

# 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** | **03/20/2023** | **Afshin E. Ahvazi** | **Analysis (WIP)** |

## 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

Afshin E. Ahvazi of Global Rain, Scrum Lead

## Interpreting Client Needs

Artemis Financial is a counseling firm that utilizes a RESTful web application to offer financial services to their clients. The company stores personal and financial information about their clients into a database; therefore, there is plenty of incentive to be targeted by attackers. Given the sensitive nature of the business, security is a primary concern for nearly every aspect of the application. It is also required by law in many US states. For example, According to the NCSL, Texas law requires governmental and third party agencies to “develop and implement best practices [...] to improve information management and analysis to increase information security” (Tex. Govt. Code § 2054.0286 & 2054.138). One can only imagine the ensuing lawsuits and fines should there be a security breach and disclosure of private data.

Proxy and man-in-the-middle attacks, SQL injections, denial of service, password bypasses… the OWASP website lists an endless number of risks such an application is vulnerable to.

As technology is always evolving, so does the relationship between this software and external sources (such as APIs, modules, and dependencies). As a result, an application is always at risk of newly introduced vulnerabilities. Consequently, we must be vigilant when using external libraries by performing frequent dependency checks.

## Areas of Security

A cursory inspection of the application code reveals a few areas of security to consider.

This application offers users an interface where they can interact with the server. In addition, the program communicates with a database to store and retrieve information. Consequently, to avoid the previously-discussed risks, the application needs input validation and cryptography. There should also be error handling in the case of any potential issues during runtime, such as SQL operation failures, to prevent crashes and unintended release of data. In addition, there are several data classes (such as Customer, myDateTime, and CRUD) so OOP concepts including encapsulation are important to prevent unauthorized access to sensitive data. And lastly, following secure coding practices and implement well known design patterns (code quality) will reduce the ever-present risk of vulnerabilities and allows to easily maintain the application.

## Manual Review

A close analysis of the code has raised more specific and intricate issues regarding the software security.

Assuming that database username and password are collected from the users via the application interface, it is extremely dangerous to simply insert un-trusted data into url, as seen in the *DocData* class. In addition, this class does not override the default *toString* method; when this method is called on a *DocData* object elsewhere, sensitive information about the application and system are carelessly mishandled. For example, within the *CRUDController* class, a method uses this information to create a *CRUD* object. This method also takes in parameters that have no impact on the outcome of the method. The ambiguity because of unimplemented application features makes this vulnerability assessment daunting.

In contrast, the GreetingController endpoint methods directly incorporate user entries into the app. However, there are no input validation techniques implemented.

Several fields in the *myDateTime* are declared as default, meaning they can be accessed by all classes within the same package. This class also tries to access array elements without any error handling.

Overall, the system does not have an authentication or authorization systems in place.

## Static Testing

The OWASP dependency check reveals 13 vulnerabilities for this project. The following is a summary of the most critical of these vulnerabilities.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Dependency** | **Dependency Description** | **Vulnerability** | **Vulnerability Description** | **Solution** |
| log4j-api-2.12.1.jar | The Apache Log4j API | [CVE-2021-44228](https://nvd.nist.gov/vuln/detail/CVE-2021-44228) | Allows attackers to execute binary code via unsecured endpoints | Upgrade to log4j 2.15.0 and newer. |
| spring-beans-5.2.3.RELEASE.jar | Spring Beans | [CVE-2022-22965](https://nvd.nist.gov/vuln/detail/CVE-2022-22965) | Application is vulnerable to remote code execution (RCE) via data binding. | Run application as Spring Boot executable JAR, avoid WAR. |
| spring-web-5.2.3.RELEASE.jar | Spring Web | [CVE-2021-22118](https://nvd.nist.gov/vuln/detail/CVE-2021-22118) | Attackers can gain administrative privileges via “privilege escalation” attack. | Upgrade Spring to version 5.3.8 or newer. |
| tomcat-embed-core-9.0.30.jar | Core Tomcat implementation | [CVE-2020-1938](https://nvd.nist.gov/vuln/detail/CVE-2020-1938) | Attackers can reveal IP addresses via accessing AJP ports. | Upgrade to Apache Tomcat 9.0.31 or newer |
| snakeyaml-1.25.jar | YAML 1.1 parser and emitter for Java | [CVE-2022-1471](https://nvd.nist.gov/vuln/detail/CVE-2022-1471) | Attackers can use a class constructor to deserialize untrusted data leading to remote code execution. | Avoid SnakeYaml's Constructor(). Use SnakeYaml's SafeConsturctor |

## Mitigation Plan

Considering the manual review of the code and the dependency check, the following proposes a plan to mitigate potential vulnerabilities and secure the application:

* Meet with the client and development team to clarify the system requirements.
* Add input validation wherever user input is taken into the system.
* Add error handling methods to error-inducing areas of code.
* Use query parametrization when using user input to access databases.
* Add appropriate encapsulation for all class elements to prevent leaks. In addition, override the *toString()* method to prevent leak of system info.
* Remove dispensable methods and functionality that unnecessarily complicate the system.
* Tack problematic dependencies as revealed by the dependency check and upgrade the appropriate packages.
* Apply secure coding practices and implement well known design patterns.