****

# 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** | **3/19/2023** | **Connor Hudson** |  |

## 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

Connor Hudson

## Interpreting Client Needs

Secure communications will be very important to Artemis Financial as they deal with sensitive financial information such as savings, retirement, investments, and insurance plans. If this information falls into the wrong hands, it could lead to financial loss, identity theft, or other serious consequences. Secure communications will help to protect the confidentiality, integrity, and availability of this information, which is critical to business operations and customer trust.

If Artemis Financial plans to handle international transactions, there are potential risks and threats associated with cross-border payments, such as exchange rate fluctuations, compliance with international regulations and sanctions, and potential fraud or money laundering. These risks can be mitigated through measures such as using secure payment gateways, verifying the identity of customers and recipients, and complying with relevant regulations and standards.

There may be relevant regulations or standards that apply to financial services providers, such as the Payment Card Industry Data Security Standard (PCI DSS) or the General Data Protection Regulation (GDPR) in the European Union. If such regulations or standards apply, Artemis Financial would need to ensure that their software application complies with the relevant requirements.

There are many potential external threats that Artemis Financial could face, including hacking, phishing, malware, or social engineering attacks. These threats are constantly evolving and can be difficult to anticipate. Therefore, it is important to have a robust security program in place that includes measures such as access controls, intrusion detection and prevention, encryption, and user education to mitigate these threats.

If Artemis Financial is seeking to modernize their operations, it is necessary to consider the role of open-source libraries and evolving web application technologies in order to stay competitive and provide the best possible user experience for their customers. This will require careful consideration of the potential security risks associated with these changes, as well as measures to mitigate these risks.

## Areas of Security

1. Input Validation: This vulnerability is important for any web-based software application, as it involves checking that user input is valid and safe before it is processed. Failure to properly validate input can lead to a range of security issues, including SQL injection and cross-site scripting (XSS) attacks.
2. Secure API Interactions: As Artemis Financials’ application includes a RESTful web API, it is important to ensure that all interactions with the API are secure. This includes using secure communication protocols such as HTTPS, properly authenticating and authorizing API users, and protecting against common API security vulnerabilities such as injection attacks.
3. Encryption Use and Vulnerabilities: As the financial data handled by Artemis Financials’ application is likely to be highly sensitive, it is important to use encryption to protect it from unauthorized access. However, if encryption is not properly implemented or configured, it can itself be a vulnerability. Ensuring proper encryption use and addressing encryption vulnerabilities should be a priority.
4. Secure Coding Practices/Patterns: Secure coding practices and patterns should be a consideration for any software development project. This includes using secure coding standards and guidelines, conducting code reviews, and ensuring that all code is written with security in mind.

## Manual Review

Upon manual review of the application, there are several potential security issues. I will organize them based on the file in which they appear:

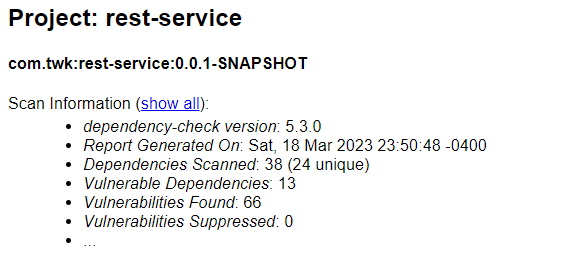
CRUDController:

1. Lack of Input Validation: The **name** parameter is accepted from the client via the **@RequestParam** annotation, but it is not being validated. This means that an attacker could potentially supply malicious input, such as SQL injection attacks or cross-site scripting (XSS) attacks, which could cause harm to the system or compromise sensitive data.
2. Lack of Authentication/Authorization: The code does not appear to have any authentication or authorization mechanism in place, which means that anyone with access to the API endpoint can perform the read operation on the data.
3. Lack of HTTPS: The code is not using HTTPS, which means that the communication between the client and the server is not encrypted. This could allow an attacker to intercept and view the data being sent between the client and the server.

