# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **11/24/24** | **Jose Munoz** |  |

## 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

Jose Munoz

**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, as a provider of financial services, has critical needs for secure communications and robust protection against external threats. The following considerations were identified:

* **Value of Secure Communications**: Secure communications are paramount for maintaining client trust and regulatory compliance. As financial data is sensitive, ensuring its confidentiality and integrity during transmission is essential.
* **International Transactions**: If Artemis Financial engages in international transactions, the company must ensure compliance with cross-border data protection laws such as GDPR and secure global data transfer protocols.
* **Governmental Restrictions**: Regulatory requirements such as Sarbanes-Oxley (SOX) and PCI DSS mandate secure storage, processing, and transmission of financial data. These restrictions must be factored into the design and implementation of the application.
* **Potential Threats**: Current and imminent threats include phishing attacks, malware, and DDoS attacks. Additionally, vulnerabilities in third-party libraries or APIs could expose the system to breaches.
* **Modernization Requirements**: Leveraging secure open-source libraries while keeping them updated is vital. Evolving web application technologies such as secure APIs and strong encryption mechanisms should also be incorporated

**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 vulnerability assessment process flow, the following areas are relevant to Artemis Financials’ application:

* **Input Validation**: Ensures all user-provided data is sanitized and prevents SQL injection or XSS attacks.
* **Secure Error Handling**: Protects sensitive information from being leaked through error messages.
* **Cryptography**: Encrypts sensitive client and financial data in transit and at rest to ensure confidentiality.
* **Secure API Interactions**: Protects data exchanged with third-party services by enforcing strict authentication and data validation.
* **Code Quality and Secure Coding Practices**: Adhering to secure coding standards minimizes vulnerabilities introduced during development.

**3. Manual Review**

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

The following vulnerabilities were identified through manual code inspection:

1. **DocData.java**: Hardcoded database credentials (root username and password) expose the application to credential theft.
2. **customer.java**: Lack of access modifiers on variables (account balance) increases the risk of unintended data access.
3. **CRUDController.java**: Returns sensitive data from DocData.toString() without sanitization.
4. **myDateTime.java**: Unimplemented methods (retrieveDateTime) could lead to undefined behavior.
5. **GreetingController.java**: Missing validation for the name parameter allows potential injection attacks.
6. **pom.xml**: Dependency on outdated library org.bouncycastle:bcprov-jdk15on:1.46 with known vulnerabilities.
7. **CRUD.java**: Inconsistent handling of object states due to overloaded constructors.
8. **MavenWrapperDownloader.java**: Hardcoded URL for Maven wrapper download could be a security risk if compromised.

**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

Using the OWASP Dependency-Check plugin, the following vulnerabilities were identified:

* **org.bouncycastle:bcprov-jdk15on:1.46**: Known CVEs include CVE-2018-1000613. The recommended solution is to update to version 1.70 or later.
* **Other Findings**: [Include additional vulnerabilities discovered during testing, with details such as severity, descriptions, and remediation steps.]

**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 address the identified vulnerabilities, the following actions are recommended:

1. **Remove Hardcoded Credentials**: Use environment variables or a secrets management tool to store database credentials securely.
2. **Implement Input Validation**: Use libraries like OWASP ESAPI or custom validation methods to sanitize user inputs.
3. **Update Dependencies**: Upgrade org.bouncycastle to version 1.70 or later and regularly review all dependencies for security patches.
4. **Enhance Secure API Practices**: Implement authentication mechanisms such as OAuth 2.0 and validate data exchanged with third-party APIs.
5. **Refactor Code for Security**: Apply proper access modifiers and encapsulation to variables and methods.
6. **Improve Logging and Error Handling**: Ensure error logs do not expose sensitive data and are stored securely.
7. **Regular Security Audits**: Perform periodic manual reviews and automated static testing to catch new vulnerabilities.