# CS 305 Project One

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **January 22 , 2025** | **Makayla Meeks** | **1st Revision** |

## 

## Client



## Developer

Makayla Meeks

**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?

Artemis Financial, a consulting company specializing in financial planning for clients, requires secure communications to protect sensitive customer data, such as savings, retirement, investments, and insurance details. Ensuring secure communication channels is essential for maintaining client trust and safeguarding the company's reputation. While Artemis Financial currently does not engage in international transactions, it must prepare for potential future expansion by considering international standards for secure communications. Additionally, the company must comply with governmental regulations such as GDPR, CCPA, and PCI DSS, which mandate encryption and data protection measures to avoid legal and financial repercussions.

The company faces several external threats, including injection attacks due to improper input validation, man-in-the-middle (MITM) attacks from insufficient encryption, and denial-of-service (DoS) attacks caused by the absence of rate-limiting mechanisms. Furthermore, outdated dependencies introduce vulnerabilities, and the lack of authentication mechanisms for REST endpoints increases the risk of unauthorized access. To modernize its software application, Artemis Financial must address these vulnerabilities while embracing current technologies. This includes maintaining secure and updated open-source libraries, implementing secure API development practices, and considering scalability and performance enhancements. By proactively addressing these security needs, Artemis Financial can strengthen its software application, mitigate risks, and align with modern industry standards.

**2. Areas of Security**

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

1. **Input Validation**:

Input validation is essential for ensuring that user inputs, such as the business name parameter in the application, are sanitized to prevent injection attacks. This step is critical as improper input handling could expose the system to SQL injection or cross-site scripting (XSS) vulnerabilities.

1. **APIs**:

Secure API interactions are necessary to protect communication between the client and server. In Artemis Financials’ application, APIs handles sensitive customer data, making it crucial to implement proper authentication, authorization, and encryption mechanisms to secure these interactions.

1. **Cryptography**:

The application must use robust encryption algorithms to protect sensitive financial information, such as customer details and transaction data, both at rest and in transit. This ensures compliance with industry regulations and guards against data breaches.

1. **Client/Server Security**:

Ensuring secure communication between the client and server is essential for distributed computing. This involves using protocols such as HTTPS and implementing secure session management to prevent attacks like session hijacking or man-in-the-middle (MITM) attacks.

1. **Code Error Handling**:

Proper error handling is vital to prevent the exposure of stack traces or system information that attackers could exploit. In Artemis Financials’ application, unhandled exceptions or improper error messages could leak sensitive system details.

1. **Code Quality**:

Following secure coding practices, such as adhering to OWASP standards, helps prevent common vulnerabilities. Artemis Financial must enforce these practices to build a resilient application and avoid issues like hardcoded credentials or insecure dependencies.

1. **Encapsulation**:

Secure data structures and encapsulation protect sensitive information by limiting access and maintaining integrity. For Artemis Financial, this involves ensuring that sensitive data, such as customer profiles, is properly encapsulated within the application.

**3. Manual Review**

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

**Input Validation and Sanitization:**

* The CRUDController.java file lacks proper input validation and sanitization for the business\_name parameter in the /read endpoint, making it vulnerable to injection attacks.

**Sensitive Data Exposure:**

* The DocData object in CRUDController.java is returned as a string without safeguards, potentially exposing sensitive information.

**Missing Authentication:**

* REST endpoints in files like GreetingController.java lack authentication mechanisms, allowing unauthorized access.

**Improper Exception Handling:**

* The CRUD.java file does not properly handle exceptions, potentially leading to application crashes or stack trace exposure.

**Unencrypted Sensitive Data:**

* Sensitive data in the customer.java file is handled without encryption, risking confidentiality during storage and transmission.

**Hardcoded Values:**

* Files such as RestServiceApplication.java contain hardcoded values, which may expose sensitive information if the source code is leaked.

**Insecure Logging:**

* Logging practices across the application do not ensure secure handling of sensitive information, potentially exposing sensitive data in logs.

**Lack of Rate Limiting:**

* REST endpoints lack rate limiting or throttling mechanisms, leaving the application vulnerable to brute force or denial-of-service (DoS) attacks.

**Default Configurations:**

* The application uses default configurations, particularly in pom.xml, which might expose it to well-known vulnerabilities in default setups.

