****

# 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** | **3/18/2023** | **Willem Kroeger** |  |

## 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

Willem Kroeger

## Interpreting Client Needs

The company is part of the financial industry, which requires strong and secure communications to ensure the safety of its customer’s sensitive personal information. Based on operating a web-based application, is it highly likely that international transactions are part of normal business with Artemis Financial?

Due to the sensitive nature of financial information and transactions, external threats likely will try various methods to gather non-encrypted information of customers and steal their money. In modernizing the web application, using open-source software will cost less to deploy and be easier to maintain but will require a vulnerability assessment and routine dependency checks. Failure to keep the software updated increases the risk of a missed vulnerability that can allow an external attacker access to Artemis Financial.

## Areas of Security

APIs – The Artemis Financial web application will likely interact with several different APIs routinely and be called upon by external budget apps. API sessions will be secure to prevent unauthorized access to sensitive information.

Cryptography – Highly sensitive personal data will be in use by Artemis Financial and will be required by law to be encrypted to prevent unauthorized access.

Code Errors – Any code errors should not display information that enables external parties to gain information about the internal database structure.

Code Quality – Code should be easily readable and follow best practices, including the functionality of classes.

Encapsulation – Data stored by Artemis Financial should be separated to the maximum extent practicable so that if an attacker does gain unauthorized access, the compartmentalization of data prevents all data from being accessed.

## Manual Review

GreetingController.java

* Greeting is a string value that could be exploited.

DocData.java

* Code comments contain username and password.
* The username and password are also automatically inputted and do not have a catch block for any errors, meaning the error would show the username and password.

Customer.java

* Deposit function is an integer, not a formatted double.
* The public showInfo function calls the private account\_number data.

## Static Testing

|  |  |  |
| --- | --- | --- |
| **Dependency** | **Description** | **Attribution** |
| bcprov-jdk15on-1.46.jar | Part of the cryptography, most issues relate to vulnerabilities in outdated versions | The bouncycastle package implemented in the Java library. |
| spring-boot-2.2.4.RELEASE.jar | Vulnerable to directory hijacking based on outdated version | Known issue with spring framework versions that are no longer supported |
| logback-core-1.2.3.jar | An attacker with required privileges could edit configuration files to execute code | Known logback issue in versions 1.2.7 and before |
| log4j-api-2.12.1.jar | Vulnerability that gives attackers access to log files and ability to run arbitrary code. | Known issue in Apache log4j, was fixed and removed from updated versions |
| snakeyaml-1.25.jar | Could be used to cause a denial of service by using significant system resources, and remote code execution based on a vulnerability within the Constructor() class | Untrusted YAML files could be supplied by an attacker as input to cause a DOS attack, evidenced by the snakeyaml package |
| jackson-databind-2.10.2.jar | Can be used to cause resource exhaustion by submitting large nested xml files | Known issue in earlier versions of Jackson-databind |
| tomcat-embed-core-9.0.30.jar | HTTP vulnerability that could cause data leakage, remote code execution, or denial of service. | Documentation from Apache Tomcat lists the conditions needed for each vulnerability |
| hibernate-validator-6.0.18.Final.jar | Could allow an attacker to bypass input validation | Documented as part of hibernate.validator |
| spring-web-5.2.3.RELEASE.jar | Can be exploited to allow an attacker to remotely execute code | Documented as part of spring framework version changelogs |
| spring-beans-5.2.3.RELEASE.jar | Can be exploited to allow an attacker to remotely execute code | Documented as part of spring framework version changelogs |
| spring-webmvc-5.2.3.RELEASE.jar | Can be exploited to allow an attacker to remotely execute code | Documented as part of spring framework version changelogs |
| spring-context-5.2.3.RELEASE.jar | Improper input handling could lead to errors with case-sensitive characters | Documented as part of spring framework version changelogs |
| spring-expression-5.2.3.RELEASE.jar | Can be exploited to cause a denial of service attack | Documented as part of spring framework version changelogs |

## Mitigation Plan

All dependency vulnerabilities discovered through static testing can be addressed by updating to the latest version of all dependencies. Updating each dependency will prevent most vulnerabilities from being accessible to attackers.

For the manual code review, each deficiency can be addressed as follows:

* For the Greeting string value, limit the length of string input into the greeting and prevent unauthorized characters from being used.
* Remove the username and password from the DocData comments and complete the catch block to sanitize the error message that would be displayed.
* For the Deposit function in Customer.java, change the data type to a formatted double, not an integer (allows for decimal values)
* The public function showInfo should be reviewed to ensure that private data, such as account numbers, is not being overshared.