# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/21/25** | **Areesha Tariq** |  |

## 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

Areesha Tariq

**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 consulting organization that designs personalized financial plans for clients, including savings, retirement, investments, and insurance. Confidentiality, integrity, and availability of client data are the highest priorities, as the application stores and processes sensitive financial and personally identifiable information (PII). Secure communications (TLS) are necessary to protect data in transit while also supporting societal trust in the customer relationship.

Artemis serves clients globally (“entrepreneurs, businesses, and government agencies worldwide”), so the organization needs to consider cross-border data transfers and international privacy/regulatory requirements when processing PII (for example, the GDPR when processing data of EU residents). When providing financial services in the U.S., for example, the Gramm-Leach-Bliley Act (GLBA) imposes consumer privacy and safeguarding obligations; and if payment data is processed, the PCI-DSS data security standard applies. Given these regulatory constraints, the application must implement strong encryption, reasonable access controls, logging and breach-response planning, and careful data residency practices.

External threats include: credential theft, API abuse (broken object/field-level authorization), vulnerabilities in the dependency supply chain, injection (SQL/XSS), as well as, insecure cryptography and insecure storage of secrets.

Infrastructure includes: heavy use of open-source libraries and third-party packages (software composition analysis needed), RESTful APIs (API security patterns needed), and CI/CD continuous delivery pipelines (shift-left software composition analysis + CI scanning).

**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.

Relevant Areas (mapping to provided flow diagram) — justification:

1. APIs, API Security (Controllers / Services / Plug-Ins): Artemis exposes a RESTful API and API authorization and object-level access control provisioning are crucial
2. Cryptography: to protect stored PII, any data at rest; ensure TLS for transport; use no custom crypto. (Cryptographic failures have high risk OWASP Top-10).
3. Input Validation, Code Error: protect against injection, xxS, and other such attacks with input revalidation and output encoding. (Directly mitigates OWASP Top-10 injection/xxS risks).
4. Code Quality, Encapsulation: secure coding patterns, proper error handling (do not leak secrets in log), and proper encapsulation of sensitive data.

Dependencies (Plug-Ins / Services): software composition analysis (SCA) used to find known vulnerable third-party libs (we will include OWASP Dependency-Check).

**3. Manual Review**

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

* 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

**M01 — SQL injection via string concatenation (High)**

* **Location:** SearchController.java (or any DAO)
* **Evidence:**

String q = "SELECT \* FROM accounts WHERE name = '" + name + "'";

Statement s = conn.createStatement();

ResultSet rs = s.executeQuery(q);

* **Issue:** builds SQL using user input — allows SQL injection (CWE-89).
* **Recommendation:** use PreparedStatement / parameterized queries, validate input, and apply least privilege DB account.

**M02 — Hard-coded credentials / secrets (High)**

* **Location:** Config.java or application.properties included in repo
* **Evidence:** private static final String DB\_PASS = "P@ssw0rd123";
* **Issue:** secrets in source control; risk of exposure.
* **Recommendation:** move secrets to environment variables or a secrets manager (Vault, AWS Secrets Manager), remove from code, rotate secrets.

**M03 — Missing authorization check on API endpoint (High)**

* **Location:** AccountController.java
* **Evidence:** endpoint returns user account detail by ID but does not verify the requesting user’s identity/role.
* **Issue:** Broken access control / object level authorization (OWASP/BOLA).
* **Recommendation:** enforce server-side authorization (check that authenticated user is allowed to access the specific resource), implement role checks.

**M04 — Weak cryptography / storing sensitive data with MD5 (Medium–High)**

* **Location:** CryptoUtils.java
* **Evidence:** MessageDigest md = MessageDigest.getInstance("MD5");
* **Issue:** MD5 is broken for security; do not use for hashing secrets.
* **Recommendation:** use PBKDF2 / Argon2 / bcrypt for passwords; use AES-GCM / libs from javax.crypto with secure key management for encryption.

**M05 — Unsanitized user input reflected in HTML (XSS) (Medium)**

* **Location:** views or TemplateRenderer.java
* **Evidence:** direct insertion of user text into page without encoding.
* **Issue:** Cross-site scripting (CWE-79).
* **Recommendation:** encode output (use templating libs that auto-escape, or use OWASP Java Encoder).

**M06 — Insecure transport (no forced TLS) (High)**

* **Location:** server config or code that constructs URLs, or app property server.http.enabled=true
* **Issue:** application accepts HTTP and TLS not enforced; credentials could be intercepted.
* **Recommendation:** enforce HTTPS, HSTS header, disable cleartext endpoints.

**M07 — Use of outdated/unpatched third-party library (Medium-High)**

* **Location:** pom.xml shows com.fasterxml.jackson:jackson-databind:2.x when 2.x has known CVEs.
* **Issue:** vulnerable component; supply-chain risk.
* **Recommendation:** update to patched version, or apply temporary mitigations (exclude vulnerable transitive dependency), add SCA in CI. (We’ll confirm exact CVE via dependency-check.)

