****

# 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** | **3/19/23** | **Emily Domoracki** | **Initial project review** |

## Client



## Developer

Emily Domoracki

## Interpreting Client Needs

Artemis Financial is trusted to keep their client’s personal information safe and secure. The company retains a lot of sensitive information about individuals; including savings accounts, retirement accounts, investment portfolios, and insurance policies. While clients benefit from having access to this data via the company’s API, it also opens the door to malicious users and various attacks such as denial of service or injections. Global Rain’s objective is to ensure Artemis Financial meets the FTC’s Standards for Safeguarding Customer Information by developing effective administrative, technical, and physical security measures – while also prioritizing the user experience for both customer and employee. The system shall account for the security implications of third-party/open source/external library and framework dependencies. Global Rain may also collaborate with Artemis Financial to outline policy concerning future technological evolution and developments.

## Areas of Security

*Input Validation*

The Artemis Financial web app requests user input numerous times. Input fields provide ample opportunity to attackers by means of injection, resource exhaustion, and overflow errors. Validating that user input meets size, value type, and legal character sequences parameters is a simple and effective barrier against malicious input.

*Secure API Interactions*

It goes without saying that the Artemis web app is a doorway to a great deal of sensitive information. Authentication is necessary to validate that a user is who they say they are, and authorization ensures they are only accessing information appropriate to their role/identity. These two concepts are fundamental to running a secure API.

*Error Handling*

It is prudent to recognize that unhandled errors can lead to unpredictable and/or volatile behavior in the software – which can be unknowingly triggered by a well-meaning user. A malicious user can exploit such vulnerabilities and glean vital information from subsequent exception reports.

*Code Quality*

Inconsistencies in code security lead to vulnerability. Adherence to input validation, query parameterization, statelessness, and POLP are what keep software and databases secure.

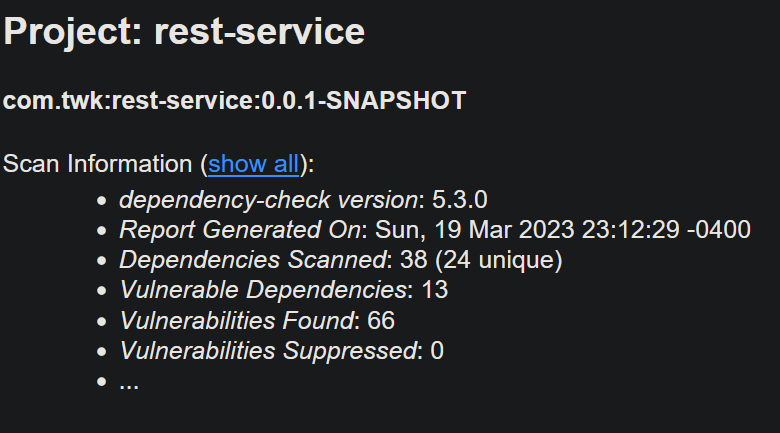
*Encapsulation*

We can maintain a high level of control and access restriction on vulnerable code by wrapping private data members together. If data members such as the account number of a client are publicly accessible to other classes, subclasses may be manipulated to gain access to those data members and information.

## Manual Review

Currently, I see no input validation measures in the GreetingController class when looking for the value for “name”. The only method with any error handling is the read\_document method in the DocData class. The exception handling in that method risks exposing system information by printing the stack trace to the user. While seemingly innocuous, the customer’s account balance is information that should also be kept private. There are some inconsistencies, but the existing code does a good job of query parameterization and encapsulation overall.

## Static Testing



The following are the dependencies present in the code with vulnerabilities as identified by the dependency check report:

* *The Bouncy Castle Crypto package*: Java library before 1.48 leaves this vulnerable to timing side-channel attacks. Java library before 1.51 is at risk of information exposure. The JCE Provider before version 1.55 is vulnerable to timing attack and does not fully validate encoding of signature. Versions before 1.55 may also generate a weak private key.
* *Spring Boot*: Versions prior to v2.2.11.RELEASE were vulnerable to temporary directory hijacking.
* *Logback-core module*: 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.
* *The Apache Log4j API*: Numerous versions before 2.3.1 had issues with improper input validation and deserialization of untrusted data. Improper validation of certificate with appender can result in leaked logs to a man-in-the-middle attack. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled.
* *YAML 1.1 parser and emitter for Java*: SnakeYAML versions before 2.2.4 vulnerable to DoS and RCE attacks.
* *General data-binding functionality for Jackson*: Versions 2.10 - (exluding) 2.10.5 are vulnerable to XXE attacks. Jackson-databind before 2.13.0 allows a Java StackOverflow exception and denial of service via a large depth of nested objects.
* *Core Tomcat implementation*: Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling'), possibility of information exposure, and vulnerability to DoS attacks exist throughout many versions of Core Tomcat.
* *Hibernate's Bean Validation (JSR-380) reference implementation*: 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.
* *Spring Web*: Pivotal Spring Framework through 5.3.16 suffers from a potential remote code execution (RCE) issue if used for Java deserialization of untrusted data. Other versions allow for malicious input to log files.
* *Spring Beans*: A Spring MVC or Spring WebFlux application running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding.
* *Spring Web MVC*: In Spring Framework versions 5.3.0 - 5.3.13, 5.2.0 - 5.2.18, and older unsupported versions, it is possible for a user to provide malicious input to cause the insertion of additional log entries.
* *Spring Context*: Improper Handling of Case Sensitivity.
* *Spring Expression Language (SpEL)*: Vulnerable to DoS attacks.

## Mitigation Plan

Most dependency vulnerabilities can be mitigated by staying updating to the latest version available. A top concern of our team is limiting access to program components that correlate directly with sensitive information about the system, company, or the clients. Ironing out the encapsulation inconsistencies and applying a global error handler that redirects the user to a generic error message will limit a lot of that access. Another priority is mitigating the risk of DoS and injection attacks by utilizing input validation measures.