****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[\_Toc33111301](#_Toc33111301)

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 5](#_Toc33111308)

[4. Secure Communications 6](#_Toc33111309)

[5. Secondary Testing 6](#_Toc33111310)

[6. Functional Testing 8](#_Toc33111311)

[7. Summary 9](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **20 Feb 2022** | **Thomas Brown** | **Initial Release** |

## 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

Thomas Brown

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

It is recommended to use a SHA (Secure Hash Algorithm), which have been constructed using the best known designs for avoiding collisions. The latest member of the SHA family of standards is SHA-3. However, SHA-256 (Which is under SHA-2) continues to be a known secure algorithm cipher.

SHA-256 is a one way cryptographic function, and is fixed in size at 256 bits (32 bytes). This helps mask the original length of the text being encrypted. SHA-256 has a non-zero chance of a collision, but with 2256 possible hash values, it is infinitesimally unlikely that a collision will occur. This hashing function is also a requirement by the US government to protect certain information.

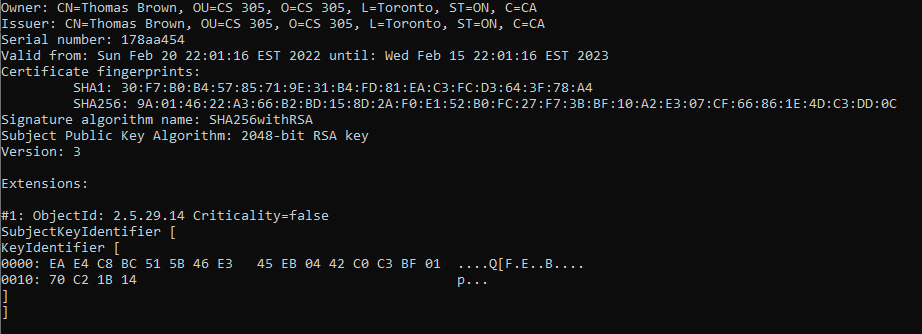
Random numbers are used in the encryption process to help protect the encryption algorithm from becoming predictable and therefore easier to break. When using algorithmic ciphers, a key is used to encrypt and decrypt the information. A symmetric key is a key that can be used to both encrypt and decrypt the data. Where a non-symmetric key can only do one action or the other, and requires at least two keys; one for encrypting and another for decrypting.

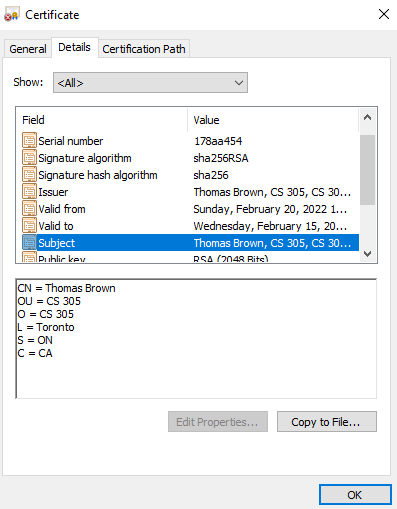
The history of encryption algorithms is relatively interesting, but in a broad overview it started with simple ciphers. Such as swapping particular numbers and letters, or shifting the alphabet by a certain number of letters. This way the original message was obscured, but it was also relatively easy to decrypt. Over time things have become more complicated, such as using more advanced methods for swapping characters, and inserting additional characters into a message to obscure its actual length. This makes it more challenging to decrypt without the key.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

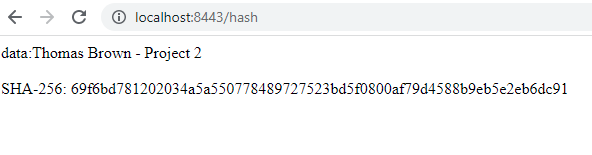




## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

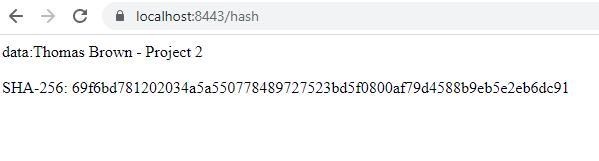
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

* Insert a screenshot below of the web browser that shows a secure webpage.

