# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **5/24/2024** | **Brandon Boggs** |  |

## 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

Brandon Boggs

**1. Interpreting Client Needs**

Artemis Financial has a critical need for secure communications due to the highly sensitive nature of the business in dealing with client information regarding their financials, including their transactions, personal retirement plans, savings, investments, and insurance. Artemis works with clients domestically and internationally; therefore, it is important to take into consideration the governing laws regarding data privacy and secure communications as well as international regulations. Artemis must consider government restrictions such as the Data Protection Act in the EU, the Safe Harbor Act in the US, and Health Insurance Portability and Accountability when dealing with the healthcare industry when considering international transactions and cooperation. External threats may include various Injection Style attacks such as SQL, XML, and JSON injection. These threats will aim to access clients’ sensitive financial information or personal information to perform various malicious actions. Query parameterization is an essential step in helping to defend against injection, as well as rigorous and intuitive input validation and error handling. To aid in ensuring secure implementation of the system, Artemis should utilize both Static and Dynamic Analysis, as well as the OWASP Dependency Check when considering modernization requirements and open-source libraries.

**2. Areas of Security**

* **Input Validation**
  + Artemis must implement strict input validation to ensure that the program only accepts valid inputs and does not allow malicious characters to enter the program that aim to discover sensitive information regarding the database. Leveraging APIs or regular expressions helps to protect potential areas of risk such as password length, character requirements, either numeric or alphanumeric, special character usage, and other input constraints. Data validation should:
    - Validate all client provided data before processing, including all form fields, URLs and HTTP header values (Data Validation – OWASP Cheat Sheat Series n.d.).
    - Validate data range and length (Data Validation – OWASP Cheat Sheat Series n.d.).
    - Validate all input against a “white” list of allowed characters (Data Validation – OWASP Cheat Sheat Series n.d.).
* **APIs**
  + The implementation of APIs needs to be validated to ensure there are no vulnerabilities. One tool to leverage to assist with this is the OWASP Dependency Check, which can check for vulnerabilities within the code. APIs need to be wrapped within a central security layer to ensure outside access is managed.
* **Cryptography**
  + All data transmitted to and from the web must be encrypted using the HTTPS protocol to protect the sensitive data from being stolen in transit by potential attackers.
* **Client/Server**
  + Sufficient access control and permissions must be assigned to ensure that the role-based security follows the method of least privilege, each user has as little access as needed to perform their desired tasks. Rate limits should be implemented to ensure that a user is not requesting massive amounts of data within a small time period for potentially harmful reasons.
* **Code Error**
  + It is imperative that system logging because of an error does not provide sensitive information to the user and potentially exposes internal information regarding either the structure or the contents of the program. Error handling and checks should be implemented to provide only a basic error message to the user, while logging detailed information internally to be reviewed and corrected.
* **Encapsulation**
  + By encapsulating the program with a central layer of security, errors are handled in an organized manner and code can be easily corrected. Access can be restricted to sensitive objects and their variables while also allowing functionality access.

**3. Manual Review**

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

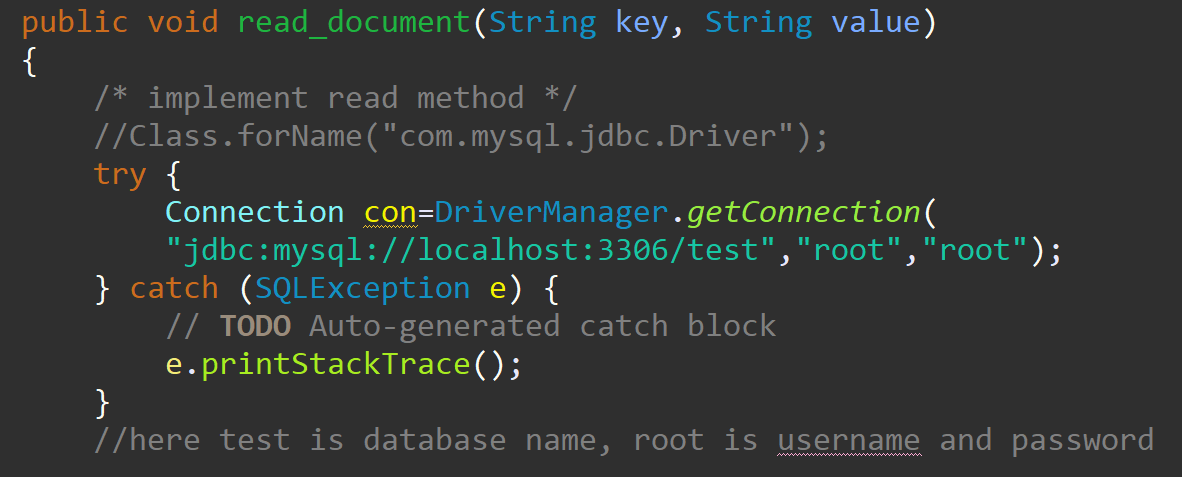
The architecture of the program looks like this:

