# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **November 13, 2024** | **Duane Wegner** | **Clean Mavin Install**  **Analyzes the security requirements and potential threats faced by Artemis Financial** |

## 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

Duane Wegner

**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?

1. What is the value of secure communications to the company?

Secure communications are very important for Artemis Financial because they protect sensitive information, like customer data and financial transactions. If communications aren’t secure, hackers could steal or alter this data. Using strong security methods like encryption helps keep this information safe, ensuring customers trust the company and its services.

2. Are there any international transactions that the company produces?

Artemis Financial may deal with international transactions, which means they have to handle things like currency exchange and work with foreign banks. These transactions need to be secure to protect the company and its customers. There are also laws in different countries, like the GDPR in Europe, that require extra care with customer data when doing business internationally.

3. Are there governmental restrictions on secure communications to consider?

Yes, different governments have rules about secure communications. Some countries, like China and Russia, have strict laws on encryption and might require companies to use specific methods. Artemis Financial has to follow these rules in each country they operate in, while still making sure their communications stay secure.

4. What external threats might be present now and in the immediate future?

Artemis Financial faces many external threats, such as hackers using phishing emails, malware, or other tricks to steal information. They could also face attacks that shut down their systems, called DDoS attacks. In the future, more advanced threats like ransomware or even new technology, such as quantum computing, could pose risks to security. Artemis needs to stay ahead of these threats to keep its systems safe.

5. What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

As Artemis Financial updates its software, it must be careful with open-source libraries, which are free tools used in software development. These libraries can have security issues if they aren't updated regularly. Artemis also needs to use the latest web technologies that come with built-in security features to protect against problems like hacking or data breaches. Staying updated with new technology is key to keeping the system secure.

**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
  + Relevance:
    - A robust architecture review is crucial to ensure that Artemis Financial’s software is designed with security in mind from the start. This includes evaluating the system's structure, identifying potential threats like unauthorized access points, and ensuring the infrastructure supports secure data handling.
  + Justification:
    - Financial applications often deal with sensitive data such as user credentials, transaction details, and financial records. A secure architecture minimizes vulnerabilities that could be exploited by attackers to gain unauthorized access or compromise system functionality.
* Input Validation
  + Relevance:
    - Ensuring that all user inputs are properly validated is a key security measure to prevent injection attacks such as SQL injection, cross-site scripting (XSS), or other forms of input manipulation.
  + Justification:
    - Financial applications typically collect sensitive user data (e.g., bank account details, personal identification information). Without proper input validation, attackers could manipulate input fields to inject malicious code, potentially accessing or altering financial data.
* APIs
  + Relevance:
    - Secure API interactions are necessary to protect data exchanges between different components of the application or with third-party services. APIs often serve as the interface between the front-end and back-end, or between Artemis Financial’s application and external services like payment gateways or financial data providers.
  + Justification:
    - APIs expose endpoints that can be targeted by attackers if not properly secured. Since financial applications often rely on external systems for payments, identity verification, or data aggregation, securing these API interactions is critical to ensure the integrity and confidentiality of the data exchanged.
* Cryptography
  + Relevance:
    - Cryptography is essential for protecting sensitive data, both in transit and at rest. This includes the use of encryption for user passwords, financial transactions, and any other sensitive information.
  + Justification:
    - Financial applications handle highly sensitive data. Encryption ensures that data is unreadable by unauthorized parties, protecting user privacy and maintaining compliance with data protection regulations (e.g., GDPR, PCI DSS). Inadequate cryptographic measures could expose financial transactions and user data to theft.
* Client/Server
  + Relevance:
    - The secure distribution and communication between the client and server is critical to prevent unauthorized interception of data or man-in-the-middle (MITM) attacks.
  + Justification:
    - Financial applications often communicate over networks, and ensuring secure client-server communication prevents attackers from intercepting or tampering with financial data during transmission.
* Code Error
  + Relevance:
    - Handling code errors securely helps avoid information leakage and ensures that vulnerabilities are addressed promptly. This includes secure exception handling and logging practices that do not expose sensitive information.
  + Justification:
    - Application errors, if not handled properly, can reveal system information or stack traces that attackers can exploit. In a financial application, any leakage of error details can provide attackers with insights into system weaknesses that can be exploited for unauthorized access or manipulation.
* Code Quality
  + Relevance:
    - Maintaining high code quality ensures that security flaws are minimized and that the code is robust against potential attacks. Secure coding practices prevent vulnerabilities from being introduced into the software.
  + Justification:
    - Code quality is directly related to the reliability and security of the application. For Artemis Financial’s software, poor coding practices can introduce vulnerabilities like buffer overflows or race conditions, which could be exploited by attackers to compromise the application.
* Encapsulation
  + Relevance:
    - Encapsulation involves ensuring that sensitive data and system components are protected from unauthorized access or modification. It includes the proper use of access controls and visibility in the code.
  + Justification:
    - Financial applications require tight control over who can access sensitive data and perform certain operations. Proper encapsulation ensures that only authorized entities can access or modify financial data, reducing the risk of data breaches.
* Code Reviews
  + Relevance:
    - Manual code reviews help identify vulnerabilities that may have been missed in automated testing, particularly in complex or high-risk areas like financial calculations or user authentication.
  + Justification:
    - Given the high stakes involved in managing financial data, regular code reviews across various application components (views, models, controllers, etc.) help ensure that security best practices are followed and potential vulnerabilities are caught before deployment.
* Summary
  + Each of these security areas applies to Artemis Financial’s software application as follows:
    - Architecture Review: To ensure a secure structure for handling sensitive data.
    - Input Validation: To prevent malicious data manipulation.
    - APIs: To secure data exchange with third-party services.
    - Cryptography: To protect sensitive data through encryption.
    - Client/Server Communication: To secure data during transmission.
    - Code Error Handling: To prevent the disclosure of sensitive system information.
    - Code Quality: To reduce the likelihood of introducing vulnerabilities.
    - Encapsulation: To protect sensitive information from unauthorized access.
    - Code Reviews: To catch security flaws missed by automated tests.

