****

# 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 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **6/20/2021** | **Joseph Mead** |  |

## 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

Joseph Mead

## 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.

[Include your findings here.]

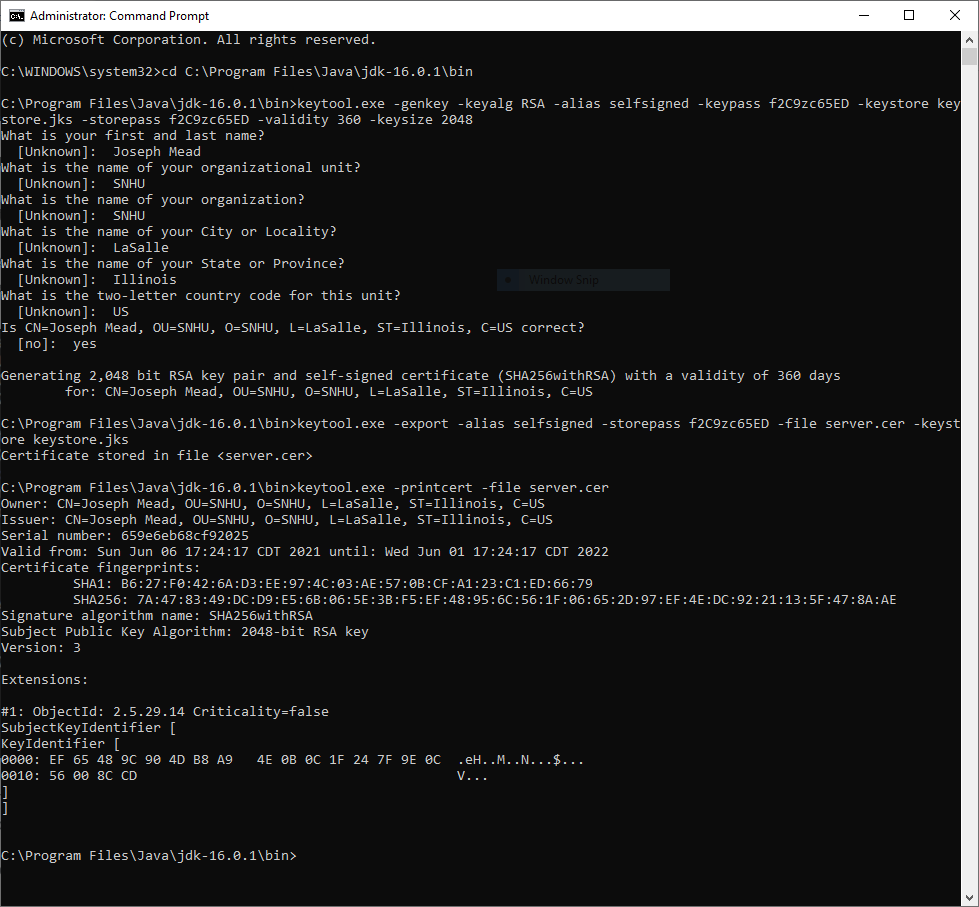
I recommend using SHA-256. SHA-256 is the industry standard, which makes it easy to recommend. It features a 256-bit key which makes it very secure. Collisions are also unlikely because there are 2256 possible hash values. For context, collisions occur when two different values result in the same value. For example, an older algorithm, MD5 is susceptible to intentional collisions. There is a program called HASHCLASH that can generate collisions in a few minutes or less (Callaghan, 2020).

