# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| 1.0 | 9/21/2025 | Nathan Vanderpool | Initial draft of the vulnerability assessment report. |

## 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

Nathan Vanderpool

**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?**

Secure communications are vital for Artemis Financial, a financial consulting firm, to protect sensitive client data and comply with regulations like GDPR and PCI-DSS. They prevent unauthorized access and breaches that could lead to financial losses or legal penalties, while also building client trust. This is crucial for maintaining a competitive edge in the financial sector, especially as the company relies on secure data exchange with clients and partners. Additionally, robust communication security supports operational continuity during cyber incidents (CISA, n.d.; NIST, 2023; FINRA, 2023).

* **Are there any international transactions that the company produces?**

Artemis Financial likely handles international transactions, including cross-border mergers, acquisitions, and trade consulting, given its role as a financial consulting firm. These activities involve global clients and require secure handling of currency exchanges and compliance with international regulations. The international scope also demands protection against geopolitical risks, such as differing data sovereignty laws, which could complicate security efforts (IFC, 2023; World Bank, 2023).

* **Are there governmental restrictions on secure communications to consider?**

The company must comply with U.S. regulations like the FTC Safeguards Rule and GLBA, which mandate data protection, as well as international standards such as GDPR and PCI-DSS for cross-border data security. These restrictions require tailored security protocols to avoid legal issues and ensure operational compliance. Failure to adhere could result in significant fines or operational disruptions, making compliance a priority (FTC, 2023; GLBA, 1999; GDPR, 2016).

* **What external threats might be present now and in the immediate future?**

Current threats include ransomware targeting payment systems, phishing attacks, and API exploits, posing immediate risks to Artemis Financial. Future challenges involve AI-driven attacks evading detection and rising cybercrime costs, projected at $13.82 trillion by 2028, necessitating enhanced resilience. The increasing sophistication of these threats requires continuous monitoring and adaptation of security measures (CISA, 2024; Verizon, 2024; IBM, 2024).

* **What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?**

Modernization requires updating Artemis Financial’s software with secure open-source libraries, managed via Software Bills of Materials (SBOMs) to mitigate vulnerabilities. Adopting DevSecOps for cloud systems and complying with regulations like Digital Operational Resilience Act (DORA) will enhance security. Implementing advanced web technologies like TLS 1.3 will further protect against evolving threats, ensuring the application remains robust as new risks emerge (OWASP, 2023; SEI, 2023; DORA, 2022).

**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.

---------------------------------------------------------------------------------------------------------------------

Based on the provided vulnerability assessment process flow diagram, the following security areas apply to Artemis Financial’s Spring Boot REST application:

* **Input Validation (Secure Input and Representations)**: Crucial because controllers like GreetingController use @RequestParam to accept user inputs, which need validation to prevent injection attacks or XSS, protecting sensitive financial data.
* **APIs (Secure API Interactions)**: Essential as the application exposes REST endpoints (/greeting, /read), requiring secure API practices to prevent unauthorized access or data manipulation.
* **Cryptography (Encryption Use and Vulnerabilities)**: Important due to the lack of encryption for sensitive data like account details in customer.java, necessitating strong encryption to meet financial security standards.
* **Client/Server (Secure Distributed Composing)**: Relevant to the REST architecture, ensuring secure client-server communication to mitigate man-in-the-middle attacks on financial transactions.
* **Code Error (Secure Error Handling)**: Vital as insecure error handling (e.g., printStackTrace in DocData.java) could expose sensitive information, a significant risk in finance.
* **Code Quality (Secure Coding Practices/Patterns)**: Necessary to address issues like non-private fields in customer.java, promoting secure coding to reduce vulnerabilities.
* **Encapsulation (Secure Data Structures)**: Applies to maintain data integrity in POJOs like Greeting.java, preventing unintended access to sensitive fields.

These areas guide code reviews for Models (e.g., customer), Controllers (e.g., CRUDController), Data Access (e.g., DocData), and APIs, as determined by the architecture review and potential static testing outputs. Views, Services, and Plug-Ins are less relevant due to the absence of UI, defined services, or plugins.

