## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/15/2024** | **Logan Riedell** |  |

## Client



## Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In this report, identify your security vulnerability findings and recommend the next steps to remedy the issues you have found.

* Respond to the five steps outlined below and include your findings.
* Respond using your own words. You may also include images or supporting materials. If you include them, make certain to insert them in the relevant locations in the document.
* Refer to the Project One Guidelines and Rubric for more detailed instructions about each section of the template.

Developer

Logan Riedell

**1. Interpreting Client Needs**

Determine your client’s needs and potential threats and attacks associated with the company’s application and software security requirements. Consider the following questions 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 on secure communications to consider?
* What external threats might be present now and in the immediate future?
* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

Secure communications are very important for Artemis Financial because they deal with sensitive client financial information. Ensuring integrity and confidentiality with this information is essential for maintaining clients' trust and meeting regulatory requirements.

It does not state explicitly whether the company handles international transactions, but I assume it would take clients from all over the world and make transfers to other banks outside of its own country. This means it must adhere to international security and privacy laws, which may be stricter.

There are governmental restrictions that the client must consider. These include regulation on data protection, data transfer over borders, and encryption standards.

Threats include phishing attacks, ransomware, and other cyberattacks that target vulnerabilities in the technology and human side of the organization. The financial sector is a high-value target due to the potential financial gain in successful attacks.

Artemis Financial should carefully select and manage libraries to mitigate vulnerabilities that attackers can exploit. Regular vulnerability scanning and updates are important. As web technologies evolve, new security challenges arise. Artemis should continuously update and test its web applications to address these challenges and secure the new functionalities or frameworks it wants to implement.

**2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

Input Validation is essential for financial services where user will input their sensitive data, financial details, personal information, and transaction data. Input validation prevents common vulnerabilities like injection flaws that could compromise the confidentiality of the data.

API

Artemis will likely use API’s to connect with other banks, financial institutions, and other third parties for different functionalities such as credit checks. Securing the API interactions ensures the data transmitted is secure and protected against leaks.

Cryptography

Strong encryption is necessary to comply with regulatory requirements and privacy laws. It prevents unauthorized access to sensitive information.

Client/Server

Financial transactions often happen over the internet, and securing both the client and server’s end is important to prevent attacks that would hijack sessions. Securing communication and utilizing authentication will provide protection to Artemis’s integrity and confidentiality

**3. Manual Review**

Continue working through the vulnerability assessment process flow diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

CRUDController.java  
The name parameter is used without a validation which may lead to injection.

customer.java

The account\_balance variable is not protected and can be changed by parts of the program that shouldn’t necessarily have access to it.

DocData.java

key and value are not cleaned or sanitized, meaning someone could sneak in commands that could steal data.

In the read\_document method, the database username and password are written right in the code easy for anyone to see.

There are also no handling SQL exceptions exposing error messages to bad parties.

myDateTime.java

The variables mySecond, myMinute, and myHour are not well protected and are open for other parts of the program to change when they shouldn’t be able to.

**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 items:

* The names or vulnerability codes of the known vulnerabilities
* A brief description and recommended solutions provided by the dependency-check report
* Any attribution that documents how this vulnerability has been identified or documented previously

**1. CVE-2024-34447**

* **Description**: Bouncycastle - Improper Validation of Certificate with Host Mismatch. The software communicates with a host that provides a certificate, but the software does not properly ensure that the certificate is actually associated with that host.
* **Recommended Solution**: Ensure proper validation of the certificate with the host.
* **Attribution**: Documented by OSSINDEX.

**2. CVE-2022-22965**

* **Description**: Spring MVC or Spring WebFlux applications running on JDK 9+ may be vulnerable to remote code execution (RCE) via data binding. The specific exploit requires the application to run on Tomcat as a WAR deployment.
* **Recommended Solution**: Apply updates per vendor instructions.
* **Attribution**: Documented by VMware and various advisories.

**3. CVE-2023-44487**

* **Description**: HTTP/2 contains a rapid reset vulnerability that allows for a distributed denial-of-service attack (DDoS).
* **Recommended Solution**: Apply mitigations per vendor instructions or discontinue use of the product if mitigations are unavailable.
* **Attribution**: Documented by CISA and multiple advisories.