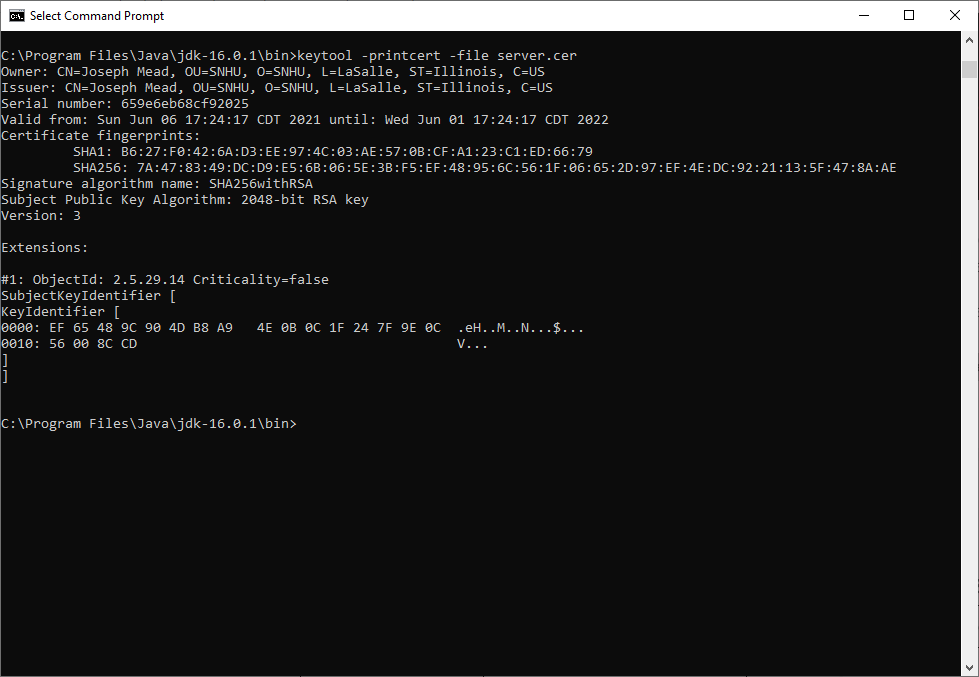
## 2. Certificate Generation

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.

[Insert screenshot(s) here.]



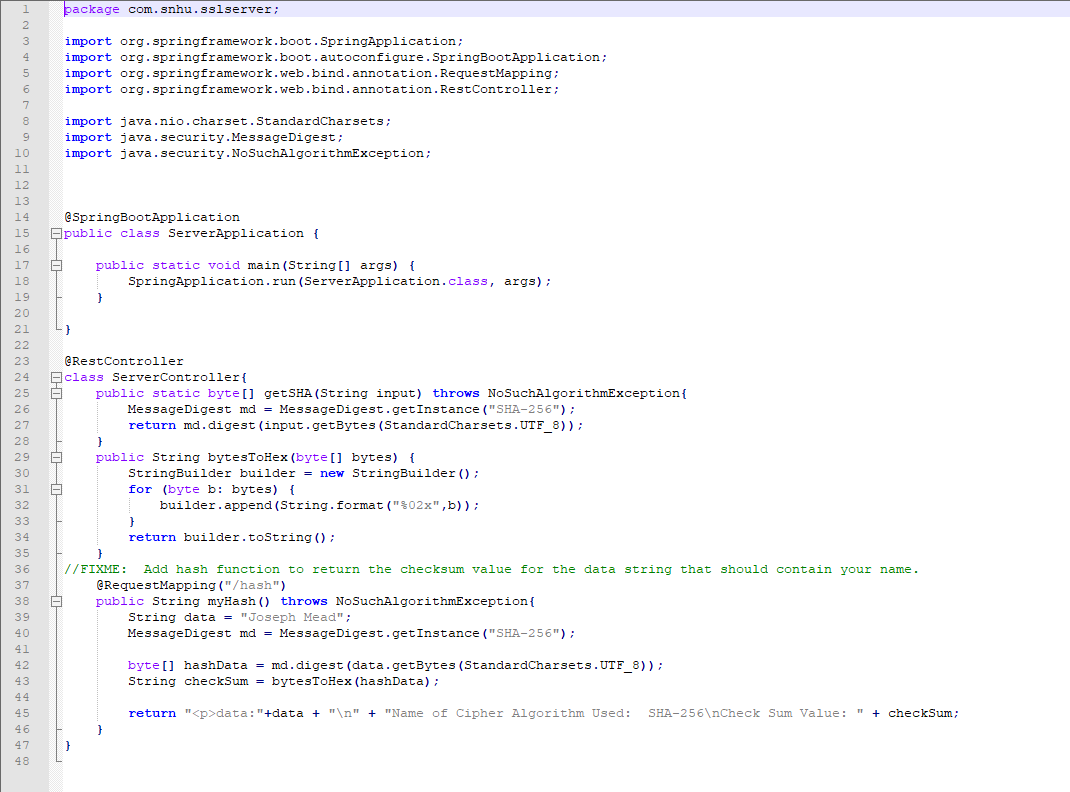


## 3. Deploy Cipher

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.

