CS 305 Project One Template

Document Revision History

Version	Date	Author	Comments
1.2	03/21/2025	Ronny Z. Valtonen	Initial 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
Ronny Z. Valtonen

1. Interpreting Client Needs

After reviewing the plans made by Artemis Financial, this would involve transactions that would require secure communications. Financial data that is being transmitted must be sent over encrypted channels (SSL/TLS) which would prevent man in the middle attacks. It could also be noted cryptography techniques such as Symmetric cryptography could be used to scramble information that is being transmitted, such that only the authorized users who own a secret key can unscramble it. This would make it virtually impossible to gain unauthorized access to sensitive data. Furthermore, there are international transactions meaning that data localization based on compliance measures must be considered. External threats that the project may face is "Cross-Site Request Forgery" (CSRF) as explained by the Iron-Clad Java textbook.[1] We must ensure that any website does not make fraudulent requests without the user knowing it. Secondly, unauthorized access must be prevented, this could be done by using something like multi-factor authentication. It is also a must to ensure that data at rest is secured using a toolkit like Keyczar which allows easy encryption by key management functionality. When considering modernization requirements, use of open-source libraries allows for quicker updates and more eyes on potential vulnerabilities. Lastly, it must be ensured that we continuously update dependencies and secure integrations to ensure emerging vulnerabilities are mitigated.

2. Areas of Security

When referring to the vulnerability assessment process flow diagram, there are four *main* areas of security that are applicable to Artemis' web application.

- **Input validation** is going to be top priority in a web application involving extremely private financial data. User input needs to be validated and sanitized to ensure injection attacks are prevented. Below in the manual review I have discussed in detail which areas of the web application are vulnerable, such as the GreetingController.java and CRUDController.java files.
- Secondly, **code quality** and secure coding practices needs to be heavily reviewed - there are some areas of hard-coded credentials. Having these design flaws could expose the web applications to unauthorized access; further discussed in detail below.
- Thirdly, **cryptography** must be secure as involving this type of data between user's and the website as well as international calls is a huge risk but has mitigation factors. This would include TLS/SSL encryption protocols during data transit as well as strong authentication.
    - It should be noted that secure API interactions - including token based authentication should be explored separately, here I have put it under cryptography. Though equally important, it is further discussed below.
    - Secondly, both data at-rest as well as data in transit both need to be protected in some way to ensure the full-lifecycle protection of sensitive data.
- To further discuss, following good practice of object-oriented principles is important for the **encapsulation** area. Restricting access to an object's internal states will be critical for protecting

[1] Manico, Jim, and August Detlefsen. *Iron-Clad Java: Building Secure Web Applications (Oracle Press)*. eBook. O'Reilly, 2014. http://dl.acm.org/citation.cfm?id=2826076.
Chpt.5

at rest data. If data is not correctly encapsulated, an attacker could access data from another area of the web-program. This also means that when performing a code review, it is far easier to review code that follows good coding practices.

- Lastly, ensuring secure **error handling** is a strong area of security that must be explored. As discussed in code quality, having design flaws could expose flaws, like using "*e.printStackTrace()*" which exposes the internal workings of the function and should instead be fed into a log. The current error could include class names, method names and line numbers which the attacker can use to map out the code structure.

3. Manual Review

1.) *Working through the vulnerability assessment process flow diagram, input validation is the first vulnerability to review.*

In CRUDController.java as well as GreetingController.java, the "name" and "business_name" are not validated. Meaning that an attacker can input code into the name and business names which could potentially raise injection attacks.

2.) *As previously mentioned, there are hardcoded credentials located in the DocData.java file.*

Many routers come with a default username/password as "Admin/Password" as the credentials which provides full access to the router and all connected devices. If someone were to guess the hardcoded "root" for the credentials, or because it is a *Public* function; someone could access the pre-built credentials details to gain unauthorized database access. Credentials should always be externalized like using environment variables to limit and reduce attacks.

3.) Secure error handling is an important part of secure coding practices.

In the DocData.java the error exception of "*e.printStackTrace()*" could expose details of the internal workings of the architecture like file names, line numbers of the code as well as method calls. The method call potential is what makes this very vulnerable due to other functions not being properly encapsulated and being public, like the function itself that contains the credentials.

*4.) The code quality for secure coding practices is a vulnerability. There are outdated dependencies within the pom.xml which is further discussed in the **Static Testing** part of this code review.*

In total, there were 159 vulnerabilities found with 15 of those being vulnerable dependencies. As one example, the **bcprov-jdk15on-1.46.jar** "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."

5.) *Lack of authentication and authorization*

There are no authentication or authorization patterns for API endpoints. The *CRUDController.java* and *GreetingController.java* have public endpoints of "/read" and "/greeting." These would need authentication checks to only authorize the correct users.

