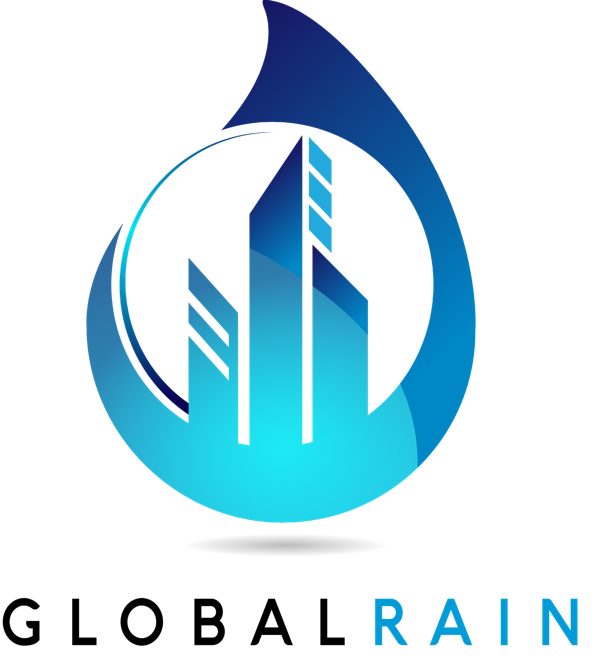
****

# CS 305 Project Two

**Practices for Secure Software Report**

**Table of Contents**

[Document Revision History 3](#_heading=h.1fob9te)

[Client 3](#_heading=h.3znysh7)

[Instructions 3](#_heading=h.2et92p0)

[Developer 4](#_heading=h.tyjcwt)

[1. Algorithm Cipher 4](#_heading=h.3dy6vkm)

[2. Certificate Generation 4](#_heading=h.1t3h5sf)

[3. Deploy Cipher 4](#_heading=h.4d34og8)

[4. Secure Communications 4](#_heading=h.2s8eyo1)

[5. Secondary Testing 4](#_heading=h.17dp8vu)

[6. Functional Testing 5](#_heading=h.3rdcrjn)

[7. Summary 5](#_heading=h.26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **8/20/2022** | **Phillip Kang** |  |

