# CS 305 Project One

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/22/2024** | **Alexander Wagner** |  |

## Client



## Developer

Alexander Wagner

**Interpreting Client Needs**

Artemis Financial’s web-based software application needs robust security measures to protect sensitive financial data and ensure secure communications. Secure communications are critical for Artemis Financial to protect sensitive customer information, financial transactions, and maintain customer trust. Secure channels prevent data breaches and unauthorized access.

If Artemis Financial conducts international transactions, secure communications become even more important to comply with global regulations and protect against international cyber threats. Compliance with regulations like GDPR, HIPAA, and others may dictate specific security requirements for handling and transmitting data. Understanding these restrictions ensures compliance and avoids legal issues.

Potential threats include phishing attacks, SQL injection, cross-site scripting (XSS), and man in the middle attacks. Future threats may involve advanced persistent threats (APTs) and evolving malware. Modernizing involves integrating open-source libraries and evolving web technologies. It is essential to keep these libraries up to date to mitigate vulnerabilities and use the newest security features.

**Areas of Security**

Based on the vulnerability assessment process flow diagram, the following areas of security apply to Artemis Financial’s software application:

**Architecture Review:** Analyzing the overall application architecture to ensure it follows security best practices is important for identifying potential structural vulnerabilities that could be exploited by attackers.

**Input Validation:** Ensuring all user inputs are validated and sanitized is essential to prevent injection attacks, such as SQL injection and cross-site scripting (XSS), which could compromise the application's data integrity and security.

**APIs:** Securing API interactions is important to prevent unauthorized access and data breaches, as APIs are often targeted by attackers to exploit vulnerabilities and gain access to sensitive data.

**Cryptography:** Implementing strong encryption for data in transit and at rest ensures that sensitive information remains confidential and protected from unauthorized access or tampering.

**Client/Server:** Securing communications between client and server helps protect data integrity and confidentiality, ensuring that data exchanged over the network is not intercepted or altered by attackers.

**Code Error Handling:** Properly handling errors to avoid information leakage and potential exploits is important for maintaining the security and stability of the application, as detailed error messages can provide attackers with valuable information.

**Code Quality:** Following secure coding practices and patterns minimizes vulnerabilities by ensuring that the code is resistant to common security issues and adheres to industry best practices.

**Encapsulation:** Securing data structures through encapsulation prevents unauthorized data access and manipulation, ensuring that sensitive information is only accessible and modifiable by authorized components within the application.

**Manual Review**

During the manual code inspection, the following vulnerabilities were identified:

**CRUDController.java**

* Lack of input validation for the `business\_name` parameter.
* No authorization checks for accessing the `/read` endpoint.
* Potential SQL injection due to direct concatenation of user input into SQL queries.

**customer.java**

* Lack of authorization checks for `showInfo` and deposit methods.
* No input validation in the `deposit` method, allowing potential malicious input.
* The `account\_balance` variable is public, which could lead to unintended access and modification from other parts of the application. It should be private with proper getter and setter methods to control access.

**DocData.java**

* Potential SQL injection in `read\_document` method due to direct use of user input in SQL queries.
* The `read\_document` method does not handle potential SQL exceptions properly, which could lead to application crashes or leakage of database error information.

**GreetingController.java**

* The `greeting` method does not validate the `name` parameter, which could lead to XSS if the `name` is rendered in the response without proper escaping.

**RestServiceApplication.java**

* The application does not implement any security measures to protect against CSRF (Cross-Site Request Forgery) attacks. This is a typical requirement for web applications.

**General Security Configuration**

* The application does not appear to use HTTPS, which is essential for secure communication over the network.

**Static Testing**

The dependency-check report identified the following vulnerabilities:

**bcprov-jdk15on-1.46.jar**

Vulnerability IDs: CVE-2024-34447, CVE-2016-1000338, CVE-2016-1000342, CVE-2016-1000343, CVE-2024-29857, CVE-2016-1000344, CVE-2016-1000352, CVE-2024-30171, CVE-2016-1000341, CVE-2016-1000345, CVE-2017-13098, CVE-2020-15522, CVE-2023-33202, CVE-2020-26939, CVE-2023-33201, CVE-2016-1000339, CVE-2015-7940, CVE-2018-5382

Highest Severity: HIGH

Description: Multiple vulnerabilities in the Bouncy Castle cryptography package.

Recommended Solution: Update to a newer version.

**hibernate-validator-6.0.18.Final.jar**

Vulnerability IDs: cpe:2.3:a:redhat:hibernate\_validator:6.0.18

Highest Severity: MEDIUM

Description: Vulnerability in Hibernate Validator.

Recommended Solution: Update to a newer version.

**jackson-databind-2.10.2.jar**

Vulnerability IDs: cpe:2.3:a:fasterxml:jackson-databind:2.10.2, cpe:2.3:a:fasterxml:jackson-modules-java8:2.10.2

Highest Severity: HIGH

Description: Multiple vulnerabilities in Jackson Databind.

Recommended Solution: Update to a newer version.

