****

# CS 305 Project One

**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** | **03-20-2022** | **Howard Gilmore** |  |

## Client



## Instructions

Deliver this completed vulnerability assessment report, identifying your findings of security vulnerabilities and articulating recommendations for next steps to remedy the issues you have found.

Respond to the five steps outlined below and include your findings. Replace the bracketed text on all pages with your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

[Insert your name here]

## 1. Interpreting Client Needs

Determine your client’s needs and potential threats and attacks associated with their application and software security requirements. Consider the following regarding how companies protect against external threats based on the scenario information:

* What is the value of secure communications to the company?

A company's and its customers' sensitive information (bank information, passwords, wiring numbers, routing IDs, etc.) is carried in all conversations and data transfers. Assessing and fixing vulnerabilities is critical to protecting data and preventing catastrophic data breaches.

* Are there any international transactions that the company produces?

in this case, Global Rain is collaborating with a software engineering firm that develops bespoke software for governments and enterprises worldwide. Presumably because they will require labor, talents, and likely facilities from throughout the globe.

* Are there governmental restrictions about secure communications to consider?

GLB Act

The Gramm-Leach-Bliley Act compels financial institutions to disclose their information-sharing activities and to protect sensitive data.

* What external threats might be present now and in the immediate future?

As a developing firm, they will face more external threats to steal money from them and their clientele. These risks might directly target the organization or indirectly target its customers. Malware and other assaults and attacks will be among the risks.

* What are the “modernization” requirements that must be considered, such as the role of open source libraries and evolving web application technologies?

Continuous modernization for security, functionality, and efficiency must be considered. Open source is helpful for adding features but should never be utilized as is since anybody may access it, including potential attackers.

[Include your findings here.]

## 2. Areas of Security

Referring to the Vulnerability Assessment Process Flow Diagram, identify which areas of security are applicable to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

* API: API calls are often used in web-based applications. Nonetheless, security in this sector should be a top focus. There should be a protected setup for the API to guarantee that the security of the system and the API connection is not jeopardized.
* Client/Server: Because this is a web-based application, communication between servers and clients is required in this section. Financial information is generally included in these letters. As a result, both the data stream and the connection in this portion will need high degrees of security.
* Cryptography: Access, transmissions, and all other types of data transfer and access fall under this category of security. Encryption and adequate certificate validation should be used to achieve this.
* Secure Coding: Code sanitization is necessary in a variety of circumstances and scenarios to ensure that the business logic of the application is consistent across all of those situations and scenarios.

## 3. Manual Review

Continue working through the Vulnerability Assessment Process Flow Diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

There were several mistakes in the code that were discovered, and some of them were rather simple to rectify. The system's efficiency is critical since the web-based software will be used by a large number of people and must thus stay efficient.

It is also necessary for the client-server connection to be secure in order to guarantee that there is no possibility of eavesdropping occurring inside the network.

There are various flaws in the code that might provide access to the system via a back door. A vulnerability that exists, for example, is in the area of data access. DocData.java, the data access method that requires the specification of the location of the database, as well as the creation of a username and password, has many security flaws. There are root users and a password, and it is not recommended that you use the "root" account because of security concerns. The password for the "root" account may thus be readily guessed and hacked. The root is used as both the login and password for the system.

This makes it simpler for hackers to get access to the system via brute force assaults.

Some mistakes were also identified in the main class of our software, where the majority of the other classes being called did not have valid call methods defined. Because the main class made use of global variables, the problems that occurred meant that the program was unable to build the subsequent classes. This problem may be readily resolved by making the necessary changes to the classes that were being invoked or by changing the function that was being called, including the scripts itself.

## 4. Static Testing

Run a dependency check on Artemis Financial’s software application to identify all security vulnerabilities in the code. Record the output from dependency check report. Include the following:

1. The names or vulnerability codes of the known vulnerabilities
2. A brief description and recommended solutions provided by the dependency check report
3. Attribution (if any) that documents how this vulnerability has been identified or documented previously

CVE-2020-9488 - Incorrect certificate validation due to a host mismatch in the Apache Log4j SMTP appender. This might enable a man-in-the-middle attack to intercept an SMTPS connection, leaking any log messages delivered via that appender.

a. It is recommended that you update to version 2.13.2 of the Apache Log4j API, which addresses the man-in-the-middle vulnerability.

