****

# 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** | **03/18/2024** | **Edward Garcia** |  |

## 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

Edward Garcia

## Interpreting Client Needs

Addressing Artemis Financials’ security demands requires a thorough understanding of their operational requirements and potential threats. Secure communication is required to protect the sensitive financial data that Artemis Financial handles. Encryption measures will help to protect data against outside attacks. Dealing with international sources, the company must use strong encryption standards to protect cross-border data transfers while adhering to international legislation. Phishing assaults on employees or consumers, malware infections, SQL injections against web applications, and Distributed Denial of Service (DDoS) attacks are the types of potential risks that the company may face. The development of Artemis Financials’ operations requires regularly updating open-source libraries to address known vulnerabilities, while also adopting newer web application technologies that prioritize security features. Merging innovative security concepts into the company’s software development process, will safeguard the company against external threats.

## Areas of Security

After analyzing the flow diagram I would focus on these areas of security in order to protect the client’s needs:

Input Validation: To prevent injection attacks, ensure that the application only processes legitimate and anticipated data. To do this we will implement input validation techniques to prevent common vulnerabilities like SQL injection and cross-site scripting.  
  
Authentication and Authorization: The application must utilize strong authentication mechanisms to ensure that only authorized users can access sensitive financial information.

Secure API Interactions: Provide security against vulnerabilities in the application's API, which communicates with external services and clients. We can help secure API interactions with tools such as TLS (Transport Layer Security) which protects the information sent by the API.

Cryptography: Using encryption correctly to protect data at rest and in transit, especially given the financial nature of the application. The proper implementation of cryptographic techniques, such as strong algorithms and key management policies, guarantees that the companies financial data is inaccessible to unauthorized parties.

Client/Server: Securing communication between the client applications and the server, particularly for distributed components. Utilizing frameworks such as OAuth2.0 and OpenID Connect can be used to securely assign permissions to different areas of the server for the client.

Code Error Handling: Implementing secure error handling to prevent leakage of sensitive information through error messages. To record errors, use logging tools such as Log4j, Log back, or the Python logging module. We will also make sure logs are saved in a secure area and do not contain sensitive information.

## Manual Review

After doing a manual review of the application’s codebase and the dependency check report I found that in CrudController.java The handling of the business\_name parameter requires validation to avoid potential security flaws such as injection attacks. In customer.java the customer class exposes account\_number and account\_balance without encapsulation. This can allow unauthorized access and manipulation of sensitive data. The class name could also be capitalized following proper class name association techniques. In DocData.java, the read\_document method has hardcoded database connection information, posing a severe security concern. Environment variables or configuration files can be used to manage sensitive information such as database URLs, usernames, and passwords. I also noticed in DocData.java that catching SQLExceptions and printing the stack trace (e.printStackTrace()) may not be enough to handle database issues gracefully. Another implementation to consider is the database connections need to be properly closed preferably in a finally block or via try-with-resources.

## Static Testing

The dependency check report found 13 vulnerable dependencies and 82 vulnerabilities.   
  
Dependency: bcprov-jdk15on-1.46.jar  
Description: The Bouncy Castle Crypto package provides a Java implementation of cryptography techniques.   
Severity: High  
Vulnerabilities: CVE-2016-1000352, CVE-2016-1000346, CVE-2016-1000345, CVE-2016-1000344, CVE-2016-1000343, CVE-2016-1000342, CVE-2016-1000341, CVE-2016-1000339, CVE-2016-1000338, CVE-2018-5382, CVE-2017-13098, CVE-2013-1624  
Recommended Solution: Update the Bouncy Castle library to the most recent stable version. This technique would help to alleviate the reported security problems with the library's outdated versions.

Dependency: spring-boot-2.2.4.RELEASE.jar  
Description: Spring Boot  
Severity: Critical  
Vulnerabilities: CVE-2023-6378, CVE-2021-42550  
Recommended Solution: Update to the most recent supported versions of Spring Boot.   
  