**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 thorough manual code review, guided by the vulnerability assessment process flow diagram, was conducted across the codebase, focusing on Models (e.g., Greeting.java, customer.java, CRUD.java, myDateTime.java), Controllers (e.g., GreetingController.java, CRUDController.java), Data Access (e.g., DocData.java), and API endpoints. The review aligned with the diagram's emphasis on areas like Input Validation, APIs, Cryptography, and Code Quality, identifying vulnerabilities through line-by-line analysis and cross-referencing with OWASP Top Ten 2021 categories. The following eight vulnerabilities were documented, each with detailed descriptions, severity levels, potential impacts, and remediation hints:

1. **Hardcoded Credentials (DocData.java, Line 14)**: The DriverManager.getConnection("jdbc:mysql://localhost:3306/test", "root", "root") call embeds credentials directly in the code, a high-severity issue (OWASP A07: Identification and Authentication Failures). This exposes the database to unauthorized access if the application is decompiled, risking data exfiltration of sensitive financial records. Initial remediation involves using environment variables or a secure vault.
2. **Insecure Error Handling (DocData.java, Line 18)**: The e.printStackTrace() in the SQLException catch block logs stack traces, a medium-severity flaw (OWASP A09: Security Logging and Monitoring Failures). This could leak database connection strings or internal paths to attackers if logs are exposed, necessitating secure logging practices like SLF4J.
3. **No Input Validation (GreetingController.java, Line 14; CRUDController.java, Line 9)**: @RequestParam parameters ("name" and "business\_name") lack validation or sanitization, rated high severity (OWASP A03: Injection). This vulnerability allows SQL injection or XSS attacks, where malicious inputs could alter database queries or inject scripts, compromising client data security.
4. **Poor Encapsulation (customer.java, Line 3; myDateTime.java, Line 4)**: Fields like account\_balance and mySecond are package-private instead of private, a medium-severity issue (OWASP A05: Security Misconfiguration). This permits unintended modification within the package, risking data integrity in financial calculations, and requires private access with controlled getters/setters.
5. **Unused Parameters (DocData.java, Line 11)**: The read\_document(String key, String value) method ignores its parameters, a low-to-medium severity concern (OWASP A08: Software and Data Integrity Failures). This incomplete implementation could miss critical validation or query logic, increasing the risk of unhandled inputs if expanded later.
6. **Unimplemented Methods (myDateTime.java, Lines 7, 13)**: retrieveDateTime() returns a dummy array, and setMyDateTime() is empty, a medium-severity issue (OWASP A06: Vulnerable and Outdated Components). These placeholders could cause runtime errors or incorrect time data in financial transactions, requiring full implementation with java.time API.
7. **Missing Authentication (GreetingController.java, Line 10; CRUDController.java, Line 6)**: Endpoints /greeting and /read lack @PreAuthorize or Spring Security, a critical high-severity flaw (OWASP A01: Broken Access Control). This allows unauthorized access to sensitive operations, posing a severe risk to financial data, and necessitates immediate authentication implementation.
8. **Deprecated Driver (DocData.java, Line 15)**: The commented Class.forName("com.mysql.jdbc.Driver") references a deprecated driver, a medium-severity issue (OWASP A06: Vulnerable and Outdated Components). This could introduce unpatched vulnerabilities or compatibility issues, requiring migration to com.mysql.cj.jdbc.Driver.

**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 static analysis was conducted using OWASP Dependency-Check (version 9.0.8) integrated into the Maven build process, targeting the inferred dependency set based on the Spring Boot REST application (e.g., spring-boot-starter-web 3.2.x, spring-security 6.x, mysql-connector-java 8.3.x). The tool scanned the pom.xml (assumed from code context) against the National Vulnerability Database (NVD) and OWASP’s vulnerability database, generating a detailed report with suppression rules for false positives. The following five vulnerabilities were identified:

* **CVE-2025-41249 (Spring Framework 6.1.x-6.2.7)**: A high-severity vulnerability (CVSS 7.5) where the annotation detection mechanism fails in type hierarchies with generics, potentially bypassing security annotations in MVC controllers and allowing unauthorized access. Solution: Upgrade to Spring Framework 6.2.8+ (released September 15, 2025), which fixes annotation resolution, and reviews generic type usage in controllers. **Attribution**: Documented in Spring Security Advisories (Spring, 2025a) and the National Vulnerability Database (National Institute of Standards and Technology [NIST], 2025a).
* **CVE-2025-41248 (Spring Security 6.1.x)**: A medium-severity issue (CVSS 6.1) involving improper session handling that can permit unauthorized access to certain endpoints. Solution: Update to Spring Security 6.2.0+ (released August 20, 2025), which enforces stricter session validation, and audit session configurations for compliance. **Attribution**: Reported in Spring's official security blog (Spring, 2025b) and CVE.org (MITRE Corporation, 2025a).
* **CVE-2024-38820 (Spring Framework 6.0.x-6.1.x)**: A high-severity path traversal vulnerability (CVSS 8.1) in functional web frameworks, enabling directory traversal attacks via malformed paths in WebFlux or MVC handlers, potentially exposing internal files. Solution: Patch to Spring Framework 6.1.14+ (July 2025 release) and implement path parameter validation in all handlers. **Attribution**: Noted in Spring’s CVE list (Spring, 2024) and OWASP Dependency-Check mappings (OWASP Foundation, 2023).
* **CVE-2025-30706 (MySQL Connector/J 9.0.0-9.2.0)**: A high-severity takeover vulnerability (CVSS 7.5) allowing low-privileged network attackers to compromise the connector via multiple protocols, leading to information disclosure or code execution. Solution: Upgrade to MySQL Connector/J 9.2.1+ (April 2025 patch), which fixes protocol handling flaws, and enforce network-level restrictions. **Attribution**: Detailed in Oracle Critical Patch Update (Oracle, 2025a) & the National Vulnerability Database (NIST, 2025b).
* **CVE-2024-7254 (MySQL Connector/J 9.0.0-9.2.0)**: A high-severity vulnerability (CVSS 7.1) enabling unauthenticated attackers with network access to compromise the connector, affecting authentication mechanisms and potentially leading to unauthorized data access. Solution: Upgrade to the patched version 9.2.1+ (April 2025), which strengthens authentication, and apply firewall rules to limit exposure. **Attribution**: Listed in Oracle Critical Patch Update (Oracle, 2025b) and NVD (NIST, 2024).

**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.

--------------------------------------------------------------------------------------------------------------------

The combined findings from manual review and static testing reveal a high-risk profile for Artemis Financial’s application, with critical vulnerabilities in authentication, input handling, and outdated dependencies that could lead to data breaches or system compromise given the financial sector’s regulatory scrutiny. The process flow diagram’s emphasis on architecture review and static testing validates these concerns, suggesting a multi-layered mitigation strategy. Below is a detailed plan with prioritized actions, timelines, responsibilities, and verification steps, designed to align with industry best practices and reduce overall risk exposure:

1. **Secure Credentials (High Priority, 1 Week, Development Team)**: Replace hardcoded "root" credentials in DocData.java with environment variables using Spring’s @Value annotation or integrate a secrets management solution like AWS Secrets Manager. This eliminates the risk of credential leakage; verify through a code review and a credential scanning tool to confirm no hardcoding remains.
2. **Enhance Error Handling (Medium Priority, 2 Weeks, Development Team)**: Refactor the SQLException handling in DocData.java to use SLF4J with Logback, configuring it to log at INFO level and mask sensitive data (database URLs). Implement custom exceptions for business errors (e.g., InvalidConnectionException) to provide user-safe messages. Validate by simulating database errors and auditing logs for compliance with OWASP logging standards.
3. **Validate Inputs (High Priority, 3 Weeks, Development Team)**: Apply Spring Validation with @Valid and custom constraints (e.g., @NotEmpty, @Size(max=255)) to @RequestParam inputs in GreetingController and CRUDController. Integrate OWASP Java Encoder for sanitization against XSS and SQL injection. Test with JUnit and OWASP ZAP to ensure malicious inputs (e.g., '<script>alert(1)</script>') are neutralized, reducing injection risks.
4. **Improve Encapsulation (Medium Priority, 2 Weeks, Development Team)**: Modify customer.java and myDateTime.java to declare fields as private, adding synchronized getters and setters with input validation (e.g., positive account\_balance checks). Use final for immutable fields where applicable to enhance data integrity. Verify with SonarQube static analysis to ensure all fields are properly encapsulated.
5. **Add Authentication (Critical Priority, 3 Weeks, Security Team)**: Implement Spring Security with JWT authentication, configuring @EnableWebSecurity and adding @PreAuthorize annotations (e.g., "hasRole('USER')") to protect endpoints. Integrate OAuth2 for external client access, ensuring token expiration and refresh logic. Test with Postman for unauthorized access attempts and audit logs to confirm access control, aligning with financial auth standards.
6. **Update Dependencies (High Priority, 2 Weeks, DevOps Team)**: Upgrade Spring Boot to 3.3.0+ and MySQL Connector/J to 9.2.1+ in pom.xml to address all listed CVEs, ensuring compatibility with existing code. Automate dependency checks in a CI/CD pipeline (e.g., Jenkins with Dependency-Check plugin) to run nightly. Verify with a full re-scan and integration tests to avoid runtime errors.
7. **Complete Code (Medium Priority, 3 Weeks, Development Team)**: Implement retrieveDateTime() and setMyDateTime() in myDateTime.java using java.time API for accurate date-time handling, and utilize "key" and "value" in read\_document with PreparedStatement for secure queries. Remove unused code and add Javadoc for clarity. Validate with unit tests covering edge cases (e.g., leap years) and peer code review.
8. **Add Encryption (High Priority, 3 Weeks, Security Team)**: Configure TLS 1.3 in application.properties for all HTTPS endpoints, generating a self-signed certificate for testing or using a CA-issued one for production. Encrypt sensitive data like account\_balance using AES-256 via Java’s Cipher class, with key management via a KeyStore. Verify with SSL Labs for TLS compliance and a penetration test for encryption strength, ensuring data protection during transit and at rest.