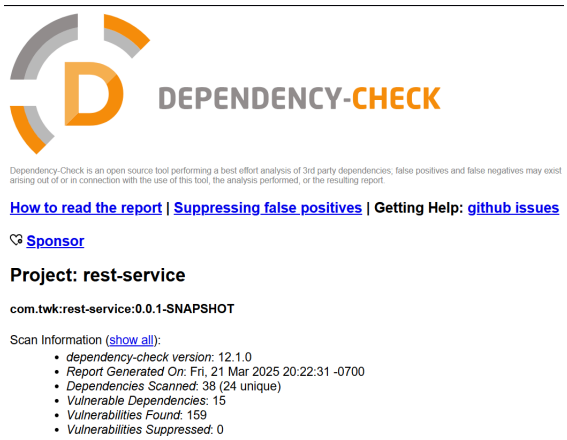
6.) Open database connection

The method "read_document" in the DocData.java opens up a database connection to "`jdbc:mysql://localhost:3306/test`" which utilizes the user-supplied inputs for queries meaning that it is currently vulnerable to injection attacks. These inputs could need to be sanitized for only authorized input.

7.) Throttling

As discussed in Iron-Clad Java, inadequate validation and verbose error reporting can lead to resource exhaustion and make the software open to DDos attacks. The CRUDController and GreetingController do not limit inputs, meaning an attacker could continuously send inputs into the software, flooding the endpoints with requests.

4. Static Testing



Here we see that there are 15 vulnerable dependencies. Below I have listed them with a brief description and recommended solution provided by the report.

1.) Bouncy Castle Crypto (CVE-2024-34447)

This is a Java implementation of cryptographic algorithms which contains the JCE provider for lightweight API cryptography.

"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."

The recommended solution is to update the package to the current up-to-date version.

2.) Hibernate's Validator (CVE-2023-1932)

There was no description of this package provided.

"A flaw was found in hibernate-validator's 'isValid' method in the org.hibernate.validator.internal.constraintvalidators.hv.SafeHtmlValidator class, which can be bypassed by omitting the tag ending in a less-than character. Browsers may render an invalid html, allowing HTML injection or Cross-Site-Scripting (XSS) attacks."

It is recommended to update this package.

3.) Jackson Databind (CVE-202-25649)

This is a general data-binding functionality for the Jackson package.

"A flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity."

It is recommended to update this package.

4.) Apache Log4j API (CVE-2020-9488)

"Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender. Fixed in Apache Log4j 2.12.3 and 2.13.1"

It is recommended to upgrade to 2.12.3 or 2.13.1.

5.) Logback Classic Module (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."

6.) Snake Yaml (CVE-2022-1471)

YAML 1.1 parser and emitter for Java

"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."

Update to version 2.0 or newer.

7.) Spring Boot (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+."

Since we are using version 2.2.4 which is an unsupported version, we should update to 3.0.6+ or 2.7.11+.

8.) Spring Boot Starter

Starter for building web, restful applications using Spring MVC.

"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+."

Upgrade to 3.0.6+ or 2.7.11+.

All Spring based dependencies need to be upgraded.

9.) Tomcat (CVE-2020-1938)

"When using the Apache JServ Protocol (AJP), care must be taken when trusting incoming connections to Apache Tomcat. Tomcat treats AJP connections as having higher trust than, for example, a similar HTTP connection."

Update to 9.0.31, may or may not need small changes to configurations.

All TomCat based dependency need to be upgrades

5. Mitigation Plan

After interpreting the results from the manual review and static testing report, the very first step to mitigate security vulnerabilities for Artemis Financial's software application is to **update all of the vulnerable dependencies** to the latest version that works with the current software application. Next it is necessary to implement **input validation and sanitization** for any incoming data for rest endpoints, this is to mitigate injection attacks. As discussed previously, CRUDController.java as well as GreetingController.java need to be fixed by ensuring the "name" and "business_name" are sanitized by making sure only valid characters are input to prevent fraudulent code from being run. Next it is important to make sure **hard-coded credentials** to the database are stored outside of the function such as using environmental variables. This will prevent attackers from seeing error codes that link to the function that stores the credentials. To stem from this, the same function uses "e.printStackTrac()" which **throws out a vulnerable error** which attackers could possibly use to reveal sensitive data within that function. Using a proper logging framework like Log4j can log the exception instead of giving it straight to the attacker. Next it is also important to follow secure coding standards like encapsulation to ensure vulnerable data is only accessible within the function. Like "customer" can be renamed to

"Customer" to make it a Private method and then using access modifiers to allow specific functions to access the data within. Lastly, implementing token-based authentication would ensure only authorized users can access API endpoints, such as using two-factor or using s key to verify like Keyczar provided by Iron-Clad Java in chapter 6, "Protecting Sensitive Data."[2] Using these mitigations the software application should be much protected by attackers.

[2] Manico, Jim, and August Detlefsen. *Iron-Clad Java: Building Secure Web Applications (Oracle Press)*. eBook. O'Reilly, 2014. http://dl.acm.org/citation.cfm?id=2826076.
Chpt.6