# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **07/21/24** | **Andrei Shostak** | **Initial** |

## 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

Andrei Shostak

**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?

**Client's Needs and Potential Threats:**

- **Secure Communications**: Artemis Financial requires secure communications to protect sensitive financial data. This includes encryption of data in transit and at rest.

- **International Transactions**: The company may engage in international transactions, necessitating compliance with various international data protection laws and standards such as GDPR.

- **Governmental Restrictions**: Compliance with governmental regulations like GDPR, CCPA, and other financial regulations is crucial.

- **External Threats**: Potential threats include phishing, malware, ransomware, SQL injection, cross-site scripting (XSS), and man-in-the-middle attacks.

- **Modernization Requirements**: The company needs to adopt modern web application technologies and ensure the secure use of open-source libraries.

**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.

**Relevant Security Areas for Artemis Financial’s Web Application:**

1. **Input Validation**: To prevent attacks such as SQL injection and XSS, ensuring all user inputs are validated and sanitized.

2. **APIs**: Secure API interactions to prevent unauthorized access and data breaches.

3. **Cryptography**: Proper use of encryption to protect data at rest and in transit.

4. **Client/Server**: Ensure secure communication between client and server, protecting against attacks like man-in-the-middle.

5. **Code Error Handling**: Secure error handling to avoid revealing sensitive information through error messages.

6. **Code Quality**: Adherence to secure coding practices to prevent vulnerabilities.

7. **Encapsulation**: Secure data structures to prevent unauthorized data access.

**3. Manual Review**

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

**Identified Vulnerabilities:**

1. **SQL Injection** in DocData.java - The code directly executes SQL queries without parameterized statements.

2. **Hardcoded Credentials** in DocData.java - Database credentials are hardcoded, which is a security risk.

3. **Sensitive Information in Logs** - The application logs sensitive information, which could be accessed by unauthorized users.

4. **Lack of Input Validation** - User inputs are not properly validated in several endpoints.

5. **Insecure API Endpoints** - API endpoints lack proper authentication and authorization mechanisms.

6. **Outdated Libraries** - The application uses outdated libraries with known vulnerabilities (e.g., Bouncy Castle).

7. **Improper Error Handling** - Error messages reveal stack traces and sensitive information.

8. **Insecure Session Management** - Sessions are not properly managed or invalidated after logout.

9. **Insecure Direct Object References** - Direct references to database objects without proper authorization checks.

10. **Cross-Site Scripting (XSS)** - User inputs are not sanitized, making the application vulnerable to XSS attacks.

**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

**Dependency Check Report Findings:**

- **bcprov-jdk15on-1.46.jar**: Multiple vulnerabilities including CVE-2013-1624 (Timing attack vulnerability) and CVE-2015-7940 (Invalid curve attack).

- **Solution**: Update to the latest version of Bouncy Castle.

- **spring-boot-2.2.4.RELEASE.jar**: Known vulnerabilities including critical issues.

- **Solution**: Update to a newer version of Spring Boot.

- **logback-core-1.2.3.jar**: Vulnerabilities such as CVE-2021-42550.

- **Solution**: Update to the latest version of Logback.

- **log4j-api-2.12.1.jar**: Multiple critical vulnerabilities.

- **Solution**: Update to the latest version of Log4j.

- **snakeyaml-1.25.jar**: Multiple vulnerabilities.

- **Solution**: Update to the latest version of SnakeYAML.

**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.

**Steps to Mitigate Identified Vulnerabilities:**

1. **SQL Injection**:

- Implement parameterized queries or prepared statements.

- Use ORM tools like Hibernate.

2. **Hardcoded Credentials**:

- Use environment variables or secure vaults to store credentials.

3. **Sensitive Information in Logs**:

- Implement log scrubbing to remove sensitive information.

- Use logging frameworks that support secure logging.

4. **Lack of Input Validation**:

- Implement robust input validation using libraries like Hibernate Validator.

- Sanitize user inputs to prevent XSS.

5. **Insecure API Endpoints**:

- Implement proper authentication (e.g., OAuth2) and authorization mechanisms.

- Validate all API inputs and outputs.

6. **Outdated Libraries**:

- Regularly update dependencies to their latest versions.

- Use tools like OWASP Dependency-Check to automate vulnerability detection.

7. **Improper Error Handling**:

- Implement custom error pages and logging.

- Avoid displaying stack traces and sensitive information in error messages.

8. **Insecure Session Management**:

- Implement secure session management practices.

- Ensure sessions are invalidated on logout.

9. **Insecure Direct Object References**:

- Implement access controls and authorization checks for all direct object references.

10. **Cross-Site Scripting (XSS)**:

- Use libraries like OWASP Java Encoder to encode user inputs.

- Implement Content Security Policy (CSP) to mitigate XSS risks.