A screenshot of a computer

Description automatically generated The main entry point for the project is within the RestServiceApplication.java. Within this file is the “boot” for the Spring application framework. In the customer.java file, there is code that handles customer information such as displaying the account number and allowing for a deposit. Within this code file there is a major security issue. There is no input validation within the deposit function which could allow for an attacker to enter a negative number for the variable “a”, and potentially withdraw funds from a user’s account. The screenshot for this code in concern is below:

A screen shot of a computer

Description automatically generatedIn DocData.java, there are functions that control access to the database. Particularly within the read\_document(String key, String value) function. In this function, a try-catch statement is utilized to check that the server being accessed is from within the system. Data validation, input validation, and encryption would need to be utilized here to ensure that the error message printing to stack trace is not displayed to the user from error. The user should not be able to enter an invalid key and gain access to the database.



The CRUDController.java file contains code that returns data directly as an HTTP response. HTTPS should be leveraged to transmit sensitive data as HTTP is not a secure means of data transmission across the web. Authentication should be implemented to ensure that the business requesting data is actually the one entering the name. Input validation needs to be utilized to ensure proper characters are being accepted as well as system logging to track errors.

A screen shot of a computer code

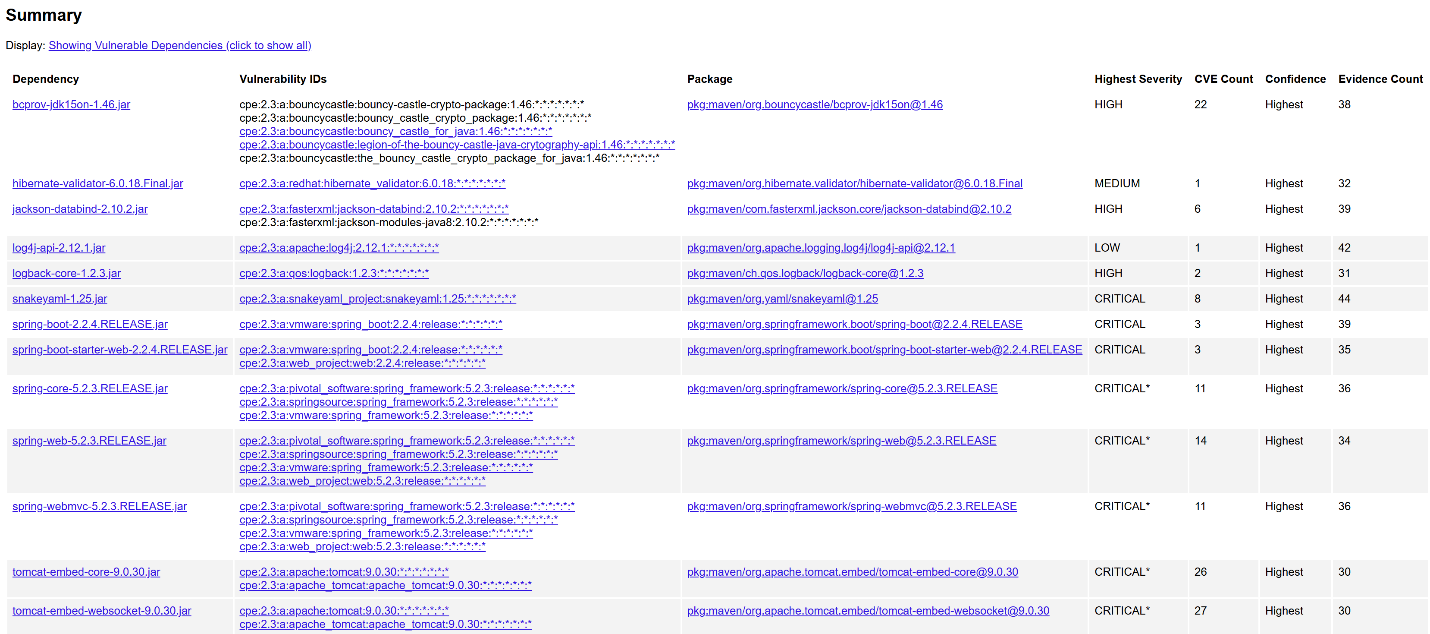
Description automatically generated

**4. Static Testing**

## Project: rest-service

#### com.twk:rest-service:0.0.1-SNAPSHOT

Scan Information ([show all](file:///C:\Users\bbogg\eclipse-workspace\rest-service\target\dependency-check-report.html)):

