`

****

# 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** |
| --- | --- | --- | --- |
| **1.3** | **12/1/2022** | **Danielle Monroe** |  |

## 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

Danielle Monroe

## 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?

As Artemis Financial has clients internationally, the policy of least privileges should be heavily enforced throughout, both, their software security requirements and application. The company will need to store their customers’ credit card numbers, personal identifiable information, or other assets, and therefore the protection of their customers’ privacy must be a top priority. Since their clients’ reside in an array of countries, the Graham-Leach-Bailey Act must be implemented. This means that, if a data leak did occur, Artemis Financial would be obligated to publicly announce it.

To deliver secure, software security requirements and application, several requirements will be implemented. First, we will wrap all third-party libraries in secure APIs. Next, we will also use the OWASP to check for vulnerabilities, allowing for clear requirements, secure coding standards, and verification of our security controls. We will, also, use both static and dynamic testing to make sure the code has no bugs and is functioning as is expected. Lastly, we will implement a web application firewall to help block hackers.

## 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.

Several areas of vulnerability will be assessed in order to make a more secure application for Artemis Financial. The first line of defense will be the principle of least privilege so that users can only access data and areas of the application that they are permitted to. Next, will be user input validation. This process entails a narrow list of acceptable characters and patterns that, if invalidated, will be rejected, and logged for analysis. All data, whether incoming or outgoing, must be encrypted as well.

Next, we will use cryptograph to make our code more secure. Aspects, such as passwords, should only be sent through an encrypted connection in order to lower our vulnerabilities. We will use static testing to make sure there is no errors in the code; errors can lead to vulnerabilities.

We’ll then wrap any third-party libraries used within the application to secure the API. This will limit the API’s functionality to only be able to conduct particular, non-operating system specific, tasks. By having commands that are not directly issued by the operating system is an attempt to block hackers from using any third-party to gain access into the application. Lastly, we will use dynamic analysis tools to test for and SQL injections and ensure code quality. We will use parameterized queries in order to keep data separate and secure.

## 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 MyDateTime.java file should have its attributes as private.
* Encapsulation is important for security and should be applied where applicable.
* The request parameters in GreetingController.java should not be hard coded.
* Customer.java should have the variable account\_balance as private.
* In the DocData.java file, the connection information should be in a variable and not hard coded into the getConnection function.
* Many of the files have no comments on them as well. Commenting should be done so that reviewers know why the code exists, how to use it, or any other pertinent information.

## 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

a. Name: Bcprov-jdk10on-1.46.jar

b. 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.

c. Vulnerabilities: This file has several vulnerabilities. One that does not properly block attacks allowing for remote attackers to make distinguishing attacks and plaintext-recovery attacks. Another that doe not validate that “a point is withing the elliptic curve” and allows for invalid curve attacks. Another that allows access to use’s private information. It also has one that doesn’t properly validate ASN.1 encoding and allowing for injection of elements that will allow ‘invisible’ data into a signed structure. There is another that allows for data to be accessed by monitoring the CPU. Another that allows for the generation of signatures that allows for information of a signature and private values to be seen by an attacker. Another that does not validate encoding properly and allows for injection into the sequence. It also has one that “generates a weak private key”. Another that allows the use of ECB mode. Another that is vulnerable to “padding oracle attack”. Another where validation is not done properly and can cause details to be shown about private keys. Another that allows an attacker to access private key from the application.

a. Name: Hibernate-validator-6.0.18.Final.jar

b. Description: Hibernate’s Bean Validation reference implementation

c. Vulnerabilities: There is one previously published vulnerability. It allows for invalid EL expressions to be seen as valid. This can allow attackers to bypass input escaping controls.

a. Name: Jackson-databind-2.10.2.jar

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

c. Vulnerabilities: This file has a vulnerability where the entity expansion is not properly secured.

a. Name: Log4j-api.2.12.1.jar

b. Description: The Apache Log4j API

c. Vulnerabilities: This file has a vulnerability that does not validate a certificate properly.

a. Name: Snakeyaml-1.25.jar

b. Description: YAML 1.1 parser and emitter for Java

c. Vulnerabilities: This file allows for the expansion of an entity, during a load operation.

a. Name: Spring-aop-5.2.3.RELEASE.jar

b. Description: Spring AOP

c. Vulnerabilities: This file allows for injection attacks to be bypassed depending on the browser. It also allows for malicious input to be put, causing additional log entries. Another vulnerability is that it allows for malicious users to read and modify files by privilege escalation.

a. Name: Spring-core-5.2.3.RELEASE.jar

b. Description: Spring Core

c. Vulnerabilities: Allows for the protections of RFD attacks to be bypassed. It also allows additional log entries to be inserted by users using malicious inputs. It is also vulnerable to privilege escalating.

a. Name: tomcat-embed-core-9.0.30.jar

b. Description: Core Tomcat Implementation

c. Vulnerabilities: This file a regression causing request smuggling by incorrectly processing invalid Transfer-Encoding headers. It also can cause high CPU usage. It also can lead to a denial of service through an OutOfMemoryException and through the payload length not being properly validated. It also allows for incorrect responses to be seen by users. The file could also reuse HTTP requests causing a possible leak of data. It also allows invalid HTTP headers to be read as valid. It may allow attackers to exploit the AJP connections to gain access. An attacker can also control contents and names of files on a server. Another vulnerability is that the file may disclose JSP source code. It also gives duplicate request headers causing two users to receive the same results. It also allows the bypassing of some protections by using valid username by attackers. It also can allow request smuggling when used with a reverse proxy. The file causes denial of service because incoming TLS packets weren’t properly validated. Lastly, the file has a memory leak.

a. Name: tomcat-embed-websocket-9.0.30.jar

b. Description: Core Tomcat Implementation

c. Vulnerabilities: HTTP request smuggling is possible through invalid Transfer-Encoding headers being incorrectly processed. It can trigger CPU usage. It can cause an OutOfMemoryException, leading to a denial of service, if a sufficient number of requests were made. A denial of service can also be caused by the incorrect validation of a payload. It also can use headers from a previous request, resulting in a user receiving the incorrect response. Information could possibly be leaked by reusing the HTTP request header value. HTTP request smuggling can also be done by invalid headers being parsed as valid. AJP connections can be exploited because the connections have higher trust than they should. The file also has incorrect default permissions. The name and contents of a file can be controlled by an attacker through this file. JSP source code was disclosed when serving resources from a network location. A duplicate request could be copied from one request to another causing two users to receive the same information. Protection can be bypassed by an attacker by using variations of a username. When a reverse proxy is used, request smuggling is possible because the HTTP transfer-encoding request header is not properly parsed. The file does not properly validate incoming TLS packets. The HHTP upgrade connections were not released when an object was introduced.

## 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.

With the aim of mitigating these security vulnerabilities, I believe, first, the policy of least privilege should be used throughout the development of Artemis Financial’s application. This will safeguard the application in even the basic of levels, as well as control access to the vital and sensitive data stored within the database. With the intention of an increased defense over the database, the use of query parameterization, variable binding, and escaping untrusted data techniques should be used to prevent query injections. Should a query injection occur, in order to reduce its impact, we need to verify the number of results being given by a request and use the principle of least authority. Along with this, the use of contextual access control should be put into effect to make sure that the user is authorized to do what is being requested; blocking their access if not authorized.

Furthermore, I suggest we reject any forbidden input with input validation. Also, the use of dynamic analysis tools, such as black list validation and output encoding to control access should be implemented where applicable. Lastly, I suggest that we use the HTTP POST request and strict transport security HTTPS response headers. By doing so, we resolve any potential leaks coming from the browser.