****

# 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 5](#_Toc32574613)

[4. Static Testing 5](#_Toc32574614)

[5. Mitigation Plan 6](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **09/16/2023** | **Bryan Pirrone** | **Initial Analysis** |

## 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

Bryan Pirrone

## Interpreting Client Needs

Artemis Financial is a consulting company that develops individualized financial plans for its customers, ranging from savings, retirement, investments, and insurance accounts. To modernize its operations, Artemis Financial aims to use the most current and effective software security. Since their organization deals with large sums of financial information, security is a high priority for them and their customers. Protection and privacy are critical for this organization to protect its customers, their assets, and the company's reputation. As a financial institution, Artemis Financial is the target of a wide range of potential attacks, such as denial of service (DoS) to data breaches.

Artemis Financial is a company with a global reach. It is necessary to consider any differences in international laws regarding the protection of private information. A legal council would be necessary to make sure all aspects that are implemented follow all applicable laws and regulations.

Careful consideration should be taken when using open-source libraries or licenses. Implementation of these systems needs to be heavily scrutinized to ensure that they are maintained and updated with regularity. If outdated systems or software were used, then those systems or software would be introducing new vulnerabilities to Artemis Financial.

## Areas of Security

Artemis Financial has a RESTful web application programming interface (API) and is requesting assistance about how to protect their organization from external threats. Since a RESTful API is in use, I would recommend focusing on input validation, APIs, code errors, and encapsulation from the Vulnerability Assessment Process Flow Diagram.

Input validation is important to ‘sanitize’ or restrict inputs by users, external systems, or other sources. This is accomplished by validating these inputs against expected constraints before processing and acceptance by the system. Input validation helps protect against DoS, cross-site request forgery (CSRF), cross-site scripting (XSS), and buffer overloads. Implementation of strong input validation helps to minimize remote code execution or injection attacks due to the insertion of malicious code, strings, or characters.

APIs are an important area of security because they offer attackers more potential entry points into the system. Since the company already uses a RESTful API, verification should take place regarding the implementation of the HTTP methods GET, POST, PUT, and DELETE.

Code error is a common software vulnerability that can result in DoS and XXS attacks, buffer overflows, and data breaches. Early identification of these bugs, coupled with high-quality coding practices, can help minimize these exploitations.

Encapsulation is an object-oriented programming (OOP) principle that protects the contents of a class or object from outside classes, methods, or objects. This form of data hiding secures the details internal to the object from the outside world. By protecting the contents of a class or object from unauthorized access, data protection and access control are bolstered. Only Get/Set functions of the associated class or object should be able to access and alter its internal contents.

## Manual Review

After an initial review of the source, I have noted several errors or inconsistencies that could lead to potential vulnerabilities.

The pom.xml file has several instances of older versions. Spring framework and dependency-check-maven are outdated and need to be updated. Older versions of software or dependencies can be vulnerable to exploits that newer revisions or versions have patched.

Get/Set functions are available for some functions, while not for others. The Greeting class utilizes Get/Set functions and has private parameters. The GreetingController class does not use Get/Set functions to help protect its internals. The Customer class has a showInfo member that is public and returns account\_number. This could lead to exposing customer account numbers.

Input validation requires significant improvement across all classes. All methods that take user input should be sanitized for characters and length to prevent buffer overloads.

Several classes have poor-quality coding, which requires improvement. The myDateTime class has a method called setMyDateTime which is not finished. The DocData class has a readable username and password that are not complex, i.e. “root” “root”. Passwords should not be readable and should utilize salting and hashing. The DocData class also has a catch that if the read\_document method fails, then e.printStackTrace(); will print the entire stack trace. These poor coding techniques are exploitable and need to be remedied.

