****

# 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** | **9/12** | **Ryan Resk** |  |

## 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

Ryan Resk

## Interpreting Client Needs

The client has requested that we work with them to create a modern application that assists Artemis Financial with providing individualized financial plans for customers. This application must be secure as financial information is sensitive. Artemis Financial deals with customers all over the globe, including government agencies. Meeting the security standards of these government agencies is crucial to the success of this application. External threats will be plentiful, especially when working with global government agencies. Modern technology must be used in order for fast changes to be pushed to the software whenever a vulnerability is detected.

## Areas of Security

**Secure API Interactions:**

Artemis Financial utilizes a RESTful web application programming interface (API), therefore secure interactions through the API are crucial to security during the use of the software.

**Secure Coding Practices:**

Secure coding practices are vital to the security of this application. Artemis Financial deals with high-value, sensitive information that will be targeted by attackers regularly. The best way to prevent attacks is to have those attacks in mind when coding the application initially. A secure code base will go a long way to accomplish this.

**Secure Data Structures:**

The data structures must be air-tight to prevent data loss, theft, or manipulation by attackers. The tampering or loss of data would be catastrophic for Artemis Financial, especially when dealing with global government agencies.

## Manual Review

**Customer.java, CRUDController.java:**

Lack of error handling, input validation, integers being used to hold account\_balance when float would be a better choice and without any form of encryption.

**CRUD.java:**

No error handling or validation before content is returned via getContent. Anything can be passed through as a string.

**DocData.java:**

Lack of secure API handling; connection is easily manipulated and there is no error handling against manipulation of the key value for the database.

## Static Testing

**CVE-2016-1000338(7.5):**

In Bouncy Castle JCE Provider version 1.55 and earlier the DSA does not fully validate ASN.1 encoding of signature on verification. It is possible to inject extra elements in the sequence making up the signature and still have it validate, which in some cases may allow the introduction of 'invisible' data into a signed structure.

The solution for this vulnerability is a patch located at (<https://github.com/bcgit/bc-java/commit/b0c3ce99d43d73a096268831d0d120ffc89eac7f#diff-3679f5a9d2b939d0d3ee1601a7774fb0>)

**CVE-2020-10693(5.3):**

A flaw was found in Hibernate Validator version 6.1.2.Final. A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages.

Updating Hibernate Validator is the only known solution to this vulnerability.

**CVE-2020-25649(7.5):**

A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity.

The solution to this vulnerability is a change to the code in FasterXML;

factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);

**CVE-2020-9488(3.7):**

Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender. Fixed in Apache Log4j 2.12.3 and 2.13.1

Upgrading to Apache 2.13.2 solves this vulnerability.

**CVE-2021-42550(6.6):**

In logback version 1.2.7 and prior versions, an attacker with the required privileges to edit configurations files could craft a malicious configuration allowing to execute arbitrary code loaded from LDAP servers.

Updating logback is the only known solution to this vulnerability.

**CVE-2017-18640(7.5):**

The Alias feature in SnakeYAML before 1.26 allows entity expansion during a load operation, a related issue to CVE-2003-1564.

Updating SnakeYAML solves this vulnerability.

**CVE-2022-27772(7.8):**

spring-boot versions prior to version v2.2.11.RELEASE was vulnerable to temporary directory hijacking. This vulnerability impacted the org.springframework.boot.web.server.AbstractConfigurableWebServerFactory.createTempDir method.

This vulnerability only effects versions prior to 2.2.11.RELEASE, therefore updating spring-boot will solve this issue.

**CVE-2022-22965(9.8):**

A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment. If the application is deployed as a Spring Boot executable jar, i.e. the default, it is not vulnerable to the exploit. However, the nature of the vulnerability is more general, and there may be other ways to exploit it.

Spring Framework versions 5.3.0 to 5.3.17, 5.2.0 to 5.2.19 are affected by this vulnerability. New versions of the framework do not suffer from this issue.

**CVE-2016-1000027(9.8):**

Pivotal Spring Framework through 5.3.16 suffers from a potential remote code execution (RCE) issue if used for Java deserialization of untrusted data. Depending on how the library is implemented within a product, this issue may or not occur, and authentication may be required. NOTE: the vendor's position is that untrusted data is not an intended use case. The product's behavior will not be changed because some users rely on deserialization of trusted data.

There is no solution to this issue as Java serialization is unsafe inherently, avoiding the exposing of HTTP Invoker endpoints is a form of mitigation for this vulnerability.

**CVE-2020-1938(9.8):**

AJP connects are treated with higher trust than HTTP connections. Therefore if these connections are available to an attacker they can be utilized to execute remote code, which is catastrophic for Artermis Financial, warranting a rating of 9.8 (critical).

This issue was fixed in an update numbered 9.0.31.

## Mitigation Plan

**Action List:**

1. Updating all dependencies would fix the vast majority of these issues.

2. Apply this code to **CVE-2020-25649** to solve the vulnerability: factory.setFeature("http://apache.org/xml/features/disallow-doctype-decl", true);

3. CVE-2016-1000027: can be mitigated by avoiding exposure of HTTP Invoker endpoints.

4. CVE-2016-1000338: apply patch located at (<https://github.com/bcgit/bc-java/commit/b0c3ce99d43d73a096268831d0d120ffc89eac7f#diff-3679f5a9d2b939d0d3ee1601a7774fb0>