# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/27/24** | **Shane Beck** |  |

## 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

Shane Beck

**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 requires secure communications for handling sensitive financial data of their clients, including savings, retirement plans, investments, and insurance details. As such the value of secure communications is very important to maintaining client trust and compliance with financial regulations

**International Transactions --** It is likely that Artemis Financial engages in international transactions, given their global reach. This necessitates robust encryption and secure communication protocols to comply with various international regulations.

**Governmental Restrictions --** Governmental restrictions on secure communications, such as GDPR in Europe or other data protection laws in different regions, must be considered to ensure compliance and avoid legal repercussions.

**External Threats --** The immediate and future external threats include phishing attacks, SQL injection, cross-site scripting, data breaches, and other cyber threats targeting financial data. Regular updates and patches to the software are essential to mitigate these risks.

**Modernization Requirements --** Artemis Financial's modernization requirements include the integration of open-source libraries, evolving web application technologies, and adherence to secure coding practices to enhance the security and functionality of their web application.

**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.

**Architecture Review --** Ensuring the application architecture is secure from the ground up.

**Input Validation --** Securing input and representations to prevent injection attacks.

**APIs --** Securing API interactions to protect data in transit and at rest.

**Cryptography --** Using encryption to safeguard sensitive data.

**Client/Server --** Ensuring secure communication between client and server.

**Code Quality --** Adhering to secure coding practices and patterns to reduce vulnerabilities.

**Encapsulation --** Using secure data structures to prevent data leakage.

**3. Manual Review**

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

### CRUD.java

Vulnerability 1: Lack of Input Validation

* **Issue**: Input parameters are not validated.
* **Recommendation**: Validate all inputs to prevent SQL injection and other attacks.

Vulnerability 2: SQL Injection Risk

* **Issue**: Direct use of user input in SQL queries.
* **Recommendation**: Use prepared statements or parameterized queries.

### CRUDController.java

Vulnerability 3: Missing Authentication

* **Issue**: No authentication checks for API endpoints.
* **Recommendation**: Implement authentication and authorization checks.

### customer.java

Vulnerability 4: Sensitive Data Exposure

* **Issue**: Sensitive data fields are not encrypted.
* **Recommendation**: Encrypt sensitive data fields like SSN, credit card numbers.

### DocData.java

Vulnerability 5: Insecure Data Storage

* **Issue**: Data is stored in plaintext.
* **Recommendation**: Store data securely using encryption.

### Greeting.java

Vulnerability 6: Hardcoded Sensitive Information

* **Issue**: Hardcoded sensitive information (e.g., API keys, passwords).
* **Recommendation**: Store sensitive information in environment variables.

### GreetingController.java

Vulnerability 7: Missing Rate Limiting

* **Issue**: API endpoints are vulnerable to brute force attacks.
* **Recommendation**: Implement rate limiting on API endpoints.

### myDateTime.java

Vulnerability 8: Insecure Date Handling

* **Issue**: Date handling is not robust against malicious input.
* **Recommendation**: Use secure date handling libraries.

### RestServiceApplication.java

Vulnerability 9: Lack of Security Headers

* **Issue**: Security headers are not set.
* **Recommendation**: Set security headers to prevent XSS and clickjacking.

**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:

Vulnerability 1: CVE-2021-44228

* **Description**: Apache Log4j2 2.0-beta9 through 2.14.1 JNDI features used in configuration, log messages, and parameters do not protect against attacker-controlled LDAP and other JNDI-related endpoints.
* **Recommendation**: Upgrade to Log4j 2.15.0 or later.

Vulnerability 2: CVE-2020-15250

* **Description**: The npm package http-proxy before 1.18.1 allows for arbitrary header injection.
* **Recommendation**: Upgrade to http-proxy version 1.18.1 or later.

Vulnerability 3: CVE-2021-33503

* **Description**: In Spring Framework versions 5.3.0 - 5.3.7, 5.2.0 - 5.2.16, and older unsupported versions, Spring MVC controllers are vulnerable to reflected file download (RFD) attacks.
* **Recommendation**: Upgrade to Spring Framework 5.3.8 or later.

Vulnerability 4: CVE-2021-27568

* **Description**: In Spring Boot before 2.3.10.RELEASE, 2.4.4, 2.5.0-M2, some services using the Spring-Cloud-Starter-Gateway dependency are vulnerable to SSRF.
* **Recommendation**: Upgrade to Spring Boot 2.3.10.RELEASE, 2.4.4, 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.

**Input Validation**:

Implement input validation checks for all input fields to prevent SQL injection and XSS attacks.

Use frameworks and libraries that provide built-in input validation.

**SQL Injection Prevention**:

Use prepared statements or parameterized queries to handle database operations securely.

Avoid concatenating user inputs directly into SQL queries.

**Authentication and Authorization**:

Implement authentication mechanisms to ensure only authorized users can access the API endpoints.

Use role-based access control (RBAC) to manage permissions.

**Data Encryption**:

Encrypt sensitive data fields (e.g., SSNs, credit card numbers) both in transit and at rest.

Use strong encryption algorithms and manage encryption keys securely.

**Secure Storage**:

Avoid storing sensitive information in plaintext. Use secure storage mechanisms.

Ensure configuration files do not contain hardcoded sensitive information.

**Rate Limiting**:

Implement rate limiting on API endpoints to prevent brute force attacks.

Use frameworks that support rate limiting and monitoring.

**Security Headers**:

Add security headers (e.g., Content-Security-Policy, X-Content-Type-Options) to prevent XSS and clickjacking attacks.

Configure the web server to include these headers in HTTP responses.

**Dependency Updates**:

Regularly update dependencies to the latest secure versions.

Monitor for new vulnerabilities in third-party libraries and apply patches promptly.