CVE-2019-17569 - A bug was created as a result of the refactoring in Apache Tomcat 9.0.28 to 9.0.30, 8.5.48 to 8.5.50, and 7.0.98 to 7.0.99. As a consequence of the defect, invalid Transfer-Encoding headers were wrongly processed, opening the door to HTTP Request Smuggling if Tomcat was behind a reverse proxy that incorrectly treated the invalid Transfer-Encoding header in a specific way. Such a reverse proxy is deemed improbable.

a. Upgrade Tomcat to one of the following versions: i. 9.0.31-1deb10u1 or higher ii. 8.5.54-0+deb9u1 or higher iii. 7.0.100 or higher

Spring Core Requirements: spring-core-5.2.3. RELEASE. jar I think the severity is moderate, and the number of CVEs is one. The dependency's trust does not represent a security issue. According to a quick Google search, 5.2.3 also offers a remedy for CVE2020-5397.

CVE-2017-18640: The Alias feature in SnakeYAML 1.18 allows entities to be expanded during a load operation, which is similar to CVE-2003-1564, which was found in SnakeYAML 1.18.

a. Do this:

i. If the YAML is not coming from a source that you don't trust, then it is a false positive. It doesn't matter. Quality is very low and there are a lot of issues that appear to be false positives in the NVD database, but it's not clear which ones.

iii. Make sure the YAML is good before you give it to SnakeYAML. For example, count \* and &.

iii. Switch to SnakeYAML Engine.

CVE-2021-27568: An issue was discovered in netplex json-smart-v1 through 2015-10-23 and json-smart-v2 through 2.4. An exception is thrown from a function, but it is not caught, as demonstrated by Number Format Exception. When it is not caught, it may cause programs using the library to crash or expose sensitive information.

CVE-2020-10693: 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.

CVE-2020-25649: 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.

CVE-2021-27568: An issue was discovered in net plex json-smart-v1 through 2015-10-23 and json-smart-v2 through 2.4. An exception is thrown from a function, but it is not caught, as demonstrated by Number Format Exception. When it is not caught, it may cause programs using the library to crash or expose sensitive information.

CVE-2021-42550: In log back 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.

CVE-2017-8046: Malicious PATCH requests submitted to servers using Spring Data REST versions prior to 2.6.9 (Ingalls SR9), versions prior to 3.0.1 (Kay SR1) and Spring Boot versions prior to 1.5.9, 2.0 M6 can use specially crafted JSON data to run arbitrary Java code.

CVE-2018-1259: Spring Data Commons, versions 1.13 prior to 1.13.12 and 2.0 prior to 2.0.7, used in combination with XMLBeam 1.4.14 or earlier versions, contains a property binder vulnerability caused by improper restriction of XML external entity references as underlying library XMLBeam does not restrict external reference expansion. An unauthenticated remote malicious user can supply specially crafted request parameters against Spring Data's projection-based request payload binding to access arbitrary files on the system.

CVE-2018-1273: Spring Data Commons, versions prior to 1.13 to 1.13.10, 2.0 to 2.0.5, and older unsupported versions, contain a property binder vulnerability caused by improper neutralization of special elements. An unauthenticated remote malicious user (or attacker) can supply specially crafted request parameters against Spring Data REST backed HTTP resources or using Spring Data's projection-based request payload binding hat can lead to a remote code execution attack.

CVE-2018-1274: Spring Data Commons, versions 1.13 to 1.13.10, 2.0 to 2.0.5, and older unsupported versions, contain a property path parser vulnerability caused by unlimited resource allocation. An unauthenticated remote malicious user (or attacker) can issue requests against Spring Data REST endpoints or endpoints using property path parsing which can cause a denial of service (CPU and memory consumption).

