****

# 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 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** | **6/14/22** | **Kerrian Offermann** | **Worked on 1** |
| **1.1** | **6/15/22** | **Kerrian Offermann** | **Worked on 2 and 3** |
| **1.2** | **6/16/22** | **Kerrian Offermann** | **Completed all prompts** |
| **1.3** | **6/17/22** | **Kerrian Offermann** | **Proofreading and Fixing HTTPS issues** |
| **1.4** | **6/18/22** | **Kerrian Offermann** | **Final touches** |

## Client



## Developer

Kerrian Offermann

## 1. Algorithm Cipher

To find the correct algorithm, I first had to look at Artemis Financial and what it is that they would like Global Rain to accomplish. Artemis Financial is a financial consulting company that creates financial plans for their customers. These financial plans include savings, retirement, investments, and insurance. At the moment, Artemis Financial has a public web interface. They would like us to secure communications by adding a data verification step (using checksum) to data being transferred with the web applications. Upon examination of these requirements, I chose to go with this algorithm cipher: TLS\_RSA\_WITH\_AES\_256\_CBC\_SHA\_256.

The first thing that was kept in mind when choosing an encryption algorithm cipher is that we are working with a web application that intends to transfer information. This means that there will be communication between client and server involved. Setting up a TLS protocol will allow a private, secure connection between two parties while RSA will serve as our public key in the exchange of certificates (Dierks & Rescorla, 2008, p.4). When it comes to the actual encryption of data, AES-256 will create an algorithm of 256 bits. CBC will take the data into blocks of 256 bits before the SHA-256 hashing algorithm works to scramble the data into undetectable alphanumeric characters.

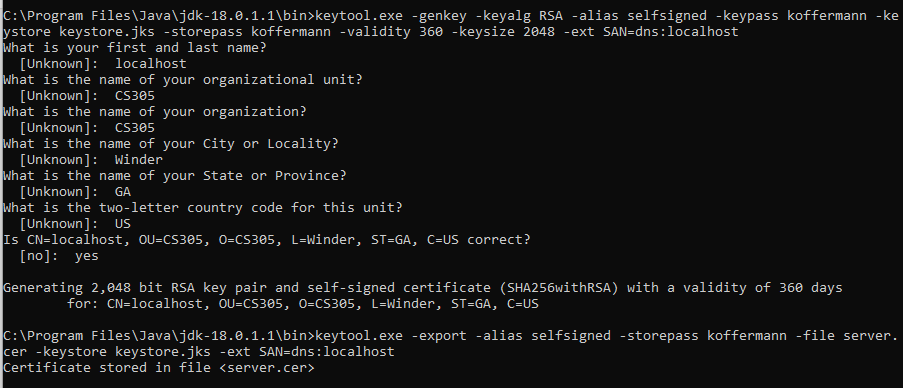
In order to conceal the data being transferred from one place to another, hash functions make data unreadable until it reaches its intended party. A hash function is incredibly useful as a cipher since it takes any input and encrypts it to a fixed length of bits (Reddy, 2022). The more bits that a hash algorithm has, the harder hackers have to work to successfully perform a collision. Collisions can do immense damage to a company like Artemis Financial since a hacker who is able to produce an identical algorithm to the algorithm in an encryption can go on to impersonate valid users and access sensitive financial records of patrons. Thankfully, the AES-256 algorithm is extremely difficult to perform a collision with and is secure enough that the federal government relies on it for the best protection (Menezes & Stebila, 2021, p. 100).

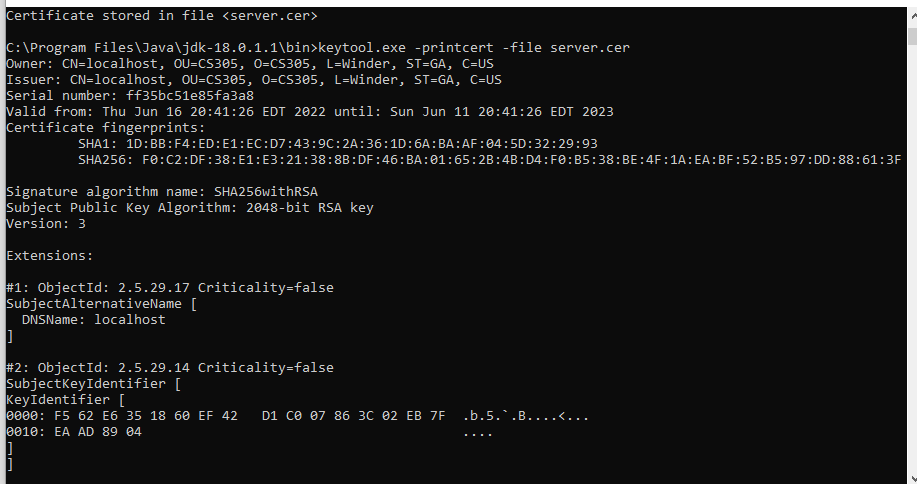
When it comes to encrypting data itself, one method of cryptography that can be used is the process of ciphering data as random numbers with a generator known as a cryptographically random numbers generator—or CRNG (Manico & Detlefsen, 2014, p. 179). Using random numbers in a cipher makes it difficult for any hacker to predict a pattern in an algorithm. When it comes to decrypting this encryption, there are two key options to consider. The first is symmetric. Symmetric keys can encrypt and decrypt data with the same key (Manico & Detlefsen, 2014, p. 165). The second type of key is a non-symmetric key which requires one public or private key to encrypt and one public or private key to decrypt (Manico & Detlefsen, 2014, p.165).

