# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5 / 22 / 25** | **Malachi Okongwu** | **First Release** |

## 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

[Malachi Okongwu]

**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?

[

a. Importance of Secure Communication

Given Artemis Financial's involvement in sensitive financial services, securing communication channels is essential. It protects both client assets and confidential data, which are foundational to maintaining trust and regulatory compliance.

b. International Transaction

Determine whether Artemis Financial conducts international financial transactions. Such activities would subject the organization to global regulations, particularly those concerning Personally Identifiable Information (PII) and data transmission security.

c. Legal Compliance and Government Regulations

Examine how U.S. legislation such as the Electronic Communications Privacy Act (ECPA), the Computer Fraud and Abuse Act (CFAA), Sarbanes-Oxley (SOX), and the Gramm-Leach-Bliley Act affects Artemis Financial. Legal consultation is recommended to ensure adherence to applicable laws.

d. Cyber Threat Analysis

Assess potential cybersecurity risks, ranging from relatively minor disruptions like Denial-of-Service (DoS) attacks to major breaches involving data exfiltration, financial loss, and reputational harm.

e. Adoption of Emerging Technologies

Evaluate the impact of integrating modern tools, including open-source libraries. This includes understanding licensing obligations, enforcing routine updates, and implementing modern encryption to safeguard data.

f. Open Source Risks and Maintenance

Highlight the risks inherent in using open-source software, such as the Log4J vulnerabilities. Ongoing monitoring and prompt patching of both open-source and proprietary components are vital for minimizing exposure.

]

**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.

[

a. Input Validation

Thorough validation and sanitization of all user inputs is vital for preventing exploits such as injection attacks. Even inputs from trusted sources can be erroneous or malicious.

b. Secure API Implementation

Beyond validating inputs, ensure secure user authentication, authorization, and encrypted transmission. Leverage frameworks like Spring but also implement custom error responses that do not expose system details.

c. Robust Error Handling

Error messages must be generic to avoid disclosing internal system architecture, yet informative enough to aid developers in debugging without creating security gaps.

d. Encapsulation Practices

Use encapsulation to restrict direct access to object fields. Private variables, accessed via getter and setter methods, help protect sensitive data from unintended exposure.

e. Data Encryption Standards

Strong encryption (e.g., RSA-2048) must be applied to both data-in-transit and data-at-rest. Ensure compliance with cryptographic regulations in all operational jurisdictions.

]

**3. Manual Review**

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

[

a. Incomplete Getter/Setter Use

Inconsistent encapsulation was found. While some classes, like Greeting, properly use private variables and accessors, others (e.g., GreetingController) do not, leaving data unprotected.

b. Unsafe User Input Handling

User inputs are not sanitized before use, particularly in the number method and the CRUD method. This opens the door for injection and cross-site scripting (XSS) attacks.

c. Exposure of Sensitive Data

The Customer class publicly exposes the account\_balance field, which should be private and accessed through controlled methods.

d. Coding Errors in Utility Classes

The myDateTime class contains unimplemented methods like setMyDateTime. In the DocData class, credentials are hardcoded (“root”, “root”), which is insecure. Even test credentials must be obfuscated, hashed, and salted.

e. Inadequate Error Reporting

Some methods, like read\_document in DocData, return full stack traces on error. This could expose internal workings to potential attackers.

Recommendation: Address these issues through rigorous code audits, apply secure coding practices, and establish stronger validation and exception handling mechanisms.

]

**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 screen

Description automatically generated

| **Dependency** | **Severity** | **Issue Summary** | **Recommended Action** |
| --- | --- | --- | --- |
| **SLF4J API** (1.7.30) | None | No known issues. | Continue monitoring and maintain version currency. |
| **Bouncy Castle** (1.46) | High | Multiple CVEs. | Update to a newer secure version. |
| **Spring Boot** (2.2.4) | Critical | Several critical vulnerabilities. | Upgrade immediately and assess for impact. |
| **Logback Core** (1.2.3) | High | Security flaws in logging mechanism. | Update and verify logging configuration. |
| **Log4j API** (2.12.1) | Critical | Highly exploitable vulnerabilities. | Urgent update and review of usage. |
| **JUL to SLF4J** (1.7.30) | None | No issues reported. | Regularly check for updates. |
| **Jakarta Annotations** (1.3.5) | None | No vulnerabilities. | Maintain with regular updates. |
| **SnakeYAML** (1.25) | Critical | Multiple parsing-related vulnerabilities. | Upgrade and sanitize YAML input. |
| **Jackson Databind** (2.10.2) | High | Data-binding risks. | Patch and review data handling. |
| **Jackson Annotations** (2.10.2) | None | No reported issues. | Keep updated. |
| **Jackson Core** (2.10.2) | None | No known risks. | Maintain and monitor. |
| **Jackson Datatype JDK8** (2.10.2) | Low | Minor concerns. | Stay up to date and monitor. |
| **Tomcat Core** (9.0.30) | Critical | 26 CVEs reported. | Update immediately and secure configurations. |

]

**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.

[

a. Assess and Prioritize Risks

Severity Ranking: Categorize vulnerabilities by risk level (Critical, High, Medium, Low).

Impact Review: Consider the effect of each vulnerability on system availability, data confidentiality, and regulatory compliance.

Remediation Order: Tackle the most severe and high-impact issues first.

b. Remediation Planning

Patch Management: Apply security patches to all vulnerable dependencies.

Code Refactoring: Modify insecure code, apply input validation, and enforce secure programming principles.

Configuration Hardening: Adjust system and application configurations to align with best practices.

c. Implementation

Test Before Deployment: Validate updates in a controlled environment to avoid regressions.

Phased Rollout: Deploy updates starting with critical systems.

CI/CD Pipeline: Automate deployment and include security checks.

d. Post-Remediation Validation

Retesting: Reassess systems post-remediation to confirm vulnerabilities are resolved.

Regression Testing: Ensure existing functionality remains unaffected.

e. Policy and Practice Updates

Root Cause Analysis: Document causes and integrate fixes into long-term practices.

Team Training: Educate development teams on secure coding and emerging threats.

f. Ongoing Monitoring

Regular Vulnerability Scans: Schedule automated scans to detect new threats.

Anomaly Detection: Use monitoring tools to identify suspicious behavior or breach attempts.

g. Documentation and Reporting

Track Changes: Maintain records of vulnerabilities, resolutions, and testing results.

Compliance Reporting: Share findings and updates with stakeholders and compliance officers.

Conclusion

Security is a continuous process requiring vigilance, structured remediation, and adaptation to emerging threats. By addressing vulnerabilities methodically, Artemis Financial can enhance its software resilience, protect critical assets, and uphold the trust of clients and partners. Implementing a proactive security strategy will ensure long-term operational integrity.

]