****

# CS 305 Project One

**Artemis Financial Vulnerability Assessment Report**

**Table of Contents**

[Document Revision History 3](#_gjdgxs)

[Client 3](#_30j0zll)

[Instructions 3](#_3znysh7)

[Developer 4](#_2et92p0)

[1. Interpreting Client Needs 4](#_tyjcwt)

[2. Areas of Security 4](#_3dy6vkm)

[3. Manual Review 4](#_1t3h5sf)

[4. Static Testing 4](#_4d34og8)

[5. Mitigation Plan 4](#_2s8eyo1)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **05/22/2022** | **William Harris** |  |

## Client



## Developer

William Harris

## 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:

Artemis financial requires a program able to operate as a server for financial details for its patrons. The patrons may be international so the program must adhere to the laws and regulations outlined by the United Nations Conference on Trade and Development regarding international data communications as well as privacy considerations. Vulnerabilities are always coming up, even in code already written. Continuous development should include dependency checks to keep up to date with newly discovered vulnerabilities.

## 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.

There are 5 areas of security that applies to Artemis Financial’s code:

1. Input Validation-The program will be receiving input, particularly in the Rest API classes.
2. APIs-The program uses Restful APIs.
3. Client/Server-This code represents the server portion of a client/server environment.
4. Code Error- There will be error reporting which will be written to a log in the system.
5. Code Quality-Code not written by the developer should be checked for architectural vulnerabilities concerning pre-written programs.

## 3. Manual Review

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

The structure of the code provides its own vulnerabilities as well. A manual inspection has discovered 6 areas of the program that are vulnerable to attack:

1. Input Validation
2. Dependency Check for Files
3. Code Quality
4. Code Error
5. Client/Server
6. Cryptography

## 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 the 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

Of the 20 unique dependencies used in our program, a native check revealed a total of 62 vulnerabilities present in 10 of them:

CVE-2013-1624-hide any timing data to prevent attacks based of analysis of timing during malformed CBC processing

CVE-2013-4152-Ensure proper access controls, privileges, and permissions are established and implemented

CVE-2021-42550–prevent unauthorized writing to configuration files of logback and/or application of unauthorized changes

CVE-2020-9488-ensure proper certificate validation concerning Apache Log4j SMTP appender

CVE-2021-45046 and CVE-2020-10693-validate all input and catch errors

CVE-2019-17569-validate all HTTPS requests to prevent request smuggling

CVE-2017-18640-refuse unauthorized entity expansion

## 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.

* Input Validation: To prevent possible input-based DoS attacks, all input should be validated against method requirements. Defining methods with wrappers will help mitigate any such attacks and ensure only acceptable input is passed to the method.
* Cryptography: As data will be passed from the client to a server, we must implement secure data transfer protocols. HTTPS allows the secure transfer of data over the internet using existing methods.
* Client/Server: This program is the Client side of client/server setup, thus, there will need to be protection against attacks designed to overwhelm and deny service to other users. The methods we use to check values, should be able to truncate or default inappropriate integer values that would result in an overflow.
* Code Error: Errors should be appropriately logged to identify and handle. Prevent code being run from the logs that are written to from untrusted and otherwise unauthorized sources. Error handling and logging: Ensure log entries that include un-trusted data will not execute as code in the intended log viewing interface or software.
* Code Quality: Memory Management: Double check that the buffer is as large as specified and check buffer boundaries if calling the function in a loop. Make sure there is no danger of writing past the allocated space.
* Dependency Check for Files: Using an up-to-date spring-data-rest-webmvc will ensure proper scanning and recognition of dependency vulnerabilities reported to the CVE and NVD databases.