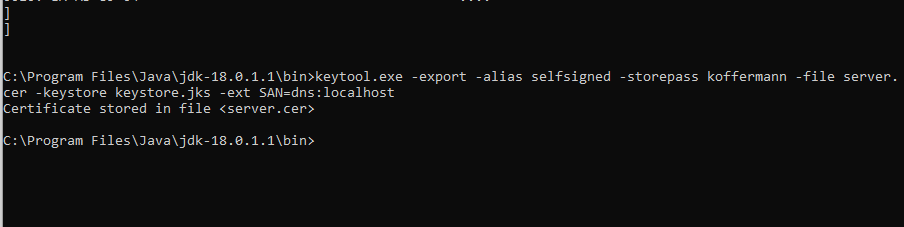
The history of encryption algorithms began in the 1970s when the National Institute of Standards and Technology (NIST) wished to establish an official block cipher standard (Menezes & Stebila, 2021, p. 98), and IBM delivered one in the form of a design known as data encryption standard. For a while, IBM’s encryption algorithm was the official government standard. However, as time passed, other developers began to notice that its bit level of 56 bits was too short and thus easier to break into (Menezes & Stebila, 2021, p. 98). This called for a new standard in encryption algorithms. A series of tests on the security level of the data encryption standard yielded a new 128-bit standard known as AES—or Advanced Encryption Standard (Menezes & Stabila, 2021, p. 98). Since its inception there has been an increase in the amount of bits that AES can use for encryption algorithm, but even with these changes the original 128-bit AES is still considered the standard after more than twenty years. As time progresses there will be newer attacks and threats, but thus far the programming community has been able to create reliable security measures at little to no cost for companies like Artemis Financial.

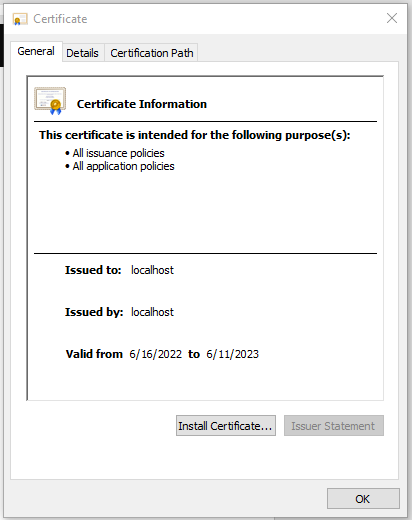
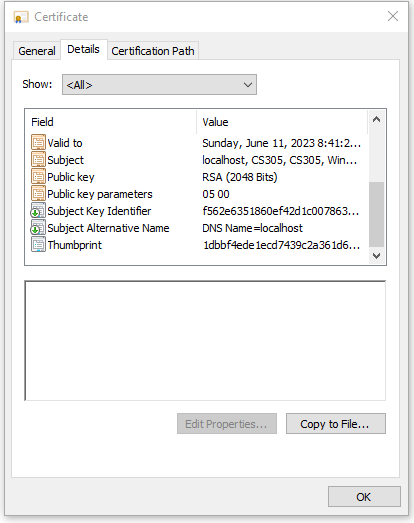
## 2. Certificate Generation

## This certificate was generated with the Subject Alternative Name extension added to the certificate. The only domain that this certificate will validate is “localhost”, but when applied to Artemis Financial it will only validate URLs related to the web application.