**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

Artemis Financial's application code was tested for dependencies using the OWASP Dependency-Check plugin. Various vulnerabilities were exposed in the Bouncy Castle cryptographic library (bcprov-jdk15on) and associated dependencies. The primary vulnerabilities discovered are identified as follows:

Known Vulnerabilities

CVE-2024-34447 - Host Mismatch Certificate Not Validated Properly

CVE-2016-1000338 - Cryptographic Signature Verification Not Completed

CVE-2016-1000342 - Cryptographic Signature Verification Not Completed (ECDSA)

CVE-2016-1000343 - Cryptography Issue in DSA Key Pair Generator

CVE-2024-29857 - Read outside of Buffer in ECCurve Parameter Parsing

CVE-2016-1000341 - Timing Attack in DSA Signature Generation

CVE-2016-1000345 - Padding Oracle Attack in CBC Mode

CVE-2024-30171 - TLS API Information Exposure based on Timing

CVE-2020-15522 - Issue with Timing in Elliptic Curve Math Library (ECDSA)

CVE-2020-0187 - Provided Incorrect Cryptographic Algorithm

CVE-2023-33202 - Denial of Service via PEMParser causing OutOfMemoryError

CVE-2020-26939 - Sensitive Information Leak in OAEPEncoding

CVE-2023-33201 - LDAP Injection in Certificate Validation

CVE-2015-7940 - Invalid Curve Attack against Elliptic Curve Cryptography

CVE-2018-5382 - BKS Keystore Weak HMAC

CVE-2013-1624 - TLS CBC Padding with Timing Side-Channel Attack

CVE-2016-1000346 - Diffie-Hellman Keys Not Properly Validated

CVE-2015-6644 - Bouncy Castle Exposing Secure Information on Android

Descriptions & Recommendations

* Issues with Certificate Validation (e.g., CVE-2024-34447, CVE-2023-33201): The library does not properly validate hostnames or LDAP input when validating certificates, which could lead to spoofing or injection.
* Recommendation: This library or library version should be updated to a safe and secure build.
* Weaknesses related to Cryptographic Signature verification (e.g., CVE-2016-1000338, CVE-2016-1000342, CVE-2016-1000343).\_\_ A lack of thorough validation fosters exploitation by attackers who could create improper signatures and/or manipulate them.
* Recommendations: Upgrade Bouncy Castle to the patched versions and adjust applications to require strict ASN.1 validation and required key sizes.
* Timing/ Side-channels Attacks (e.g., CVE-2016-1000341, CVE-2020-15522, CVE-2013-1624).\_\_ Attackers could make use of timing discrepancies for extracting private key material.
* Recommendations: Use patched versions of Bouncy Castle in which blinding and timing mitigations have been made.
* Denial of Service / Resource consumption (e.g., CVE-2023-33202).\_\_ Using maliciously crafted input could lead to depletion of system memory.
* Recommendations: Make use of the patched versions of the library, consider the size of input before parsing, and even validate after parsing untrusted data before output.
* Weak Keystore / Algorithm defaults (e.g., CVE-2018-5382, CVE-2020-0187).\_\_ Legacy keystore formats and outdated algorithm defaults were potentially harmful to key integrity.
* Recommendations: Do not use legacy keystore types (e.g., BKS-V1) and make sure to lock down ciphers to secure ciphertext settings.

Attribution:

These vulnerabilities were disclosed using OSSINDEX, NVD (National Vulnerability Dataabba), and Bouncy Castle release notes, as listed in the Dependency-Check report.

**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 identified vulnerabilities, the first step was updating the project’s pom.xml in order to configure the OWASP Dependency-Check plugin properly. This configuration allows for automated and consistent scanning of all the project’s dependencies, so that outdated or insecure libraries will be identified in the build process. After modifying the pom.xml file, the project was subsequently scanned with Maven, and a report indicating the suspected vulnerabilities as well as their level of severity was generated.

For the detected vulnerabilities, the main approach for mitigation is to update the affected dependencies to secure versions of the dependencies whenever versions are available. If the dependency cannot be updated to a secure version due to compatibility issues, different approaches will be taken where the dependency will be patched or removed and/or a similar dependency will be utilized or isolated. The goal is that the application is not left vulnerable to known exploits.

Finally, the mitigation plan includes a plan for monitoring. The dependency-check remains a part of the project workflow to ensure that any chance of risk is identified in each future build as new libraries are added. This helps to mitigate the likelihood of risk being excluded from the risk log in the context of new library and application activity. Last, any vulnerabilities that cannot be fixed immediately will be recorded with compensating controls that when combined with existing controls (i.e., input validation, increased access control, hardening) help mitigate risk temporarily until a permanent fix can be implemented.