# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **23May24** | **Jacob Batrano** |  |

## 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

Jacob Batrano

**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?

As a developer at Global Rain, working with Artemis Financial has given me a valuable perspective on the necessity of secure communications for financial consulting services. Artemis Financial deals with sensitive customer data involving savings, retirement plans, investments, and insurance policies, making security a top priority.

In evaluating their needs, it’s clear that Artemis Financial conducts transactions that might span across international boundaries. This introduces complexities related to compliance with international data protection regulations, such as GDPR in Europe and other regional laws that govern cross-border data transfers and financial transactions.

Given the nature of their services, the potential external threats include phishing attacks aimed at obtaining confidential financial information, man-in-the-middle attacks during data transmission, and more sophisticated threats like ransomware that could lock down critical financial data. As the company is also seeking to modernize its operations, the adoption of cloud services and the integration of third-party APIs could further expose the network to unauthorized access if not properly secured.

In response, our security measures must not only protect against these threats but also ensure compliance with governmental restrictions on secure communications. The value of secure communications is not just about safeguarding data but also maintaining the trust and confidence of Artemis Financial's clients. Ensuring robust encryption methods, secure data transmission protocols, and comprehensive data privacy measures will be crucial.

As we look to the future, we must also consider emerging threats and the role of evolving technologies like open-source libraries, which might introduce new vulnerabilities. The modernization of Artemis Financial's applications must include an evaluation of these aspects to mitigate potential security risks effectively.

**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.

My finding are below.

1. **Input Validation:** Ensuring all input to the system is validated to prevent injection attacks and ensure only properly formatted data enters our workflows.
2. **Secure Input and Representations:** Confirming that all data representations are secure, preventing data leaks or inappropriate data exposure.
3. **Architecture Review:** A comprehensive analysis of the application's architecture to identify potential security issues in the design and setup.
4. **Code Review:** A detailed examination of the application's code across different modules like Views, Client/Server architecture, and APIs to identify security flaws like improper error handling, insecure data structures, and weak cryptographic practices.
5. **Cryptography:** Focused on the use and implementation of encryption within the application to secure sensitive data against unauthorized access and ensure data integrity.
6. **APIs:** Reviewing API interactions to ensure they are secure against common vulnerabilities like broken access control and security misconfigurations.

**3. Manual Review**

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

1. Input Validation:
   * File: GreetingController.java
   * Issue: The method that handles greetings does not validate the name parameter input, which can lead to Cross-Site Scripting (XSS) if the input is reflected back to the user without proper sanitization.
2. Authentication and Authorization:
   * File: CRUDController.java
   * Issue: There are CRUD operations exposed without checks for user authentication or authorization, potentially allowing unauthorized access to sensitive data or functions.
3. Error Handling:
   * File: RestServiceApplication.java
   * Issue: The application does not implement a global exception handler. As a result, detailed error messages or stack traces might be sent to the client, leading to information disclosure.
4. Data Encryption:
   * File: customer.java
   * Issue: Sensitive data such as customer information is handled without encryption when stored or transmitted, risking data breaches and non-compliance with data protection regulations.
5. Session Management:
   * File: DocData.java
   * Issue: There appears to be no secure session management practice in place. Session tokens are not properly secured against hijacking, and there’s no mechanism to prevent session fixation.
6. Dependency Management:
   * File: CRUD.java
   * Issue: The file includes libraries that are outdated and have known vulnerabilities, which need updating to prevent exploitation.
7. Logging and Monitoring:
   * File: GreetingController.java
   * Issue: Logging of sensitive information such as user inputs is performed without masking, which could lead to information leakage in log files accessible to attackers.

