****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 5](#_Toc33111307)

[3. Deploy Cipher 5](#_Toc33111308)

[4. Secure Communications 5-6](#_Toc33111309)

[5. Secondary Testing 7-8](#_Toc33111310)

[6. Functional Testing 8-9](#_Toc33111311)

[7. Summary 10](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/3/2021** | **Elizabeth Hodgman** | **Algorithm Cipher, Certificate Generation** |
| **2.0** | **12/6/2021** | **Elizabeth Hodgman** | **Deploy Cipher, Secure Communications, Secondary Testing, Functional Testing** |
| **3.0** | **12/7/2021** | **Elizabeth Hodgman** | **Summary** |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

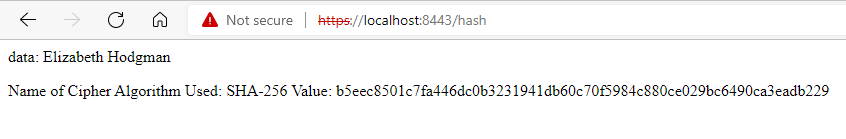
Elizabeth Hodgman

## 1. Algorithm Cipher

Artemis Financial is looking for an algorithm cipher that can ensure the integrity of their files. So, using an algorithm that can accurately compare the file’s encrypted value to a previously calculated value is a must. If the values match, the file is safe. If the values don’t match, then the file has been corrupted or modified. Since collisions can result in false positives, making it seem like the file has not been tampered with, an algorithm with a very low collision rate is needed. So, SHA-256 would be a great option to build this system. The National Institute of Standards and Technology (NIST) also recommends using this algorithm, instead of the previous hashing standards, MD5 and SHA-1. SHA-256 takes the file’s contents and generates a value made up of randomized characters. These characters make it harder for a hacker to figure out the contents. SHA-256 has possible values. Thus, making it nearly impossible for it to generate the same value, or a collision, for different files. Additionally, there has not been any collisions found in the history of using SHA-256. Unlike the previous standard hashing algorithms, such as MD5 and SHA-1, SHA-256, has yet to be compromised. As stated, SHA-1, the predecessor of SHA-2, would not be a good fit because it was found to have a lot of vulnerabilities. One issue is that SHA-1 only generates a 160-bit hash, versus SHA-256’s 256-bit hash. Due to collisions being found, SHA-1 is no longer seen as “secure.” Additionally, MD5, a message digest hash function, only produces a 128-bit hash. Therefore, due to being highly susceptible to collisions, MD5 is also not secure to use. Unlike MD5 and SHA-1, with no vulnerabilities found, SHA-256 is still considered cryptographically secure today.

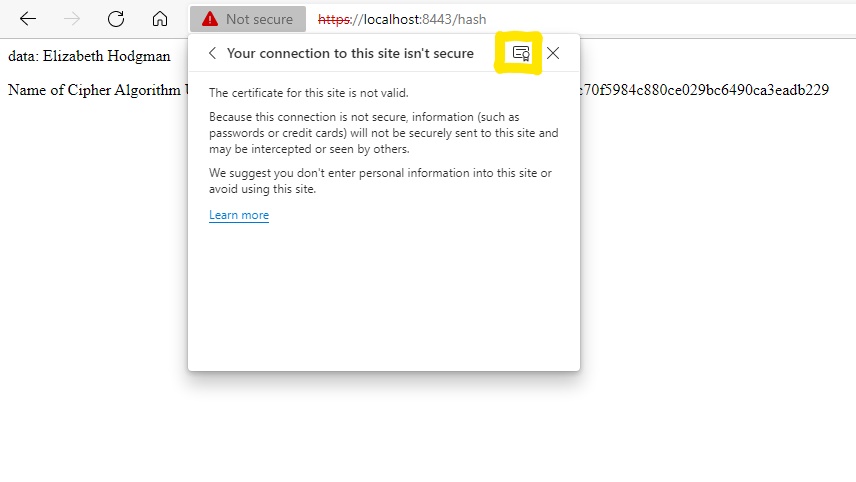
## 2. Certificate GenerationText Description automatically generated