DocData:

1. SQL Injection Vulnerability: The **read\_document** method accepts two parameters, **key** and **value**, which are used to construct a SQL query without proper validation or sanitization. This can lead to SQL injection attacks if an attacker can manipulate the input to execute arbitrary SQL commands.
2. Lack of Authentication and Authorization: The **read\_document** method does not enforce any authentication or authorization checks, which means anyone can access the database and execute SQL queries.
3. Unencrypted Database Connection: The database connection is established using the **jdbc:mysql://localhost:3306/test** URL without any encryption, which means the data in transit can be intercepted and read by an attacker.

## Static Testing



Graphical user interface, text, application, email

Description automatically generated

The following vulnerability codes were generated in the report descending from highest severity:

Critical:

* CVE-2022-38749
* CVE-2022-38750
* CVE-2022-38751
* CVE-2022-38752
* CVE-2022-41854
* CVE-2022-42003 (2x)
* CVE-2022-42004

High:

* CVE-2016-1000338
* CVE-2016-1000339
* CVE-2016-1000341
* CVE-2016-1000342
* CVE-2016-1000343
* CVE-2016-1000344
* CVE-2016-1000345
* CVE-2016-1000346
* CVE-2016-1000352
* CVE-2017-13098
* CVE-2018-5382
* CVE-2019-17569
* CVE-2020-0187
* CVE-2020-25649
* CVE-2020-36518
* CVE-2020-5421
* CVE-2020-8022
* CVE-2020-9484
* CVE-2020-9488
* CVE-2020-11996
* CVE-2020-13934
* CVE-2020-13935
* CVE-2020-13943
* CVE-2020-17527
* CVE-2020-1935
* CVE-2020-1938
* CVE-2021-22060
* CVE-2021-22096
* CVE-2021-24122
* CVE-2021-25122
* CVE-2021-25329
* CVE-2021-30640
* CVE-2021-33037
* CVE-2021-41079
* CVE-2021-42550
* CVE-2021-43980
* CVE-2021-44228
* CVE-2021-44832
* CVE-2021-45046
* CVE-2021-45105
* CVE-2021-4235 (2x)
* CVE-2022-1471
* CVE-2022-22950
* CVE-2022-25857
* CVE-2022-27772
* CVE-2022-29885
* CVE-2022-3064
* CVE-2022-34305
* CVE-2022-42252

Medium:

* CVE-2013-1624
* CVE-2015-6644
* CVE-2015-7940
* CVE-2016-1000027
* CVE-2017-18640
* CVE-2020-26939
* CVE-2021-22118
* CVE-2021-25329 (2x)

Low:

* CVE-2020-13943 (2x)

## Mitigation Plan

CRUDController:

1. Input Validation: Validate the **name** parameter to ensure that it is in the expected format and does not contain any malicious code. For example, you can use a regular expression to validate that the **name** parameter only contains alphanumeric characters.
2. Authentication/Authorization: Implement an authentication and authorization mechanism to ensure that only authorized users can access the API endpoint. For example, you can use JSON Web Tokens (JWTs) to authenticate users and restrict access to certain resources.
3. HTTPS: Use HTTPS to encrypt the communication between the client and the server to prevent eavesdropping and data interception. You can obtain an SSL certificate from a trusted certificate authority to enable HTTPS on your server.

DocData:

1. Use parameterized SQL queries instead of constructing the SQL query using input values to prevent SQL injection attacks.
2. Implement authentication and authorization checks to ensure that only authorized users can access the database.
3. Use an encrypted database connection (e.g., HTTPS) to protect the data in transit.

Dependencies:

The majority of the CVE codes generated lie in vulnerabilities within the Apache Tomcat and Log4j libraries and the BouncyCastle JCE. Many of the issues flagged may be resolved by updating these libraries to their most current versions. Other vulnerabilities lie within features that have been removed entirely from the vendors. The remainder of the codes are in regard to features that are not present within this program thus false positives. In summation, by keeping an updated dependency library, the risk factors can be greatly reduced.