# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **7/20/2024** | **Armon Wilson** |  |

## 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

Armon Wilson

**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?

Secure communications are crucial for Artemis Financial as they ensure data confidentiality, integrity, authentication, authorization, regulatory compliance, and uphold the company's reputation and client trust. These communications prevent unauthorized access to sensitive financial data, protect client privacy, and ensure the integrity of financial transactions.

The prompt does not specify whether Artemis Financial conducts international transactions. However, it is prudent to assume that, as a consulting company with a global client base, secure communications are critical. International transactions can introduce additional security challenges due to cross-border data transfers and varying regulatory requirements, such as GDPR in Europe and CCPA in California.

Governmental restrictions on secure communications may exist depending on the jurisdictions in which Artemis Financial operates. These could include data localization requirements, encryption standards, or limitations on specific technologies. Examples of relevant regulations include GDPR (General Data Protection Regulation) in the European Union, CCPA (California Consumer Privacy Act) in California, and HIPAA (Health Insurance Portability and Accountability Act) in the United States.

External threats to Artemis Financial’s web-based application include data breaches, phishing attacks, malware and ransomware, denial-of-service (DoS) attacks, and injection attacks, such as SQL injection and cross-site scripting (XSS). Each of these threats can compromise the security and functionality of the application, leading to potential data loss, financial damage, and reputational harm.

To modernize and mitigate these threats, Artemis Financial should consider enhancing the security of open-source libraries, adopting new security frameworks and tools, conducting regular security assessments and updates, and adhering to secure coding practices.

**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 Vulnerability Assessment Process Flow Diagram and the nature of Artemis financials’ software application, the following areas of security are relevant:

1. **Input Validation:** Since the application is web-based and interacts with users, validating input is crucial to prevent vulnerabilities like injection attacks (e.g., SQL injection, cross-site scripting). For example, login forms, search bars, and any user input fields must be validated to ensure that only expected data is processed.
2. **Secure Input and Representations:** This is important to ensure that data entered by users is properly sanitized and encoded to prevent malicious input from being interpreted as code. Techniques include using prepared statements for SQL queries and encoding user inputs in HTML to prevent cross-site scripting (XSS)..
3. **Architecture Review:** An architecture review is necessary to identify potential design flaws that could be exploited by attackers. This includes analyzing the overall structure of the application, its components, and how they interact. It also involves checking for common issues such as improper separation of concerns, insecure communication channels, and lack of redundancy for critical components.
4. **Code Review:** A manual code review is essential to identify vulnerabilities that may not be detected by automated tools. This includes checking for insecure coding practices, logic errors, and potential backdoors. Automated tools can quickly identify common issues, while manual reviews can catch more complex or context-specific vulnerabilities.
5. **Encapsulation:** Proper encapsulation of data and functionality helps to limit the impact of potential vulnerabilities. This means restricting access to sensitive data and ensuring that objects can only be modified in controlled ways. Applying the principle of least privilege ensures that components only have access to the data and functionality they need.
6. **APIs:** Since Artemis Financial's software application is a RESTful web API, it's crucial to review the security of the API endpoints, authentication mechanisms, and data exposure. This includes implementing measures such as rate limiting, API key management, and using secure tokens for authentication.

The areas of Cryptography and Secure Error Handling may not be directly applicable in this assessment, as the provided information doesn't indicate the use of encryption or specific error handling mechanisms within the application. These areas could become relevant depending on the specific implementation of the software.

**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. **Insecure Dependency**: The project's pom.xml file includes an outdated version of the Bouncy Castle library (version 1.46). This library has known vulnerabilities that could be exploited by attackers.
2. **Hardcoded Credentials**: In the DocData.java file, there are hardcoded database credentials ("root," "root"). This is a security risk as it allows anyone with access to the code to connect to the database.
3. **SQL Injection Vulnerability:** The read\_document method in DocData.java appears to be vulnerable to SQL injection. The method does not properly sanitize the input parameters (key and value) before using them in a SQL query.
4. **Lack of Input Validation:** The CRUDController.java file does not validate the input parameter (name) received from the /read endpoint. This could lead to various vulnerabilities, including injection attacks.
5. **Information Exposure:** The CRUD class in CRUD.java has two constructors that take the same input (content) and store it in two different fields (content and content2). This could potentially lead to unintended information exposure if both fields are returned in a response.
6. **Unprotected Customer Information**: The customer.java file contains methods (showInfo and deposit) that directly access and modify customer account information without proper authorization or access controls.
7. **Insufficient Logging:** The code base lacks sufficient logging mechanisms. Logging is essential for security as it helps to track and investigate suspicious activities or errors.
8. **Insecure Direct Object Reference**: In the CRUDController.java file, the CRUD method directly returns a DocData object. This could potentially expose sensitive information if the object contains data that should not be accessible to the client.
9. **Lack of Exception Handling**: The read document method in DocData.java does not properly handle exceptions that may occur during database operations. This could lead to unexpected behavior and potential information leakage.
10. **Insufficient Transport Layer Protection**: The application does not enforce HTTPS for secure communication, leaving it vulnerable to man-in-the-middle attacks where data can be intercepted and modified in transit.

