****

# Artemis Financial Vulnerability Assessment Report

Table of Contents

[Document Revision History 3](#_Toc32574607)

[Client 3](#_Toc32574608)

[Developer 4](#_Toc32574610)

[1. Interpreting Client Needs 4](#_Toc32574611)

[2. Areas of Security 7](#_Toc32574612)

[3. Manual Review 8](#_Toc32574613)

[4. Static Testing 9](#_Toc32574614)

[5. Mitigation Plan 24](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **September 17, 2023** | **Matthew Pool** | **Final** |

## 

## Client



## Developer

Matthew Pool

## Interpreting Client Needs

Artemis Financial is a financial planning/consulting company that operates on a global level and is responsible for quite an impressive and diverse portfolio, including savings and retirement accounts, investments, and insurance. At Global Rain, our mission statement says “Security is everyone’s responsibility”, and we truly believe that! New security vulnerabilities are continually being discovered and new, innovative and creative ways to counter them are constantly being created. Because Artemis Financial works at an international level involving financial transactions and government agencies (and other businesses), their “rest-service” software requires more than the typical RESTful web Application Programming Interface (API), as will be explained in this document, starting with domestic and international laws (including financial sector regulations), followed by general web app security vulnerability concerns, as follows:

Financial & International Government Standards & Regulatory Compliance:

* **(GLBA) Gramm-Leach-Bliley Act**: U.S. Law for financial info management & data protection
* **(SOX) Sarbanes-Oxley Act**: U.S. law for data integrity & confidentiality
* **(MiFID II) Markets in Financial Instruments Directive II**: European transparency regulations
* **(PSD2) Revised Payment Service Directive**: European directive for payment services
* **(GDPR**) **General Data Protection Regulation**: European Union law for data protection
* **(FISMA**) **Federal Information Security Management Act**: U.S. law for federal data systems
* **+** Other industry- and regional-specific laws **(HIPAA**, etc.**)**

RESTful Web App Security Vulnerabilities:

Following the Vulnerability Assessment Process Flow Diagram (VAPFD) resource from SNHU, seven general areas concerning software security vulnerabilities exist in web app architectures:

**VAPFD Web App Vulnerability Areas**:

* **Input Validation**: crucial in preventing injection attacks
* **APIs**: provide abstraction and encapsulation for accessing methods indirectly
* **Cryptography**: provides data encryption and secure communication
* **Client/Server**: web applications use client/server architecture with both secured
* **Code Error**: errors in code can provide the opportunity to take advantage of exploits
* **Code Quality**: best practices ensure the least possibility for attacks like injection or DoS
* **Encapsulation**: hides implementation details from the user and reduces possible attacks

**Specific Threats/Attacks**:

* **DoS (Denial of Service)**: threats that cause a software program to use an excessive number of resources, resulting in unavailable service or poor performance for legitimate users
  + **Example**: Initiating multiple connections rapidly via a script like
    - **for** **i in {1..10000}; do curl http://target.com; done**
* **SQL Injection**: mixing query data fragments with untrusted data to run untrusted code or access/manipulate database data
  + **Example**: inserting **' OR '1'='1** into SQL queries
* **HTML Injection**: web page defacement with malicious script execution
  + **Example**: inserting **<script>alert('hacked')</script>** into a web page
* **XML Injection**: unauthorized data access (or DoS) by inserting malicious script in XML parser
  + **Example**: inserting **<!ENTITY xxe SYSTEM "file:///etc/passwd">** into XML data
* **JSON Injection**: data tamper & logic bypass by inserting script into JSON parser/web service
  + **Example**: Inserting **{"isAdmin": true}** into JSON payloads
* **Command Injection**: injecting harmful system commands
* **Cross-Site Scripting (XSS)**: injection of malicious scripts into web pages viewed by other users, allowing access to cookies or session tokens or modifying web pages
  + **Example**: Inserting **<script>stealCookies()</script>** in a comment or forum post
* **Cross-Site Request Forgery (CSRF)**: attacker tricks victim into performing actions without their consent like clicking a link they didn’t want to click, resulting in unauthorized actions performed on behalf of an authenticated user
  + **Example**: Inserting an **<img src="http://bank.com/withdraw?amount=1000">** tag in a forum, causing users to unknowingly make withdrawals
* **Regular Expression Denial-of-Service (ReDoS)**: uses regular expression computation that grow exponentially with input size
  + **Example**: Inserting a complex regex like **^(a+)+$** matched with long strings of '**a**'s
* **URL Open Redirect**: misleading URLs leading to malicious sites or granting remote access
  + **Example**: Modifying **http://example.com/redirect?url=trusted.com** to **http://example.com/redirect?url=malicious.com**
* **Unauthorized Access**: unauthorized function/data access or misuse
  + **Example**: Exploiting a missing authentication check to access restricted API endpoints, like **GET http://target.com/api/admin/users**
* **Misconfiguration**: default or risky configurations create unnecessary exploits
  + **Example**: Leaving unsafe default configurations in a third-party plugin