**4. CVE-2020-25649**

* **Description**: A flaw in FasterXML Jackson Databind where it did not have entity expansion secured properly, allowing XML external entity (XXE) attacks.
* **Recommended Solution**: Apply updates per vendor instructions.
* **Attribution**: Documented by FasterXML and OSSINDEX.

**5. CVE-2016-1000342**

* **Description**: In the Bouncy Castle JCE Provider version 1.55 and earlier, ECDSA does not fully validate ASN.1 encoding of signatures on verification. It is possible to inject extra elements in the sequence making up the signature and still have it validate.
* **Recommended Solution**: Apply the latest updates from Bouncy Castle.
* **Attribution**: Documented by OSSINDEX and other advisories.

**6. CVE-2022-27772**

* **Description**: Spring Boot versions prior to v2.2.11.RELEASE were vulnerable to temporary directory hijacking impacting the method **org.springframework.boot.web.server.AbstractConfigurableWebServerFactory.createTempDir**.
* **Recommended Solution**: Upgrade to Spring Boot v2.2.11.RELEASE or later.
* **Attribution**: Documented by OSSINDEX and GitHub Security Advisories.

**7. CVE-2023-33201**

* **Description**: LDAP injection vulnerability in Bouncy Castle. The certificate's Subject Name is inserted into an LDAP search filter without any escaping during the certificate validation process.
* **Recommended Solution**: Ensure proper escaping of LDAP search filters.
* **Attribution**: Documented by OSSINDEX and other advisories.

**8. CVE-2016-1000339**

* **Description**: In Bouncy Castle JCE Provider versions up to 1.55, the AES encryption implementation was vulnerable to side-channel attacks, potentially leaking information about the AES key being used.
* **Recommended Solution**: Upgrade to version 1.56 or later where the leak is fixed.
* **Attribution**: Documented by OSSINDEX and other advisories.

**9. CVE-2023-6378**

* **Description**: A serialization vulnerability in logback receiver component allows a Denial-Of-Service attack by sending poisoned data.
* **Recommended Solution**: Upgrade to a version that addresses this issue.
* **Attribution**: Documented by OSSINDEX and other advisories.

**10. CVE-2020-1938**

* **Description**: Apache Tomcat treats AJP connections as having higher trust. If such connections are available to an attacker, they can be exploited.
* **Recommended Solution**: Apply updates per vendor instructions.
* **Attribution**: Documented by Apache and CISA Known Exploited Vulnerabilities.

**11. CVE-2022-1471**

* **Description**: SnakeYaml's **Constructor()** class does not restrict types during deserialization, potentially leading to remote code execution.
* **Recommended Solution**: Use SnakeYaml's **SafeConstructor** and upgrade to version 2.0 or later.
* **Attribution**: Documented by Google Security and OSSINDEX.

**12. CVE-2023-46589**

* **Description**: Vulnerability in Apache Tomcat that could allow for remote code execution.
* **Recommended Solution**: Apply the latest updates from Apache Tomcat.
* **Attribution**: Documented by Apache Security Team.

**13. CVE-2023-20861**

* **Description**: Spring Framework vulnerability could lead to data exposure and other security issues.
* **Recommended Solution**: Upgrade to the latest version of Spring Framework.
* **Attribution**: Documented by VMware and various advisories.

**5. Mitigation Plan**

Interpret the results from the manual review and static testing report. Then identify the steps to mitigate the identified security vulnerabilities for Artemis Financial’s software application.

To keep their application safe, we need to use parameterized queries and clean up user inputs to prevent SQL injection attacks. This could mean double-checking any data that users input before using it in the database. It is also a good idea to keep sensitive information like passwords out of the code and store them somewhere else encrypted.  
  
Setting up error handling will prevent malicious parties from discovering too much about the application.

Regularly updating the application and the libraries it uses to their latest versions.

Using tools like the dependency check regularly can save a lot of time.

Regular training on secure coding practices for our developers.