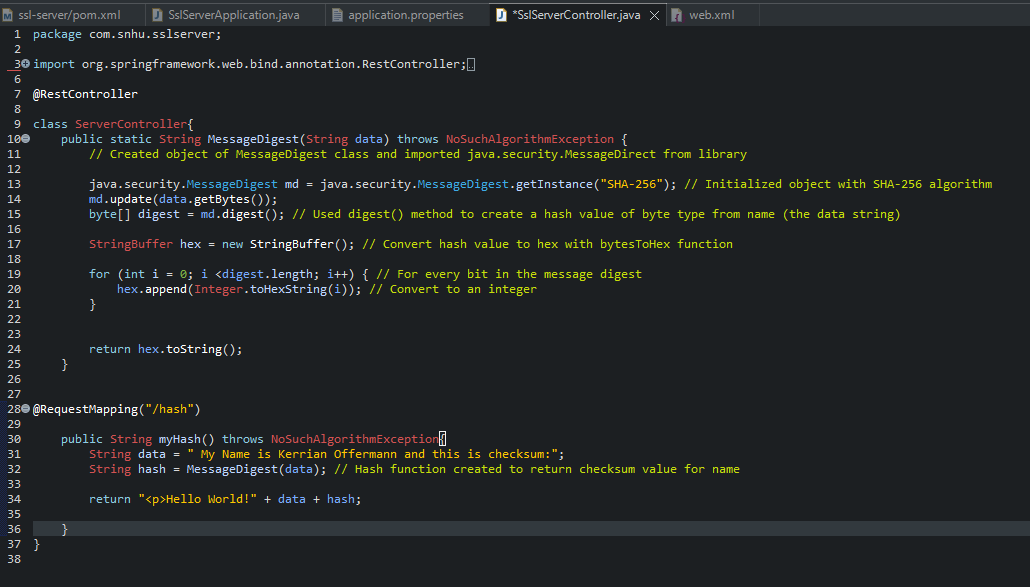


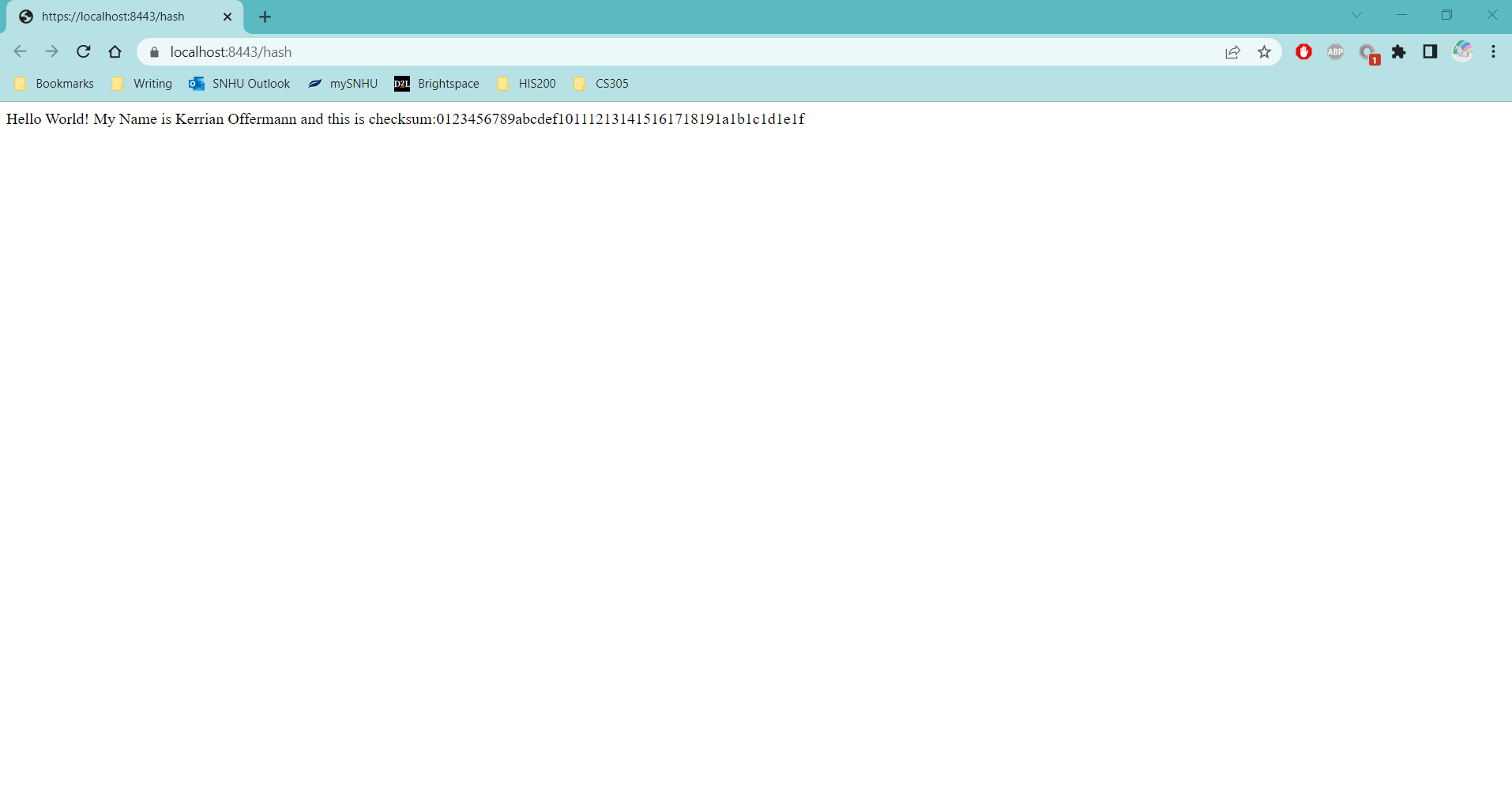




## 3. Deploy Cipher

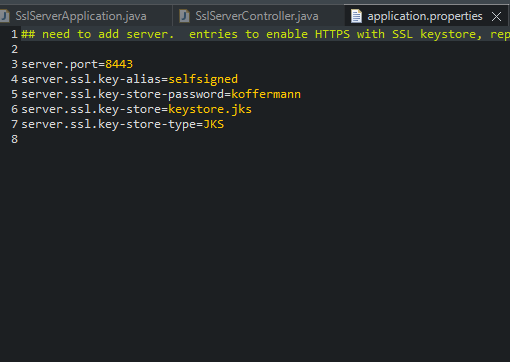
## The refactored code below uses the java security library to utilize the SHA-256 hash algorithm and create a checksum verification. When the code executes correctly, it will allow a local host web browser to open with my name and a unique, encrypted data string.