**Unvalidated Redirects:**

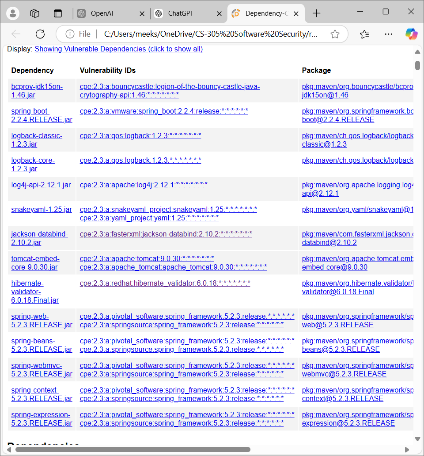
* Potential unvalidated redirects in routing logic might allow attackers to redirect users to malicious websites.

**4. Static Testing**

Run a dependency check on Artemis Financials’ 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

**A screenshot of a computer

Description automatically generated**

**Known Vulnerabilities and Details:**

**1. bcprov-jdk15on-1.46.jar**

* **Vulnerability Code:** CPE 2.3: a:bouncycastle:legion-of-the-bouncy-castle-java-cryptography-api
* **Description:** This library has known cryptographic weaknesses that can compromise data integrity and confidentiality.
* **Recommended Solution:** Upgrade to the latest version of the library that addresses these issues.
* **Attribution:** Identified via the National Vulnerability Database (NVD) and referenced in the OWASP Dependency-Check tool.

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

* **Vulnerability Code**: CPE 2.3: a:vmware:spring\_boot
* **Description:** The version of Spring Boot used is outdated and vulnerable to multiple issues, including remote code execution.
* **Recommended Solution:** Upgrade to Spring Boot 2.5.0 or later to address known vulnerabilities.
* **Attribution:** Documented in the NVD and highlighted in the dependency-check report.

**3. logback-classic-1.2.3.jar**

* **Vulnerability Code:** CPE 2.3: a:qos.logback:logback-classic
* **Description:** Vulnerabilities in this version can lead to improper logging practices that may expose sensitive data.
* **Recommended Solution:** Upgrade to the latest stable version of Logback Classic to mitigate risks.
* **Attribution:** Tracked in the NVD and flagged during the dependency analysis.

**4. log4j-api-2.12.1.jar**

* **Vulnerability Code:** CPE 2.3: a:apache:log4j
* **Description:** The Log4j version used has potential vulnerabilities for remote code execution attacks.
* **Recommended Solution:** Upgrade to Log4j 2.17.0 or higher to mitigate known risks.
* **Attribution**: Well-documented vulnerability in the NVD and widely discussed in security bulletins.

**5. jackson-databind-2.10.2.jar**

* **Vulnerability Code:** CPE 2.3: a:fasterxml:jackson-databind
* **Description:** Vulnerable to data binding exploits that can lead to deserialization attacks.
* **Recommended Solution:** Upgrade to a more recent version of Jackson Databind (e.g., 2.12.0 or later).
* **Attribution**: Highlighted in the dependency-check report based on CVE listings**.**

**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 Financials’ software application.

To address the identified vulnerabilities in Artemis Financials’ software application, several steps must be taken. First, outdated and vulnerable dependencies, such as bcprov-jdk15on-1.46.jar, spring-boot-2.2.4.RELEASE.jar, logback-classic-1.2.3.jar, log4j-api-2.12.1.jar, and jackson-databind-2.10.2.jar, must be upgraded to their latest stable versions to mitigate security risks such as cryptographic weaknesses, remote code execution, and data binding exploits. Dependency updates should be performed regularly to ensure that the application remains secure against newly discovered vulnerabilities.

Additionally, input validation and sanitization should be implemented across all REST endpoints to prevent injection attacks and ensure that user inputs are safely processed. Sensitive data, such as customer information, must be encrypted both in transit and at rest to protect confidentiality. Logging practices should be reviewed to avoid exposing sensitive information and ensure compliance with security best practices.

Authentication and authorization mechanisms should be added or strengthened to secure access to the application’s endpoints. Finally, regular security audits, including automated dependency checks and manual code reviews, should be conducted to identify and address potential vulnerabilities proactively. Implementing these measures will significantly enhance the security posture of the application and protect both the organization and its users from potential threats.