**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
* **Dependency:** bcprov-jdk15on-1.46.jar
  + **Description:** The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7.
  + **Vulnerabilities:** 
    - CVE-2024-34447 (OSSINDEX)
      * bouncycastle - Improper Validation of Certificate with Host Mismatch.
      * Fully check the hostname of the certificate and provide the user with adequate information about the nature of the problem and how to proceed. If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the host name.
      * **Attribution:** <https://cwe.mitre.org/data/definitions/297.html>
    - CVE-2024-29857 (OSSINDEX)
      * An issue was discovered in ECCurve.java and ECCurve.cs in Bouncy Castle Java (BC Java) before 1.78, BC Java LTS before 2.73.6, BC-FJA before 1.0.2.5, and BC C# .Net before 2.3.1. Importing an EC certificate with crafted F2m parameters can lead to excessive CPU consumption during the evaluation of the curve parameters.
      * Upgrade to the latest version of Bouncy Castle to address this vulnerability (Release 1.78.1).
      * **Attributions:** 
        + <https://nvd.nist.gov/vuln/detail/CVE-2024-29857>
        + <https://cwe.mitre.org/data/definitions/400.html>
    - CVE-2023-33202
      * Bouncy Castle for Java before 1.73 contains a potential Denial of Service (DoS) issue within the Bouncy Castle org.bouncycastle.openssl.PEMParser class. This class parses OpenSSL PEM encoded streams containing X.509 certificates, PKCS8 encoded keys, and PKCS7 objects. Parsing a file that has crafted ASN.1 data through the PEMParser causes an OutOfMemoryError, which can enable a denial of service attack. (For users of the FIPS Java API: BC-FJA 1.0.2.3 and earlier are affected; BC-FJA 1.0.2.4 is fixed.)
      * Upgrade to the latest version of Bouncy Castle to address this vulnerability. Design throttling mechanisms into the system architecture. The best protection is to limit the amount of resources that an unauthorized user can cause to be expended. A strong authentication and access control model will help prevent such attacks from occurring in the first place. The login application should be protected against DoS attacks as much as possible. Limiting the database access, perhaps by caching result sets, can help minimize the resources expended. To further limit the potential for a DoS attack, consider tracking the rate of requests received from users and blocking requests that exceed a defined rate threshold.
      * **Attribution:** 
        + <https://nvd.nist.gov/vuln/detail/CVE-2023-33202>
        + <http://cwe.mitre.org/data/definitions/400.html>
    - CVE-2023-33201 (OSSINDEX)
      * Bouncy Castle for Java before 1.74 is affected by an LDAP injection vulnerability. The vulnerability only affects applications that use an LDAP CertStore from Bouncy Castle to validate X.509 certificates. During the certificate validation process, Bouncy Castle inserts the certificate's Subject Name into an LDAP search filter without any escaping, which leads to an LDAP injection vulnerability.
      * Upgrade to the latest version of Bouncy Castle to address this vulnerability. Certificates should be carefully managed and checked to assure that data are encrypted with the intended owner's public key. If certificate pinning is being used, ensure that all relevant properties of the certificate are fully validated before the certificate is pinned, including the hostname.
      * **Attribution:** <https://cwe.mitre.org/data/definitions/295.html>
    - CVE-2020-26939 (OSSINDEX)
      * In Legion of the Bouncy Castle BC before 1.61 and BC-FJA before 1.0.1.2, attackers can obtain sensitive information about a private exponent because of Observable Differences in Behavior to Error Inputs. This occurs in org.bouncycastle.crypto.encodings.OAEPEncoding. Sending invalid ciphertext that decrypts to a short payload in the OAEP Decoder could result in the throwing of an early exception, potentially leaking some information about the private exponent of the RSA private key performing the encryption.
      * Upgrade to the latest version of Bouncy Castle to address this vulnerability. Ensure that error messages only contain minimal details that are useful to the intended audience and no one else. The messages need to strike the balance between being too cryptic (which can confuse users) or being too detailed (which may reveal more than intended). The messages should not reveal the methods that were used to determine the error. Attackers can use detailed information to refine or optimize their original attack, thereby increasing their chances of success. If errors must be captured in some detail, record them in log messages, but consider what could occur if the log messages can be viewed by attackers. Highly sensitive information such as passwords should never be saved to log files. Avoid inconsistent messaging that might accidentally tip off an attacker about internal state, such as whether a user account exists or not.
      * **Attribution:**
        + <https://cwe.mitre.org/data/definitions/203.html>
        + <https://nvd.nist.gov/vuln/detail/CVE-2020-26939>
* **Dependency:** hibernate-validator-6.0.18.Final.jar
  + **Description:** Hibernate's Bean Validation (JSR-380) reference implementation.