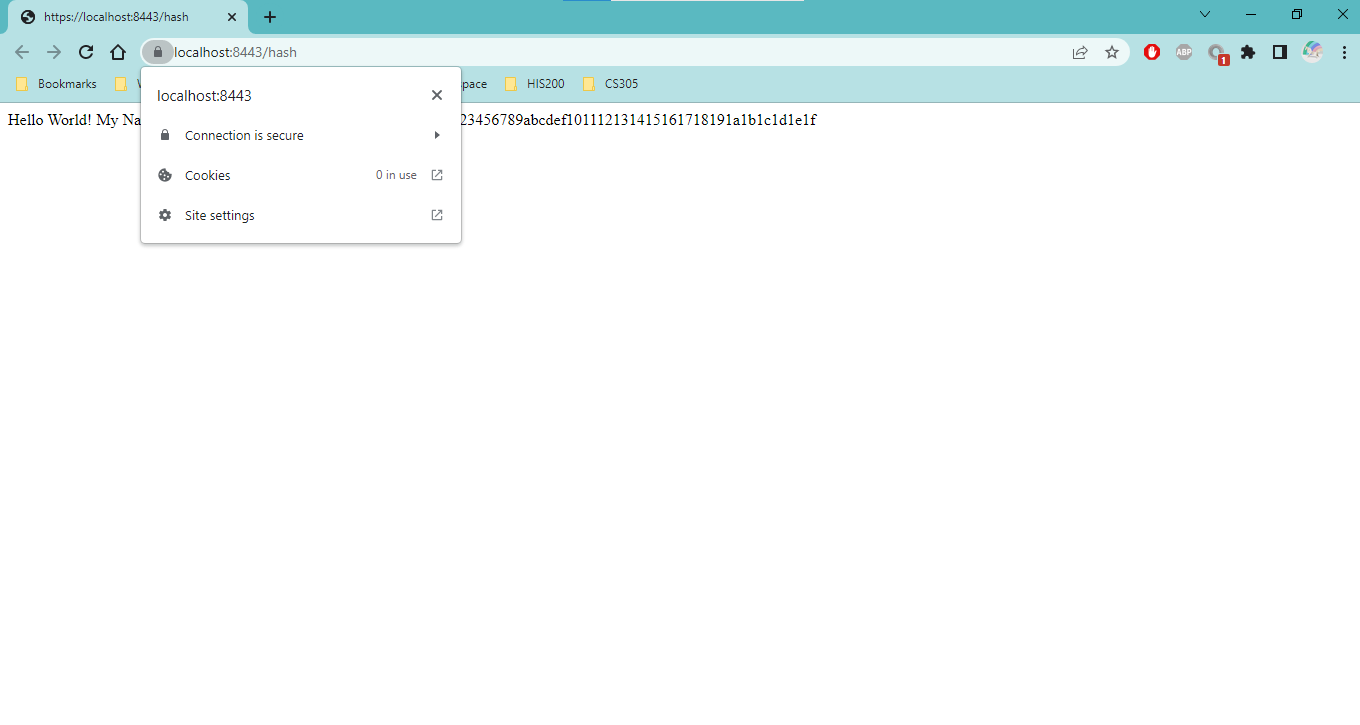


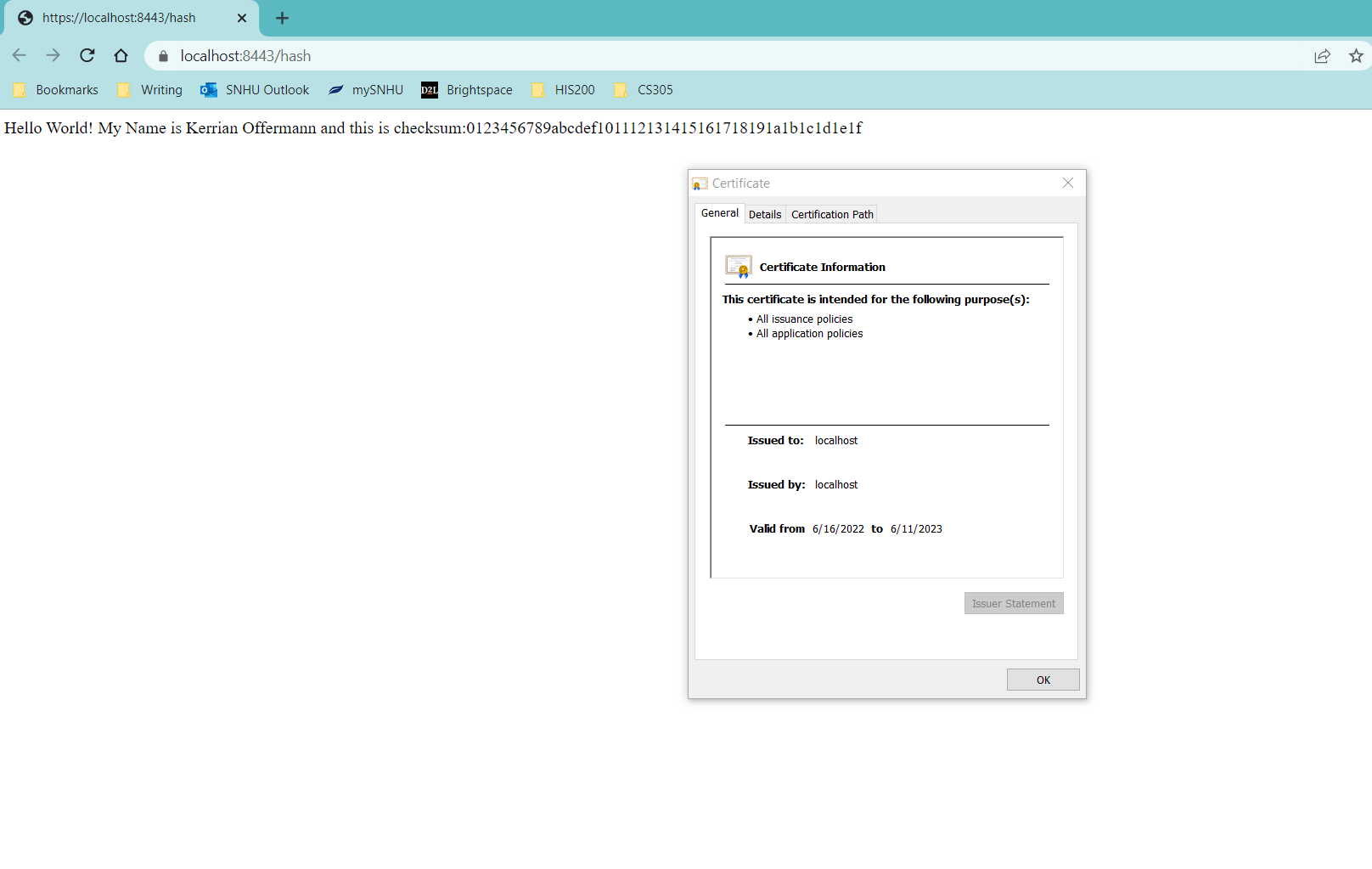
## 4. Secure Communications

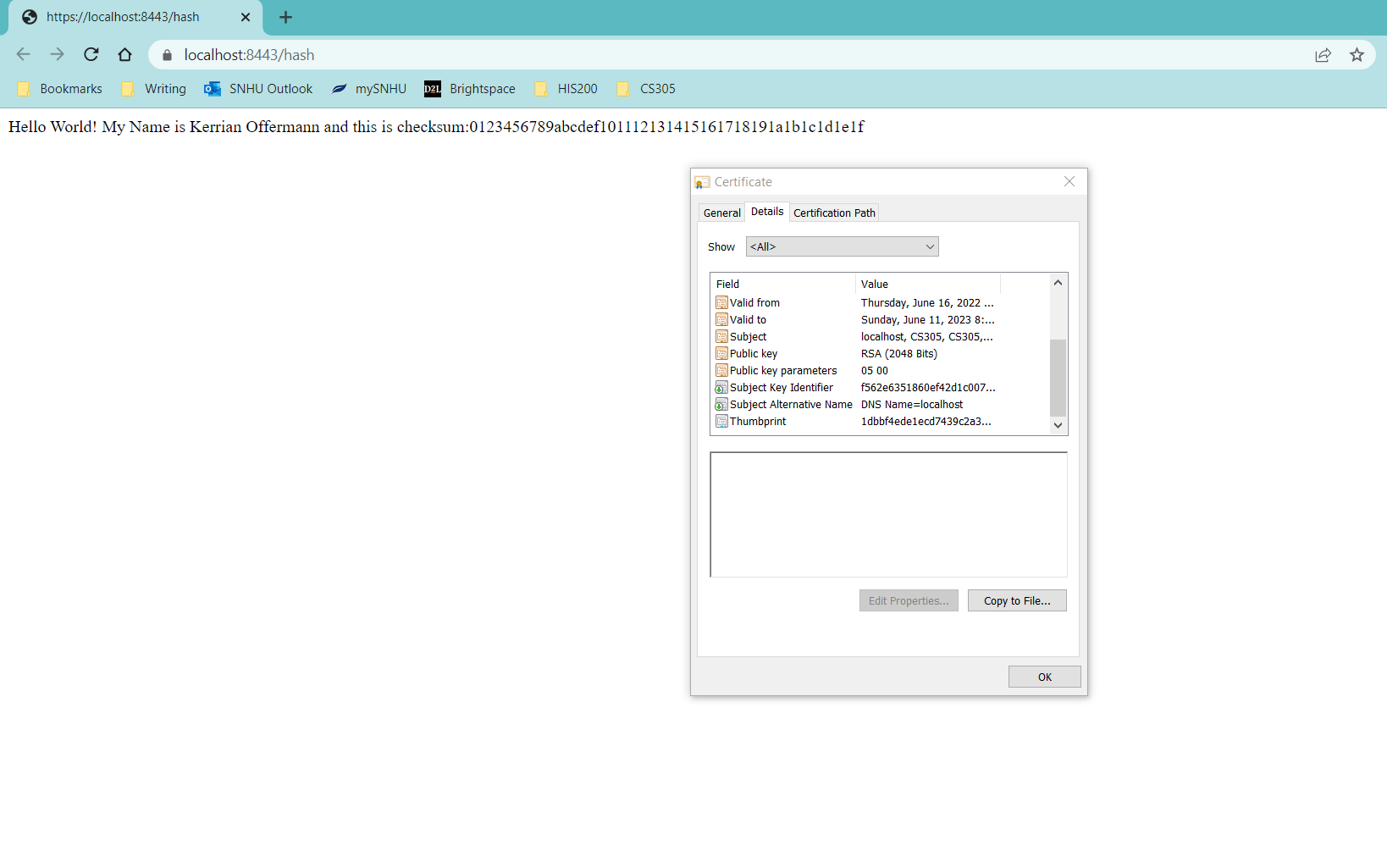
In order to convert to the HTTPS protocol, the information from the previously generate certificate had to essentially be connected to the application itself. The certificate files were moved from the java folder to the main folder of the web application, and the SSL information was updated under application.properties to match the information on the certificate.