[Insert screenshot(s) here.]

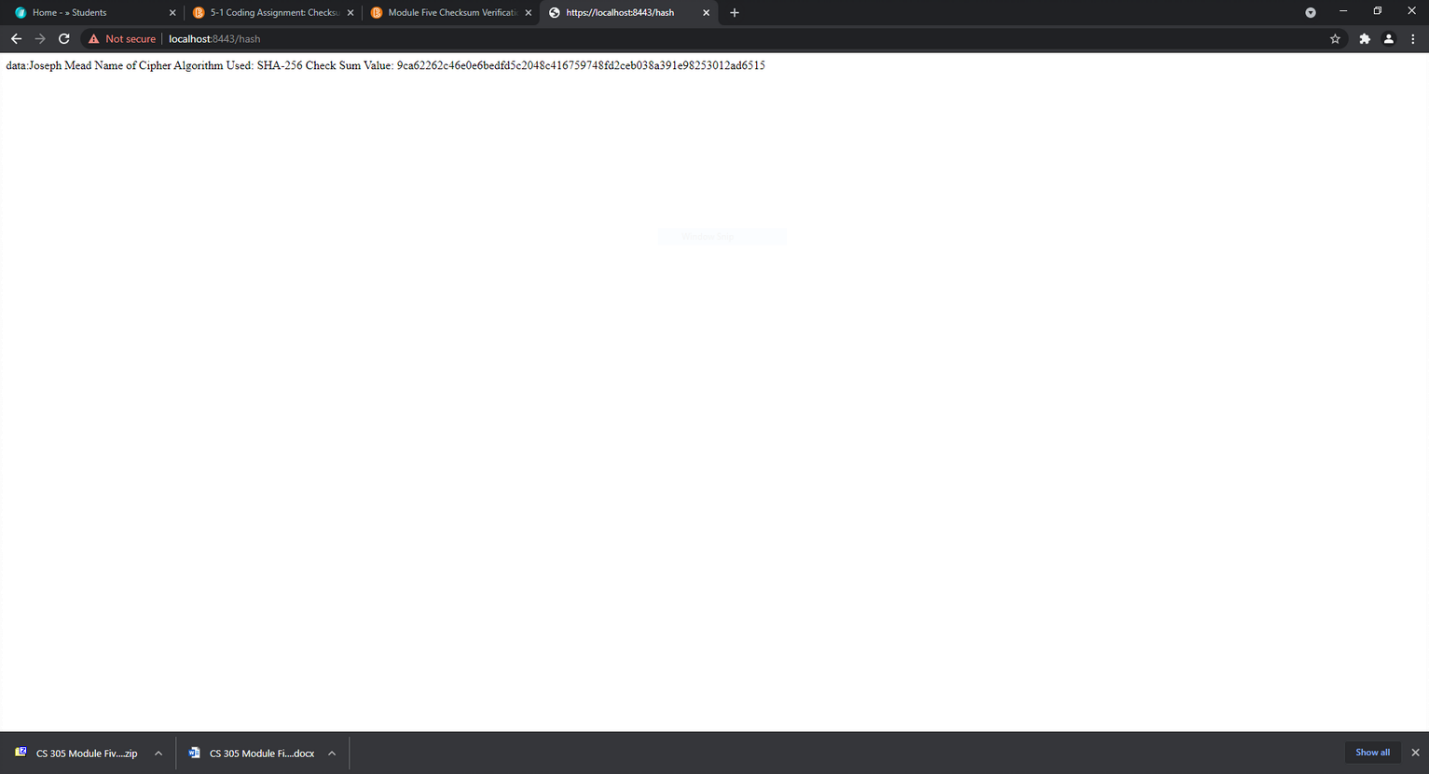


## 4. Secure Communications

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.

[Insert screenshot(s) here.]

