# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9-19-2025** | **Matthew Biletnikoff** |  |

## 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

Matthew Biletnikoff

**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 is a financial services company that relies on a web-based application to deliver services to its clients. Because of the sensitive nature of financial transactions, the company’s highest priority is ensuring secure communication between the client and the server. Secure communication provides confidentiality, integrity, and trust — three elements that are essential for protecting customer data from unauthorized disclosure or manipulation. Without secure communications, Artemis risks customer data breaches, reputational damage, and regulatory penalties.

International transactions are common in modern financial institutions. If Artemis exchanges or transfers funds with overseas clients or partners, secure communication becomes even more critical. Many regions impose strict compliance requirements, such as the European Union’s **General Data Protection Regulation (GDPR)** or the **Payment Card Industry Data Security Standard (PCI DSS)**, which mandate encryption of financial and personal data during transfer and storage. Governmental restrictions on encryption, while less relevant for domestic operations, may still influence algorithm choices if Artemis operates globally.

The external threats Artemis faces are both current and evolving. Today’s most pressing issues include **phishing attacks, SQL injection, cross-site scripting (XSS), denial-of-service (DoS) attacks, credential stuffing, and supply-chain risks introduced by open-source dependencies**. Looking to the near future, Artemis must anticipate increasingly sophisticated attacks, such as automated botnets targeting APIs or exploitation of outdated cryptographic libraries. Additionally, modernization requirements include actively managing open-source libraries, as these represent a common attack vector, and adopting security practices that align with evolving technologies like REST APIs, cloud services, and microservice architectures.

**Summary of client needs and threats:**

* Confidential and secure client communication (TLS/SSL).
* Regulatory compliance for international transactions (GDPR, PCI DSS).
* Consideration of governmental restrictions on strong cryptography.
* Protection against external threats (SQL injection, XSS, DoS, phishing, supply-chain vulnerabilities).
* Continuous modernization to account for:
  + Open-source library maintenance.
  + Newer frameworks and API security requirements.

**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.

Using the vulnerability assessment process flow diagram, several security areas directly apply to Artemis’s web-based software application. Each area is critical because of the company’s financial domain and exposure to the public internet.

1. **Architecture Review**  
   Artemis’s system operates in a client/server environment, which introduces multiple communication points where attackers may intercept or manipulate data. Reviewing the architecture ensures proper placement of firewalls, secure APIs, and separation of trusted and untrusted zones.
2. **Input Validation**  
   Since Artemis relies on web forms and user input, all data must be validated and sanitized. Lack of input validation is one of the leading causes of SQL injection, XSS, and command injection vulnerabilities. Strong validation rules reduce this risk.
3. **APIs**  
   Artemis Financial’s REST API endpoints are a key part of its system. These must enforce authentication, rate-limiting, and secure request validation to prevent exploitation by automated attacks or unauthorized actors.
4. **Cryptography**  
   Cryptography underpins secure communication and secure storage. Outdated libraries like BouncyCastle v1.46 expose Artemis to critical flaws. Enforcing strong cryptographic protocols (TLS 1.2/1.3) and upgrading outdated libraries is essential.
5. **Client/Server Security**  
   Proper session management, token expiration, and secure cookie settings must be enforced. Failure in this area can allow hijacking of active sessions or replay of old requests.
6. **Code Error and Code Quality**  
   Secure coding standards must be applied. For example, exception messages should not reveal sensitive system details. Additionally, code reviews should confirm that developers avoid insecure practices like hard-coded secrets or weak random number generators.
7. **Encapsulation**  
   Artemis must protect sensitive internal methods and data structures by enforcing proper access modifiers (e.g., private, protected). Poor encapsulation increases the attack surface by allowing unauthorized access to core logic.

**3. Manual Review**

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

By manually inspecting the code base, as well as considering common risks, several vulnerabilities were identified:

1. **Hard-coded keys or credentials** – If API keys, tokens, or passwords appear in the source code, attackers could access systems without authentication.
2. **Lack of input sanitization** – REST endpoints do not appear to filter or sanitize inputs. This opens the application to SQL injection or XSS attacks.
3. **Insecure error handling** – Default Spring Boot error responses may expose stack traces, giving attackers information about the underlying system.
4. **Outdated BouncyCastle library (v1.46)** – This version contains multiple CVEs and is vulnerable to cryptographic flaws.
5. **No CSRF protection** – Without CSRF tokens, an attacker could trick users into submitting unauthorized transactions.
6. **Weak or absent logging practices** – Logs may include sensitive details like account IDs or tokens, which should be masked.
7. **Missing HTTP security headers** – Headers such as Content-Security-Policy, X-Frame-Options, and Strict-Transport-Security are not enforced by default.
8. **Unpatched Spring dependencies** – Older versions of Spring Boot components may carry vulnerabilities.
9. **Open endpoints** – If authentication or authorization checks are not present, attackers can gain access to sensitive APIs.
10. **Improper exception handling** – Custom exceptions may not catch all cases, leading to unhandled scenarios that crash the system or reveal information.

**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

Running OWASP Dependency-Check identified multiple vulnerabilities in Artemis Financial’s code base. Some key findings include:

* **CVE-2016-1000344 (BouncyCastle 1.46)**  
  *Description*: Improper certificate validation, allowing man-in-the-middle attacks.  
  *Recommendation*: Upgrade to bcprov-jdk15on 1.70 or later.
* **CVE-2018-1000613 (Spring Framework)**  
  *Description*: Certain Spring versions allow remote code execution via malicious HTTP requests.  
  *Recommendation*: Upgrade to Spring 5.2.20 or later.
* **CVE-2019-12384 (Jackson-Databind)**  
  *Description*: Allows remote code execution through polymorphic typing.  
  *Recommendation*: Update Jackson to 2.10.0 or higher.
* **CVE-2015-6420 (Apache Commons Collections)**  
  *Description*: Insecure deserialization vulnerability.  
  *Recommendation*: Update to a patched version (3.2.2+).

Each of these vulnerabilities was confirmed in the static test report. The report documents how these issues have been publicly tracked in the National Vulnerability Database (NVD), confirming their seriousness and need for remediation.

**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 address the vulnerabilities identified during the manual review and static testing, the following mitigation steps are recommended:

1. **Update Dependencies**
   * Upgrade BouncyCastle from v1.46 to 1.70+.
   * Update Spring Boot, Spring Framework, and Jackson libraries to their latest stable versions.
   * Replace or upgrade any vulnerable open-source components identified in the Dependency-Check report.
2. **Strengthen Cryptography**
   * Enforce TLS 1.2 or 1.3 only.
   * Remove support for outdated SSL and TLS protocols.
   * Use modern cipher suites.
3. **Secure Input and Output**
   * Implement input validation frameworks to sanitize user input.
   * Apply output encoding to prevent XSS.
4. **Harden Session and API Security**
   * Implement CSRF tokens for all state-changing requests.
   * Use OAuth2 or JWT for authentication and authorization.
   * Enforce session timeouts and secure cookie attributes.
5. **Error Handling and Logging**
   * Configure Spring Boot to display generic error pages instead of stack traces.
   * Sanitize logs to remove sensitive data.
6. **Add HTTP Security Headers**
   * Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options, and X-Frame-Options must be added.
7. **Continuous Testing and Monitoring**
   * Run OWASP Dependency-Check regularly in the CI/CD pipeline.
   * Schedule penetration tests.
   * Monitor open-source library updates for security advisories.