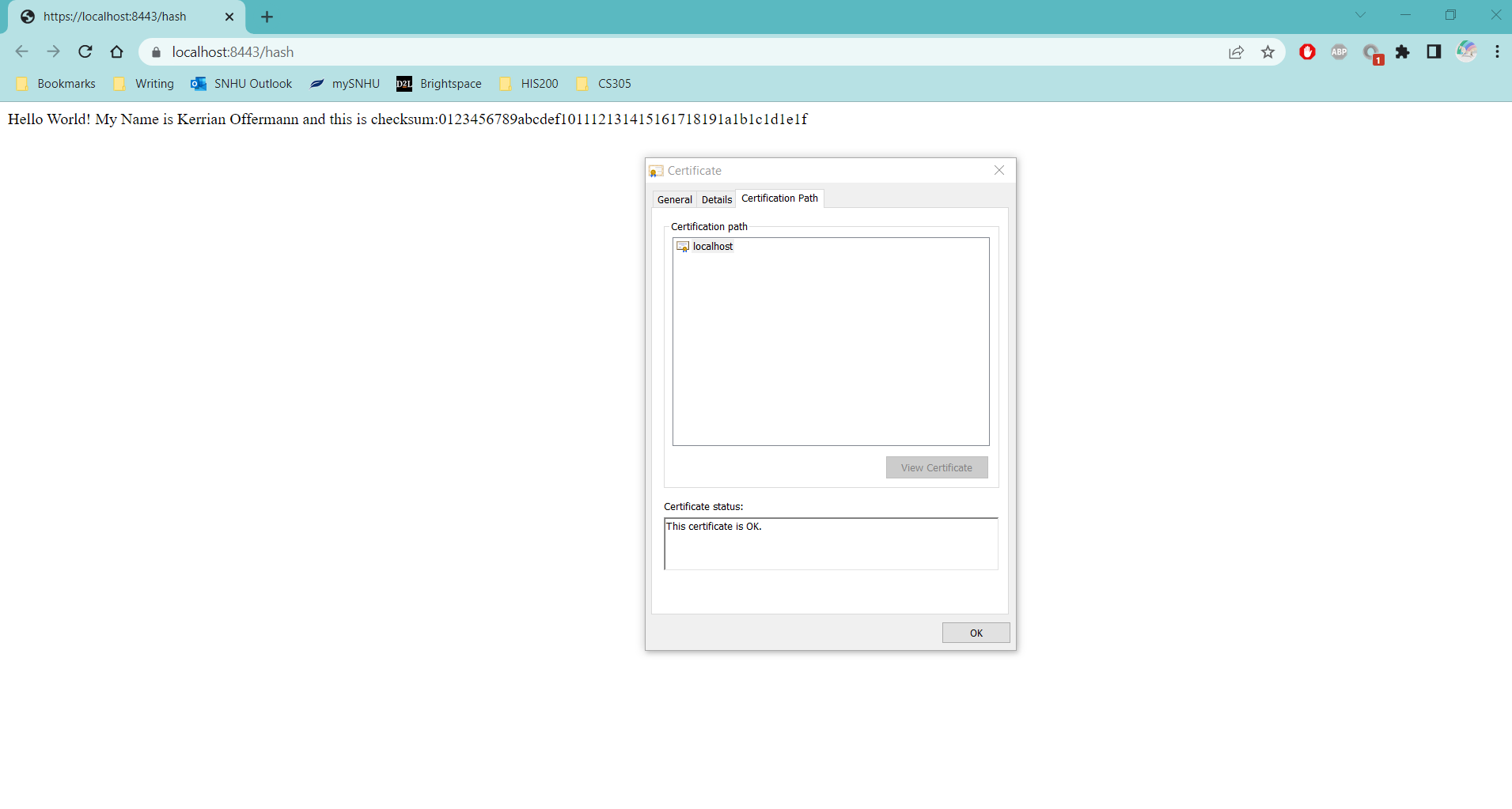


Proof that the connection is secure can be found below. This information is access by selecting the lock icon next to the URL and selecting “Connection is secure” for more information about the certificate itself. Note: this was done through Google Chrome. Other browsers may have different methods of accessing data about how secure a connection may be.





