# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **July 21, 2024** | **Alexander Ouellet** | **Initial Submission for Module 3** |

## Client



## Developer

Alexander Ouellet

**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 to protect sensitive financial information. Given the nature of their business, the following aspects are critical:

* **Value of Secure Communications**: Ensures the protection of client data, maintaining confidentiality and trust.
* **International Transactions**: If Artemis Financial handles international transactions, compliance with international data protection laws (e.g., GDPR) is necessary.
* **Governmental Restrictions**: Must adhere to regulations such as PCI-DSS for financial transactions and possibly SOX compliance.
* **External Threats**: Likely threats include phishing attacks, ransomware, and SQL injection attacks.
* **Modernization Requirements**:
  + **Open-Source Libraries**: Regular updates and security patches for libraries used in their application.
  + **Evolving Web Application Technologies**: Adoption of modern security practices, such as multi-factor authentication and end-to-end encryption.

**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.

**Applicable Security Areas**:

1. **Input Validation**: Critical to prevent injection attacks by validating all inputs from users.
2. **APIs**: Secure API interactions to prevent unauthorized access and data leaks.
3. **Cryptography**: Ensures data encryption in transit and at rest to protect sensitive information.
4. **Client/Server Communication**: Secure distributed components to prevent man-in-the-middle attacks.
5. **Code Error Handling**: Proper error handling to prevent leakage of sensitive information through error messages.
6. **Code Quality**: Ensuring adherence to secure coding practices to minimize vulnerabilities.
7. **Encapsulation**: Protects data structures to maintain data integrity and confidentiality.

Each of these areas is relevant because they directly impact the security and integrity of the financial information managed by Artemis Financial's web 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.

**Identified Vulnerabilities**:

1. **Class: UserInputHandler.java**
   * **Vulnerability**: Lack of input validation.
   * **Description**: User inputs are directly used in SQL queries, making the application vulnerable to SQL injection attacks.
2. **Class: PaymentProcessor.java**
   * **Vulnerability**: Hard-coded credentials.
   * **Description**: Sensitive information is hard-coded, which can be easily extracted by attackers.
3. **Class: ApiController.java**
   * **Vulnerability**: Insecure API endpoints.
   * **Description**: APIs are not secured with proper authentication, allowing unauthorized access.
4. **Class: DataEncryptor.java**
   * **Vulnerability**: Weak encryption.
   * **Description**: Uses outdated encryption algorithms that are susceptible to attacks.
5. **Class: SessionManager.java**
   * **Vulnerability**: Insecure session management.
   * **Description**: Sessions are not properly invalidated after logout, leading to potential session hijacking.
6. **Class: ErrorHandler.java**
   * **Vulnerability**: Detailed error messages.
   * **Description**: Error messages reveal sensitive information about the system architecture.
7. **Class: FileUploader.java**
   * **Vulnerability**: Unrestricted file upload.
   * **Description**: Allows upload of any file type, which can be exploited to upload malicious files.
8. **Class: EmailService.java**
   * **Vulnerability**: Email injection.
   * **Description**: User inputs are directly used in email body, making it vulnerable to email injection attacks.
9. **Class: LoggingService.java**
   * **Vulnerability**: Log injection.
   * **Description**: User inputs are directly logged without sanitization, leading to log injection vulnerabilities.
10. **Class: CacheManager.java**
    * **Vulnerability**: Insecure caching.
    * **Description**: Sensitive data is cached without encryption, which can be accessed by unauthorized users.

**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**:

1. **hibernate-validator-6.0.18.Final.jar**
   * **CVE-2020-10693**: Improper Input Validation.
   * **Solution**: Update to a newer version.
2. **jackson-databind-2.10.2.jar**
   * **CVE-2020-25649**: XML External Entity Injection.
   * **Solution**: Update to a newer version.
3. **log4j-api-2.12.1.jar**
   * **CVE-2020-9488**: Improper validation of certificate.
   * **Solution**: Update to a newer version.
4. **logback-core-1.2.3.jar**
   * **CVE-2023-6378**: Deserialization of Untrusted Data.
   * **Solution**: Update to a newer version.
5. **mongo-java-driver-2.4.jar**
   * **CVE-2021-20328**: Improper Certificate Validation.
   * **Solution**: Update to a newer version.
6. **snakeyaml-1.25.jar**
   * **CVE-2022-1471**: Remote code execution.
   * **Solution**: Update to a newer version.
7. **spring-core-5.2.3.RELEASE.jar**
   * **Various vulnerabilities**.
   * **Solution**: Update to a newer version.
8. **spring-webmvc-5.2.3.RELEASE.jar**
   * **Various vulnerabilities**.
   * **Solution**: Update to a newer version.
9. **tomcat-embed-core-9.0.30.jar**
   * **Various vulnerabilities**.
   * **Solution**: Update to a newer version.

**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. **Input Validation**:
   * Implement input validation and sanitization for all user inputs.
   * Use whitelisting techniques for allowed inputs.
2. **Hard-Coded Credentials**:
   * Remove hard-coded credentials and use secure storage mechanisms like environment variables.
3. **Insecure API Endpoints**:
   * Implement proper authentication and authorization for all API endpoints.
   * Use OAuth2 or JWT for securing APIs.
4. **Weak Encryption**:
   * Replace outdated encryption algorithms with strong, modern algorithms.
   * Regularly review and update encryption practices.
5. **Session Management**:
   * Ensure sessions are properly invalidated after logout.
   * Implement secure cookie attributes like HttpOnly and Secure.
6. **Error Messages**:
   * Remove detailed error messages and replace them with generic messages.
   * Log detailed errors on the server-side only.
7. **Unrestricted File Upload**:
   * Implement file type validation and restrict uploads to safe file types.
   * Scan uploaded files for malware.
8. **Email Injection**:
   * Sanitize user inputs used in email bodies.
   * Use parameterized methods for email content.
9. **Log Injection**:
   * Sanitize user inputs before logging.
   * Implement secure logging practices.
10. **Insecure Caching**:
    * Encrypt sensitive data before caching.
    * Implement cache expiration policies.