Dependency: logback-core-1.2.3.jar  
Description: logback-core module  
Severity: High  
Vulnerabilities: CVE-2023-20883, CVE-2023-20873, CVE-2022-27772  
Recommended Solution: Update logback to a stable version

Dependency: log4j-api-2.12.1.jar  
Description: The Apache Log4j API  
Severity: Critical  
Vulnerabilities: CVE-2021-44832, CVE-2021-45105, CVE-2021-45046, CVE-2021-44228, CVE-2020-9488  
Recommended Solution: Update log4j to the latest version

Dependency: snakeyaml-1.25.jar

Description: YAML 1.1 parser and emitter for Java

Severity: Critical

Vulnerabilities: CVE-2022-1471, CVE-2022-41854, CVE-2022-38752, CVE-2022-38751, CVE-2022-38750, CVE-2022-38749, CVE-2022-25857, CVE-2017-18640

Recommended Solution: Upgrade to a more secure version of the library and use safer procedures when parsing untrusted YAML content:

Dependency: jackson-databind-2.10.2.jar

Description: General data-binding functionality for Jackson works on core streaming API

Severity: High

Vulnerabilities: CVE-2023-35116, CVE-2021-46877, CVE-2022-42004, CVE-2022-42003, CVE-2020-36518, CVE-2020-25649

Recommended Solution: Upgrade Jackson to a more secure version.

Dependency: tomcat-embed-core-9.0.30.jar

Description: Core Tomcat implementation

Severity: Critical

Vulnerabilities: CVE-2024-21733, CVE-2023-46589, CVE-2023-45648, CVE-2023-42795, CVE-2023-44487, CVE-2023-41080, CVE-2023-28708, CVE-2022-42252, CVE-2021-43980, CVE-2022-34305, CVE-2022-29885, CVE-2021-41079, CVE-2021-33037, CVE-2021-30640, CVE-2021-25329, CVE-2021-25122, CVE-2021-24122, CVE-2020-17527, CVE-2020-13943, CVE-2020-13935

Recommended Solution: Update Apache Tomcat to a stable version.

Dependency: hibernate-validator-6.0.18.Final.jar

Description: Hibernate's Bean Validation (JSR-380) reference implementation.

Severity: Medium

Vulnerabilities: CVE-2020-10693

Recommended Solution: Update Hibernate Validator to a stable version.

Dependency: spring-web-5.2.3.RELEASE.jar

Description: Spring Web

Severity: High

Vulnerabilities: CVE-2016-1000027, CVE-2020-5421, CVE-2021-22096, CVE-2021-22118, CVE-2024-22243

Recommended Solution: Update Spring Web to a stable version.

Dependency: spring-beans-5.2.3.RELEASE.jar

Description: Spring Beans

Severity: High

Vulnerabilities: CVE-2022-22965  
Recommended Solution: Update Spring Beans to a stable version.

Dependency: spring-webmvc-5.2.3.RELEASE.jar

Description: Spring Web MVC

Severity: Medium

Vulnerabilities: CVE-2021-22060  
Recommended Solution: Update Spring Web MVC to a stable version.

Dependency: spring-context-5.2.3.RELEASE.jar

Description: Spring Context

Severity: Medium

Vulnerabilities: CVE-2022-22968  
Recommended Solution: Update Spring Context to a stable version.  
  
Dependency: spring-expression-5.2.3.RELEASE.jar

Description: Spring Expression Language (SpEL)

Severity: Medium

Vulnerabilities: CVE-2022-22950  
Recommended Solution: Update Spring Expression Language to a stable version.

A screenshot of a computer

Description automatically generated

## Mitigation Plan

To address the identified security vulnerabilities, the mitigation plan includes several steps. First, we'll upgrade all vulnerable dependencies to the most recent, patched versions. Second, we will update the application's codebase by adding input validation to CrudController.java to prevent injection attacks, encapsulating sensitive data in customer.java, and ensuring proper naming standards are followed. Third, in DocData.java, we will replace hardcoded database credentials with secure environment variables and verify that database connections are properly maintained and closed. Finally, we will incorporate continuous security monitoring and testing procedures into our development lifecycle to quickly find and patch new vulnerabilities.