## 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

Phillip Kang

## 1. Algorithm Cipher

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

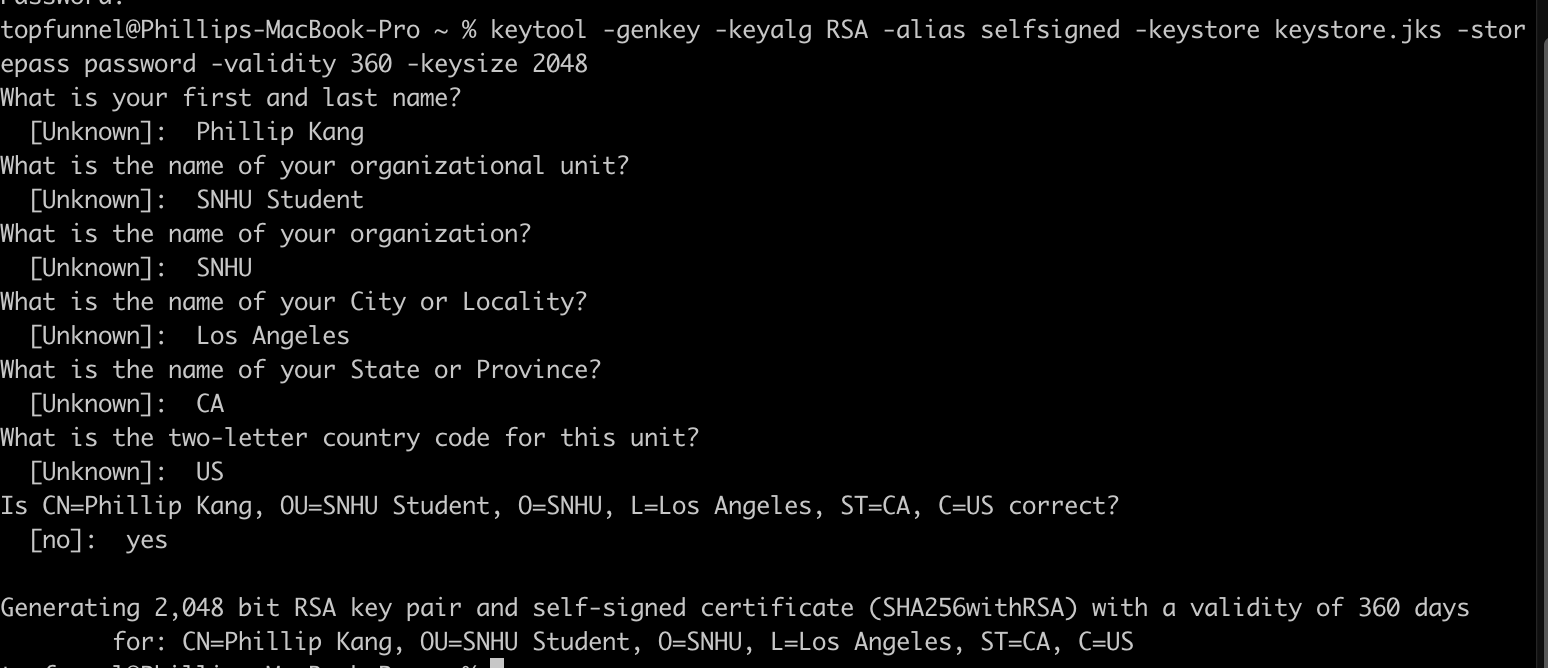
* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

An encryption algorithm cipher takes bytes and rearranges them through an algorithm in a way that is reversible to get the original byte sequence. Hash functions are run n many times by the bit level of the cipher, which means the attacker would have to perform at most 2^n number of attacks to be able to decrypt. By using a sufficiently large enough random number, the cipher will be hard enough to break to deter most would-be attackers. Additionally, having a non-symmetric key makes it systematically harder for the attacker to determine the cipher since the ciphers to decrypt and encrypt data would be different. Hacking the cipher on the client side is not enough to break server side encryption. Encryption algorithms started with symmetric, simple ciphers such as RSA, but have grown to asymmetric, more complex ciphers such as elliptic curve based ECDSA. Nowadays, asymmetric keys are the standard to enforce common encryption such as HTTPS and password hashing.