**log4j-api-2.12.1.jar**

Vulnerability IDs: cpe:2.3:a:apache:log4j:2.12.1

Highest Severity: LOW

Description: Vulnerability in Apache Log4j.

Recommended Solution: Update to a newer version.

**logback-core-1.2.3.jar**

Vulnerability IDs: cpe:2.3:a:qos:logback:1.2.3

Highest Severity: HIGH

Description: Multiple vulnerabilities in Logback Core.

Recommended Solution: Update to a newer version.

**snakeyaml-1.25.jar**

Vulnerability IDs: cpe:2.3:a:snakeyaml\_project:snakeyaml:1.25

Highest Severity: CRITICAL

Description: Multiple vulnerabilities in SnakeYAML.

Recommended Solution: Update to a newer version.

**spring-boot-2.2.4.RELEASE.jar**

Vulnerability IDs: cpe:2.3:a:vmware:spring\_boot:2.2.4

Highest Severity: CRITICAL

Description: Multiple vulnerabilities in Spring Boot.

Recommended Solution: Update to a newer version.

**spring-boot-starter-web-2.2.4.RELEASE.jar**

Vulnerability IDs: cpe:2.3:a:vmware:spring\_boot:2.2.4, cpe:2.3:a:web\_project:web:2.2.4

Highest Severity: CRITICAL

Description: Multiple vulnerabilities in Spring Boot Starter Web.

Recommended Solution: Update to a newer version.

**spring-core-5.2.3.RELEASE.jar**

Vulnerability IDs: cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3, cpe:2.3:a:springsource:spring\_framework:5.2.3, cpe:2.3:a:vmware:spring\_framework:5.2.3

Highest Severity: CRITICAL

Description: Multiple vulnerabilities in Spring Core.

Recommended Solution: Update to a newer version.

**spring-web-5.2.3.RELEASE.jar**

Vulnerability IDs: cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3, cpe:2.3:a:springsource:spring\_framework:5.2.3, cpe:2.3:a:vmware:spring\_framework:5.2.3, cpe:2.3:a:web\_project:web:5.2.3

Highest Severity: CRITICAL

Description: Multiple vulnerabilities in Spring Web.

Recommended Solution: Update to a newer version.

**spring-webmvc-5.2.3.RELEASE.jar**

Vulnerability IDs: cpe:2.3:a:pivotal\_software:spring\_framework:5.2.3, cpe:2.3:a:springsource:spring\_framework:5.2.3, cpe:2.3:a:vmware:spring\_framework:5.2.3, cpe:2.3:a:web\_project:web:5.2.3

Highest Severity: CRITICAL

Description: Multiple vulnerabilities in Spring WebMVC.

Recommended Solution: Update to a newer version.

**tomcat-embed-core-9.0.30.jar**

Vulnerability IDs: cpe:2.3:a:apache:tomcat:9.0.30, cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30

Highest Severity: CRITICAL

Description: Multiple vulnerabilities in Apache Tomcat Embed Core.

Recommended Solution: Update to a newer version.

**tomcat-embed-websocket-9.0.30.jar**

Vulnerability IDs: cpe:2.3:a:apache:tomcat:9.0.30, cpe:2.3:a:apache\_tomcat:apache\_tomcat:9.0.30

Highest Severity: CRITICAL

Description: Multiple vulnerabilities in Apache Tomcat Embed WebSocket.

Recommended Solution: Update to a newer version.

**Mitigation Plan**

To address the identified security vulnerabilities, the following steps should be taken:

**Regularly Update Dependencies:** Update project dependencies like Bouncy Castle, Hibernate Validator, Jackson Databind, Log4j, Logback, SnakeYAML, Spring Boot, Spring Core, Spring Web, Spring WebMVC, and Apache Tomcat to the latest versions to mitigate identified vulnerabilities. This ensures the code uses up to date libraries with known vulnerabilities patched.

**Implement Input Validation and Sanitization:** Prevent SQL injection and XSS attacks by ensuring all user inputs are validated and sanitized before processing. This applies to vulnerabilities identified in `CRUDController.java`, `customer.java`, `DocData.java`, and `GreetingController.java`.

**Authorization Checks:** Add authorization checks for sensitive operations to ensure that only authorized users can access or modify sensitive data. This is particularly important for the `showInfo` and `deposit` methods in `customer.java` and the `/read` endpoint in `CRUDController.java`.

**Use TLS for Secure Communication:** Encrypt data in transit using TLS to ensure secure communication channels between clients and servers. This will address the lack of secure communication identified in the general security configuration.

**Encrypt Sensitive Data at Rest:** Encrypt sensitive data at rest to prevent unauthorized access in case of data breaches. Use strong encryption algorithms and key management practices to enhance data security.

**Logging and Monitoring:** Implement logging and monitoring to detect and respond to security incidents promptly. Set up alerts for suspicious activities and regularly review logs for anomalies. This will help in early detection and mitigation of potential security breaches.

**Proper Error Handling:** Ensure proper error handling to prevent information leakage. Avoid displaying detailed error messages to users and securely log errors for internal review. This addresses issues identified in `DocData.java` regarding SQL exception handling.