CVE-2020-1938: When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connections to Apache Tomcat. Tomcat treats AJP connections as having higher trust than, for example, a similar HTTP connection. If such connections are available to an attacker, they can be exploited in ways that may be surprising. In Apache Tomcat 9.0.0.M1 to 9.0.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99, Tomcat shipped with an AJP Connector enabled by default that listened on all configured IP addresses. It was expected (and recommended in the security guide) that this Connector would be disabled if not required. This vulnerability report identified a mechanism that allowed: - returning arbitrary files from anywhere in the web application - processing any file in the web application as a JSP Further, if the web application allowed file upload and stored those files within the web application (or the attacker was able to control the content of the web application by some other means) then this, along with the ability to process a file as a JSP, made remote code execution possible. It is important to note that mitigation is only required if an AJP port is accessible to untrusted users. Users wishing to take a defence-in-depth approach and block the vector that permits returning arbitrary files and execution as JSP may upgrade to Apache Tomcat 9.0.31, 8.5.51 or 7.0.100 or later. A number of changes were made to the default AJP Connector configuration in 9.0.31 to harden the default configuration. It is likely that users upgrading to 9.0.31, 8.5.51 or 7.0.100 or later will need to make small changes to their configurations.

CVE-2020-9484: When using Apache Tomcat versions 10.0.0-M1 to 10.0.0-M4, 9.0.0.M1 to 9.0.34, 8.5.0 to 8.5.54 and 7.0.0 to 7.0.103 if a) an attacker is able to control the contents and name of a file on the server; and b) the server is configured to use the PersistenceManager with a FileStore; and c) the PersistenceManager is configured with sessionAttributeValueClassNameFilter="null" (the default unless a SecurityManager is used) or a sufficiently lax filter to allow the attacker provided object to be deserialized; and d) the attacker knows the relative file path from the storage location used by FileStore to the file the attacker has control over; then, using a specifically crafted request, the attacker will be able to trigger remote code execution via deserialization of the file under their control. Note that all of conditions a) to d) must be true for the attack to succeed.

CVE-2021-24122: When serving resources from a network location using the NTFS file system, Apache Tomcat versions 10.0.0-M1 to 10.0.0-M9, 9.0.0.M1 to 9.0.39, 8.5.0 to 8.5.59 and 7.0.0 to 7.0.106 were susceptible to JSP source code disclosure in some configurations. The root cause was the unexpected behaviour of the JRE API File.getCanonicalPath() which in turn was caused by the inconsistent behaviour of the Windows API (FindFirstFileW) in some circumstances.

CVE-2021-25122: When responding to new h2c connection requests, Apache Tomcat versions 10.0.0-M1 to 10.0.0, 9.0.0.M1 to 9.0.41 and 8.5.0 to 8.5.61 could duplicate request headers and a limited amount of request body from one request to another meaning user A and user B could both see the results of user A's request.

## 5. Mitigation Plan

After interpreting your results from the manual review and static testing, identify the steps to remedy the identified security vulnerabilities for Artemis Financial’s software application.

[Include your findings here.]

1. Authentication - Use Spring Security for authentication. Whether any part of such web app is prohibited, usernames and passwords are advised. For data encryption, use the Spring Security Crypto Encryptors class. This subclass will employ symmetric cryptography to decode credential codec.

2. Access control may be done using the Spring Security framework or a bespoke solution. Easy to establish role-based identity management. If indeed the app is complicated, authorization access using Apache Shiro or more powerful essential element password protection via Spring Security should be explored.

3. Input Validation- The REST controllers must be annotated using @Validated or other Spring-provided proper input attributes. Whenever feasible, employ unrestricted verification over ban list verification. The output of parsing numeric strings into Java numerical classes should be validated for length and range.

4. Exception Handling - controllers functions must be annotated with @ExceptionHandler to indicate how and where to handle controller occurrences. Errors reports should clarify what occurred without disclosing API inner workings.

The client/server connections should then be protected and encrypted using SSL with Spring Security.

Bank details and balance must be handled through an authorized 3rd API. An public banking/financial API like the Citigroup API, for instance, may enable a web app access user bank information and transfer money across accounts. Artemis Financial should use these trusted financial APIs to handle sensitive financial data and never keep client data unprotected or without professional assistance.

7. Activity Logging- This software already uses Log4j but does not utilize it. It should be used to log server requests.

8. Whenever contemplating encapsulation, apply the standard user concept. Class and package-specific characteristics and procedures shouldn't be publicly disclosed.

The action plan is as follows :

Update Apache Log4APi to the Latest Version

Update Spring Framework to Latest Versions

Update Bouncy Castle to Latest Versions