# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **May 26th 2024** | **Karina Washington** | Initial draft completed, including an overview of client needs and detailed security assessment.  Conducted thorough manual code review and identified potential security vulnerabilities.  Implemented findings from static testing to identify known vulnerabilities in the code base.  Developed a comprehensive mitigation plan to address identified security issues. |

## 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

Karina Washington

**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? Secure communications are paramount for Artemis Financial, given the nature of their business involving sensitive financial data such as savings, retirement plans, investments, and insurance details. Secure communications ensure the protection of this data from interception, unauthorized access, and breaches. This protection fosters trust with clients and complies with legal and regulatory requirements.
* Are there any international transactions that the company produces? Considering Artemis Financials’ global client base, it is highly likely that they handle international transactions. These transactions necessitate stringent security measures to protect data as it moves across different jurisdictions, each with varying regulations and potential vulnerabilities.

* Are there governmental restrictions on secure communications to consider? Financial institutions like Artemis Financial must comply with several regulations regarding secure communications. These regulations include General Data Protection Regulation (GDPR) for protecting EU citizens' data. Payment Card Industry Data Security Standard (PCI-DSS) for securing card payment transactions. Other national and international financial data protection laws, which mandate the implementation of secure communication protocols to protect sensitive financial information.

* What external threats might be present now and in the immediate future?   
  Artemis Financial faces several external threats, including but not limited to multiples attacks.

First, phishing attacks, there attempts to trick employees or clients into providing sensitive information through deceptive emails or websites. Also, Man-in-the-Middle (MitM) Attacks is when Interception and alteration of communication between two parties without their knowledge. Another attack can be SQL Injection. Attackers exploit vulnerabilities in the application to execute arbitrary SQL code, compromising the database. One more attack can be Cross-Site Scripting (XSS). Attackers inject malicious scripts into webpages viewed by other users. Finally, Denial of Service (DoS) Attacks can be attempts to make the application unavailable to users by overwhelming it with traffic.

* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies? Secure Open-Source Libraries is when utilizing the latest and most secure versions of open-source libraries to mitigate risks associated with outdated or vulnerable code.

Also, encryption standards while adopting and implementing the latest encryption standards for data at rest and in transit to ensure robust data protection. Another modernization can be API security to ensure all API interactions are secure, authenticated, and authorized to prevent unauthorized access and data breaches. Additionally, secure coding practices to maintain and enforcing secure coding standards to prevent vulnerabilities during the development phase. This includes regular code reviews and adherence to security best practices. Finally, evolving web application technologies to keep up to date with the latest web technologies and frameworks that offer improved security features and performance.

The findings to secure communications are essential for protecting sensitive financial data and maintaining client trust. Given the likelihood of international transactions, robust security for data in transit across jurisdictions is necessary. Compliance with governmental restrictions such as GDPR, PCI-DSS, and other financial data protection laws is mandatory. The company faces external threats including phishing, MitM attacks, SQL injection, XSS, and DoS attacks. Modernization requirements involve the use of secure open-source libraries, the latest encryption standards, API security, secure coding practices, and the adoption of evolving web technologies. These findings highlight the critical areas that need to be addressed to ensure the security and modernization of Artemis financials’ web-based software 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

Conducting an architecture review is fundamental to ensure the overall design of Artemis Financial’s software application adheres to security best practices. This involves evaluating the system’s components and their interactions to identify potential security weaknesses. For example, ensure the use of multi-layered security controls such as firewalls, intrusion detection systems, and secure network design. Implementing principles like least privilege and defense-in-depth will help protect against various types of attacks.

Input Validation

Input validation is critical for preventing injection attacks, such as SQL injection and cross-site scripting (XSS). Every input field should be validated to ensure it conforms to expected formats and ranges. For instance, when accepting user inputs such as usernames or financial amounts, the application should check for invalid characters and length constraints. Using prepared statements for database queries instead of directly concatenating user inputs can help mitigate SQL injection risks.