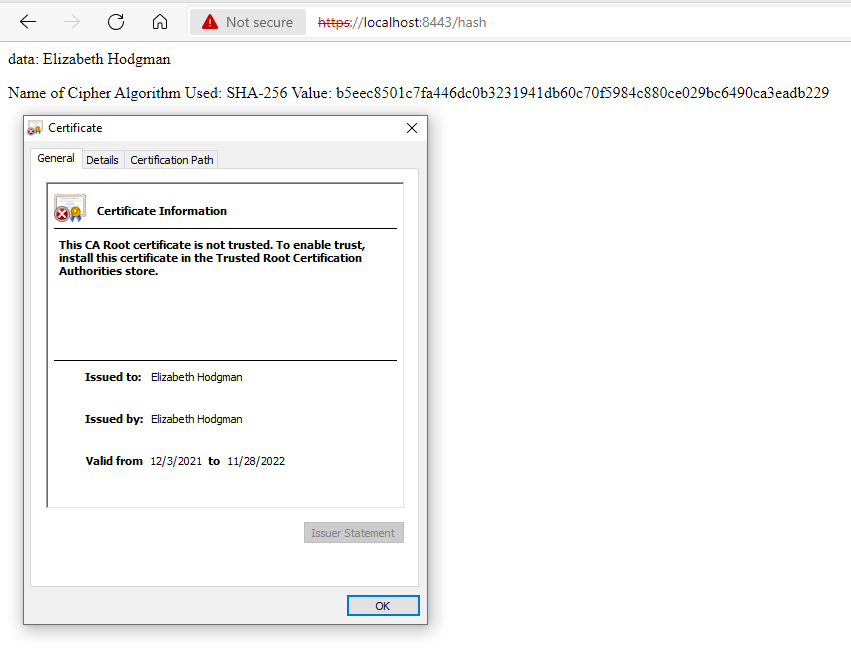
## 3. Deploy Cipher



## 4. Secure Communications

For an HTTPS to become trusted and fully secure, the generated certificate would need to be sent to a Certificate Authorities (CA) to be verified. Since we are connecting an unverified, self-signed, certificate to this program, the HTTPS will not show that it is secure. In this screen shot, it shows that the HTTPS is still not secure, but the certificate is properly connected to the program. When “Not secure” is clicked on, it shows an icon that a certificate is associated with the URL (top screenshot, outlined in yellow). Then, if the icon is clicked, the certificate information is opened, as shown in the bottom screen shot. Again, if that certificate were to become verified by a CA, the HTTPS would show it is secure. Also, in the application.properties file, I added security.require-ssl=true. This line only enables HTTPS for Spring Boot and only allows communications through SSL secured links.