  + Vulnerabilities:
    - CVE-2020-10693
      * A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages.
      * Upgrade to version 6.0.20.Final or higher. Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a list of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. When performing input validation, consider all potentially relevant properties, including length, type of input, the full range of acceptable values, missing or extra inputs, syntax, consistency across related fields, and conformance to business rules. As an example of business rule logic, "boat" may be syntactically valid because it only contains alphanumeric characters, but it is not valid if the input is only expected to contain colors such as "red" or "blue." Do not rely exclusively on looking for malicious or malformed inputs. This is likely to miss at least one undesirable input, especially if the code's environment changes. This can give attackers enough room to bypass the intended validation. However, denylists can be useful for detecting potential attacks or determining which inputs are so malformed that they should be rejected outright.
      * **Attribution:** 
        + <https://cwe.mitre.org/data/definitions/20.html>
        + <https://nvd.nist.gov/vuln/detail/CVE-2020-10693>
* **Dependency:** logback-core-1.2.3.jar
  + **Description:** Logback is intended as a successor to the popular log4j project.
  + Vulnerabilities:
    - CVE-2021-42550
      * Logback before version 1.2.10 is vulnerable to a denial of service attack when a large (gigabytes) lookahead is used in PatternLayout.
      * Update logback-core to version 1.2.10 or higher. Make fields transient to protect them from deserialization. An attempt to serialize and then deserialize a class containing transient fields will result in NULLs where the transient data should be. This is an excellent way to prevent time, environment-based, or sensitive variables from being carried over and used improperly.
      * **Attribution:** 
        + <https://cwe.mitre.org/data/definitions/502.html>
        + <https://nvd.nist.gov/vuln/detail/CVE-2023-6378>
* **Dependency:** tomcat-embed-core-9.0.30.jar & tomcat-embed-websocket-9.0.30.jar
  + Description: The Apache Tomcat software is an open source implementation of the Java Servlet, JavaServer Pages, Java Expression Language and Java WebSocket technologies.
  + Vulnerabilities:
    - CVE-2020-13935
      * When using Apache Tomcat versions 10.0.0-M1 to 10.0.0-M4, 9.0.0.M1 to 9.0.30, 8.5.0 to 8.5.51 and 7.0.0 to 7.0.99 if a) an attacker is able to control the contents and name of a file on the server; and b) the server allows HTTP PUT requests to be made to a context mapping which is configured with readonly initialized to false, then the attacker can overwrite any file in the web application directory.
      * **Mitigation:**
        + The payload length in a WebSocket frame was not correctly validated. Invalid payload lengths could trigger an infinite loop. Multiple requests with invalid payload lengths could lead to a denial of service.
        + - Upgrade to Apache Tomcat 10.0.0-M7 or later
        + - Upgrade to Apache Tomcat 9.0.37 or later
        + - Upgrade to Apache Tomcat 8.5.57 or later
      * **Attribution:**
        + <https://nvd.nist.gov/vuln/detail/CVE-2020-13935>
        + <https://lists.apache.org/thread/r7m5zthg1k9grytzqz0cwlnfb7wjfonz>
    - CVE-2020-9484
      * When running Apache Tomcat 9.0.0.M1 to 9.0.30, 8.5.0 to 8.5.50 and 7.0.0 to 7.0.99 on Windows with HTTP PUTs enabled (e.g. via setting the readonly initialisation parameter of a Context to false) it was possible to upload a JSP file to the server via a specially crafted request. This could then be requested and any code it contained would be executed by the server.
      * If:
        + a) an attacker is able to control the contents and name of a file on the server; and
        + b) the server is configured to use the PersistenceManager with a FileStore; and
        + c) the PersistenceManager is configured with sessionAttributeValueClassNameFilter="null" (the default unless a SecurityManager is used) or a sufficiently lax filter to allow the attacker provided object to be deserialized; and d) the attacker knows the relative file path from the storage location used by FileStore to the file the attacker has control over; then, using a specifically crafted request, the attacker will be able to trigger remote code execution via deserialization of the file under their control.
        + Note that all of conditions a) to d) must be true for the attack to succeed.
      * **Mitigation:**
        + - Upgrade to Apache Tomcat 10.0.0-M5 or later
        + - Upgrade to Apache Tomcat 9.0.35 or later
        + - Upgrade to Apache Tomcat 8.5.55 or later
        + - Upgrade to Apache Tomcat 7.0.104 or later
        + Alternatively, users may configure the PersistenceManager with an appropriate value for sessionAttributeValueClassNameFilter to ensure that only application provided attributes are serialized and deserialized.
      * **Attribution:** 
        + <https://nvd.nist.gov/vuln/detail/CVE-2020-9484>
        + <https://lists.apache.org/thread/jgqg9ftb9zwpd0kt7mxb5sj0hfnd4xym>

