****

# 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** | **07/15/2022** | **Janece Gates** |  |

## 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

Janece Gates

## 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?

The client, Artemis Financial, is a financial consultation company. Because of the nature of their business, the client will need to keep their own clients’ financial information on record, which means that this information will need to be kept secure. One potential attack associated with their application and software security requirements is that a hacker or malicious programmer could attempt to gain unauthorized access to one or more of their clients’ financial records. This would not only negatively affect their client, but would also negatively impact the business and reputation of Artemis.

Another issue for Artemis to worry about is the potential for their rivals to access non-financial information that is just sensitive enough to motivate their clients to leave them in favor of their rivals. This means that not only the financial information needs to be secured in the database, it should certainly be the priority, however all information related to the company or its clients should be kept as secure as possible.

Artemis must also be sure to keep itself appraised of all updates to technologies that they use in their system. Sometimes, an update to a particular technology could have unintended results and actually make what was once a secure technology become unsecure. This means that it is not enough to simply make sure that all technology used during the development of the project is secure, but they also must keep up with the updates and vulnerability reports of the technologies being utilized in their system and periodically verify their security.

## 2. Areas of Security

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

The areas of security that are applicable to Artemis Financial’s software application are:

* Input Validation
  + This area of security is applicable to Artemis Financial’s software application because given that the nature of their business entails providing a platform on which their clients can interact directly with the system, the application will certainly need to accept input from the user. If this input is not validated, a malicious user could potentially provide input that would crash or otherwise comprise the system.
* APIs
  + This area of security is applicable to Artemis Financial’s software application because, as discussed earlier, the company is going to be using a lot of 3rd party software, using such software is a double-edged sword. The developer is not responsible for creating certain features that will be included in the application, however this also means that the developer has minimal control over the behavior of the 3rd party software. The security of such software will need to be researched, tested and verified before being officially used in the system by clients.
* Cryptography
  + This area of security is applicable to Artemis Financial’s software application because, as also mentioned earlier, financial information is especially sensitive to clients and as such needs to be kept secure. Part of the process of securing information kept in storage is encrypting the data.
* Client / Server
  + This area of security is applicable to Artemis Financial’s software application because it’s application will need a distributed structure in order to service any of its clients needs at any given moment. This means that the connection and communication processes between the client and the server will need to be made secure.
* Code Error
  + This area of security is applicable to Artemis Financial’s software application, and is applicable to most applications ever created. Allowing the code to run into errors that have not been accounted for can lead to unpredictable behavior by the system. Sometimes these behaviors can be neutral in the sense that they don’t negatively impact the performance of the system but often times, they can be dangerous.
* Code Quality
  + This area of security is applicable to Artemis Financial’s software application and, similar to the above point, to most applications in general. Sloppy coding can lead to crashes and the exposure of sensitive aspects of the system that should remain secured.

## 3. Manual Review

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

During my manual review of the code base for Artemis Financial’s software application, I found the following vulneravilities:

* In the myDateTime Class, the accessor methods have not been implemented yet.
* Either in the GreetingController or the Greeting Class itself, there should be some sort of input validation. Currently, whatever the query parameter value is for “name”, it will accept that data without question and use it for the inner workings of the application. This is very dangerous as the user could potentially feed the program a string that would cause it to malfunction.
* In the DocData Class, the connection to the mysql database provides a password that is stored, in plaintext, right inside of the code file.
* In the customer class, the account\_number field is very unsecured for how sensitive the information is. The account number should rather be stored encrypted in a remote database and records should be accessed using public information such as a user’s customer ID. Only non-sensitive information should be stored in plaintext in a program.
* In the CRUD controller, the request parameter “business\_name”, is not only without input validation, but it’s actually not even utilized by the application after extraction. Furthermore, the DocData class does not have a custom to\_string() method, this means that the generic version from Java is being used, resulting in sensitive information about the DocData being exposed.

## 4. Static Testing

Run a dependency check on Artemis Financial’s 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

After running the dependency check on Artemis Financial’s software application code base, these are the security vulnerabilities that were identified:

* bcprov-jdk15on-1.46.jar
  + Code:
  + Description: The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7.
* hibernate-validator-6.0.18.Final.jar
  + Code:
  + Description: Hibernate's Bean Validation (JSR-380) reference implementation.
* jackson-databind-2.10.2.jar
  + Code:
  + Description: General data-binding functionality for Jackson: works on core streaming API
* jakarta.annotation-api-1.3.5.jar
  + Code:
  + Description: Jakarta Annotations API
* log4j-api-2.12.1.jar
  + Code:
  + Description: The Apache Log4j API
* logback-core-1.2.3.jar
  + Code:
  + Description: logback-core module
* snakeyaml-1.25.jar
  + Code:
  + Description: YAML 1.1 parser and emitter for Java
* spring-boot-2.2.4.RELEASE.jar
  + Code:
  + Description: Spring Boot
* spring-core-5.2.3.RELEASE.jar
  + Code:
  + Description: Spring Core
* spring-web-5.2.3.RELEASE.jar
  + Code:
  + Description: Spring Web
* tomcat-embed-core-9.0.30.jar
  + Code:
  + Description: Core Tomcat implementation
* tomcat-embed-websocket-9.0.30.jar
  + Code:
  + Description: Core Tomcat implementation

## 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 Financial’s software application.

* In regards to the vulnerability in the myDateTime Class, one simply needs to implement the methods.
* In regards to the vulnerability in the GreetingController/Greeting Class, once the value for the “name” parameter has been extracted, the data should go through a series of checks/validations to make sure that it is safe to use for the inner workings of the application
* In regards to the vulnerability in the DocData Class, the password should be stored in a password manager of some sort.
* In regards to the vulnerability in the customer class, the account number should be stored encrypted in a remote database and records should be accessed using public information such as a user’s customer ID. Only non-sensitive information should be stored in plaintext in a program.
* In regards to the vulnerability in the CRUD controller, the request parameter “business\_name” should be used or eliminated. Also, the DocData class needs a toString method that utilizes the values of the string members in the output rather than exposing memory information.