****

# 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 5](#_Toc32574612)

[3. Manual Review 6](#_Toc32574613)

[4. Static Testing 7](#_Toc32574614)

[5. Mitigation Plan 8](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **3/18/2022** | **Erin Walter** | **Added Client needs and security errors** |
| **2.0** | **3/19/2022** | **Erin Walter** | **Added Manual review, static testing, and mitigation plan** |
| **3.0** | **3/20/2022** | **Erin Walter** | **Made edits** |

## 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

Erin Walter

## 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?

Artemis financial is a financial consulting company who is looking to modernize their operations. They develop financial savings, retirement, investment, and insurance plans for their customers. When dealing with a customer’s personal information and financial information security is extremely important. Artemis financial needs to keep their customers’ information safe, to avoid any one getting any money stolen or their identity stolen, for instance. They also need to build a trusting relationship with their customers to ensure them that they are handling their personal information and financial data correctly and ethically and that their information and their financials are safe in their hands.

The company likely has a database of clients and different accounts that the clients have, and especially for things like investments and retirement accounts, Artemis is researching different stocks and mutual funds and making decisions on behalf of the client, so there is going to be internal transactions such as company communications about trades and stocks or products they are putting their clients’ money into, as well as external transactions such as investing in different stocks and mutual funds or different types of accounts like a 401k or Roth IRA account, or different types of insurance companies that they are dealing with and managing policies and accounts for their clients.

As a financial company, there are many government restrictions that they have to keep in mind. For instance, how they communicate with clients and how they handle clients’ personal and financial information (PIFI). Even though they aren’t a bank and wouldn’t necessarily fall under government regulations for banks, there are also government regulations for Financial Institutions such as many Securities and Exchange Commission (SEC) acts like the Securities Act of 1933 that states that “investors must receive financial and other significant information concerning securities being offered for public sale,” meaning that they have to give their clients financial information on the stocks that are in their portfolios. Another act is the Securities Exchange Act of 1934, which regulates conduct in the markets such as prohibiting Insider Trading. Also, at many Financial Consulting companies, there are also certifications such as “Certified Financial Planner” or “Certified Financial Advisor” certificates for their employees to get and maintain in order to do business with their clients.

External threats that the company faces would be for attacks that intend to get their clients’ personal information either to sell this information or if it’s some sort of ransomware to hold this data hostage and lock up their system. Another threat is an attacker to gain access to their clients’ log in and account information and to steal money and drain the accounts of their clients. Additionally, if an attacker is able to get access to the system and the correct privileges, the system could be at risk of this attacker to move around clients’ money and steal from their clients.

Some modernization to be considered would be to modernize security in the system. Making sure that certain secure coding patterns are followed in the code, as well as checking on the libraries and dependencies that are used in the code currently to minimize any vulnerabilities and update to the versions of these external libraries that do not have bugs or security risks associated with them. It will also be important to look into how their users access their system, how much access they have within the system, and to make sure that the way they log in and log out is secure, and that the risk of an outsider attacker getting access to an account is minimized by things like Multi-factor authentication, or locking accounts after a certain number of attempts. Another thing to look at in modernizing the system is to make sure that the code surrounding any database queries is secure, such as coding using query parameterization to protect against SQL injection attacks.

## 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.

Input Validation: Input validation is going to be very important both on the front end and back end of the system. On the front end, it’s going to be important to validate input for both clients who are accessing their accounts as well as for financial consultants or planners who are working on clients accounts. For instance, if a financial planner is looking to invest a certain percentage or dollar amount of a client’s account in a certain mutual fund, input should be validated to make sure that they aren’t accidentally mis-typing the number they wish to invest (or trying to invest more than is actually in the account). On the back end, making sure that data is sanitized and that they are using query parametrization when interacting with databases is also important to protect against accidental entries or against attacks on the company’s data.

APIs: Since the company will be connecting to multiple external sources, such as places where they are going to be investing in stocks or mutual funds, different savings accounts, different insurance types and companies, and different sorts of retirement accounts so that they can manage their money, it is going to be imperative to have secure API interactions and to securely handle client data and information throughout these API interactions.

Cryptography: Artemis Financial is going to need to encrypt client information especially since they have personal financial information about their clients as well as sensitive data such as Name, Address, Phone Number, Social Security Number, and Beneficiary information for these accounts that they manage. Since they are dealing with different external companies for their clients’ financial records, they need to encrypt this data and keep it secure while it is entering, exiting, and passing through their system.

Code Error/Quality: Since Artemis Financial’s system handles a lot of sensitive, private information for their clients, coding securely and using secure coding patterns is going to be very important to ensure security at a coding level. Also, secure error handling is going to be important in the code as well, to make sure that the program isn’t crashing or possibly exposing information upon an error that is thrown. Also, since they are dealing with many external data sources and systems, it is important to sanitize data when it enters the system, and properly handle the errors that this incoming information could cause. This will also help to with any possible attacks that may come in, so that the errors that a potential attack could cause are properly handled in the code.

Encapsulation: The code should be encapsulated into different methods and objects that each have a single purpose or that each perform actions on the same data. With different classes and methods encapsulated and coded by purpose and by data source, it is easier to secure data by controlling access to certain classes and methods within the code, and to keep sensitive data separate from other data so that there is less of a potential for attackers to get access to this sensitive data. Also by writing a central Security Manager method to validate credentials and then using this method throughout the system to validate security of the user in the system, this will cut down on duplicating code and credentials check, and will make it much easier to update the code in the future because only one method will need to be updated rather than running a search of all security checks that are scattered throughout the program.