The following dependencies were identified as false positives in the dependency check report:

* **Dependency:** jackson-databind-2.10.2.jar
  + **Reasoning:** The vulnerabilities reported for this dependency are related to deserialization of untrusted data and resource exhaustion. However, the project does not use these features in a way that would expose it to these vulnerabilities.
* **Dependency:** log4j-api-2.12.1.jar
  + **Reasoning:** The reported vulnerability (CVE-2020-9488) is related to the SMTP appender, which is not used in this project.
* **Dependency:** spring-boot-2.2.4.RELEASE.jar & spring-boot-starter-web-2.2.4.RELEASE.jar
  + **Reasoning:** The vulnerabilities reported for these dependencies are related to specific deployment scenarios and features that are not relevant to this project.
* **Dependency:** spring-core-5.2.3.RELEASE.jar, spring-web-5.2.3.RELEASE.jar, & spring-webmvc-5.2.3.RELEASE.jar
  + **Reasoning:** The vulnerabilities reported for these dependencies are related to specific configurations and features (data binding, SpEL expressions, file uploads) that are not used in a way that would expose the project to these vulnerabilities.

**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.

* **Insecure Dependency (bcprov-jdk15on-1.46.jar):** Update the Bouncy Castle library to the latest version (1.74 or higher). This will address the identified vulnerabilities related to improper certificate validation, CPU consumption, denial of service, LDAP injection, and information exposure.
* **Hardcoded Credentials (DocData.java):** Remove the hardcoded credentials from the code and store them securely using environment variables, configuration files, or a secrets management tool.
* **SQL Injection Vulnerability (DocData.java):** Sanitize user input using prepared statements or parameterized queries. This will prevent attackers from injecting malicious SQL code into the application.
* **Lack of Input Validation (CRUDController.java):** Implement input validation for the name parameter in the /read endpoint. This can be done using regular expressions, whitelisting, or other appropriate validation techniques.
* **Information Exposure (CRUD.java):** Refactor the CRUD class to avoid storing the same input in multiple fields. If both fields are necessary, ensure that only the appropriate field is returned in the response.
* **Unprotected Customer Information (customer.java):** Implement proper authorization and access controls for the showInfo and deposit methods. This could involve using authentication tokens, role-based access control, or other security mechanisms.
* **Insufficient Logging:** Add comprehensive logging mechanisms to the application to track user actions, errors, and exceptions. This will help in identifying and investigating security incidents.
* **Insecure Direct Object Reference (CRUDController.java):** Avoid directly returning domain objects like DocData. Instead, create a data transfer object (DTO) that only exposes the necessary information to the client.
* **Lack of Exception Handling (DocData.java):** Implement proper exception handling in the read\_document method to catch and log any errors that occur during database operations. This will prevent unexpected behavior and potential information leakage.
* **Insufficient Transport Layer Protection:** Enforce HTTPS for all communication between the client and server. This can be done by configuring the server to redirect HTTP requests to HTTPS and obtaining a valid SSL/TLS certificate.
* **Denial of Service (logback-core-1.2.3.jar):** Update logback-core to version 1.2.10 or higher.
* **Remote Code Execution (tomcat-embed-core-9.0.30.jar):** Update Apache Tomcat to the latest version.
* **Improper Input Validation (hibernate-validator-6.0.18.Final.jar):** Upgrade to version 6.0.20.Final or higher.