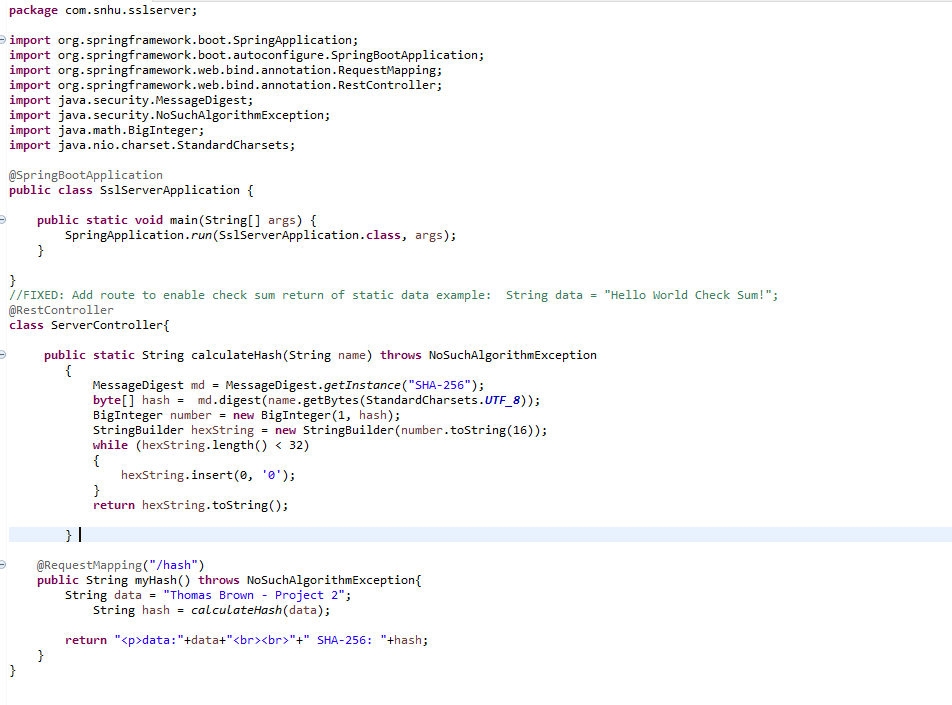


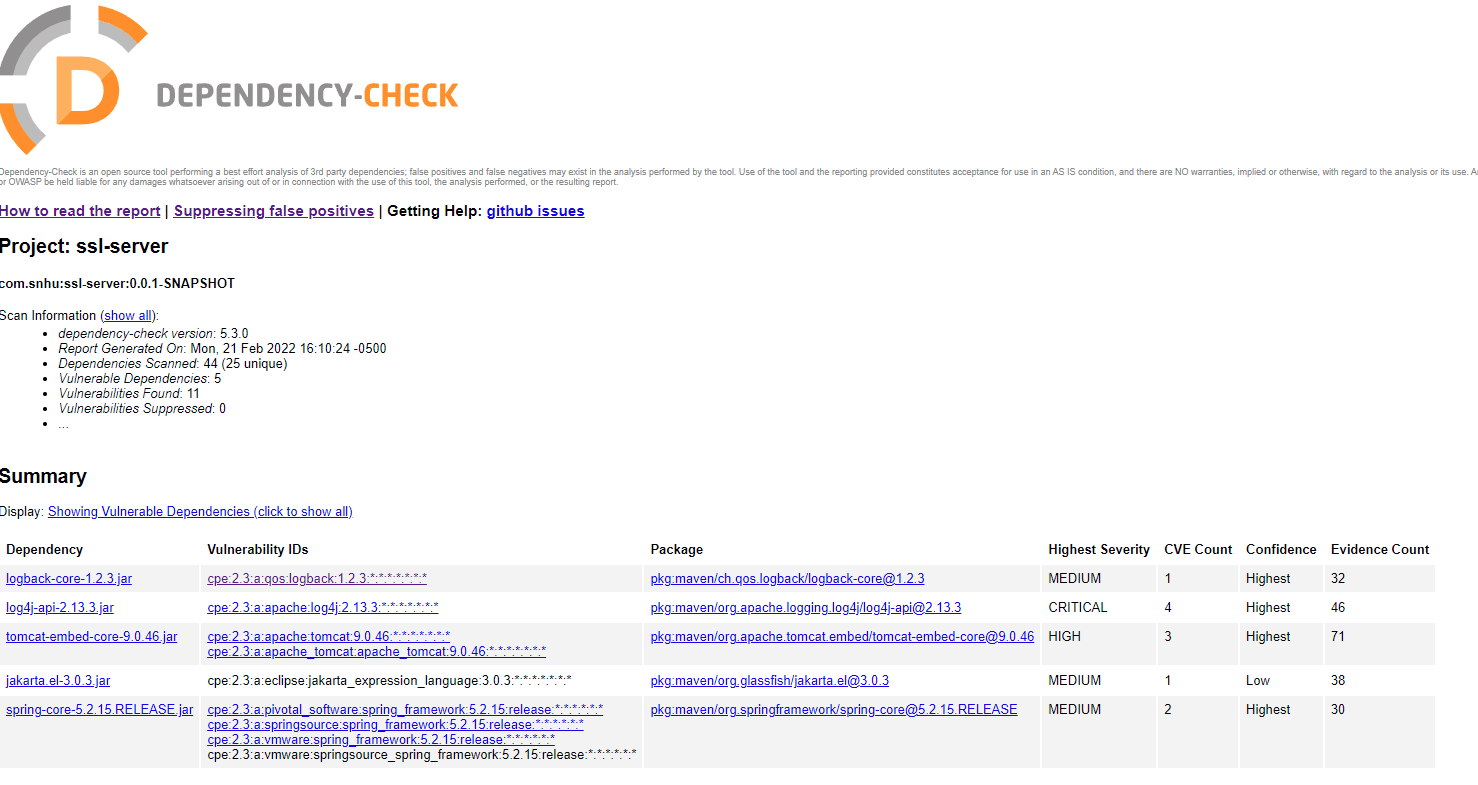
## 5. Secondary Testing

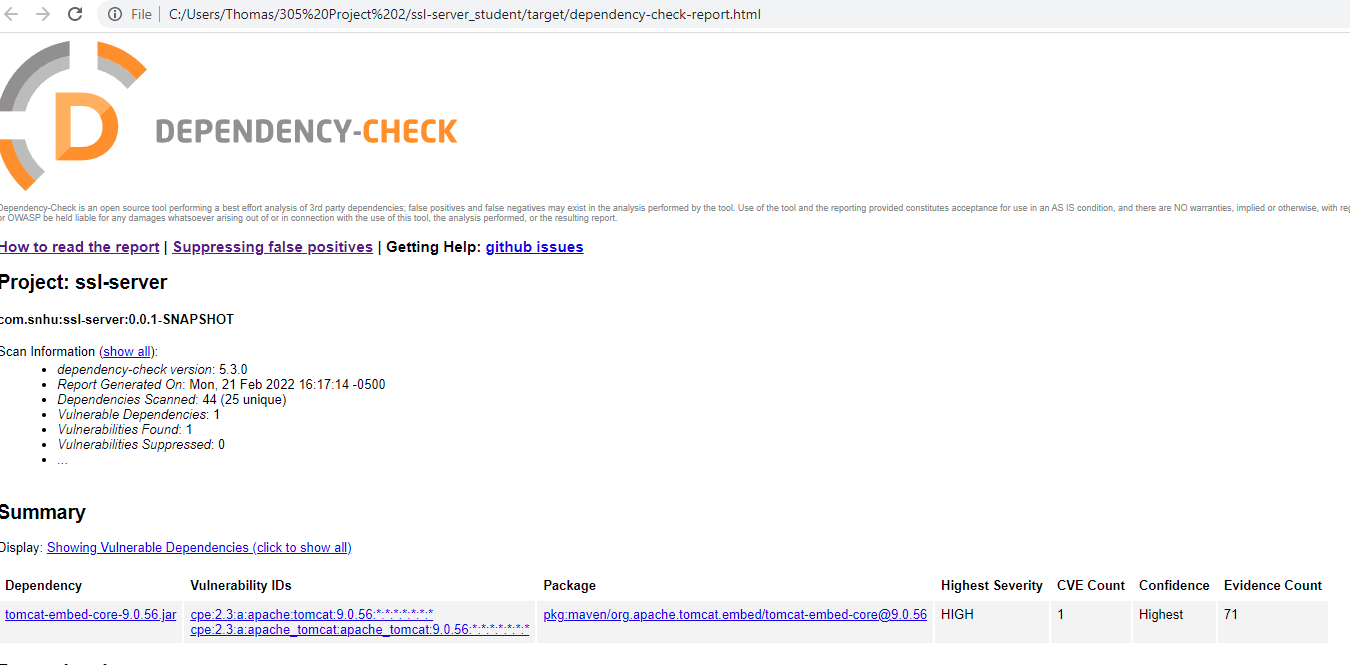
Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

The screenshots below show the refactored code without error, and a screenshot of the dependency check report. In the pom.xml file, the version for the springframework.boot was changed to the latest release of 2.3.12.RELEASE in order to avoid the majority of vulnerabilities that occur from out of date versions. The original report using the old version had 32 vulnerabilities found, while the updated version only had 11 vulnerabilities. A review of the vulnerabilities listed confirmed that updating to a more recent version should eliminate the issues listed. This was confirmed after updating the version and rerunning the check using version 2.6.3 (the latest build), which revealed a single vulnerability.