## Static Testing

A screenshot of a computer

Description automatically generated

|  |  |  |
| --- | --- | --- |
| **Dependency** | **Description** | **Solution** |
| bcprov-jdk15on-1.46.jar | The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. Vulnerable to EBC mode and signature vulnerabilities. | Upgrade Bouncy castle to a later version. |
| hibernate-validator-6.0.18.Final.jar | Hibernate’s Bean Validation (JSR-380) reference implementation. Vulnerable to input sanitation bypassing. | Update Hibernate’s Bean Validation to a later version. |
| jackson-databind-2.10.2.jar | General data-binding functionality for Jackson: works on core streaming API. Vulnerable to XML external entity (XXE) attacks, stack overflow crashes, DoS attacks, and resource exhaustion crashes. | Update jackson-databind to a version 2.15.2 or later. |
| log4j-api-2.12.1.jar | The Apache Log4j API. Vulnerable to man-in-the-middle attacks. | Update Apache Log4j to version 2.12.3 or 2.13.1 or later. |
| logback-core-1.2.3.jar | Logback-core module. Vulnerable to an attacker with the required privileges to edit configuration files to be able to craft malicious configuration, allowing to execute arbitrary code loaded from LDAP servers. | Update logback-core to version 1.2.9 or later. |
| snakeyaml-1.25.jar | YAML 1.1 parser and emitter for Java. Vulnerable to deserialization of untrusted data, XXE attacks, DoS attacks, and stack overflow crashes. | Upgrade snakeyaml to version 2.0 or later. |
| spring-boot-2.2.4.RELEASE.jar | Spring Boot. Vulnerable to security bypassing, temporary directory hijacking, and DoS attacks. | Upgrade spring-boot to version 3.0.6 or later. |
| spring-boot-starter-web-2.2.4.RELEASE.jar | Starter for building web, including RESTful, applications using Spring MVC. Uses Tomcat as the default embedded container. Vulnerable to security bypassing, temporary directory hijacking, and DoS attacks. | Upgrade spring-boot-starter-web to version 3.0.6 or later. |
| spring-core-5.2.3.RELEASE.jar | Spring Core. Vulnerable to remote code execution (RCE) via data binding, privilege escalation, RFD protection bypassing, DoS attacks, stack overflow crashes, and code injection. | Upgrade spring-core to a later version. |
| spring-web-5.2.3.RELEASE.jar | Spring Web. Vulnerable to RCE attacks, deserialization of untrusted data, code injection, privilege escalation, RFD attacks, and DoS attacks. | Upgrade spring-web to a later version. |
| spring-webmvc-5.2.3.RELEASE.jar | Spring Web MVC. Vulnerable to code injection, privilege escalation, exposure of resources, RFD attacks, DoS attacks, and stack overflow crashes. | Upgrade spring-webmvc to a later version. |
| tomcat-embed-core-9.0.30.jar | Core Tomcat implementation. Vulnerable to remote code execution via higher trust privileges, high CPU usage resulting in server unresponsiveness, DoS attacks due to memory issues, DoS attacks due to infinite loops, data breaches, deserialization of untrusted data, uncontrolled resource consumption, XXS attacks, open redirect attacks, and HTTP Request/Response smuggling. | Upgrade Apache to version 10.1 or later. |
| tomcat-embed-websocket-9.0.30.jar | Core Tomcat implementation. Vulnerable to remote code execution via higher trust privileges, incorrect default permissions, high CPU usage resulting in server unresponsiveness, DoS attacks due to memory issues, DoS attacks due to infinite loops, data breaches, deserialization of untrusted data, uncontrolled resource consumption, XXS attacks, open redirect attacks, and HTTP Request/Response smuggling. | Upgrade Apache to version 10.1 or later. |

## Mitigation Plan

All dependencies need to be updated, including any libraries. This is important to mitigate existing security vulnerabilities. This may result in new code generation, so this needs to be accomplished first. Following recompiling, internal tests need to be verified with passing results. Sanitation can be implemented for the cases stated previously as well as password security enhancements. I would also recommend that peer programming be utilized to help generate good-quality code and to reinforce this concept in the organization's culture. These steps should mitigate any existing vulnerabilities found during the manual review and static test.