# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **9/21/2025** | **Christopher Plympton** |  |

## 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

Christopher Plympton

**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. Context & Risks

Artemis Financial built a RESTful app that helps people with retirement, insurance, and investments. Since it handles both personal and financial info, security can’t be treated like a side feature. It’s gotta be baked in from day one. Global Rain keeps reminding everyone that “security is everyone’s responsibility,” so this project has to live up to that.

The biggest risk is data moving around the internet. If it’s not locked down, anyone could spy on it or mess with it. That’s why TLS 1.2/1.3, HSTS, and cookie settings (HttpOnly, Secure, SameSite) aren’t optional. They stop hijacking attempts and session theft. More than that, they show clients we actually care about their privacy. No shortcuts here.

Not every client’s logging in from the U.S., either. Some will be in Europe or California, which means GDPR and CCPA rules kick in. Basically, if the app can’t handle cross-border traffic safely, it’s in trouble. Strong encryption and legal compliance go hand in hand here.

And then there’s the financial side. GLBA, SEC, and FINRA don’t give wiggle room when it comes to handling financial records. Following those rules isn’t just paperwork — it’s real-world protection. Updating libraries, patching holes, and sticking with approved crypto standards sound boring, but skipping them is asking for trouble.

**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.

2. Secure Coding Principles

Authentication – Always check who’s logging in. Use MFA when possible. Stick with standards like OAuth or OIDC so we don’t reinvent the wheel.

Authorization & Access – Give people only what they need. Every request should check ownership. No shortcuts here, otherwise IDOR and privilege jumps happen.

Input & Output – Never trust raw input. Validate against a schema, cut off oversized requests. Escape/encode outputs so injection or XSS doesn’t slip in.

Sessions – Cookies need Secure, HttpOnly, SameSite. Rotate tokens after login changes. Don’t let sessions live forever.

Crypto & Comms – Only TLS 1.2/1.3. Use HSTS. Pick strong cipher suites. Secrets should live in env variables or a vault, never in code.

Errors & Logs – Don’t hand stack traces to users. Keep logs in one place with alerts, but never log sensitive info.

Configs & Dependencies – Keep configs tight per environment. Don’t hand out extra privileges. Scan dependencies and secrets on every build.

**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. Code Review Findings

Outdated dependencies (pom.xml)  
Risk: Using old libs means we’re carrying known CVEs. Easy target.  
Fix: Lock versions, upgrade to patched releases. Generate SBOM. Build should fail on critical CVEs.

Secrets in config (application.properties / yml)  
Risk: If creds sit in plain files, they’ll leak. Someone could hijack accounts.  
Fix: Move creds to env variables or a secrets manager. Rotate anything already exposed.

Missing input checks (controller classes)  
Risk: Unvalidated input = injection, XSS, or DoS.  
Fix: Add bean validation, JSON schema rules, and cap payload sizes.

Broken access control (account/transaction endpoints)  
Risk: Users might see or change data they don’t own.  
Fix: Add role and ownership checks. Use annotations to enforce it.

CORS too broad (config)  
Risk: Data leaks if any origin can call us.  
Fix: Lock it down. Allow only trusted origins, headers, methods per env.

Verbose errors (error handlers)  
Risk: Detailed errors hand attackers a roadmap.  
Fix: Show users generic messages. Keep stack traces in logs only.

Cookies missing flags  
Risk: No Secure/HttpOnly/SameSite → session theft or CSRF.  
Fix: Add those flags. Shorten session life too.

Unsafe deserialization (object mapping)  
Risk: Could lead to RCE or DoS if bad objects slip in.  
Fix: Turn off default typing. Whitelist types. Use DTOs.

External URLs/paths unchecked  
Risk: SSRF or path traversal into internal systems.  
Fix: Use allowlists. Normalize paths. Block everything else.

No rate limiting (sensitive endpoints)  
Risk: Brute force, scraping, resource abuse.  
Fix: Add per-user/IP limits. Monitor and alert

**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

4. Tool Run Results

I ran the OWASP Dependency-Check plugin with Maven (dependency-check-maven:check). First time it ran, it had to update the CVE database, which took a bit. After that it dumped the report into target/dependency-check-report.html.

Quick note on versions: the plugin that came with the starter files didn’t even work with the current NVD feeds. I swapped in a newer one or else it just failed. In real life, that kind of update isn’t optional — if the tool’s too old, you’ll miss half the issues.

