****

# 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** | **07/05/2023** | **Joseph Daniszewski** |  |

## 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

Joseph Daniszewski

## Interpreting Client Needs

Artemis Financial values secure communications because their web-based software application deals with sensitive financial information. Secure communications ensure the confidentiality, integrity, and authenticity of data during transmission, protecting it from unauthorized access or modification.

Artemis Financial does make international transactions, which increases the risk of exposure to external threats. It is important to consider international regulations and compliance requirements related to secure communications, such as data protection laws like GDPR (General Data Protection Regulation).

External threats that might be present now and in the immediate future include:

* Malicious attacks: Artemis Financial's application may be targeted by hackers attempting to gain unauthorized access, steal sensitive information, or disrupt services.
* Injection attacks: Attackers may attempt SQL injection or other code injection techniques to exploit vulnerabilities in the application and access or manipulate data.
* Cross-Site Scripting (XSS): Attackers can inject malicious scripts into web pages viewed by users, leading to the execution of unauthorized actions or theft of sensitive information.
* Denial of Service (DoS) attacks: Hackers may attempt to overwhelm the application with a high volume of requests, causing it to become unresponsive or unavailable.
* Social engineering: Attackers may try to manipulate individuals within Artemis Financial through deception to gain unauthorized access or sensitive information.

Modernization requirements to consider include:

* The role of open-source libraries: Open-source libraries provide developers with pre-built components and functionality, enabling faster development. However, they can introduce security vulnerabilities if not properly maintained or if outdated versions with known vulnerabilities are used.
* Evolving web application technologies: As web application technologies advance, new security challenges and vulnerabilities emerge. It is essential to stay up to date with the latest security best practices and adapt to changes in the threat landscape.

## Areas of Security

Based on the Vulnerability Assessment Process Flow Diagram, the following areas of security are relevant:

* Architecture Review: Analyzing the application's architecture helps identify potential vulnerabilities related to the design and implementation of the software.
* Input Validation: Ensuring secure input and representations is crucial to prevent injection attacks, such as SQL injection or Cross-Site Scripting (XSS).
* APIs: Secure API interactions are important to protect against unauthorized access, data leakage, and ensure the integrity and authenticity of data exchanged through the API.
* Cryptography: Proper encryption usage and vulnerability management are important to protect sensitive data at rest and in transit.
* Client/Server: Secure distributed composing involves implementing secure communication protocols and handling authentication and authorization properly.
* Code Error: Secure code handling helps prevent coding errors, such as buffer overflows or unhandled exceptions, which can lead to security vulnerabilities.
* Encapsulation: Secure data structures and proper data encapsulation ensure that data is protected and accessed only by authorized components or entities.

## Manual Review

Some vulnerabilities I have identified within the code base are:

* In the CRUD class, there is a potential information disclosure vulnerability as the “getContent()” and “getContent2()” methods return the content strings without proper access control or validation.
* In the customer class, the” account\_number” and “account\_balance” fields are declared without proper access control, potentially exposing sensitive customer information.
* In the “DocData” class, the “read\_document” method establishes a database connection without implementing proper error handling or securely managing sensitive credentials.
* In the “GreetingController” class, the greeting method does not implement proper input validation or sanitization for the name parameter, potentially exposing the application to input-based attacks.
* In the “myDateTime” class, the “retrieveDateTime” and “setMyDateTime” methods are declared without proper access control, potentially exposing sensitive date and time information.

## Static Testing

The dependency check identified the following vulnerabilities in the code:

* bcprov-jdk15on-1.46.jar:
  + Vulnerability IDs: Multiple vulnerability IDs related to the Bouncy Castle Crypto Package with the highest severity.
* hibernate-validator-6.0.18.Final.jar:
  + Vulnerability IDs: One vulnerability ID related to Hibernate Validator with medium severity.
* jackson-databind-2.10.2.jar:
  + Vulnerability IDs: Multiple vulnerability IDs related to Jackson Databind with the highest severity.
* log4j-api-2.12.1.jar:
  + Vulnerability IDs: One vulnerability ID related to Log4j API with low severity.
* logback-core-1.2.3.jar:
  + Vulnerability IDs: One vulnerability ID related to Logback Core with medium severity.
* snakeyaml-1.25.jar:
  + Vulnerability IDs: Multiple critical vulnerability IDs related to SnakeYAML.
* spring-boot-2.2.4.RELEASE.jar:
  + Vulnerability IDs: Two vulnerability IDs related to Spring Boot with the highest severity.
* spring-boot-starter-web-2.2.4.RELEASE.jar:
  + Vulnerability IDs: Two vulnerability IDs related to Spring Boot Starter Web with the highest severity.
* spring-core-5.2.3.RELEASE.jar:
  + Vulnerability IDs: Multiple critical vulnerability IDs related to Spring Core.
* spring-web-5.2.3.RELEASE.jar:
  + Vulnerability IDs: Multiple critical vulnerability IDs related to Spring Web.
* spring-webmvc-5.2.3.RELEASE.jar:
  + Vulnerability IDs: Multiple critical vulnerability IDs related to Spring Web MVC.
* tomcat-embed-core-9.0.30.jar:
  + Vulnerability IDs: Multiple critical vulnerability IDs related to Apache Tomcat.
* tomcat-embed-websocket-9.0.30.jar:
  + Vulnerability IDs: Multiple critical vulnerability IDs related to Apache Tomcat WebSocket.

## Mitigation Plan

To mitigate the above security vulnerabilities, all the listed packages should be updated to the latest version. These updates should be applied as part of regular software maintenance to ensure the web application remains secure against known vulnerabilities.