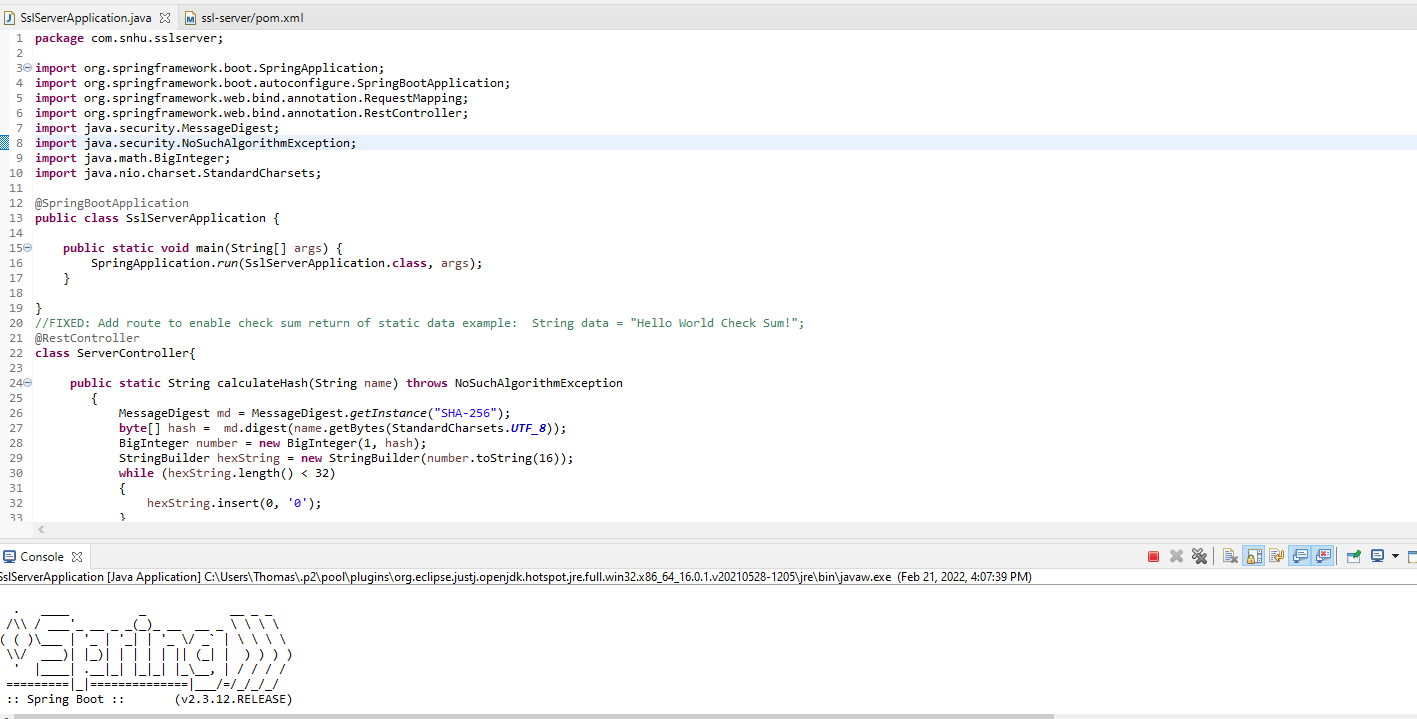




## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.



The code was executed without error. There are minimal risks with this code. Assuming the code is updated to make use of user input, some verification would need to be performed prior to applying the hash. Direct use of a string is not typically recommended. In the application.properties file, there is also a concern with having the data be readable. As anyone with access to this file would be able to see all the relevant information, including the password. Ensuring that no one without authorization can access this file can minimize the risks.

The code syntax is acceptable, although it could be improved by using more comments. This will make the code easier to update and maintain in the future. With a code this size, this may not be a significant concern.

## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

Using the Vulnerability Assessment Process Flow Diagram, the areas of security that were addressed by refactoring the code are: APIs, Code Quality, and Cryptography. Utilizing the self-signed certificate allows for a secure connection and interaction APIs. The code quality was improved to be functional and ensuring that good coding practices were followed. And the Cryptography was addressed through the use of the hash function and software certificate. The refactored code has been compiled into a zip folder and attached to this report.