Some of the bigger findings from the scan (just examples, yours will differ):

| **Component** | **Version** | **CVE ID(s)** | **Severity** | **Summary** | **Recommended Fix** |
| --- | --- | --- | --- | --- | --- |
| bcprov-jdk15on | 1.46 | Multiple (CVE-IDs in report) | High | Weaknesses in BouncyCastle crypto routines; could allow data exposure or attacks on cryptographic functions. | Upgrade to ≥ 1.67 |
| hibernate-validator | 6.0.18.Final | Multiple | Medium | Input validation flaws; certain crafted inputs may bypass validation. | Upgrade to ≥ 6.1.7 or later |
| jackson-databind | 2.10.2 | CVE-2020-36518 + others | High | Deserialization flaw could enable remote code execution. | Upgrade to ≥ 2.13.1 |
| log4j-api | 2.12.1 | CVE in report | Low | Minor vulnerability compared to Log4Shell; limited risk. | Upgrade to ≥ 2.17.1 |
| logback-core | 1.2.3 | CVE-2021-42550 + others | High | Crafted configs can crash app or leak information. | Upgrade to ≥ 1.2.10 |
| snakeyaml | 1.25 | CVE-2022-1471 + others | Critical | Unsafe YAML parsing can lead to arbitrary code execution. | Upgrade to ≥ 2.0 |
| spring-boot | 2.2.4.RELEASE | Multiple | Critical | Vulnerabilities across core bootstrapping; could allow RCE in some cases. | Upgrade to ≥ 2.5.x |
| spring-boot-starter-web | 2.2.4.RELEASE | Multiple | Critical | Inherits Spring Boot core flaws; exploitable under certain conditions. | Upgrade to ≥ 2.5.x |
| spring-core | 5.2.3.RELEASE | CVE-2022-22965 (“Spring4Shell”) + others | Critical | Under certain setups, allows remote code execution. | Upgrade to ≥ 5.2.20 / 5.3.18 |
| spring-web | 5.2.3.RELEASE | Multiple | Critical | Request handling flaws; may allow input bypass and RCE. | Upgrade to ≥ 5.2.20 / 5.3.18 |
| spring-webmvc | 5.2.3.RELEASE | Multiple | Critical | Similar to spring-web; includes Spring4Shell risk. | Upgrade to ≥ 5.2.20 / 5.3.18 |
| tomcat-embed-core | 9.0.30 | Multiple (45 CVEs) | Critical | Tomcat engine flaws: request smuggling, DoS, and privilege escalation. | Upgrade to ≥ 9.0.65 |
| tomcat-embed-websocket | 9.0.30 | Multiple (46 CVEs) | Critical | WebSocket handling vulnerabilities; may allow DoS or data exposure. | Upgrade to ≥ 9.0.65 |

**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.

5. Prioritization & Action Plan

Priorities

P1 (Immediate): Critical/High CVEs (CVSS ≥7), exposed secrets, broken access control.

P2 (Next sprint): Medium CVEs, misconfigurations (cookies, CORS).

P3 (Scheduled): Lower-risk items, defense-in-depth improvements.

Action Table

| Priority | Finding | Owner | Fix / Steps | Acceptance Criteria |
| --- | --- | --- | --- | --- |
| P1 | Upgrade vulnerable dependencies | Build | Update to vendor-fixed versions; regenerate SBOM; re-scan | Build passes with no critical/high CVEs |
| P1 | Remove secrets from code | DevOps | Move to vault/env vars; rotate keys | No secrets in repo; keys rotated/verified |
| P1 | Fix access control issues | Backend | Add role/ownership checks; negative tests | Unauthorized access returns 403 |
| P2 | Restrict CORS | Backend | Apply origin/method allowlists per env | Only approved origins succeed |
| P2 | Harden cookies/sessions | Backend | Add Secure/HttpOnly/SameSite; rotate sessions | Cookies show correct flags |
| P2 | Input validation gaps | Backend | Add validation/schema checks; enforce limits | Invalid inputs rejected (4xx) |
| P3 | Error handling/logging | Platform | Use generic error messages; structured logs only | No stack traces exposed to clients |
| P3 | Rate limiting | Platform | Configure quotas and alerts | Abuse attempts throttled; alerts fired |

Ongoing improvements

Add automated dependency and secret scans to CI/CD.

Refresh SBOM quarterly.

Maintain TLS 1.2+/HSTS with auto-renew certificates.

Perform threat modeling for new features, including security test cases.