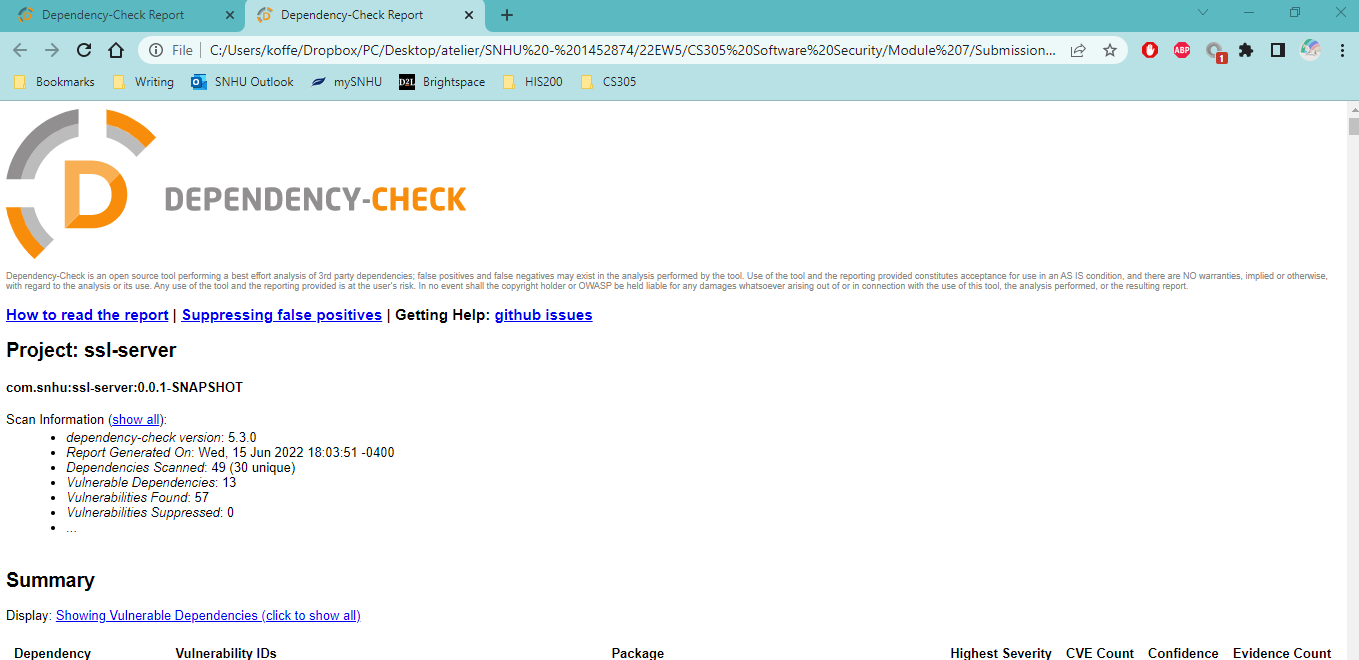




## 

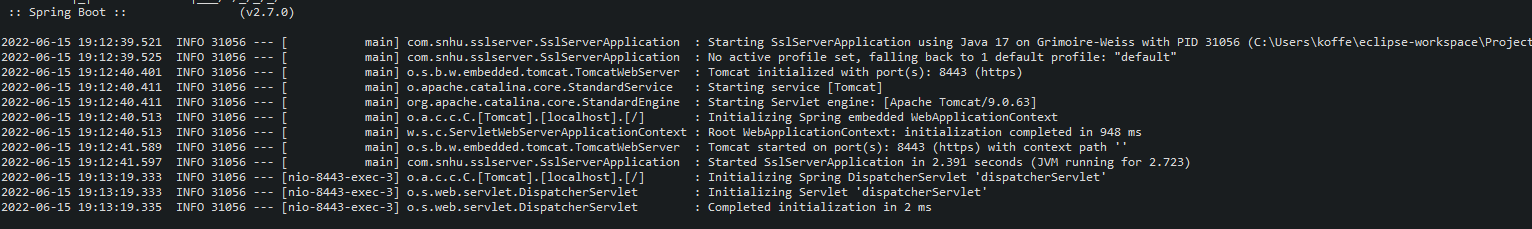
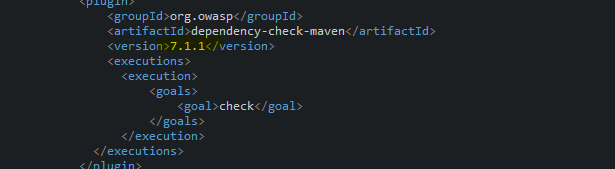
## 5. Secondary Testing