Future External Threats:

As mentioned earlier, new threats are constantly being uncovered, and new types of security vulnerabilities are on the horizon. Quantum computing has been gaining traction over the years with increasing numbers of qubits (the basic unit of quantum information that uses the quantum property of superposition), allowing a quantum computer to crack encryption algorithms in a fraction of the time of a classical computer. Various complex forms of artificial intelligence are gaining popularity and already pose an existential threat to democracy and other major aspects of our lives. Deepfake technology, a subset of Generative Adversarial Networks (GANs) that creates fake media, is one way A.I. is being used by bad actors. However, governments and intelligence agencies (and other clever and cautious minds around the world) are aware of these threats and working on creative ways to combat them. Ongoing software security analyses and maintenance (updates) will be required with any lasting system, including the RESTful web application for Artemis Financial.

Modernization Requirements:

There are a few other technical requirements to consider concerning modernization of the software app. Currently, a relational SQL database is used to take advantage of ACID (Atomicity, Consistency, Isolation, Durability) transactions, which offer strong data integrity. To easily scale out, a hybrid database approach could be taken advantage of by employing a document-oriented NoSQL database too, to take advantage of logging, caching, and real-time analytics. A cloud infrastructure should also be implemented to allow efficient and cost-effective performance, scalability, reliability, recoverability, and security! It is also highly recommended that accessibility considerations and integrations be made to be ADA- (Americans with Disabilities) and WCAG- (Web Content Accessibility Guidelines) compliant. Open-source libraries, such as dependencies utilized by Maven, will be used when feasible and kept up to date. An agile Scrum framework will be utilized in continually providing security analysis and implementation that meet all security software requirements, as they continue to evolve.

## Areas of Security

Because the Artemis Financial software app is a RESTful web API, the following VAPFD areas are of concern in terms of software security:

**Controllers**: handle HTTP requests and routing

* **vulnerabilities**: insecure endpoints, improper authentication/authorization

**Data Access**: uses java.sql (SQL) for accessing data in the database

* **vulnerabilities**: SQL/NoSQL injection, insecure data storage, lacking input validation

**APIs**: RESTful web API with CRUD methods to get (read) data and set (create/update) data

* **vulnerabilities**: improper authentication, no rate limiting, no data validation

The software application software was created using the Spring Framework and also includes the following VAPFD areas of concern:

**Models**: data structures and data validation throughout app

* **vulnerabilities**: insecure data handling/validation, weak encryption/hashing

**Plugins**: OWASP Dependency-Check is implemented into the software application as a plugin

Not relevant to Artemis Financial software application:

**Services**: REST Service Application bootstrap class created by Spring Framework to handle business logic/operations can be ignored during manual review (unless modified)

* **vulnerabilities**: business logic bypass, improper access control, error handling, insecure communication (unencrypted)

**Views**: CLI (command-line interface) output is not considered a “View” in the context of MVC architecture, and this application only uses a CLI API without additional graphical representation

* **vulnerabilities**: XSS, data leakage through error messages, CSRF

## Manual Review

**SECURITY VULNERABILITIES**

**DocData.java**

* username and password are both hardcoded (and in comment)
* weak (or no) password rules
* username and password are the same
* read\_document() doesn’t use placeholders (‘?’) for key and value variables (no parameterized queries to automatically escape characters)
* auto-generated error catch block may expose sensitive information
* unused passed values (key and value)
* no SSL/TLS used for database connection (main-in-the-middle attacks)
* make id variable ‘final’ since an id will not change once assigned for an instance