## 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.

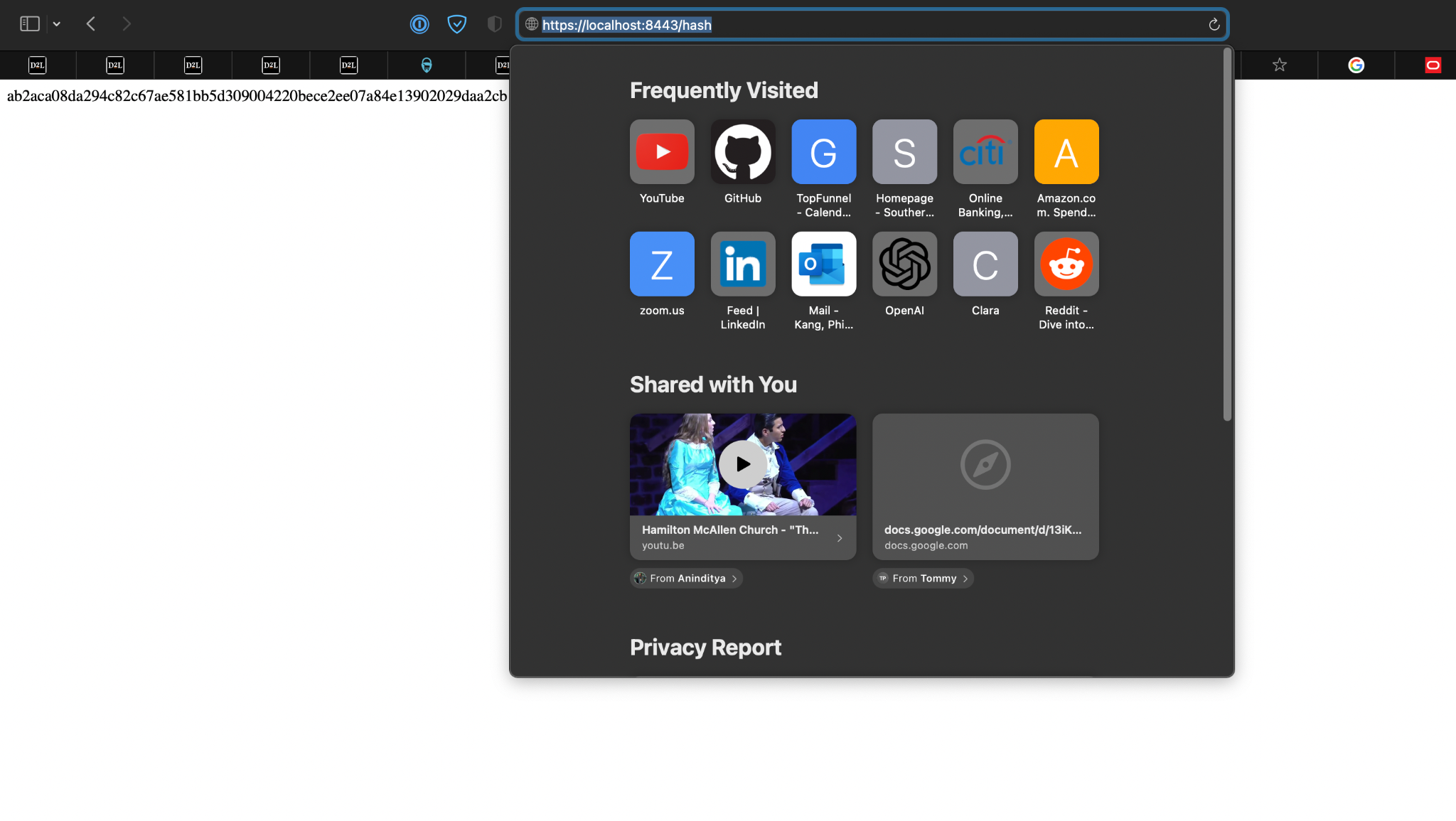
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.

[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.

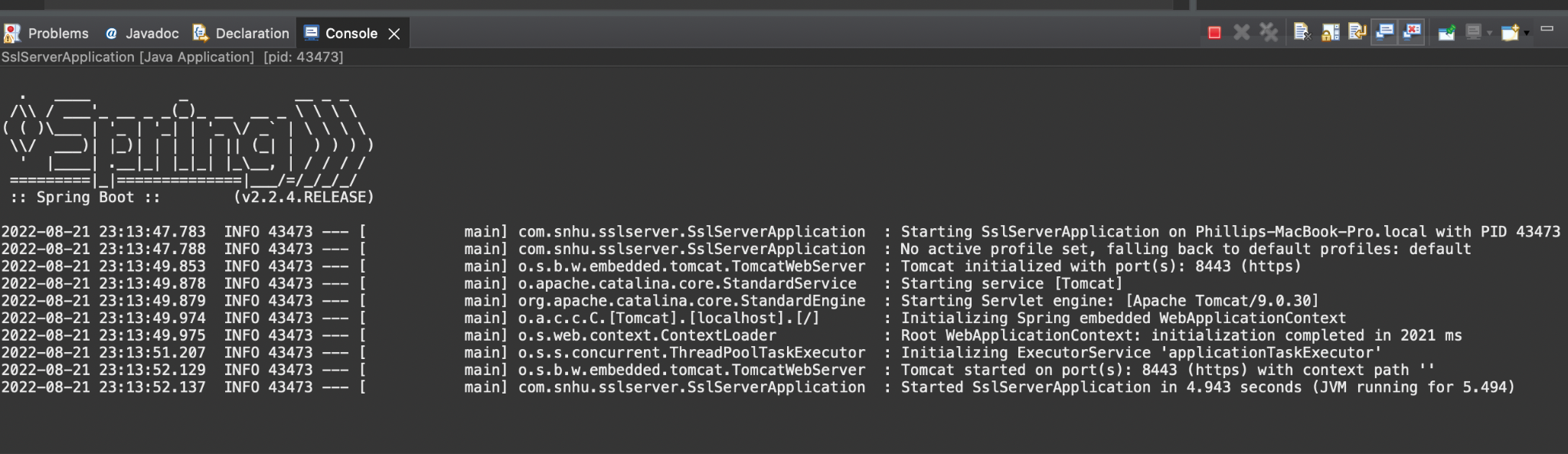
* Insert a screenshot below of the web browser that shows a secure webpage.

