - Enhance user guidance and feedback to improve overall usability, ensuring that users can easily understand how to interact with the software.
* **Overall Security Awareness:**
  + **Issue**: The assessment highlights several security gaps, emphasizing the need for increased security awareness and practices within the development team.
  + **Mitigation Steps**:
    - Conduct security training and awareness programs for the development team to foster a security-conscious culture.
    - Integrate security practices into the development lifecycle, including regular security code reviews and threat modeling sessions.