**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

1. **Dependency:** **bcprov-jdk15on-1.46.jar**
   * **Vulnerability ID:** CVE-2013-1624
   * **Description:** The TLS implementation in the Bouncy Castle Java library before 1.48 and C# library before 1.8 does not properly consider timing side-channel attacks on a noncompliant MAC check operation during the processing of malformed CBC padding. This allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets.
   * **Recommended Solution:** Upgrade to a later version of Bouncy Castle that has addressed these timing side-channel attacks. Ensure all cryptographic operations are compliant with current standards.
   * **Attribution:** Documented by various security advisories including Red Hat and Secunia advisories.
2. **Dependency:** **spring-boot-2.2.4.RELEASE.jar**
   * **Vulnerability ID:** CVE-2022-27772
   * **Description:** Spring Boot versions prior to 2.2.11.RELEASE were vulnerable to temporary directory hijacking, impacting the method **AbstractConfigurableWebServerFactory.createTempDir**.
   * **Recommended Solution:** Upgrade to Spring Boot version 2.2.11.RELEASE or later to mitigate this vulnerability.
   * **Attribution:** This vulnerability was disclosed through various security channels including GitHub security advisories.
3. **Dependency:** **logback-core-1.2.3.jar**
   * **Vulnerability ID:** CVE-2017-5929
   * **Description:** Logback before 1.2, when a socket server and a TCP socket receiver are configured, allows remote attackers to execute arbitrary code via a crafted serialized object, related to Java deserialization.
   * **Recommended Solution:** Update to at least logback-core 1.2 or higher, which fixes deserialization vulnerabilities by properly validating input before object reconstruction.
   * **Attribution:** Widely reported in various security databases including NVD and Sonatype.
4. **Dependency:** **log4j-api-2.12.1.jar**
   * **Vulnerability ID:** CVE-2019-17571
   * **Description:** A socket receiver in Log4j 2.x allows remote code execution because the user input is deserialized.
   * **Recommended Solution:** Upgrade to Log4j 2.13.0 or newer, which disables socket servers by default as part of the security enhancements.
   * **Attribution:** Documented and fixed following extensive reporting on remote code execution risks associated with Log4j.
5. **Dependency:** **snakeyaml-1.25.jar**
   * **Vulnerability ID:** CVE-2017-18640
   * **Description:** The SnakeYAML library allows remote code execution because the user input is deserialized.
   * **Recommended Solution:** Upgrade to version 1.26 or higher where deserialization of untrusted data has been addressed by using safe constructors.
   * **Attribution:** Detailed in various security advisories including those by Sonatype and the YAML project maintainers.
6. **Dependency:** **jackson-databind-2.10.2.jar**
   * **Vulnerability ID:** CVE-2020-25649
   * **Description:** A flaw in the **jackson-databind** which could allow an unauthenticated user to perform code execution by inserting maliciously crafted JSON into fields.
   * **Recommended Solution:** Upgrade to **jackson-databind** version 2.10.5.1 or above, where this flaw is fixed.
   * **Attribution:** Identified and widely reported as part of the Jackson project's ongoing security reviews.
7. **Dependency:** **tomcat-embed-core-9.0.30.jar**
   * **Vulnerability ID:** CVE-2020-1938
   * **Description:** The "Ghostcat" vulnerability allows attackers to read configuration files and other files on the server via crafted AJP requests if AJP is enabled.
   * **Recommended Solution:** Disable AJP protocol if not used or upgrade to Apache Tomcat version 9.0.31 or later where AJP defaults to listen on localhost.
   * **Attribution:** First reported by the Apache Tomcat team and documented in the National Vulnerability Database (NVD).

**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.

As part of my role at Global Rain. I've developed a comprehensive mitigation plan addressing the vulnerabilities identified in our recent security assessment of Artemis Finacial. The plan prioritizes vulnerabilities based on their severity and potential impact, ensuring that critical issues such as remote code execution or data leaks are addressed promptly. To mitigate these vulnerabilities effectively, I recommend upgrading outdated libraries and frameworks, implementing secure coding practices, and adjusting configuration settings to tighten security.

The mitigation strategies include specific actions such as updating **jackson-databind** to a version that patches known serialization vulnerabilities, validating all inputs, and using prepared statements for database interactions to prevent SQL injection and XSS. We'll also disable unnecessary services and enforce strong authentication mechanisms.

To ensure the long-term security of Artemis Financial’s application, we will establish a routine for conducting regular security audits and continuous testing. This will help us catch and mitigate new vulnerabilities before they can be exploited. Post-implementation, we will conduct a thorough review to confirm that the vulnerabilities have been effectively mitigated and that no new issues have been introduced. This process of continuous improvement will be critical in adapting to evolving security threats and enhancing our security practices based on lessons learned from each assessment.

By adhering to this structured approach, we aim not only to protect Artemis Financial's web application from existing threats but also to strengthen its defenses against future vulnerabilities. This commitment to security is a core part of our mission at Global Rain and essential for maintaining the trust and confidence of our clients at Artemis Financial.