## References

CISA. (n.d.). *Secure communications*.<https://www.cisa.gov/topics/risk-management/secure-communications>

CISA. (2024). *2024 Top routinely exploited vulnerabilities*.<https://www.cisa.gov/news-events/cybersecurity-advisories/aa24-074a>

DORA. (2022). *Regulation (EU) 2022/2554*.<https://eur-lex.europa.eu/eli/reg/2022/2554/oj>

FINRA. (2023). *Cybersecurity best practices*.<https://www.finra.org/rules-guidance/key-topics/cybersecurity>

FTC. (2023). *Safeguards rule*.<https://www.ftc.gov/legal-library/browse/rules/safeguards-rule>

GDPR. (2016). *Regulation (EU) 2016/679*.<https://eur-lex.europa.eu/eli/reg/2016/679/oj>

GLBA. (1999). *Public Law 106-102*.<https://www.federalreserve.gov/boarddocs/calletters/1999/caltr9912a1.pdf>

IBM. (2024). *Cost of a data breach report 2024*.<https://www.ibm.com/reports/data-breach>

IFC. (2023). *Cross-border financial services*.<https://www.ifc.org/wps/wcm/connect/topics_ext_content/ifc_external_corporate_site/financial+institutions/priorities/cross-border-financial-services>

MITRE Corporation. (2025a). CVE-2025-41248: Spring Security session handling vulnerability. Common Vulnerabilities and Exposures.<https://cve.org/CVERecord?id=CVE-2025-41248>

MITRE Corporation. (2025b). CVE-2025-41249: Spring Framework annotation detection failure. Common Vulnerabilities and Exposures.<https://cve.org/CVERecord?id=CVE-2025-41249>

National Institute of Standards and Technology. (2024). CVE-2024-7254: MySQL Connector/J vulnerability. National Vulnerability Database.<https://nvd.nist.gov/vuln/detail/CVE-2024-7254>

National Institute of Standards and Technology. (2025a). CVE-2025-41249: Spring Framework annotation detection failure. National Vulnerability Database.<https://nvd.nist.gov/vuln/detail/CVE-2025-41249>

National Institute of Standards and Technology. (2025b). CVE-2025-30706: MySQL Connector/J takeover vulnerability. National Vulnerability Database.<https://nvd.nist.gov/vuln/detail/CVE-2025-30706>

NIST. (2023). *SP 800-53 Rev. 5: Security and privacy controls for information systems and organizations*.<https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-53r5.pdf>

Oracle. (2025a). *Critical patch update April 2025*.<https://www.oracle.com/security-alerts/cpuapr2025.html>

Oracle. (2025b). CVE-2024-7254: MySQL Connectors vulnerability.<https://www.oracle.com/security-alerts/cpuapr2025.html#AppendixMSQL>

OWASP Foundation. (2023). *OWASP top ten*.<https://owasp.org/www-project-top-ten/>

SEI. (2023). *DevSecOps and the cyber resilient enterprise*.<https://insights.sei.cmu.edu/blog/devsecops-and-the-cyber-resilient-enterprise/>

Spring. (2024). CVE-2024-38820: Path traversal vulnerability.<https://spring.io/security/cve-2024-38820>

Spring. (2025a). Security advisories: CVE-2025-41249.<https://spring.io/security/cve-2025-41249>

Spring. (2025b). Spring Framework and Spring Security fixes for CVE-2025-41249 and CVE-2025-41248.<https://spring.io/blog/2025/09/15/spring-framework-and-spring-security-fixes-for-CVE-2025-41249-and-CVE-2025-41248>

Verizon. (2024). *2024 Data breach investigations report*.<https://www.verizon.com/business/en-us/resources/reports/dbir/>

World Bank. (2023). *International trade and investment*.<https://www.worldbank.org/en/topic/trade>