

# 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]** | **[Your name]** |  |

## 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

Liam Nunes

## Interpreting Client Needs

Interpreting Client Needs: Review the scenario to determine your client’s needs and potential threats and attacks associated with their application and software security requirements. Document your findings in your vulnerability assessment report. Consider the scenario information and the following questions regarding how companies protect against external threats:

1. What is the value of secure communications to the company?
2. Does the company make any international transactions?
3. Are there governmental restrictions about secure communications to consider?
4. What external threats might be present now and in the immediate future?
5. What are the modernization requirements that you must consider? For example:
   1. The role of open-source libraries
   2. Evolving web application technologies

The client, Artemis Financial, a financial consulting company that works with their customers to develop customized financial plans. As such it is imperative that they have top notch security as their customers trust them with their financial and personal information. If there security was to be breached it would cost them the trust of all of their customers, potentially ruining the business. It is unknown but highly likely that they make international transactions. There are without a doubt many governmental restrictions requirement secure information and communications as they are dealing directly with individuals protected information. External threats could include people trying to steal their clients information and money to threats directly to their business and attacking them directly for their money and reputation. They should be running the most up to date software possible at all times to prevent any vulnerabilities and regular maintenance is a must.

## Areas of Security

Use what you’ve learned in step 1 and refer to the Vulnerability Assessment Process Flow Diagram provided. Think about the functionality of the software application to identify which areas of security apply to Artemis Financial’s web application. Document your findings in your vulnerability assessment report and justify why each area is relevant to the software application.

Cryptography- Artemis Financial needs to be able to encrypt customer and financial information when using the web application so that information cannot be stolen

Input Validation- Customers are going to be needing to input information onto the web application such as login information and potentially some from of request, so ensuring secure input is very important.

Code quality- The code for the web application but be held to a high standard as any vulnerability could leave a weak point in the web application.

APIs- The APIs are going to need to be able securely handle requests from the company and from customers while making sure that no bad actors can take advantage of them to get information they should not have access to.

## Manual Review

Refer to the seven security areas outlined in the Vulnerability Assessment Process Flow Diagram. Use what you’ve learned in steps 1 and 2 to guide your manual review. Identify all vulnerabilities in the Project One Code Base, linked in Supporting Materials, by manually inspecting the code. Document your findings in your vulnerability assessment report. Be sure to include a description that identifies where the vulnerabilities are found (specific class file, if applicable).

* In the customer class the account\_balance is not set to private which could allow it to be accessed when not intended
* There appears to be a lack of encryption for customer information in the given application, unless that is meant to be handled at a different level
* There is no validation for input in the greetings method of the GreetingController class

## Static Testing

Integrate the dependency-check plug-in into Maven by following the instructions outlined in the Integrating the Maven Dependency-Check Plug-in tutorial provided in Supporting Materials. Run a dependency check on Artemis Financial’s software application to identify all security vulnerabilities in the code. Specifically, identify all vulnerabilities in the code base by analyzing results from running the code through a static test. Include these items from the dependency-check report in your vulnerability assessment report:

* 1. The names or vulnerability codes of the known vulnerabilities
  2. A brief description and recommended solutions that are found in the dependency-check report
  3. Attribution (if any) that documents how this vulnerability has been identified or how it was documented in the past

Dependency: bcprov-jdk15on-1.46.jar

Vulnerability: cpe:2.3:a:bouncycastle:legion-of-the-bouncy-castle-java-crytography-api:1.46:\*:\*:\*:\*:\*:\*:\*

versions of Bouncy Castle JCE 1.55 and earlier allowed ECB mode which is regarded as unsafe

Solution: Update to latest version as the use of that mode was disabled

Dependency: logback-core-1.2.3.jar

Vulnerability: cpe:2.3:a:qos:logback:1.2.3:\*:\*:\*:\*:\*:\*:\*

In logback version 1.2.7 and earlier and attacker with required permissions could craft a malicious configuration to execure arbetrary code from LDAP servers

Solution:Update to Latest version of logback

Dependency: log4j-api-2.12.1.jar

Vulnerability: pe:2.3:a:apache:log4j:2.12.1:\*:\*:\*:\*:\*:\*:\*

Apache log4j versions prior to 2.17.0 ar venruable to remote code execution when the attacker has control of the target LDAP server

Solution: update to latest Apache log4j

Dependency: snakeyaml-1.25.jar

Vulnerability: cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25:\*:\*:\*:\*:\*:\*:\*

using snakeyaml to parse untrusted yaml files may open up vunrabilities to DoS attacks

Solution: Do not use snakeyaml to parse untrusted yaml files

Dependency: hibernate-validator-6.0.18.Final.jar

Vulnerability: cpe:2.3:a:redhat:hibernate\_validator:6.0.18:\*:\*:\*:\*:\*:\*:\*

In hibernate validator verison 6.1.2.final a bug enabled invalid el expressions to be evaluated as if they were valid

Solution: update to latest version

Dependency: spring-core-5.2.3.RELEASE.jar

Vulnerability:cpe:2.3:a:vmware:spring\_framework:5.2.3:release:\*:\*:\*:\*:\*:\*

versions of spring framework prior to 5.3.0 file uploads are vunerable to DoS attacks

Solution: Update to latest verson of spring framework

## Mitigation Plan

Interpret the results from the manual review and static testing report. Identify steps to mitigate the identified security vulnerabilities by creating an action list that documents how to fix each vulnerability in your vulnerability assessment report.

The first reccomendation that I would have is to immediately update all of the dependancies to the latest versions, this will prevent a lot of potential areas of concern. I would also reccomed fixing some spots in the code. Setting account\_balance to private in the Custoemr class will prevent access to it except from the getter in the class. I would also reccomed making sure that anywhere where there is a string input, such as in the greeting controller classes greeting method, input validation is implemented to ensure that noting bad can be injected into the string field. Finally I would reccomed ensuring that all personal data is encrypted and only accoiunts that should have acess to it can view the information.