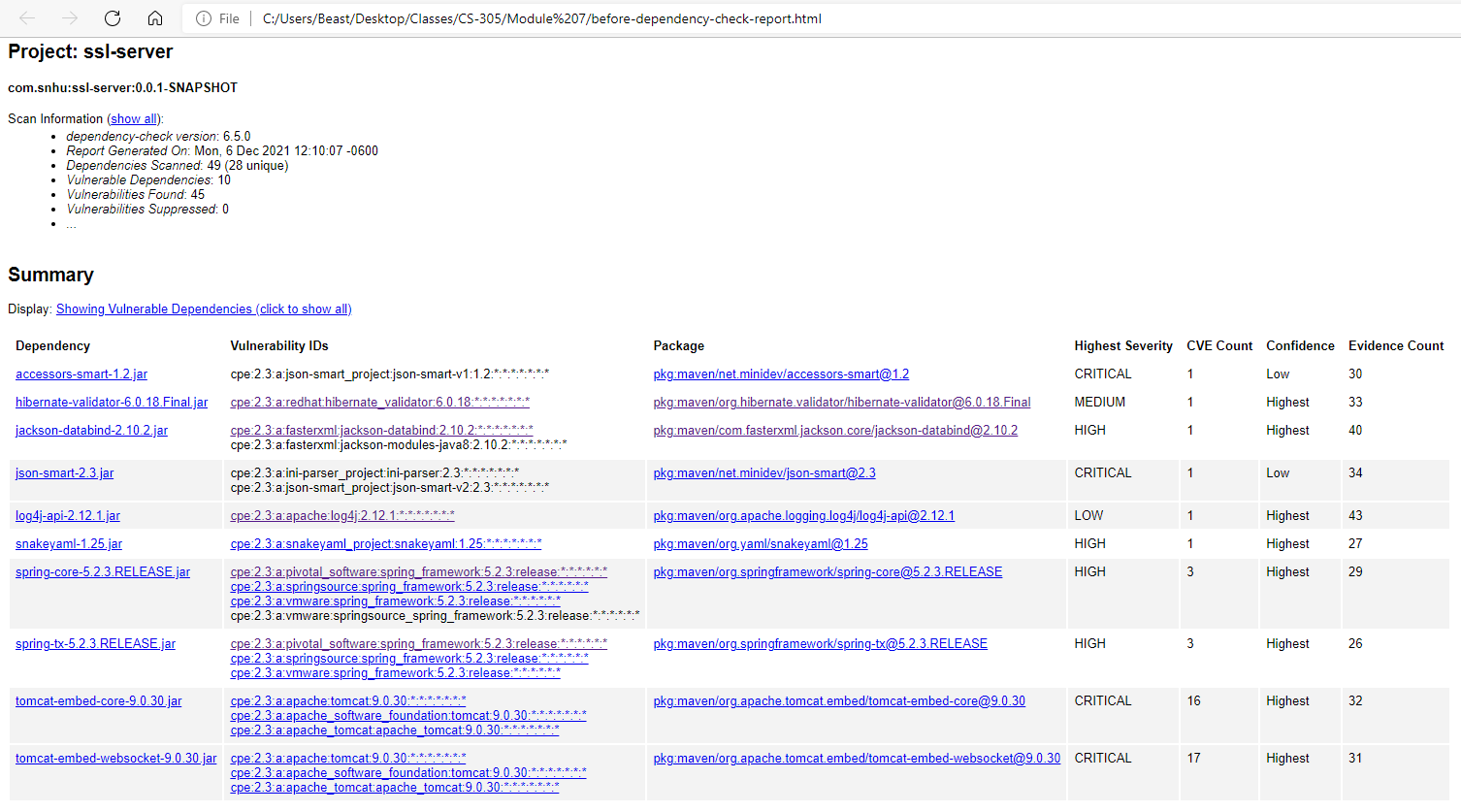




## 5. Secondary Testing

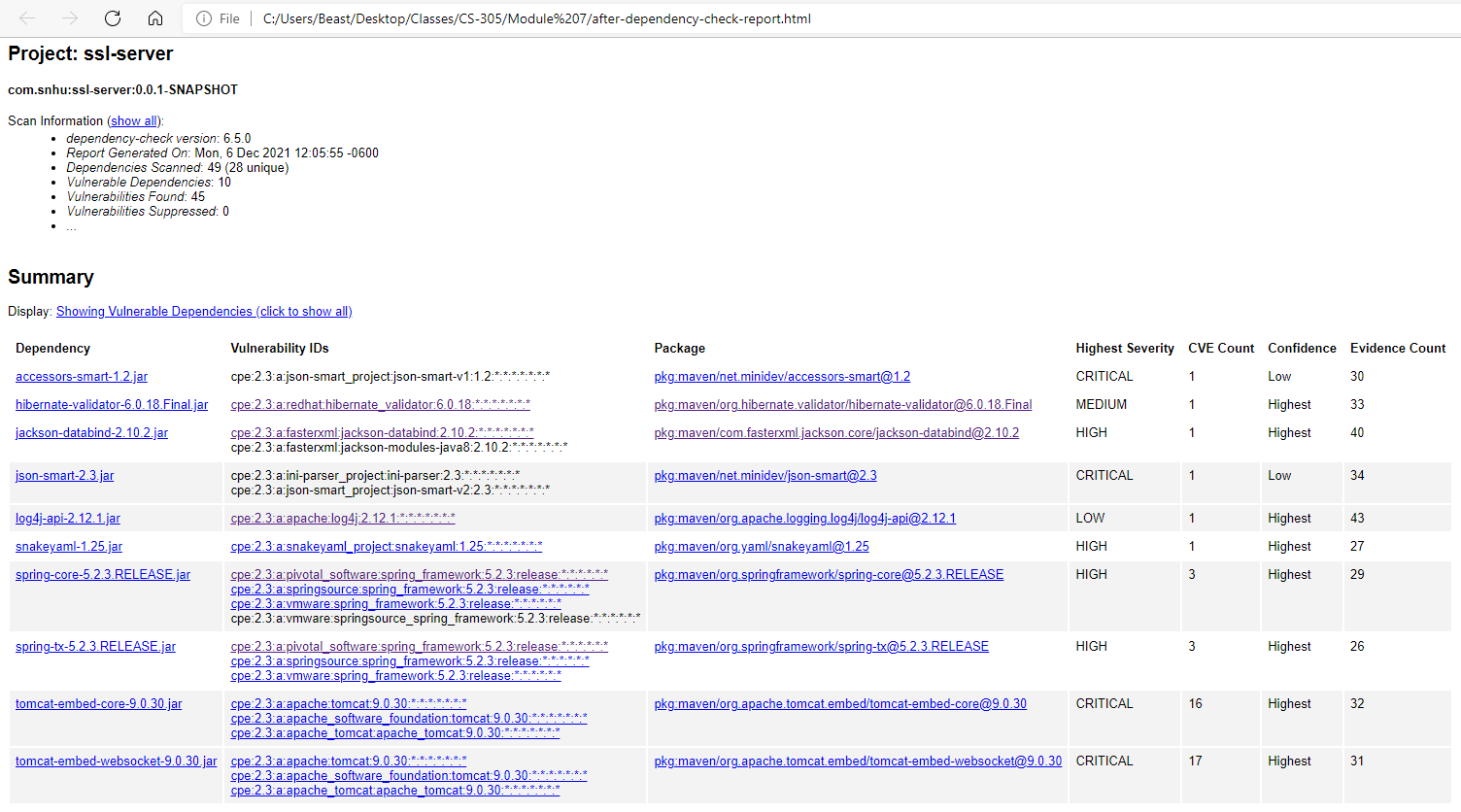
I ran a dependency check before adding any new code to the SslServerApplication.java file. Then, I ran a second check after adding the new code. Looking at the dependency checks before and after refactoring the code, they both have the same output. Thus, the code that I added did not introduce any new vulnerabilities.

