****

# CS 305 Project One

**Artemis Financial Vulnerability Assessment Report**

**Table of Contents**

[Document Revision History 3](#_heading=h.gjdgxs)

[Client 3](#_heading=h.30j0zll)

[Instructions 3](#_heading=h.3znysh7)

[Developer 4](#_heading=h.2et92p0)

[1. Interpreting Client Needs 4](#_heading=h.tyjcwt)

[2. Areas of Security 4](#_heading=h.3dy6vkm)

[3. Manual Review 4](#_heading=h.1t3h5sf)

[4. Static Testing 4](#_heading=h.4d34og8)

[5. Mitigation Plan 4](#_heading=h.2s8eyo1)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/22/22** | **Michael Martin** |  |

## 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

Michael Martin

## 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 that develops individualized financial plans, likely for foreign and domestic clients. Using a secure communications platform would benefit the company greatly, putting a high value on it. At the moment, the restrictions for secure communication are based on the certifications required by the company to do business with specific entities, such as government or healthcare clients. The restrictions would indicate a specific minimum required encryption algorithm, which is typically AES with the bit size at 128 or 256. The current threats Artemis Financial has to work to protect themselves against are quite broad. When a company has an active online presence, they must work to protect against Phishing, DDOS attacks, malware, and bruteforce attempts. With regards to modernization of their software, some important considerations are to keep dependencies up to date, and ensure chosen libraries have active support. Keeping those in mind will help ensure that vulnerabilities are patched and the software is secure.

## 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 - Ensuring that only allowed strings, int, etc are accepted by the program will help prevent injections and program crashes, which subsequently would allow attackers access to the program

APIs - Ensuring the APIs are secure is important, as APIs allow programs to interact with one another. If an API is not secure, an attacker could intercept the traffic or spoof traffic, therefore potentially gaining access to sensitive information.

Cryptography - This is applicable as a direct reflection of using secure communications, as well as encrypted web traffic to transfer sensitive information to and from the client and employee.

Code Error - Code Error is important as it’ll help ensure that attackers can not gain any useful information or access via error codes or invalid inputs.

Code Quality - Having good quality code will prevent any known vulnerabilities from being within the program as well as ensure that error codes and invalid inputs are handled correctly.

## 3. Manual Review

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

I start the manual review by getting familiar with the code and the pom.xml. Next, I start to review the areas of security I listed above. I started with Input Validation, and with pom.xml and was unable to find a reference to the Apache Commons Validator. I then took a look at the included Java files and I did not see any Java programming referring to the input validation (Secure Coding Guidelines nor Scanner).

When checking for potential API vulnerabilities, I noticed the myql test included the database name, and root user password in plain text within DocData.java. Other than this, I did not notice any more API vulnerabilities (RESTful, Springframework, etc).

Next in the manual review is to look for Cryptography vulnerabilities, upon my inspection of the program, I did not see any reference to encryption algorithms or secure communication.

Next is to check for Code Error or error handling, while looking through the code, I noticed that a try catch statement was implemented within DocData.java. This try catch was designed to output any errors when attempting to contact the database. Otherwise, Code Error is absent from all other files, some files such as customer.java could use a way to deal with incorrect account balances and numbers.

The final check is Code Quality, this entails checking the code for known vulnerabilities or bad code. I did not notice anything upon my read through.

## 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

bcprov-jdk15on-1.46.jar

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.

CVE-2013-1624

CVE-2015-6644

CVE-2015-7940

CVE-2016-1000338

CVE-2016-1000339

CVE-2016-1000341

CVE-2016-1000342

CVE-2016-1000343

CVE-2016-1000344

CVE-2016-1000345

CVE-2016-1000346

CVE-2016-1000352

CVE-2017-13098

CVE-2018-1000613

CVE-2018-5382

CVE-2020-26939

spring-boot-2.2.4.RELEASE.jar

Spring Boot

CVE-2013-4152

CVE-2013-7315

CVE-2014-0054

CVE-2016-1000027

CVE-2022-22965

CVE-2022-22968

CVE-2022-27772

logback-core-1.2.3.jar

logback-core module

CVE-2021-42550

log4j-api-2.12.1.jar

The Apache Log4j API

CVE-2020-9488

CVE-2021-44228

CVE-2021-44832

CVE-2021-45046

CVE-2021-45105

snakeyaml-1.25.jar

YAML 1.1 parser and emitter for Java

CVE-2017-18640

jackson-databind-2.10.2.jar

General data-binding functionality for Jackson: works on core streaming API

CVE-2020-25649

CVE-2020-36518

tomcat-embed-core-9.0.30.jar

Core Tomcat implementation

CVE-2019-17569  
 CVE-2020-11996

CVE-2020-13934

CVE-2020-13935

CVE-2020-13943

CVE-2020-17527

CVE-2020-1935

CVE-2020-1938

CVE-2020-8022

CVE-2020-9484

CVE-2021-24122

CVE-2021-25122

CVE-2021-25329

CVE-2021-30640

CVE-2021-33037

CVE-2021-41079

CVE-2021-42340

CVE-2022-29885

tomcat-embed-websocket-9.0.30.jar

Core Tomcat implementation

CVE-2019-17569

CVE-2020-11996

CVE-2020-13934

CVE-2020-13935

CVE-2020-13943

CVE-2020-17527

CVE-2020-1935

CVE-2020-1938

CVE-2020-8022

CVE-2020-9484

CVE-2021-24122

CVE-2021-25122

CVE-2021-25329

CVE-2021-30640

CVE-2021-33037

CVE-2021-41079

CVE-2022-29885

hibernate-validator-6.0.18.Final.jar  
 Hibernate's Bean Validation (JSR-380) reference implementation.

CVE-2020-10693

spring-core-5.2.3.RELEASE.jar

Spring Core

CVE-2016-1000027

CVE-2020-5421

CVE-2021-22060

CVE-2021-22096

CVE-2021-22118

CVE-2022-22950

CVE-2022-22965

CVE-2022-22968

spring-jcl-5.2.3.RELEASE.jar

Spring Commons Logging Bridge

CVE-2022-22950

h2-1.4.200.jar

H2 Database Engine

CVE-2021-23463

CVE-2021-42392

CVE-2022-23221

CWE-94: Improper Control of Generation of Code ('Code Injection')

gson-2.8.6.jar

Gson JSON library

CVE-2022-25647

failureaccess-1.0.1.jar

Contains

com.google.common.util.concurrent.internal.InternalFutureFailureAccess and

InternalFutures. Most users will never need to use this artifact. Its

classes is conceptually a part of Guava, but they're in this separate

artifact so that Android libraries can use them without pulling in all of

Guava (just as they can use ListenableFuture by depending on the

listenablefuture artifact).

CVE-2020-8908

maven-model-3.0.jar

Maven Model

CVE-2020-36460

maven-core-3.0.5.jar

Maven Core classes.

CVE-2021-26291

maven-model-builder-3.0.5.jar

The effective model builder, with inheritance, profile activation, interpolation, ...

CVE-2020-36460

## 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.

The majority of the vulnerabilities can be resolved by updating the dependencies and keeping them updated to the most recent stable releases. Looking at the code, implementing more error handling and adding input validation should help secure the code.