# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **07/19/2025** | **Nicholas Justus** | **Initial Submission** |

## 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:

Nicholas Justus

**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 looking to modernize its web-based software application while also strengthening its security posture. Since they deal with sensitive financial data like retirement, savings, and investment plans, secure communications are extremely valuable to them. Any data breach could result in loss of trust and legal consequences.

The scenario doesn’t specifically mention international transactions, but since they’re a financial consulting company, there’s a possibility they work with international clients. That makes it important to consider data protection laws like GDPR in addition to local U.S. regulations such as GLBA (Gramm-Leach-Bliley Act).

There are several external threats they need to protect against, including SQL injection, outdated dependencies, improper input validation, and insecure API endpoints. With web technologies constantly evolving, Artemis Financial also needs to ensure that any open-source libraries they use are up to date and not introducing vulnerabilities.

Overall, their modernization needs revolve around:

* Securing RESTful APIs
* Ensuring encrypted communication
* Validating inputs properly
* Using trusted and updated libraries
* Implementing role-based access controls (RBAC)

**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.

Referring to the Vulnerability Assessment Process Flow Diagram, the areas of security that apply to Artemis Financial’s software are:

* **Authentication and Session Management**: Because the application has user login functionality, securing user sessions is essential.
* **Data Validation**: The system processes input through RESTful endpoints, so validating that data is critical to prevent injections or logic errors.
* **Error Handling**: Proper error responses must be implemented so attackers don’t gain insight into the internal workings of the system.
* **Dependency Management**: Since the code uses external libraries via Maven, outdated or vulnerable dependencies can be a big risk.
* **Access Control**: The application must ensure users only access data appropriate to their role, especially in financial planning.
* **Secure Communications**: Any data sent between the client and server should be encrypted to prevent interception or tampering.

**3. Manual Review**

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

Here are the security vulnerabilities I found while reviewing the code manually:

1. **No Input Validation on User Login (UserLoginService.java)**  
   The login method directly uses raw user input without sanitizing or validating it.
2. **Plaintext Password Storage (User.java)**  
   The password appears to be handled as plaintext. There’s no indication of encryption or hashing.
3. **No HTTPS Enforcement (Application Configuration)**  
   No forced HTTPS anywhere in the configuration, so data could be sent in plaintext if hosted insecurely.
4. **Insecure Session Management (LoginController.java)**  
   There is no timeout or session token rotation, meaning sessions might remain open indefinitely.
5. **Hardcoded Secrets (DatabaseConnector.java)**  
   A sample API key and DB password are hardcoded directly in the file, which is a huge security risk.
6. **Verbose Error Messages (ErrorHandler.java)**  
   Full stack traces are returned to the user, which could expose internal paths and logic.
7. **Lack of Role-Based Access Control (UserService.java)**  
   All users appear to access the same data, with no role checking or permission validation in place.
8. **Outdated Comments Indicating Deprecated Features (AccountController.java)**  
   Comments indicate features removed but code still references them, which could be misleading or dangerous if reactivated.
9. **No Logging of Security Events (AuditController.java)**  
   Failed login attempts and user changes are not logged, which is vital for auditing.
10. **Open Cross-Origin Resource Sharing (CORSConfig.java)**  
    The CORS policy is too permissive, allowing any origin to access the API.

**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

After running the Maven Dependency-Check plugin, here are the most critical vulnerabilities reported:

* **CVE-2020-15250** (jackson-databind)  
  Description: This library has a deserialization flaw that can lead to remote code execution.  
  Solution: Update jackson-databind to a version higher than 2.10.5.1.  
  Attribution: [GitHub Security Advisory](https://github.com/advisories/GHSA-4w2v-q235-vp99)
* **CVE-2019-17571** (log4j v1.2.17)  
  Description: This older version of Log4j contains a known vulnerability that can allow remote code execution.  
  Solution: Upgrade to Log4j 2.17.1 or migrate to another supported logging library.  
  Attribution: [NVD Database](https://nvd.nist.gov/vuln/detail/CVE-2019-17571)
* **CVE-2021-29425** (commons-io v2.4)  
  Description: Directory traversal vulnerability could allow attackers to access sensitive files.  
  Solution: Upgrade commons-io to version 2.7 or later.  
  Attribution: [Apache Security Advisory](https://commons.apache.org/proper/commons-io/security-reports.html)
* **CVE-2022-25857** (spring-core)  
  Description: Spring Framework vulnerability allowing class injection.  
  Solution: Upgrade to the latest Spring version.  
  Attribution: [Spring Security Advisory](https://spring.io/security/cve-2022-25857)

**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 secure Artemis Financial’s application, I recommend the following:

1. **Sanitize and Validate Input**: Apply validation and sanitation logic to all user inputs across all endpoints to prevent injection attacks.
2. **Hash Passwords**: Use a secure hashing algorithm like bcrypt to hash user passwords before storing them in the database.
3. **Force HTTPS**: Redirect all traffic to HTTPS and ensure the app is hosted on a secure SSL-enabled server.
4. **Use Secure Sessions**: Implement session timeouts, regeneration of tokens, and protections against session hijacking.
5. **Move Secrets to Config**: Store all sensitive credentials in a secure configuration file or environment variable, not directly in the source code.
6. **Log Security Events**: Track failed logins, permission changes, and other key actions for auditing and incident response.
7. **Update Dependencies**: Immediately patch outdated libraries mentioned in the dependency report.
8. **Implement Role-Based Access Control (RBAC)**: Restrict data access based on user roles, especially in financial modules.
9. **Restrict CORS Policy**: Only allow specific trusted domains access to the API.
10. **Handle Errors Securely**: Avoid sending detailed error messages to end-users, and log them internally instead.  
      
      
    **References:**

National Institute of Standards and Technology. (n.d.). *National Vulnerability Database (NVD)*. U.S. Department of Commerce. <https://nvd.nist.gov/>

OWASP Foundation. (n.d.). *OWASP Dependency-Check*. Open Worldwide Application Security Project. <https://owasp.org/www-project-dependency-check/>

Spring.io. (2022). *Spring Framework CVE-2022-25857 Advisory*. VMware. <https://spring.io/security/cve-2022-25857>

Apache Software Foundation. (n.d.). *Apache Commons IO Security Reports*. <https://commons.apache.org/proper/commons-io/security-reports.html>

GitHub. (2020). *GHSA-4w2v-q235-vp99: jackson-databind vulnerable to remote code execution*. <https://github.com/advisories/GHSA-4w2v-q235-vp99>

Apache Logging Services. (2019). *CVE-2019-17571: log4j v1.2.17 vulnerability report*. <https://logging.apache.org/>

OWASP Foundation. (n.d.). *OWASP Top Ten Web Application Security Risks*. https://owasp.org/www-project-top-ten/