APIs

Ensuring the security of APIs is vital since they can be a gateway for unauthorized access if not properly secured. Implement strong authentication and authorization mechanisms for all API endpoints. For example, use OAuth2 for authorization and ensure APIs return minimal information in error messages to avoid revealing internal implementation details. Additionally, rate limiting can prevent abuse of the API endpoints.

Cryptography

Implementing robust cryptography ensures that sensitive financial data is protected both in transit and at rest. For data in transit, use HTTPS with strong TLS configurations to encrypt communications between the client and server. For data at rest, encrypt sensitive data stored in databases using strong encryption algorithms like AES-256. Regularly update cryptographic protocols and algorithms to address emerging vulnerabilities.

Client/Server

Securing the communication between the client and server is crucial to protect against man-in-the-middle (MitM) attacks. Ensure that all data exchanged between the client and server is encrypted using HTTPS. Additionally, implement measures such as HSTS (HTTP Strict Transport Security) to force secure connections and prevent protocol downgrade attacks. Employing mutual TLS can further enhance security by ensuring both the client and server authenticate each other.

Code Error

Proper error handling ensures that sensitive information is not leaked through error messages and helps maintain the integrity of the application. Use generic error messages for end-users to prevent exposing stack traces or detailed system information that could be exploited by attackers. For example, instead of displaying a database connection error, display a generic "Internal Server Error" message and log the detailed error information securely for internal review.

Code Quality

Following secure coding practices is essential to prevent common vulnerabilities and ensure the maintainability and robustness of the application. Conduct regular code reviews and use static analysis tools to detect security issues early in the development process. Adhere to coding standards and guidelines, such as the OWASP Top Ten, to mitigate risks associated with poor coding practices. For example, consistently using input sanitization and output encoding throughout the codebase can prevent many types of attacks.

These areas of security are crucial for ensuring that Artemis Financial’s web-based software application remains secure, reliable, and compliant with relevant regulations. Implementing these measures will help protect the company’s sensitive financial data and maintain the trust of its clients.

**3. Manual Review**

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

3. Manual Review

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

CRUDController.java

Vulnerability: Potential exposure of sensitive data.

Description: The method directly returns the result of doc.toString(), which may contain sensitive data.

DocData.java

Vulnerability: SQL Injection.

Description: The method read document uses dynamic SQL query construction, which can be exploited for SQL injection.

File: DocData.java

customer.java

Vulnerability: Insecure handling of account balance.

Description: Lack of input validation and authorization checks when updating the account balance.

GreetingController.java

Vulnerability: Cross-Site Scripting (XSS).

Description: The greeting endpoint does not sanitize user input, leading to potential XSS.

myDateTime.java

Vulnerability: Inadequate encapsulation.

Description: Direct access to date and time values without validation or security checks.

CRUD.java

Vulnerability: Improper use of constructors.

Description: Constructors allow inconsistent state creation which can be exploited.

RestServiceApplication.java

Vulnerability: Default configurations.

Description: Using default configurations without security hardening.

**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 names or vulnerability codes of the known vulnerabilities

Example: CVE-2023-0001, CVE-2023-0002

A brief description and recommended solutions provided by the dependency-check report

CVE-2023-0001: Description of the vulnerability, such as a known issue in an outdated library.

Recommended Solution: Update the library to the latest version.

Any attribution that documents how this vulnerability has been identified or documented previously

CVE-2023-0001: Identified in previous security bulletins and by the dependency-check tool.

CRUDController.java

**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.

CRUDController.java

Mitigation: Implement proper data sanitization and ensure only non-sensitive data is returned.

DocData.java

Mitigation: Use prepared statements and parameterized queries to prevent SQL injection.

customer.java

Mitigation: Implement input validation and authorization checks for account balance updates.

GreetingController.java

Mitigation: Sanitize user inputs to prevent XSS attacks.

myDateTime.java

Mitigation: Ensure encapsulation and proper validation of date and time values.

CRUD.java

Mitigation: Review and refactor constructors to ensure consistent and secure state initialization.

RestServiceApplication.java

Mitigation: Review and update configurations to follow security best practices.

Dependency Vulnerabilities

Mitigation: Regularly update dependencies to the latest versions and monitor for new vulnerabilities.A screen shot of a computer

Description automatically generated