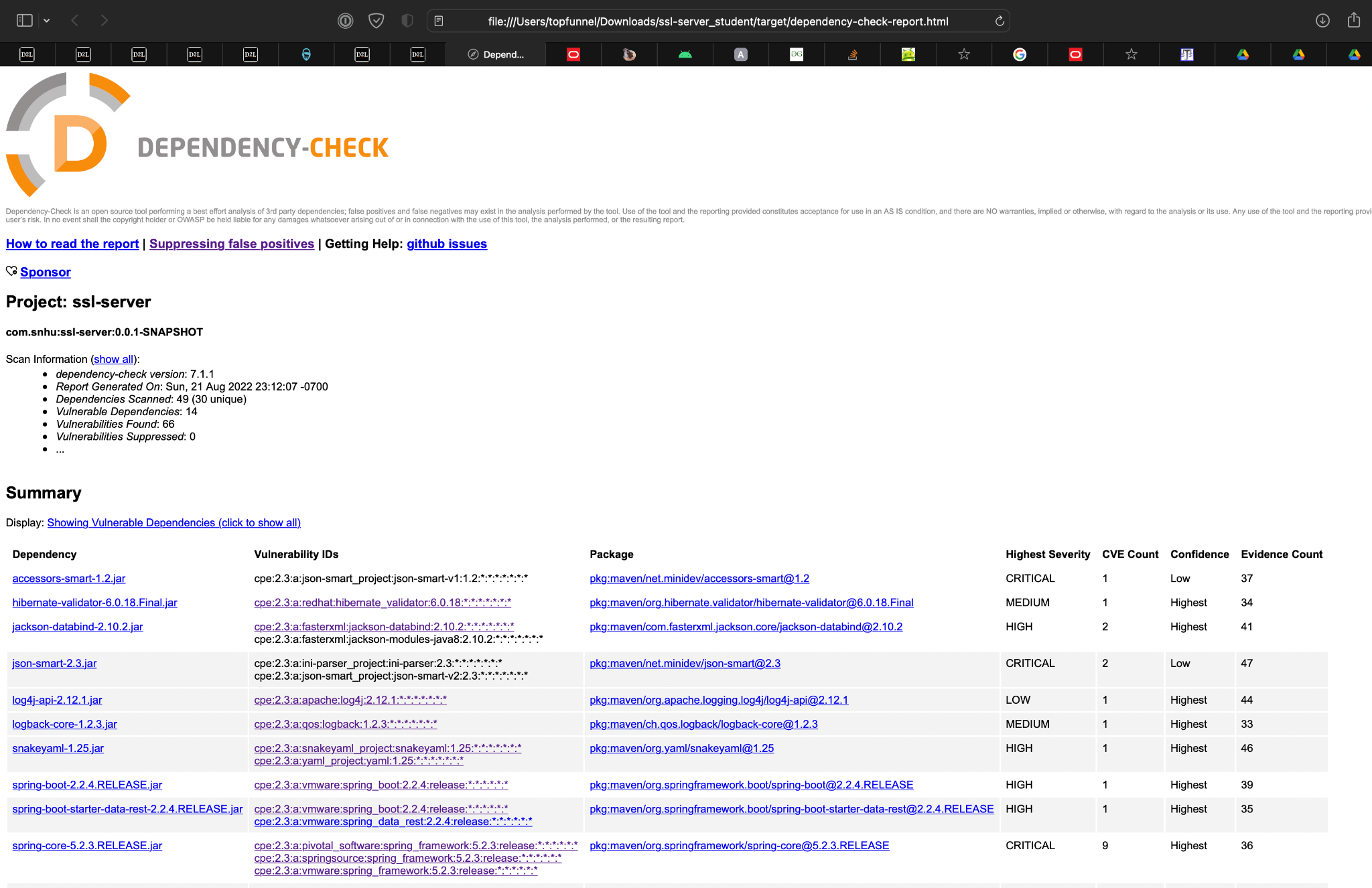


## 5. Secondary Testing

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

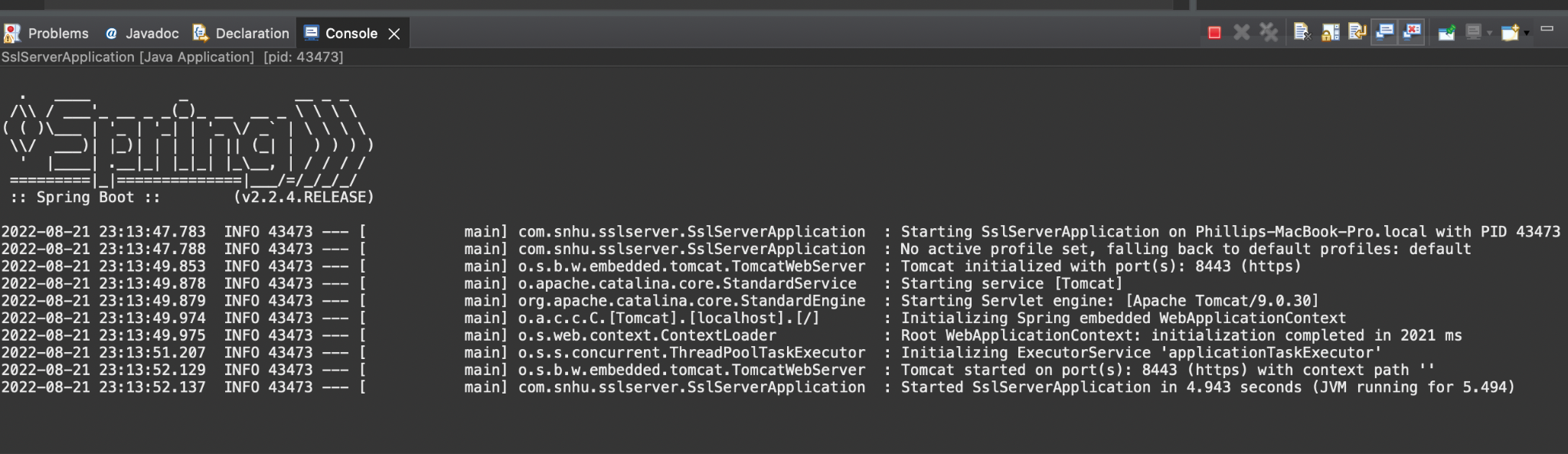




## 6. Functional Testing

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.



## 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.

The code has been refactored by adding the HTTPS layer by using certificates to check that communications between the server and client are encrypted. We also added cipher hashing to ensure that strings passed to the server are correctly encrypted. To continue maintaining the current security, it is important to continue running static tests to review that there are no vulnerabilities, update libraries as they are available, and to continue having test coverage by running functional tests across the codebase.