**myDateTime.java**

* no visibility identifiers used for mySecond, myMinute, and myHour, resulting in package-private attributes that may be accessed from other classes in the package without intended authorization

**customer.java**

* no visibility identifier used for account\_balance, resulting in package-private attributes that may be accessed from other classes in the package without intended authorization
* deposit() lacks parameter validation, which could lead to harmful inputs
* showInfo() lacks authentication/authorization and could lead to data leakage of sensitive information (account\_number)

**CRUDController.java**

* business\_name parameter not validated and could be used for SQL Injection attacks if used in database queries
* /read endpoint lacks authentication/authorization and could expose information
* no rate limiting which could lead to DoS attack
* lacking HTTPS implementation can lead to data exposure as data is transmitted in plain text
* stack traces or sensitive information may be leaked unless proper error-handling is implemented

**CRUD.java**

* content and content2 may be exposed unless authorization/authentication to access them are implemented and validation/sanitization are implemented

**Greeting.java**

* id & content may be exposed unless authorization/authentication to access them are implemented and validation/sanitization are implemented

**GreetingController.java**

* lack of rate limiting on number of API calls, leading to potential DoS attacks
* lack of authentication/authorization for the endpoint /greeting

## Static Testing

A close-up of a sign

Description automatically generated

A screenshot of a computer

Description automatically generated

**Dependency Updates**:

* org.bouncycastle:bcprov-jdk15on 1.46 -> 1.70

**Plugin Updates**:

* org.springframework.boot:spring-boot-maven-plugin 2.2.4 RELEASE -> 3.1.3

**Plugin Validation Issues**:

* org.codehaus.mojo:versions-maven-plugin:2.7
* org.apache.maven.plugins:maven-site-plugin:3.8.2

**Minimum Maven 3.1.0**:

* org.owasp:dependency-check-maven 5.3.0 -> 8.4.0

[WARNING] **Plugin validation issues** were detected in 8 plugin(s)

[WARNING]

[WARNING] \* org.apache.maven.plugins:maven-resources-plugin:3.1.0

[WARNING] \* org.apache.maven.plugins:maven-jar-plugin:3.1.2

[WARNING] \* org.apache.maven.plugins:maven-compiler-plugin:3.8.1

[WARNING] \* org.apache.maven.plugins:maven-install-plugin:2.5.2

[WARNING] \* org.apache.maven.plugins:maven-clean-plugin:3.1.0

[WARNING] \* org.owasp:dependency-check-maven:5.3.0

[WARNING] \* org.springframework.boot:spring-boot-maven-plugin:2.2.4.RELEASE

[WARNING] \* org.apache.maven.plugins:maven-surefire-plugin:2.22.2

FALSE POSITIVES (NO VULNERABILITIES)

A screenshot of a computer

Description automatically generated

THREATS

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

## Mitigation Plan

As previously stated, the Artemis Financial software application must adhere to strict guidelines set forth by various governmental and industry regulations. On top of this, RESTful web applications are prone to certain security vulnerabilities and must be kept up to date, and security requirements must continually be checked, as new vulnerabilities and threats are discovered.

**The following changes should be made to the rest-service package classes listed below**:

**DocData.java**

* remove hardcoded username and password and comment
* pass in the username and access salted SHA-256 encrypted password using AES and POST, along with masking, as well as forcing the password to be different than the username
* input validation annotations and custom logic to enforce password rules to require at least 8 digits and 1 digit and special character
* use parameterized queries with placeholders (‘?’) for key and value variables in read\_document() and implement their use (or completely remove if unused)
* use global and localized error catching blocks that log and return purged information
* use SSL (TLS) for all database connections

**myDateTime.java**

* make mySecond, myMinute, and myHour variables ‘private’

**customer.java**

* make account\_balance ‘private’
* make account\_number ‘final’
* add Spring Security dependency for authentication/authorization check for showInfo()
* validate parameters passed to deposit()

**CRUDController.java**

* add validation for business\_name parameter
* add Spring Security dependency for authentication/authorization for endpoint
* use Bucket4j library for rate limiting
* use global and localized error catching blocks that log and return purged information