* *dependency-check version*: 9.2.0
* *Report Generated On*: Fri, 24 May 2024 10:00:10 -0400
* *Dependencies Scanned*: 38 (22 unique)
* *Vulnerable Dependencies*: 13
* *Vulnerabilities Found*: 135
* *Vulnerabilities Suppressed*: 0
* ...
* **bcprov-jdk15on-1.46.jar**
  + **Cve-2024-34447: bouncycastle – Improper Validation of Certificate with Host Mismatch**
  + **The software communicates with a host that provides a certificate, but the software does not properly ensure that the certificate is actually associated with that host.**
  + **Phase: Implementation**
  + **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.**
* **Hibernate-validator-6.0.18.Final.jar**
  + **CVE-2020-10693**
  + **A flaw was found in Hibernate Validator version 6.1.2.Final. 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.**
  + **CWE-20 Improper Input Validation**
  + **Use an input validation framework such as Struts or the OWASP ESAPI Validation API. Note that using a framework does not automatically address all input validation problems; be mindful of weaknesses that could arise from misusing the framework itself (CWE-1173).**
* **Logback-core-1.2.3.jar**
  + **CVE-2023-6378**
  + **A serialization vulnerability in logback receiver component part of**
  + **logback version 1.4.11 allows an attacker to mount a Denial-Of-Service**
  + **attack by sending poisoned data.**
  + **CWE-502 Deserialization of Untrusted Data**
  + **When deserializing data, populate a new object rather than just deserializing. The result is that the data flows through safe input validation and that the functions are safe.**
* **Snakeyaml-1.25.jar**
  + **CVE-2022-1471**
  + **SnakeYaml's Constructor() class does not restrict types which can be instantiated during deserialization.��Deserializing yaml content provided by an attacker can lead to remote code execution. We recommend using SnakeYaml's SafeConsturctor when parsing untrusted content to restrict deserialization. We recommend upgrading to version 2.0 and beyond.**
  + **CWE-502 Deserialization of Untrusted Data, CWE-20 Improper Input Validation**
  + **If available, use the signing/sealing features of the programming language to assure that deserialized data has not been tainted. For example, a hash-based message authentication code (HMAC) could be used to ensure that data has not been modified.**
* **Spring-boot-2.2.4.RELEASE.jar**
  + **CVE-2023-20873**
  + **In Spring Boot versions 3.0.0 - 3.0.5, 2.7.0 - 2.7.10, and older unsupported versions, an application that is deployed to Cloud Foundry could be susceptible to a security bypass. Users of affected versions should apply the following mitigation: 3.0.x users should upgrade to 3.0.6+. 2.7.x users should upgrade to 2.7.11+. Users of older, unsupported versions should upgrade to 3.0.6+ or 2.7.11+.**
  + **NVD-CWE-noinfo**
* **Tomcat-embed-core-9.0.30.jar**
  + **CVE-2020-1938 suppress**
  + **CISA Known Exploited Vulnerability:**
  + **Product: Apache Tomcat**
  + **Name: Apache Tomcat Improper Privilege Management Vulnerability**
  + **Date Added: 2022-03-03**
  + **Description: Apache Tomcat treats Apache JServ Protocol (AJP) connections as having higher trust than, for example, a similar HTTP connection. If such connections are available to an attacker, they can be exploited.**
  + **Required Action: Apply updates per vendor instructions.**
  + **Due Date: 2022-03-17**

**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.

Based on the results of the manual review and the static testing report, input validation needs to be properly implemented on a critical level. Currently, there are many ways for an attacker to gain access to the system because of a lack of input validation. Also, the outdated software needs to be updated to the current version to ensure that all exposed threats that were found in the old versions are eradicated. There were also instances where data being transmitted from a user to the system could be potentially poisonous and could allow potential DOS attacks. This is also a result of the lack of input validation utilized within the codebase. A critical step is to handle error log backs to the user, action needs to be taken to ensure that error stacks are not being printed via HTTP to the user and they are instead logged internally within the system while printing basic error messages to the user so the architecture of the program and the contents of the database are not being leaked to potential attackers.