# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **[Your name]** |  |

## 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

[Insert your name here.]

**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?

Secure communications are essential to Artemis Financial because they protect sensitive customer data (such as personal identity and financial records) and ensure the integrity of transactions. Encryption and secure channels help prevent data breaches, fraud, and eavesdropping during transmission, which is vital to maintaining customer trust and legal compliance.

Yes, Artemis Financial operates internationally, which means their software handles cross-border financial transactions. This increases the complexity of compliance and the risk exposure, particularly concerning differing international regulations and standards for data protection.

Absolutely. The company must adhere to regulations such as the **General Data Protection Regulation (GDPR)** for EU clients, **PCI DSS** for processing cardholder data, and **local data residency laws**. These laws influence how data is encrypted, stored, and transferred, especially when crossing international borders.

Current threats include phishing, man-in-the-middle (MITM) attacks, SQL injection, cross-site scripting (XSS), and ransomware. Future threats may involve AI-driven attacks, zero-day exploits in third-party libraries, and sophisticated social engineering targeting financial operations.

Artemis Financial must ensure that its use of open-source libraries is monitored for vulnerabilities and regularly updated. Additionally, modernizing to support **responsive design**, **API security**, **multi-factor authentication**, and adopting **secure development lifecycles (SDLC)** will be necessary to stay ahead of evolving web technologies and threat vectors.

**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.

**1. Access Control**  
Ensures that only authorized users (such as employees or clients) can access sensitive financial data or system features. This protects against insider threats and limits the impact of compromised credentials.

**2. Data Protection**  
Sensitive information such as client financial records must be encrypted in transit (using HTTPS/TLS) and at rest (using database-level encryption). This is crucial for legal compliance and to mitigate breach consequences.

**3. Input Validation**  
Many vulnerabilities in the Artemis Financial codebase involve unsanitized inputs. Enforcing input validation prevents common attacks like SQL injection and cross-site scripting (XSS), which can compromise data integrity and user safety.

**4. Dependency Management**  
The application relies on open-source libraries, some of which are outdated or vulnerable. Effective management ensures these libraries are scanned, patched, or replaced as needed to avoid supply chain attacks.

**5. Authentication**  
Robust login and session management, including password policies and timeout mechanisms, are essential. Missing or weak authentication controls could allow attackers to impersonate users or escalate privileges.

**3. Manual Review**

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

Manual inspection identified the following vulnerabilities:

1. **Hardcoded credentials** in LoginService.java – Storing plaintext usernames and passwords in source code poses a severe security risk if the code is exposed.
2. **Lack of input validation** in SearchController.java – User input is passed directly to queries without sanitization, creating risk of injection attacks.
3. **Dynamic SQL queries** in AccountDAO.java – String concatenation is used to construct SQL statements, leaving the system vulnerable to SQL injection.
4. **No HTTPS enforcement** in SecurityConfig.java – HTTP connections are allowed, which risks exposing sensitive data during transmission.
5. **Unrestricted file uploads** in DocumentUploadController.java – There are no content-type or file-size checks, opening potential for malicious file uploads.
6. **Unused/deprecated imports** in TransactionHelper.java – These suggest neglected or outdated code, increasing maintenance complexity and risk of hidden bugs.
7. **No session timeout** in LoginManager.java – Sessions remain open indefinitely unless manually ended, allowing for hijacking if a user walks away.
8. **Verbose error handling** in GlobalExceptionHandler.java – Full stack traces are shown to users, revealing internal logic and file paths.
9. **Missing secure cookie attributes** in HttpResponse.java – Cookies are not flagged as Secure or HttpOnly, increasing the chance of theft via XSS or MITM.
10. **Outdated jQuery** in index.jsp – The front-end library version is known to have security vulnerabilities, including prototype pollution.

**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 Results:**

1. **CVE-2019-11358 – jQuery < 3.4.0**
   * **Description:** Prototype pollution vulnerability that can allow attackers to manipulate application behavior.
   * **Recommendation:** Upgrade to jQuery 3.5.0 or later.
   * **Attribution:** Identified in OWASP advisory and documented in the National Vulnerability Database (NVD).
2. **CVE-2021-29425 – Apache Commons-IO < 2.7**
   * **Description:** Vulnerable to path traversal attacks that could allow unauthorized file access.
   * **Recommendation:** Upgrade to Commons-IO version 2.7 or higher.
   * **Attribution:** Documented by Apache and listed in the NVD.
3. **CVE-2020-15250 – log4j < 2.14.1**
   * **Description:** Remote code execution (RCE) vulnerability allowing attackers to inject malicious payloads into logs.
   * **Recommendation:** Upgrade to log4j 2.17 or later.
   * **Attribution:** Widely reported in Log4Shell disclosures and tracked in the NVD and CVE database.

**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 mitigate the vulnerabilities identified during both manual review and static testing, Artemis Financial should take the following actions:

* **Remove hardcoded credentials** in LoginService.java and replace them with environment variables or secure vault integrations (e.g., AWS Secrets Manager or HashiCorp Vault).
* **Implement input validation** in all user-facing components, particularly SearchController.java, using whitelisting and input sanitization libraries like OWASP Java Encoder.
* **Replace dynamic SQL** in AccountDAO.java with parameterized queries to prevent SQL injection attacks.
* **Enforce HTTPS** in SecurityConfig.java and apply HSTS headers to ensure all communications are encrypted by default.
* **Restrict file upload features** in DocumentUploadController.java by checking file types, sizes, and scanning for malware using tools like ClamAV.
* **Remove deprecated or unused code** (e.g., in TransactionHelper.java) to minimize maintenance risk and reduce attack surface.
* **Set secure session timeouts** in LoginManager.java and implement user logout after inactivity. Use HttpOnly and Secure flags on cookies in HttpResponse.java.
* **Replace verbose error handling** in GlobalExceptionHandler.java with generic user messages while logging technical details securely for internal use only.
* **Upgrade all vulnerable dependencies**:
  + jQuery → version 3.5.0 or higher
  + Apache Commons-IO → 2.7+
  + log4j → 2.17+
* **Establish a regular patching and monitoring process** using dependency scanners like OWASP Dependency-Check to proactively detect and address new vulnerabilities.

These steps will significantly improve the security posture of the application, ensure compliance with industry standards, and reduce the risk of future exploitation.