**Dependency Check Before Refactored Code:**

****

**Refactored Code:**

**Dependency Check After Refactored Code:**

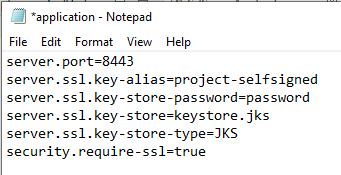
****

## 6. Functional Testing

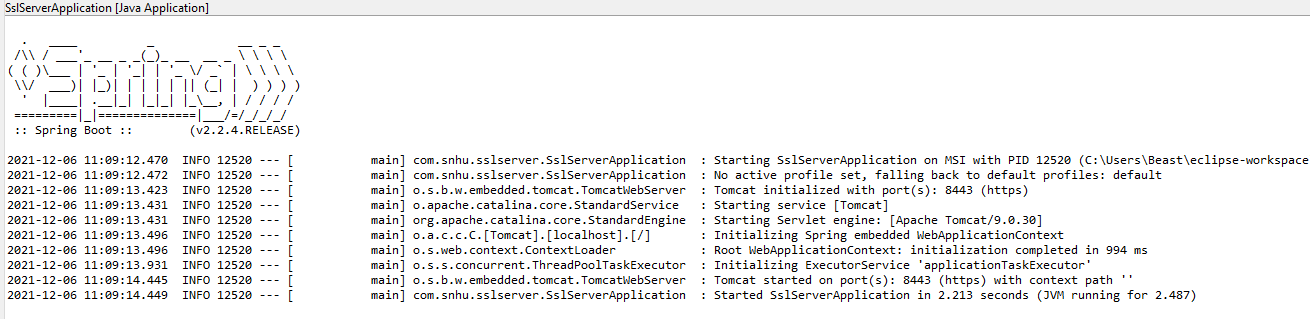
To allow the code to function securely, there were a few things that had to get refactored. First, I made the self-signed security certificate. Generating the certificate allowed me to move the keystore into the project and utilize the certificate to secure the program. Once that was completed, I had to edit the application.properties file so the program could connect to the browser properly. Without editing the application.properties file, the program could not run. After getting the application.properties file set up, I started adding the hashing algorithm cipher. The Code Execution image shows that the application booted up without any errors.

**Refactored Code:**

**Refactored Application File:**



**Code Execution:**



## 7. Summary

When building this program, I addressed API Security, Cryptography, Client/Server, and Code Quality in my refactored code. First, to secure the API, I generated a self-signed certificate that is connected to the HTTPS. The certificate encrypts the data using SHA-256 hashing with RSA. This security prevents hackers from retrieving data in transit. Then, the certificate uses a 2048-bit RSA public key to encrypt data that is being sent. Though, without sending the self-signed certificate to a Certificate Authority (CA), the HTTPS is not completely trusted. If a CA were to authorize the certificate, the HTTPS would show that the connection is secure. Furthermore, if the certificate were to become CA certified, Artemis Financial’s clients would be confident that their information is encrypted and safe. Which would make the company more trustworthy. Also, the SSL requirement was another security layer that was added. This layer ensures secure communications with the program. Securing the API also ties into the Client/Server security as well. Everything sent between the client’s computer and the server will be encrypted using SHA-256 with RSA. Next, addressing cryptography, I developed a SHA-256 hashing cipher to encrypt any data that is inputted into the program. With this example, I used my name as the inputted data. Then, my name was converted into an encrypted hash value. Though, in Artemis Financial’s case, they could input a file. If the file’s hash value matches with a previously calculated value, then it would verify the file’s integrity. Data encryption is a layer of security that helps to prevent sensitive information to be accessed, stolen, or modified. Lastly, I made sure that my cipher code was properly built and working. This allowed me to verify that the code I developed was of good quality.

To maintain this program’s current security, it is important to occasionally run dependency checks. As technology advances, so does the knowledge of hackers. Malicious users will always push to find vulnerabilities within dependencies. Thus, the program should always have the most secure dependencies available. Running the dependency checks will show if any bugs were found after development that need to be mitigated. Using up-to-date dependencies lessens the likelihood of hackers gaining access to vulnerable areas of the program. Also, the program’s security would benefit from adding input validation and error handling. If these areas of security were developed, it would aid in avoiding SQL injecting attacks and data leakage.