****

# 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** | **[Date]** | **Jeremy Leuck** |  |

## 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

Jeremy Leuck

## 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?
* Are there any international transactions that the company produces?
* Are there governmental restrictions about secure communications to consider?
* What external threats might be present now and in the immediate future?
* What are the “modernization” requirements that must be considered, such as the role of open-source libraries and evolving web application technologies?
* Secure communication is vital for the company and its program. We must secure communication and data transmission between the server and the client. This will lead to no loss of data, and no tampering of data from outside the system. The user and company will both be protected if communication is secure.
* Yes, international transactions will be required by the company. For example, if someone who has access to the bank is on a trip, they will want to access their account from where they are located.
* There are government restrictions are in place to secure communication seeing as it is a financial institution, but the company is also encouraged to produce as much protection as possible. As said above, this will allow us to protect the data and keep the user private from outside dangers.
* External threats will always cause problems for the company. The data will be vulnerable to attackers since hackers learn more and more about the system in place and will be able to eventually break it. This leads to the fact that the company and developers will constantly need to improve and advance the program without leaving holes in the security.
* Open-source libraries, a form of modernization requirements, will improve the security of the system all around. This adds layers of protection and functionality to the program by allowing everyone to work on it and not just a few devs who are given the access to the security code.

## 2. Areas of Security

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

* Input Validation: Seeing as we have input functions, we need to be able to secure and validate all possible inputs to ensure system security and prevent any sort of data leaks from an outside attack.
* API’s: Since the command function will be accessed from outside the system, we will implement an API which will allow access to the code. We will also implement input validation tests on the API’s to ensure the safety of them as well.
* Cryptography: Seeing as the information is of top importance, we need to implement proper encryption of the data which will protect the system and the customer data like all other tests so far.
* Client/Server access: Since we installed API’s to the system, we need to ensure the data is safe during requests of transferring data, as well as simple connections between Client and Server access.
* Code Error: We will manually run a code review to look at all layers, input functions, and possible flaws in the code that may not have been accounted for when writing the program. This will ensure a machine and human eye checks the program for issues.
* Encapsulation: Seeing as data is being stored and needed to be accessed, we must properly ensure that the data structures are protected and not being messed up.

## 3. Manual Review

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

* In the pom.xl file in the program, there is a line that reads “spring-data-rest-webmvc”. This implies that we are dependent on sprint data, and this is now out of date. Seeing as the new version in 3.3.0 and not 2.6.5, this can cause breaches due to hackers having more access to the older version and know more ways to breach the file. Not only that, but we should never put a new program out with shoddy/out of date technology.
* GreetingController.java: In the code, we take data in from a parser between the lines 15-17. We then will add this text, that was turned into a string, into the template. This can cause problems for the server since this is how we store all the data, as someone can send a fake string format into the data center and ruin the system later.
* DocData.java: Even though it is not a huge issue, the variable “con” is declared but is never used. This is not an issue due to the fact that it is test, it is in the wrong package and could really wind-up throwing issues if someone is able to gain access to this program.

## 4. Static Testing

Run a dependency check on Artemis Financials’ 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

* [bcprov-jdk15on-1.46.jar](#l1_991c96a4e31e6c19e2b9136c8955bd423f2d): 7.4 High. In the Bouncy Castle JCE Provider version 1.55 and earlier the ECIES implementation allowed the use of ECB mode. This mode is regarded as unsafe and support for it been removed from the provider.
* [hibernate-validator-6.0.18.Final.jar](#l2_7fd00bcd87e14b6ba66279282ef15efa30dd): 5.3 Medium. 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.
* [jackson-databind-2.10.2.jar](#l5_0528de95f198afafbcfb0c09d2e43b6e0ea6): 7.5 High. Does 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.
* [log4j-api-2.12.1.jar](#l10_a55e6d987f50a515c9260b0451b4fa217dc): This flaw is related to APIs and cloud attacks. This allows attackers to stick themselves into a program that is controlled off site, which causes a huge issue for a majority of programs.
* [logback-core-1.2.3.jar](#l11_864344400c3d4d92dfeb0a305dc87d95367): Currently, according to the NVD, solutions are being reanalyzed due to the fact that the vulnerability has recently been modified. Unfortunately for developers this means code is open to attacks, however the attackers themselves are still learning to newer dependency as well.
* [snakeyaml-1.25.jar](#l14_8b6e01ef661d8378ae6dd7b511a7f2a33fa): This seems like a simple lack of testing issue. There is an obvious flaw since there is a loading operation that causes a security breach. This should be a fix that devs can improve upon quickly and fix the bugs of that particular operation.
* [spring-aop-5.2.3.RELEASE.jar](#l15_9cdd9a1dd636331767fffcc7fe49a7bb00e): Again, this is a scary attack threat seeing as it seems to be a lack of judgement on the testing side.
* [spring-core-5.2.3.RELEASE.jar](#l17_3734223040040e8c3fecd5faa3ae8a1ed6d): Having a leak of memory is dangerous for data theft but also program functionality.
* [tomcat-embed-core-9.0.30.jar](#l18_ad32909314fe2ba02cec036434c0addd19b): Refer to last bullet.
* [tomcat-embed-websocket-9.0.30.jar](#l20_33157f6bc5bfd03380ebb5ac476db0600a0): Refer to last bullet.

## 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 Financials’ software application.

* Update the pom.xl file line to the most modern version to keep the program safe and running for as long as possible.
* We can translate the string twice in order to make it harder to breach, or perhaps limit the string size so that dangerous data cannot be implemented into the program. We can also protect the data by verifying length of input, as well as implementing blocks/errors if the input is possibly harmful.
* Seeing as this isn’t necessarily an issue, it doesn’t need a big fix. I would suggest either moving this to the testing folder however, seeing as that will be protected and organized in a better way.
* As a team we should change the controls that developers have put into place to control input handling, not allowing the data to be stripped
* Once again we face the problem of having data leaks and integrity problems. We need to implement walls for the code for external breaches when expanding entities.
* We can fix this by not only validating input of API’s but also validation of cloud access in multiple forms.
* Seeing as this is a typical issue because of other similar flaws, we can check other programs that have encountered this problem and fixed them in a similar means of open sourced sandbox languages.
* This is something simple like specific browsers cause breaks in the security, which can be fixed on the testing side.. We can fix this by validating the client side access and creating a list of safe browsers, only those able to access it.
* This can be fixed by creating a function that ensures all data is locked once the program is closed, or the connection is lost. That way storage will not be filled by unneeded memory that was accidently leaked.