# CS 305 Project One

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **January 26, 2025** | **Brett Plemons** |  |

## 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

Brett Plemons

**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 is a consulting company dealing with sensitive client financial data. Their needs include:

1. **Value of Secure Communication:** Critical to project sensitive customer data, maintain client trust, and ensure compliance with regulations.
2. **International Transactions:** If the company deals internationally, it must comply with international regulations like GDPR and secure cross-border data transfers.
3. **Governmental Restrictions:** Artemis Financial must comply with financial and data protection laws such as FINRA, SEC, or regional banking regulations.
4. **External Threats:**

* Injection Attacks (SQL, XML, etc.)
* Dependency vulnerabilities (e.g., Log4j, Spring vulnerabilities)
* Man-in-the-middle (MITM) attacks due to weak TLS configuration
* Cross-Site Scripting (XSS)

1. **Modernization Requirements:**

* Secure integration of open-source libraries and frameworks.
* Mitigating vulnerabilities in evolving web technologies like RESTful APIs.

**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.

Based on the analysis, the relevant areas of security for Artemis Financial’s web application include:

1. **Input Validation:** To prevent SQL injection and other input-related vulnerabilities.
2. **Authentication and Password Management:** To secure access to client-sensitive data.
3. **Session Management:** To prevent session hijacking or fixation attacks.
4. **Cryptographic Practices:** To ensure secure encryption of data in transit and at rest.
5. **Error Handling and Logging:** To avoid leaking sensitive information.
6. **Data Protection:** To safeguard client financial and personal data.

**3. Manual Review**

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

**Architecture Review**

1. Critical services and sensitive data operations are not encapsulated behind proper authorization mechanisms.
   1. Example: Direct access to sensitive operations via public methods without authentication.
2. Lack of centralized logging or auditing system for monitoring critical operations.
   1. Mitigation: Implement centralized logging and auditing mechanisms.

**Input Validation**

1. Missing input validation for user-provided data in REST endpoints.
   1. Example: The CRUDController allows raw user input without sanitization in create() and update() methods.
   2. Mitigation: Implement safelisting for acceptable inputs and sanitize inputs before processing.
2. Potential for SQL injection due to lack of validation before database queries in DocData.
   1. Mitigation: Use parameterized queries and validate inputs at the controller level.

**API Security**

1. Exposed endpoints without authentication or authorization.
   1. Example: GreetingController’s /greeting endpoint allows public access.
   2. Mitigation: Require authentication tokens (e.g., OAuth) for all API endpoints.
2. Missing HTTPS enforcement.
   1. Mitigation: Configure HTTPS and secure headers for all API interactions.

**Cryptography**

1. Cryptographic keys are hardcoded in source files.
   1. Example: The RestServiceApplication uses hardcoded keys for encryption.
   2. Mitigation: Move keys to environment variables or a secure secrets management system.
2. Use of insecure hashing algorithms (e.g., MD5).
   1. Mitigation: Replace MD5 with a stronger algorithm like bcrypt or SHA-256.

**Client/Server Security**

1. Insecure session handling.
   1. Example: No session timeouts or token expiration implemented.
   2. Mitigation: Implement session timeout policies and ensure tokens have expiration times.
2. Cookies not marked with Secure or HttpOnly.
   1. Mitigation: Configure cookies to include Secure and HttpOnly attributes.

**Code Errors and Quality**

1. Stack traces exposed in error responses.
   1. Example: Controllers return raw stack traces in responses when exceptions occur.
   2. Mitigation: Use generic error messages and log stack traces only in secure server logs.
2. Inconsistent exception handling across services.
   1. Mitigation: Centralize exception handling using a middleware or global error handler.

**Encapsulation**

1. Direct exposure of sensitive data through public class fields.
   1. Example: Public fields in the Customer model expose sensitive user information.
   2. Mitigation: Use private fields and expose data through controlled getter methods.
2. Lack of immutability in data structures where necessary.
   1. Mitigation: Refactor mutable structures into immutable ones where appropriate.

**Code Review Categories**

1. **Views:** Missing XSS protection in HTML responses.
   1. Example: Direct rendering of user input without escaping in the GreetingController.
   2. Mitigation: Escape all user-provided data rendered in HTML using libraries like OWASP Java Encoder.
2. **Models:** No validation for fields in the Customer class.
   1. Mitigation: Add validation annotations (e.g., @NotNull, @Size) for all fields.
3. **Controllers:** Missing input validation in controller logic.
   1. Example: CRUDController directly passes user inputs to services without validation.
   2. Mitigation: Validate all inputs at the controller level before passing to services.
4. **Data Access:** Unsafe queries in DocData due to concatenated SQL strings.
   1. Mitigation: Use parameterized queries to prevent SQL injection.
5. **Services:** Privilege escalation vulnerabilities.
   1. Example: Services do not check if the user has the right permissions.
   2. Mitigation: Implement role-based access control checks in service methods.

**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

The results of the dependency-check static test revealed the following vulnerabilities:

| **Dependency** | **Vulnerability ID(s)** | **Severity** | **Description and Fix** |
| --- | --- | --- | --- |
| log4j-api-2.12.1 | CVE-2021-44228 | Critical | Log4j RCE vulnerability. Upgrade to version 2.16.0 or later. |
| spring-boot-2.2.4 | CVE-2020-5421 | Critical | Data exposure vulnerability in Spring Boot. Upgrade to 2.3.1.RELEASE or later. |
| bcpkix-jdk15on-1.46 | Multiple | High | Outdated BouncyCastle library. Upgrade to a later version that addresses CVEs. |
| jackson-databind | CVE-2020-8840 | High | Deserialization vulnerability. Upgrade to version 2.10.3 or later. |
| hibernate-validator | CVE-2020-10693 | Medium | Improper validation in Hibernate. Upgrade to 6.0.20.Final or later. |
| snakeyaml-1.25 | CVE-2022-1471 | Critical | YAML parsing vulnerability. Upgrade to version 1.28 or later. |

**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.

Here’s the action plan for mitigating identified vulnerabilities:

| **Vulnerability** | **Mitigation Steps** |
| --- | --- |
| Hardcoded credentials | Move credentials to secure environment variables or a secrets manager. |
| Lack of input validation | Use safelist validation for all inputs, validate types, ranges, and formats. |
| No parameterized queries | Refactor database queries to use parameterized or prepared statements. |
| Weak password hashing | Implement bcrypt for password hashing |
| Insufficient error handling | Replace stack traces in error responses with generic error messages. |
| Unsecured endpoints | Enforce HTTPS for all API endpoints and configure secure headers (e.g., Content-Security-Policy, X-Frame-Options). |
| log4j-api-2.12.1 | Upgrade to Log4j version 2.16.0 or later to address CVE-2021-44228 |
| spring-boot-2.2.4 | Upgrade to Spring Boot version 2.3.1.RELEASE or later. |
| bcpkix-jdk150n-1.46 | Upgrade to the latest version of the BouncyCastle library. |
| jackson-databind | Upgrade to Jackson Databind version 2.10.3 or later. |
| hibernate-validator | Upgrade Hibernate Validator to 6.0.20.Final or later. |
| snakeyaml-1.25 | Upgrade SnakeYAML to version 1.28 or later. |