## 3. Manual Review

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

Models: Currently there are models for customer, Greeting, and myDateTime. Currently, there is no error handling nor input validation for these methods, so if the incorrect data type comes into these methods, the program will crash and the error has no catch for the method. Additionally, there is currently no access control nor security restricting access to these methods.

Controllers: Currently there is a CRUD controller and a Greeting controller. Personally, I don’t think the CRUD controller is the best choice and this should be better distributed between different models and databases and different data. For instance, instead of having one CRUD controller that allows this functionality for all databases and models, make a CustomerController and an AccountController to separate the functionality between customer information and account information. This way these controllers can be given different privacy and access layers and access control can be better implemented at the controller level. There also is no access control to the endpoints for either the CRUD/read nor the Greeting/greeting endpoints. So I would implement access control here. Also, there is no input validation nor sanitization within these methods, which leaves the program vulnerable to attacks from the outside.

Data Access: Currently there is a CRUD class within the code that looks to be set up to handle CRUD functions to the data base. The data is not sanitized nor validated before being passed into these methods, nor is query parameterization used. While this currently does not call to a database, if the getContent() calls were set up to get data from a database, and create, update, and delete methods were added, the data would need to be validated and sanitized, as well as query parameterization used to correctly and safely put and update data in the database. Additionally, access control to these methods will be important as well. There is also a DocData method that looks to read documents from the database. A key value pair is passed into the method, however currently never used in the method. Right now it says “root” and “root” so this needs to be changed so there is validation in this method. Also, there is only one type of catch statement written for this, so more error handling needs to be implemented.

APIs: currently there are GET calls within the CRUD controller to read data, and within the Greeting controller to get greeting. There doesn’t look to be any security surrounding who has access to these endpoints or the data in these endpoints, so security and access control should be implemented. There also is no input validation nor sanitization in these methods to make sure that the data type that is being passed in matches the models that are being used. There also is no error handling in case there is an invalid data type passed in, or any sort of HTTP error is thrown.

## 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

As seen in the report summary, there were 9 vulnerable dependencies found in the dependency check.





* Bcprov-jdk15on-1.46.jar: Bouncy Castle Java library has issues with improper verification of cryptographic software. Vulnerabilities were found in versions before 1.61, so can be updated to version 1.61 or newer.
* Logback-core-1.2.3.jar: Logback versions 1.2.7 and older found to have issues with deserialization of untrusted data. There is a patch that has been published for this.
* Log4j-api-2.12.1.jar: Log4j found to have issues with improper certificate validation, deserialization of untrusted data, improper input validation in. There is a fix in versions 2.17.0, 2.12.3, and 2.3.1.
* Snakeyaml-1.25.jar: SnakeYAML bug found that causes improper restriction of recursive entity references in DTDs, there is a patch that has been published for this.
* Jackson-databind-2.10.2.jar: Bug found that caused improper restriction of XML External Entity Reference, leave vulnerable to XML external entity attacks. This can be updated to version 2.13.0, as there is a fix in version 2.13.0.
* Tomcat-embed-core-9.0.30.jar: Bugs in this version found where there is inconsistent interpretation of HTTP Requests, an infinite loop can be created, there could be a memory leak, or possible information exposure, or improper privilege management. Affects versions 10.0.0-M1 to 10.0.5, 9.0.0-M1 to 9.0.45, 8.5.0 to 8.5.65, but can be fixed by upgrading to a different version, and there are also patches that can mitigate risks from some of the bugs that were found.
* Hibernate-validator-6.0.18.Final.jar: there was a flaw found causing improper input validation, however, this can be upgraded to version 6.0.20 as the vulnerability was not found in this version.
* Spring-core-5.2.3.RELEASE.jar: Protections against RFD attacks may be bypassed depending on the browser used through use of jsessionid path parameter. A newer version of the spring framework could be used to mitigate this bug.
* Spring-jcl-5.2.3.RELEASE.jar: Protections against RFD attacks may be bypassed depending on the browser used through the use of a jsessionid path parameter, however can be mitigated by updating to a newer version of the spring framework.

## 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.

After reviewing the dependency check, first the bugs can be mitigated in the software by updating to different releases for the libraries, or by installing patches that have been released for the vulnerabilities if there hasn’t been a fix in the library.

A centralized, encapsulated SecurityManager method should be written that can verify credentials of the user before any of the other methods or endpoints are called. This method can be called for other methods before running code to determine if the user has the correct credentials to access the method.

Next, some of the coding patterns can be reviewed and made more secure. For instance, encapsulation can be implemented and the CRUDcontroller and CRUD class can be broken down into separate methods for each separate part of the system. A CustomerController, an AccountController, and other controllers broken down by specific purpose and functionality can be implemented.

Also, access control to each encapsulated area of code can be added in so that some users only have access to certain methods. For instance, a client will only have access to their own account and their own customer login page, and can get information for their accounts, but so as to not risk a client deleting or changing information in the database, not giving them access to the Put, Post, or Delete methods within the controller or data access layer.

Next, input validation and sanitization needs to be implemented throughout the current code and this needs to be maintained as new code is added so that when information comes in through client or employee input (or from outside potential attack) this input is validated and cleaned up to be in the correct format and the correct errors are thrown if needed so that the program continues to run and doesn’t break and properly handles errors (for instance in the UI giving user feedback before accepting the input and potentially corrupting a database down the line).

In the data access layer, query parameterization needs to be implemented so that this guards against SQL injection attempts.

Last, different try catch blocks can be placed in the code to handle all possible errors. For instance, different HTTP error codes should be considered and accounted for (400 errors, 500 errors), as well as different code errors (data type mismatch exception, argument out of range exception) so that the application behaves appropriately instead of crashing and showing the error in the UI to the user.