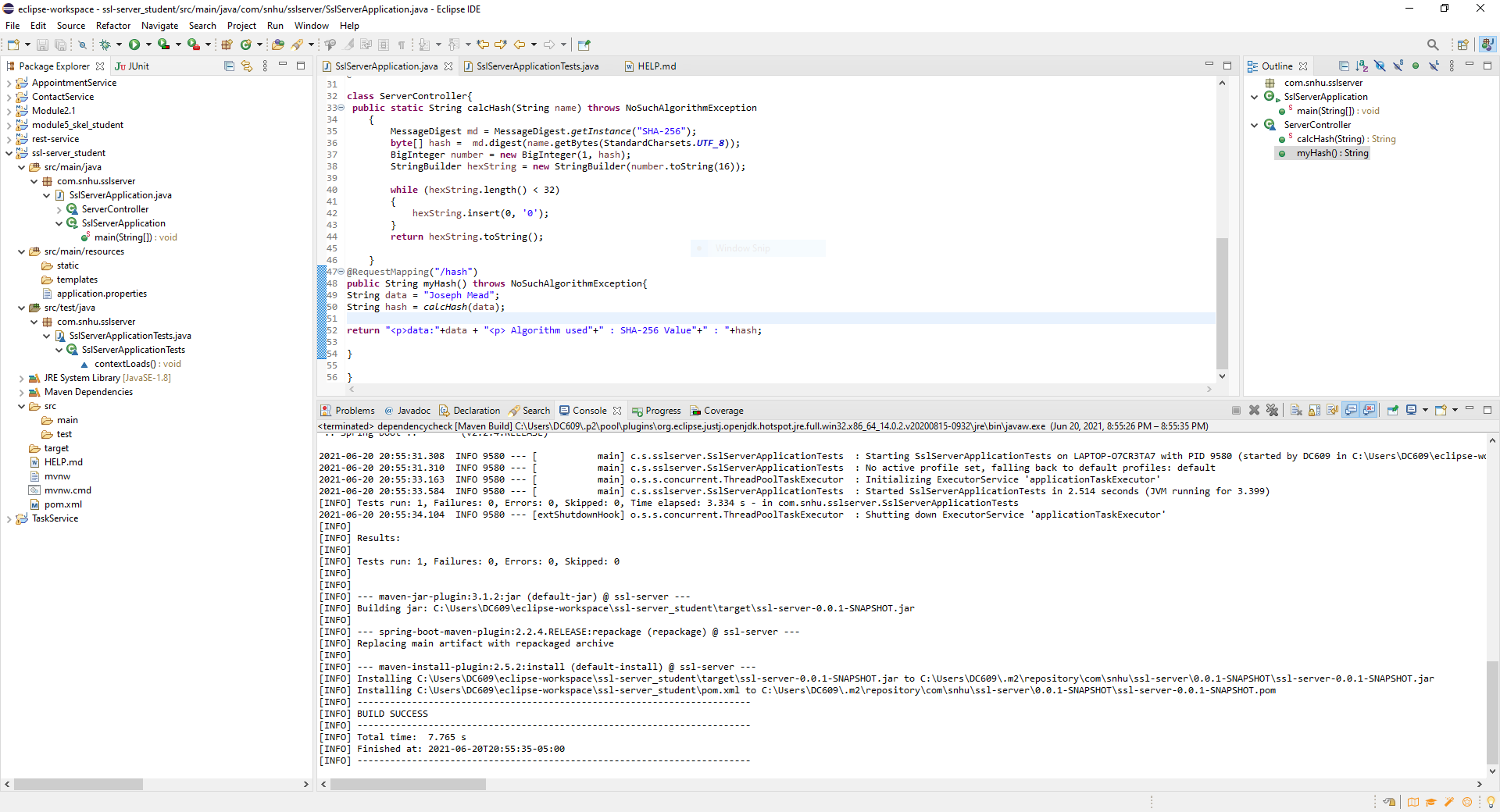


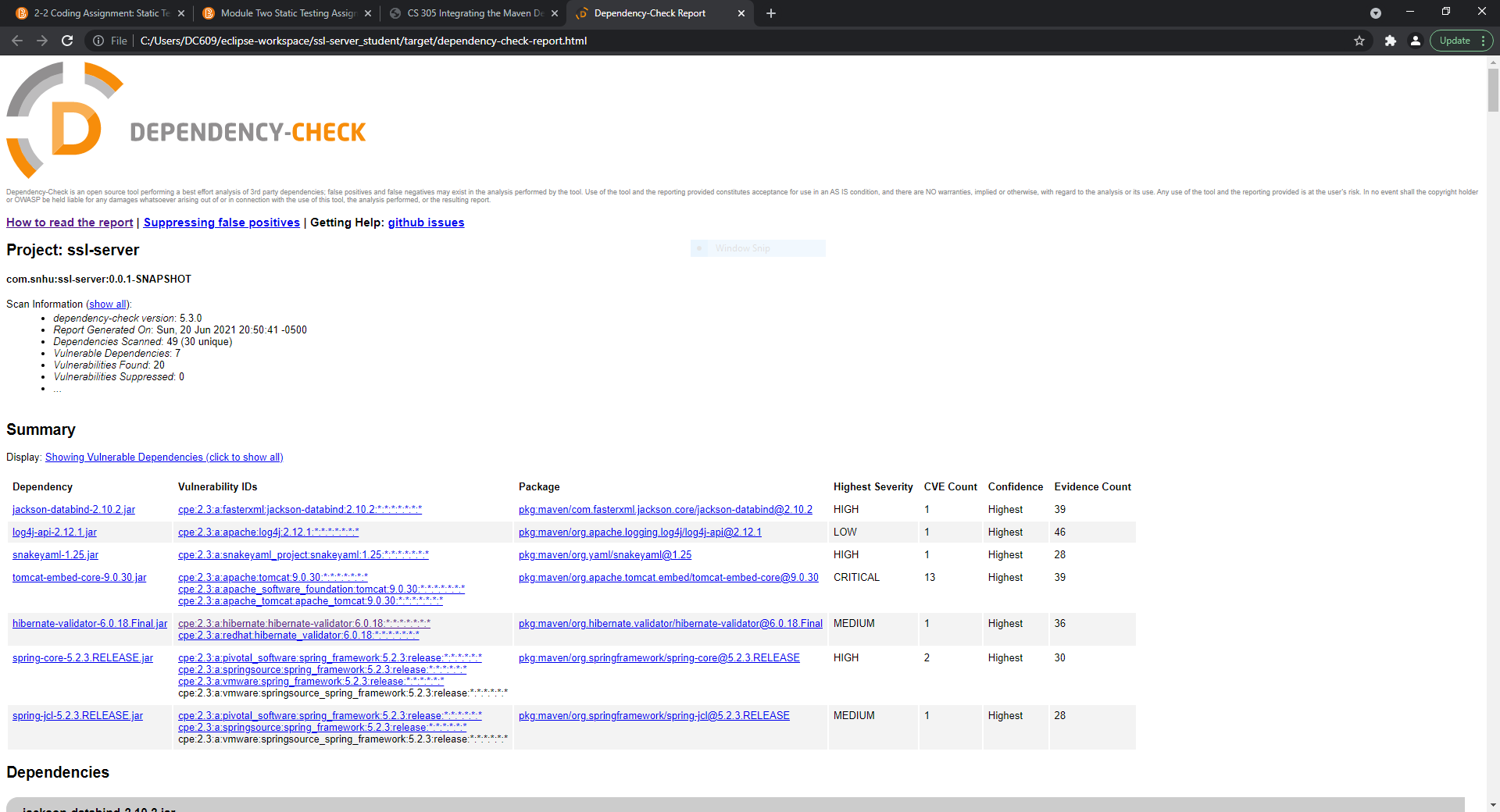
## 5. Secondary Testing

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

[Insert screenshots here.]