**3. Manual Review**

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

* Outdated Spring Boot Version: The application uses spring-boot-2.2.4.RELEASE, which has several known vulnerabilities. An upgrade to a more recent version is necessary.
* Insecure Log4j Version: The version of Log4j (log4j-api-2.12.1.jar) used in the application is vulnerable to remote code execution (RCE) attacks. It is recommended to update to Log4j 2.17.1 or higher.
* Outdated Tomcat Version: The application uses tomcat-embed-core-9.0.30.jar, which has vulnerabilities such as the potential for denial-of-service attacks. An update to a more secure version is recommended.
* Vulnerable Jackson Databind Version: The jackson-databind-2.10.2.jar is outdated and vulnerable to deserialization issues, which could lead to remote code execution. It is recommended to upgrade to the latest stable version.
* Insecure YAML Library: The snakeyaml-1.25.jar library used is vulnerable to certain types of attacks due to improper YAML processing. Upgrading to a more secure version is advised.
* Outdated Hibernate Validator Version: The version hibernate-validator-6.0.18.Final.jar is not up to date and could lead to potential security flaws. An update is needed to resolve these issues.
* Insecure Spring Framework Versions: Multiple Spring Framework libraries (e.g., spring-web-5.2.3.RELEASE, spring-beans-5.2.3.RELEASE) used in the application have known vulnerabilities. The application should be updated to the latest Spring version.

**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
* A screenshot of a computer

  Description automatically generatedBouncyCastle Library (bcprov-jdk15on-1.46.jar):
  1. Vulnerability ID: CVE-2020-28052
  2. Severity: High
  3. Recommended Solution: Upgrade to a more recent version of BouncyCastle (1.70 or later).
* Spring Boot (spring-boot-2.2.4.RELEASE.jar):
  1. Vulnerability ID: CVE-2020-5398
  2. Severity: Critical
  3. Recommended Solution: Upgrade to Spring Boot 2.3 or higher.
* Log4j (log4j-api-2.12.1.jar):
  1. Vulnerability ID: CVE-2021-44228
  2. Severity: Critical
  3. Recommended Solution: Upgrade to Log4j 2.17.1 or later to mitigate RCE vulnerabilities.
* SnakeYAML (snakeyaml-1.25.jar):
  1. Vulnerability ID: CVE-2017-18640
  2. Severity: Critical
  3. Recommended Solution: Upgrade to SnakeYAML 1.27 or later.
* Jackson Databind (jackson-databind-2.10.2.jar):
  1. Vulnerability ID: CVE-2020-36518
  2. Severity: High
  3. Recommended Solution: Upgrade to Jackson 2.12 or higher.
* Tomcat (tomcat-embed-core-9.0.30.jar):
  1. Vulnerability ID: CVE-2020-9484
  2. Severity: Critical
  3. Recommended Solution: Upgrade to Tomcat 9.0.43 or later.
* Hibernate Validator (hibernate-validator-6.0.18.Final.jar):
  1. Vulnerability ID: CVE-2020-10327
  2. Severity: Medium
  3. Recommended Solution: Upgrade to Hibernate Validator 6.0.20 or higher.

**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.

* Upgrade Outdated Libraries: Update the affected libraries (e.g., Spring Boot, Log4j, Jackson Databind, SnakeYAML) to the latest stable versions. This will address known vulnerabilities and reduce the attack surface.
* Review Authentication and Authorization: Implement multi-factor authentication (MFA) for all users, particularly for those with access to sensitive financial data. Role-based access control should be enforced to ensure that only authorized users can access specific data.
* Enhance Input Validation: Ensure that all user inputs are validated, especially those coming from external sources, to prevent injection attacks and other malicious activities.
* Encrypt Sensitive Data: All sensitive data, including financial details and personally identifiable information (PII), should be encrypted both in transit and at rest using strong encryption algorithms.
* Continuous Monitoring: Implement a continuous security monitoring system to detect any anomalies or suspicious activities. This includes setting up logging and alerting systems that notify the development team of any security events.
* Conduct Regular Security Audits: Schedule regular security audits and penetration testing to identify new vulnerabilities and assess the effectiveness of the current security measures.