**CRUD.java**

* add Spring Security dependency for authorization/authentication to access content and content2
* implement validation/sanitization for content and content2

**Greeting.java**

* add Spring Security dependency for authorization/authentication to access id and content
* implement validation/sanitization for id and content

**GreetingController.java**

* use Bucket4j library for rate limiting
* add Spring Security dependency for authentication/authorization for /greeting endpoint

**Dependency Updates**:

* org.bouncycastle:bcprov-jdk15on 1.46 -> 1.70

**Plugin Updates**:

* org.springframework.boot:spring-boot-maven-plugin 2.2.4 RELEASE -> 3.1.3

**Plugin Validation Issues**:

* org.codehaus.mojo:versions-maven-plugin:2.7
* org.apache.maven.plugins:maven-site-plugin:3.8.2

**Other Plugin** Issues:

* org.apache.maven.plugins:maven-resources-plugin:3.1.0
* org.apache.maven.plugins:maven-jar-plugin:3.1.2
* org.apache.maven.plugins:maven-compiler-plugin:3.8.1
* org.apache.maven.plugins:maven-install-plugin:2.5.2
* org.apache.maven.plugins:maven-clean-plugin:3.1.0
* org.owasp:dependency-check-maven:5.3.0
* org.springframework.boot:spring-boot-maven-plugin:2.2.4.RELEASE
* org.apache.maven.plugins:maven-surefire-plugin:2.22.2

**Minimum Maven 3.1.0**:

* org.owasp:dependency-check-maven 5.3.0 -> 8.4.0

***Based on the previous results, the following steps should be taken to address vulnerabilities***:

**GLOBAL CODE MODIFICATIONS**

* implement HTTPS into application configuration settings
* implement Spring global error catching

**UPDATES**

* Update Spring Framework
* Update OWASP Dependency-Check plugin
* Update out-of-date dependencies
* Update out-of-date libraries and other plugins
* Apply any applicable vendor security patches
* Run OWASP (Maven) Dependency-Check & check results

**CVE DEPENDENCY UPDATES**

*Apply the listed strategies for any of the following CVE Vulnerabilities that remain:*

* **CVE-2013-1624**: Update org.bouncycastle:bcprov-jdk15on 1.46 -> 1.70
* **CVE-2022-27772**: Update org.springframework.boot:spring-boot-maven-plugin:2.2.4.RELEASE
* **CVE-2021-42550:** Update org.apache.maven.plugins:maven-site-plugin:3.8.2
* **CVE-2017-18640**: Update org.springframework.boot:spring-boot-maven-plugin:2.2.4.RELEASE
* **CVE-2020-25649**: Update org.springframework.boot:spring-boot-maven-plugin:2.2.4.RELEASE
* **CVE-2019-17569**: Update org.springframework.boot:spring-boot-maven-plugin:2.2.4.RELEASE
* **CVE-2020-10693**: Update to patched version 6.15.Final (or above)
* **CVE-2016-1000027 (OSSINDEX)**: Update spring-web (org.springframework) dependency in pom.xml
* **CVE-2022-22965 (OSSINDEX)**: Update spring-bean (org.springframework) dependency in pom.xml
* **CVE-2021-22060 (OSSINDEX)**: Update spring-webmvc (org.springframework) dependency in pom.xml
* **CVE-2022-22968 (OSSINDEX)**: Update spring-context (org.springframework) dependency in pom.xml
* **CVE-2022-22950 (OSSINDEX)**: Update spring-expression (org.springframework) dependency in pom.xml

*Save pom.xml, create a Maven clean installation, and verify security vulnerabilities have been fixed by the preceding steps.*

*Various tools from OWASP and other open-sources may be used to perform final vulnerability assessment and analysis.*

**Sources**:

* *Common Vulnerabilities and Exposures* (*CVE*).(2023).

<https://cve.mitre.org/cve/search_cve_list.html>

* *National Vulnerability Database* (*NVD*). (2023).

<https://nvd.nist.gov/vuln/search>

* *Common Weakness Enumeration* (*CWE*). (2023).

<https://cwe.mitre.org/index.html>

* *Secure Coding Guidelines for Java SE*: <https://www.oracle.com/java/technologies/javase/seccodeguide.html>
* *OWASP Secure Coding Practices Checklist*: <https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v2.pdf>