****

# Artemis Financial Vulnerability Assessment Report

**Table of Contents**

[Document Revision History 3](#_gjdgxs)

[Client 3](#_30j0zll)

[Instructions 3](#_1fob9te)

[Developer 4](#_3znysh7)

[1. Interpreting Client Needs 4](#_2et92p0)

[2. Areas of Security 4](#_tyjcwt)

[3. Manual Review 4](#_3dy6vkm)

[4. Static Testing 4](#_1t3h5sf)

[5. Mitigation Plan 4](#_4d34og8)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **Peter F. Tumulty** |  |

## 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

Peter F. Tumulty

## Interpreting Client Needs

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

Artemis Financial develops customized financial plans for its customers, covering savings, retirement, investments, insurance, and more. Given that the company possesses data on their clients' financial information, protecting this information is paramount. Therefore, all data communicated over their API services must be highly secure and monitored at all times.

**Does the company make any international transactions?**

Without reviewing the financial information of Artemis Financial's clients and understanding the legal structure of the business, we cannot be certain that the company is engaged in international transactions. However, considering that Global Rain, the company we represent, primarily develops software for 'entrepreneurs, businesses, and government agencies around the world', we can operate under the assumption that Artemis Financial has some aspect of their business dealing with international transactions. Learning whether Artemis Financial does, in fact, engage in international transactions should be a priority for the team interacting with the client. This information ought to be discovered during the initial meetings while Artemis Financial is in the process of becoming a client of Global Rain.

**Are there governmental restrictions about secure communications to consider?**

When it comes to developing financial software, there are several governmental restrictions and compliances that need to be met. Artemis Financial's API, for instance, must adhere to FFIEC Compliance. This includes guidelines like implementing a layered approach to user privilege access to information, multi-factor authentication, etc. ensuring that the software design incorporates controlled access. It's not just about ticking boxes; it's about safeguarding user data and maintaining trust.

Additionally, the API needs to comply with FAPI Security Standards, set by a group within the OpenID Foundation focused on enhancing API security. A key initiative here is the secure authentication of all APIs that handle confidential information.

While there are more governmental restrictions and compliance issues to consider, the specifics should be guided by the project owners and legal teams at Global Rain and Artemis Financial.

**References:**

* Authentication and Access to Financial Institution Services and Systems. (n.d.). www.ffiec.gov. <https://www.ffiec.gov/press/pdf/Authentication-and-Access-to-Financial-Institution-Services-and-Systems.pdf>
* How OpenID Connect Works. (n.d.). openid.net. <https://openid.net/developers/how-connect-works/>

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

Based on a general overview of the client's needs, it is challenging to determine the specific threats Artemis Financial may face while developing their API. However, three immediate and prevalent threats are: malicious actors attempting to infiltrate their system to access client financial information, subpar coding standards that could unintentionally expose private data to unauthorized users, or even worse, to the general public, and the potential infiltration of viruses such as trojans, botnets, rootkits, spyware, and ransomware.

**What are the modernization requirements that you must consider?**

Considering Artemis Financial's request for us to audit the design of their API from a security perspective, we can also recommend technologies to address any issues we discover. If the issue is related to code practices, we can cite online sources that support best practices. Open-source libraries are an essential resource for providing technologies to address security issues from a design perspective. For any given technological problem, there is likely an open-source-based solution. However, it isn’t wise to select just any open-source-based solution. The selected open-source solution should be thoroughly researched to ensure it is well-supported by a quality community, has a log of its developmental history, and any known flaws are identified with instructions on how to patch the flaw. Once a technological solution is chosen, it should be rigorously tested within the system in a controlled environment before being released into production. Only after several rounds of testing and the support of the open-source library have been established should it be fully integrated into the codebase.

## Areas of Security

**Input Validation**

It's crucial to verify that all inputs to this API, whether they are GET, POST, UPDATE, PATCH, or DELETE requests, are serialized and parsed with security considerations in mind. Primarily, the serialization and parsing should be conducted on the client side before being passed to the API. However, some work also needs to be done on the API side. Fortunately, the API is developed using the Java programming language, a strongly-typed language that ensures the data processed by the API adheres to the data types specified by the developers.

