****

# 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** | **9-12-23** | **Tiffany McDonnell** | **Initial implementation** |

## Client



## Developer

Tiffany McDonnell

## Interpreting Client Needs

* Artemis is a company that deals with finance. Since we are dealing with finances, Artemis could be considered heavily targeted for attacks. Therefore, security must be of the utmost importance. Artemis has not only their own security but that of their clients as well. Security will have to be carefully watched and stay up to date to keep all data secure from vulnerabilities.
* Currently Artemis is not dealing with anything international. Due to the fact that this is a finance application and people who travel may want to access their funds, international transactions may end up needing to be implemented. If international transactions are implemented, then there is a lot to consider in the security and other country's laws.
* There are many governmental restrictions to consider. The first is ECPA(Electronics Communications Privacy Act). This act is put in place a restriction to help protect a person's private data during an electronic communication, which in our case would be fund transfers. Another one to consider is GLBA(Gramm-Leach-Baliley Act). Although its not on the books for this application at this time, it is a financial application. This is put in place to make the company let the users know how their data is being shared when applicable. As long as the application stays simple and only allows deposits, then this act does not need to be considered but the company may want to expand into investments.
* Being a financial application Hackers are a threat. There are many people out there who would try to access either the company's or a clients funds and take it for themselves. Another threat that could become an issue would be DOS attack. A denial-of-service attack would freeze multiple accounts and make funds accessible. This kind of attack is also dangerous to the company’s reputation for security because it will become well-known quickly.
* Open-source libraries are very useful and very commonly used. Most of which are already coded in a secure manner but should always be considered untrusted data and treated as such. All technology is consistently growing and improving, but at the same time so are the threats against it. We much always stay up to date in order to keep all data and applications secure.

## Areas of Security

For this application the first area of security to focus on would be Input Validation. Any outside source of data must be validated and sanitized in order to keep the application safe. Any type of input could potentially be harmful whether on purpose or not. Since it can be harmful it all should be treated as untrusted data and be sanitized.

The second area of security that must be focused on is API interactions. The application runs using a RESTful API and must stay secure. This area of security is used to help validate not just data but the users are who they say they are. This helps to protect not just the company but the clients as well.

Code errors is another area of security to consider. The application must be able to handle any errors that may show up. If issues are not handled properly then the application could break or shut down. Being a financial application dealing with other people's funds and possibly their livelihood, all errors must be prevented. Following code errors encapsulations should also be done so there are no unwanted changes to the code that could potentially break the application.

Encryption is another must because of being a financial application. All data going from system to user and vice versa must be encrypted to data cannot be seen from outside views. This will ensure that as a user logs in no one can see they username, password, or other personal data can be viewed for an unwanted user to access.

## Manual Review

Starting with the main method, the input is very clearly not being sanitized with input validation. The string that is inserted automatically gets run and then checked by the Spring framework. If there is something harmful within the code or input, it would be too late to stop since it was already being run. Both the CRUDController and GreetingController are doing something similar. They both are taking in a string but nowhere is it validating and sanitizing input being used.

There are a few different object classes within the code. The first one is the Greeting class. It contains two private variables, a constructor, and 2 public methods. One method is used to obtain the greeting’s ID, and the other to obtain the greeting’s content. The CRUD class contains 2 private variables, a constructor, an overloaded constructor, and 2 public methods. One method is used to obtain CRUD’s content1 and the other to obtain CRUD’s content2. The DocData class has one private member, a default constructor, and two methods. The first method is used to return and ID, while the other does a bit of error handling. Another object class is the customer class. It contains one private member and a regular variable. It also has 2 methods; one returns a customer’s account balance and the other adds a deposit to the account balance. The last class has 3 variables that hold hours, minutes, and seconds. It has 2 methods. One of which returns the time and the other sets the time.

## Static Testing

After following a dependency check on the application, there were 75 vulnerabilities found within 13 different dependencies. The dependencies were as follows:

* bcprov-jdk15on-1.46.jar
* spring-boot-2.2.4.RELEASE.jar
* logback-core-1.2.3.jar
* log4j-api-2.12.1.jar
* snakeyaml-1.25.jar
* jackson-databind-2.10.2.jar
* tomcat-embed-core-9.0.30.jar
* hibernate-validator-6.0.18.Final.jar
* spring-web-5.2.3.RELEASE.jar
* spring-beans-5.2.3.RELEASE.jar
* spring-webmvc-5.2.3.RELEASE.jar
* spring-context-5.2.3.RELEASE.jar
* spring-expression-5.2.3.RELEASE.jar

A screenshot of a computer

Description automatically generated

Looking into the most critical vulnerabilities in “log4j-api-2.12.1.jar”, the big issue is that the current version being used is vulnerable to a MITM attack. Being a financial application, this is a big deal and a heavy threat. A hacker could potentially find a way to read the communication from user to system to gain access to a client’s funds. Another vulnerable dependency is “tomcat-embed-core-9.0.30.jar”. This vulnerability is vulnerable to HTTP request smuggling. If someone went through with this type of attack, they could easily bypass all security protocols. This would allow them to access sensitive data that should be secured from unwanted users.

## Mitigation Plan

For a mitigation plan to be effective, all proper areas of security must be taken care of. The first one is a big must. All input must be validated and sanitized. For an extra measure, an API should be used. The next area that should be secured would be making sure that all types of errors are handled appropriately. If they are not handled the application can break and data leaked. Along with the application breaking, encapsulation should be used to prevent any unwanted changes to the code and help prevent any errors from occurring later. The last area should be cryptography. Encryption must be used while transfer of data occurs. This will help protect a user’s identity and keep it secure. For an extra measure, all dependencies used need to stay up to date. As vulnerabilities are found, patches get released helping to secure from new attacks.

**Citations**

*How to read the reports*. dependency-check – How To Read The Reports. (n.d.). https://jeremylong.github.io/DependencyCheck/general/thereport.html

Manico, J., & Detlefsen, A. (n.d.). *Iron-clad Java*. O’Reilly Online Learning. https://learning.oreilly.com/library/view/iron-clad-java/9780071835886/?sso\_link=yes&sso\_link\_from=SNHU

The New York Times. (2021, September 6). *The State of Consumer Data Privacy Laws in the US (and why it matters)*. The New York Times. https://www.nytimes.com/wirecutter/blog/state-of-privacy-laws-in-us/

*What is HTTP request smuggling? tutorial & examples: Web security academy*. What is HTTP request smuggling? Tutorial & Examples | Web Security Academy. (n.d.). https://portswigger.net/web-security/request-smuggling

*What is MITM (man in the middle) attack: Imperva*. Learning Center. (2019, December 29). https://www.imperva.com/learn/application-security/man-in-the-middle-attack-mitm/