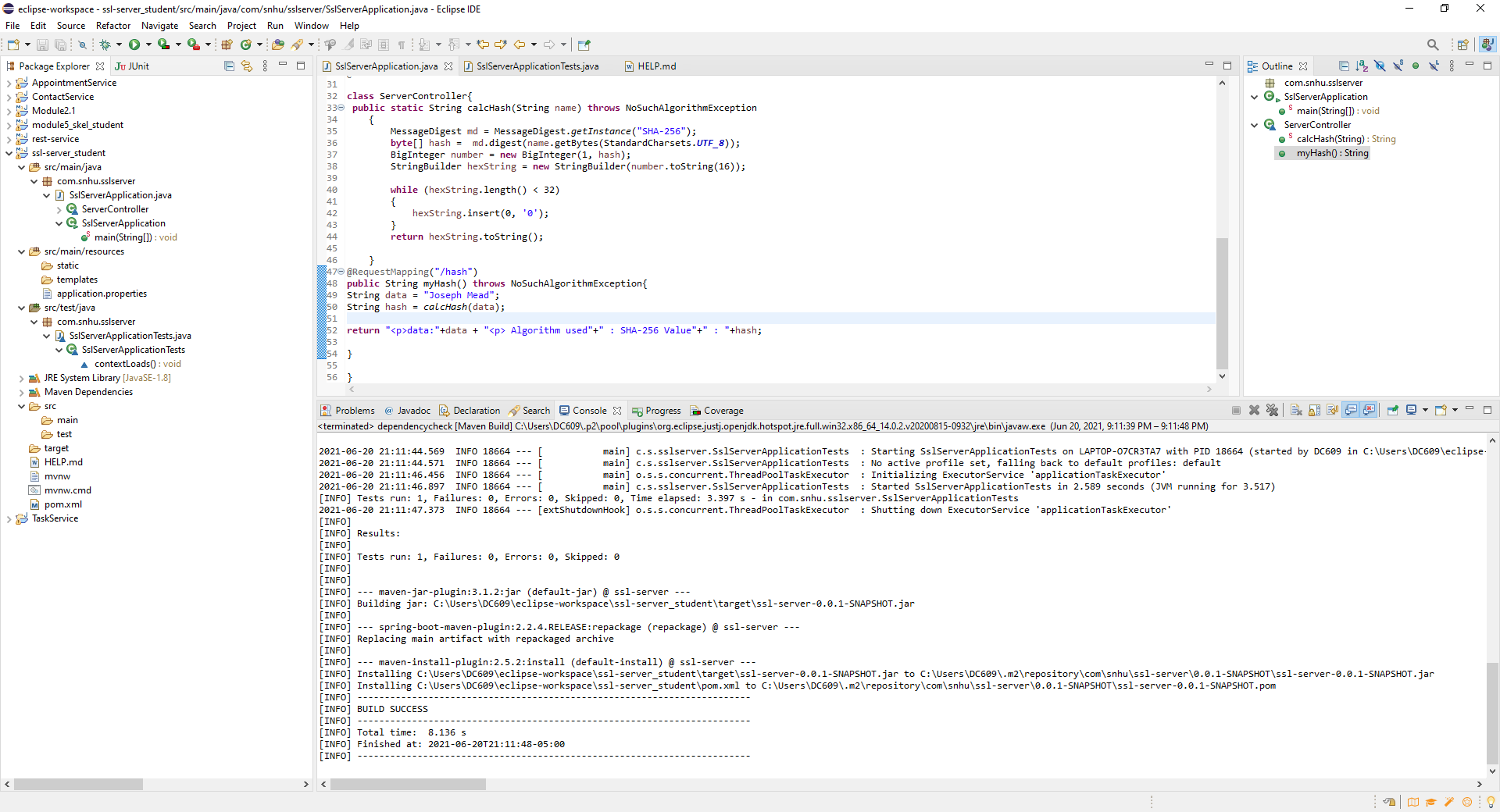


## 6. Functional Testing

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.

[Insert screenshot(s) here.]



## 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.

Each of these areas of secure architecture have been addressed:

Input Validation- Added a checksum function which verifies the authenticity of the server to the user

APIs – Employs a web API with SHA-256 encryption

Cryptography- Modified code with a SHA hashing function that encrypts strings. (Presently, just my name).

Code Error – Added a dependency that automatically checks the code against the known errors in the CVE database and generates a report of those errors.

This year, there have been a record number of hacks that have resulted in billions of dollars or lost production and compromise the security of data within our government and the country’s biggest corporations. Using the industry standards in encryption and authentic verifications is the bare minimum required to secure user’s data. It’s also essential to periodically review the code as the database of known vulnerabilities is growing all of the time.

# Works Cited

Callaghan, P. (2020, August 19). *Why You Should Use SHA-256 in Evidence Authentication*. Retrieved from Pagefreezer.com: https://blog.pagefreezer.com/sha-256-benefits-evidence-authentication