**Cryptography (encryption use and vulnerabilities)**

The server hosting the API should be configured to use SSL/TLS certificates, ensuring that it is trusted by the client. From a client's perspective, the web application utilizing these API endpoints should be served using the HTTPS protocol, thereby ensuring that the connection between the client and server is fully encrypted.

**APIs (secure interactions)**

In compliance with U.S. and international governance regulations, all API endpoints should be secured with authentication middleware. Furthermore, authentication should be organized in layers, where users are authorized to access only segmented aspects of the data. The API should be designed to take these different layers of authentication into consideration.

**Client/Server (secure distributed composing)**

The server on which the codebase runs should also be secure. This means having updated certificates and locking down specific directories, as well as employing server management best practices. As previously mentioned, the client/server communication should be encrypted end-to-end, using SSL/TCP certificates on the server and the HTTPS protocol on the client. Furthermore, a security and server architect should audit the server's architecture, ensuring the read/write access of any sensitive directories in the operating system is appropriately restricted.

**Code Error (secure error handling)**

This should be implemented for 2 reasons: inform the user there is a network problem and contact customer support & to inform the developers where the source of the problem is located in the code base. The codebase should be designed to include error handling. There are two main benefits to having proper error handling in any codebase, especially when security is a priority. The first benefit is that it provides a way to communicate an issue to the user and offer instructions on the next steps. Additionally, it informs the maintainer of the codebase about which area of the code is the source of the problem.

**Code Quality (secure coding practices)**

When it comes to code quality, well-documented and commented code is crucial for fellow developers, and using a strongly typed language like Java ensures the data passed through the application remains consistent. Additionally, maintaining high-quality Java code is vital for preventing security breaches, as it helps in identifying and mitigating risks like SQL injection and cross-site scripting.

**Encapsulation (secure data structure)**

Thankfully, the API is designed using the Java programming language, which is object-oriented, so encapsulation is inherently integrated into the design. However, novice programmers can still write code that disrupts the encapsulation aspects. Therefore, it is important to follow coding best practices for the Java programming language.

## Manual Review

**APIs (secure interactions)**

Both CRUDController.java and GreetingController.java should be updated to include a dependency that incorporates an authentication wrapper around the endpoint. Then, the method should be updated to incorporate logic controlling what happens when the user has or does not have access to use this endpoint.

**Code Error (secure error handling)**

Both CRUDController.java and GreetingController.java should include proper error handling. Implementing try/catch blocks in the code can logically structure the endpoints. In the try block, the code executes the purpose of the endpoint successfully. Meanwhile, the catch block handles unexpected code executions. Additionally, this catch block is where we can implement functionality to manage situations where the user lacks authorization to use the endpoint.

**Code Quality (secure coding practices)**

There are several instances where this codebase needs updating to ensure code quality and secure coding practices. The first issue that needs to be fixed is the removal of the comment that provides the database name and password in the DocData.java file. Having this information available in the codebase is a big red flag and should be removed as soon as possible. Instead, please put those details in an encrypted text file and share it directly with the development team. Another issue in the DocData.java file is that the connection string contains database details that should also be private. Please update this string to include environment variables that can be hidden, instead of hardcoding the password, username, database name, etc.

The final aspect that needs addressing is returning the full instance of the CRUD object in the CRUDController.java. Returning a full instance of the CRUD object exposes all methods and internal variables, which should not happen when the user accesses the endpoint, regardless of the user's authorization. Please update the 'Read' method to only execute the CRUD object's read method and any established variables or data that are designed to be used with this method. After reviewing the codebase, though it is still in development, these issues should be addressed before further development

## Static Testing

| **Dependency** | **Vulnerability IDs** | **Package** | **Highest Severity** | **CVE ID** | **Description** |
| --- | --- | --- | --- | --- | --- |
| bcprov-jdk15on-1.46.jar | [cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Abouncycastle&cpe_product=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api&cpe_version=cpe%3A%2F%3Abouncycastle%3Alegion-of-the-bouncy-castle-java-crytography-api%3A1.46) | [pkg:maven/org.bouncycastle/bcprov-jdk15on@1.46](https://ossindex.sonatype.org/component/pkg:maven/org.bouncycastle/bcprov-jdk15on@1.46?utm_source=dependency-check&utm_medium=integration&utm_content=5.3.0) | High | CVE-2023-6378 | A serialization vulnerability in logback receiver component part of logback version 1.4.11 allows an attacker to mount a Denial-Of-Service attack by sending poisoned data. |
| og4j-api-2.12.1.jar | [cpe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache&cpe_product=cpe%3A%2F%3Aapache%3Alog4j&cpe_version=cpe%3A%2F%3Aapache%3Alog4j%3A2.12.1) | [pkg:maven/org.apache.logging.log4j/log4j-api@2.12.1](https://ossindex.sonatype.org/component/pkg:maven/org.apache.logging.log4j/log4j-api@2.12.1?utm_source=dependency-check&utm_medium=integration&utm_content=5.3.0) | Critical | CVE-2021-45046 | It was found that the fix to address CVE-2021-44228 in Apache Log4j 2.15.0 was incomplete in certain non-default configurations. |
| snakeyaml-1.25.jar | [cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Asnakeyaml_project&cpe_product=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml&cpe_version=cpe%3A%2F%3Asnakeyaml_project%3Asnakeyaml%3A1.25)  [cpe:2.3:a:yaml\_project:yaml:1.25:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Ayaml_project&cpe_product=cpe%3A%2F%3Ayaml_project%3Ayaml&cpe_version=cpe%3A%2F%3Ayaml_project%3Ayaml%3A1.25) | [pkg:maven/org.yaml/snakeyaml@1.25](https://ossindex.sonatype.org/component/pkg:maven/org.yaml/snakeyaml@1.25?utm_source=dependency-check&utm_medium=integration&utm_content=5.3.0) | Critical | CVE-2022-1471 | SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization. Deserializing yaml content provided by an attacker can lead to remote code execution. |
| tomcat-embed-core-9.0.30.jar | [cpe:2.3:a:apache:tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache&cpe_product=cpe%3A%2F%3Aapache%3Atomcat&cpe_version=cpe%3A%2F%3Aapache%3Atomcat%3A9.0.30)  [cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aapache_tomcat&cpe_product=cpe%3A%2F%3Aapache_tomcat%3Aapache_tomcat&cpe_version=cpe%3A%2F%3Aapache_tomcat%3Aapache_tomcat%3A9.0.30) | [pkg:maven/org.apache.tomcat.embed/tomcat-embed-core@9.0.30](https://ossindex.sonatype.org/component/pkg:maven/org.apache.tomcat.embed/tomcat-embed-core@9.0.30?utm_source=dependency-check&utm_medium=integration&utm_content=5.3.0) | High | CVE-2023-46589 | Improper Input Validation vulnerability in Apache Tomcat.Tomcat from 11.0.0-M1 through 11.0.0-M10, from 10.1.0-M1 through 10.1.15, from 9.0.0-M1 through 9.0.82 and from 8.5.0 through 8.5.95 did not correctly parse HTTP trailer headers. |
| hibernate-validator-6.0.18.Final.jar | [cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\*](https://nvd.nist.gov/vuln/search/results?form_type=Advanced&results_type=overview&search_type=all&cpe_vendor=cpe%3A%2F%3Aredhat&cpe_product=cpe%3A%2F%3Aredhat%3Ahibernate_validator&cpe_version=cpe%3A%2F%3Aredhat%3Ahibernate_validator%3A6.0.18) | [pkg:maven/org.hibernate.validator/hibernate-validator@6.0.18.Final](https://ossindex.sonatype.org/component/pkg:maven/org.hibernate.validator/hibernate-validator@6.0.18.Final?utm_source=dependency-check&utm_medium=integration&utm_content=5.3.0) | Medium | 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. |

## Mitigation Plan

* Remove the database authentication details out of the codebase and put them in a secure file that should be shared with the developer directly.
* Refactor the codebase to use best coding practices:
  + Implement error handling logic for each API method
  + Add authentication middleware to each API endpoint
  + Using OOP best practices and do not return the entire initialization of the object, but only the intended methods
* Finally, repair the following critical dependencies: Bouncy Castle Crypto package, Logback Core, Apache Log4j2, SnakeYaml, Apache Tomcat by either installing the latest version of the dependency or following the specific instructions from the CVE.