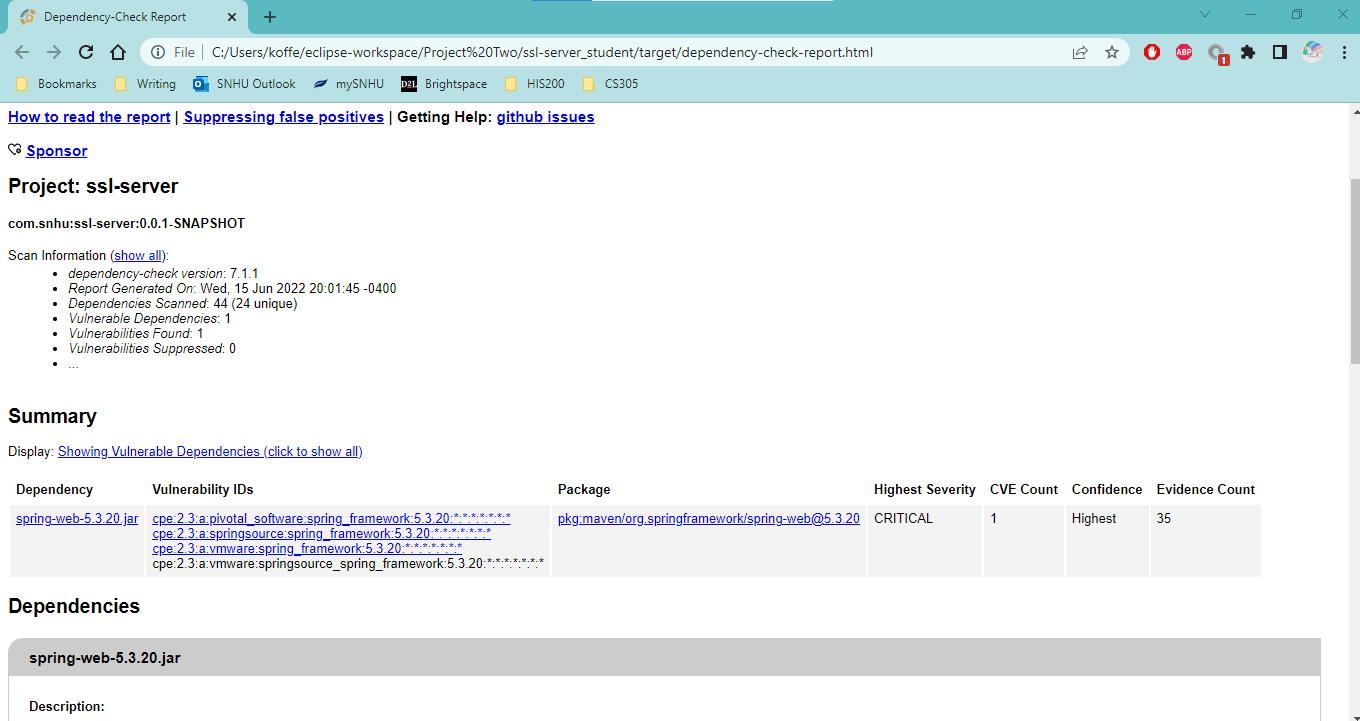
**Before Refactor:** Prior to refactoring the provided code, the following vulnerabilities were found through a dependency check.

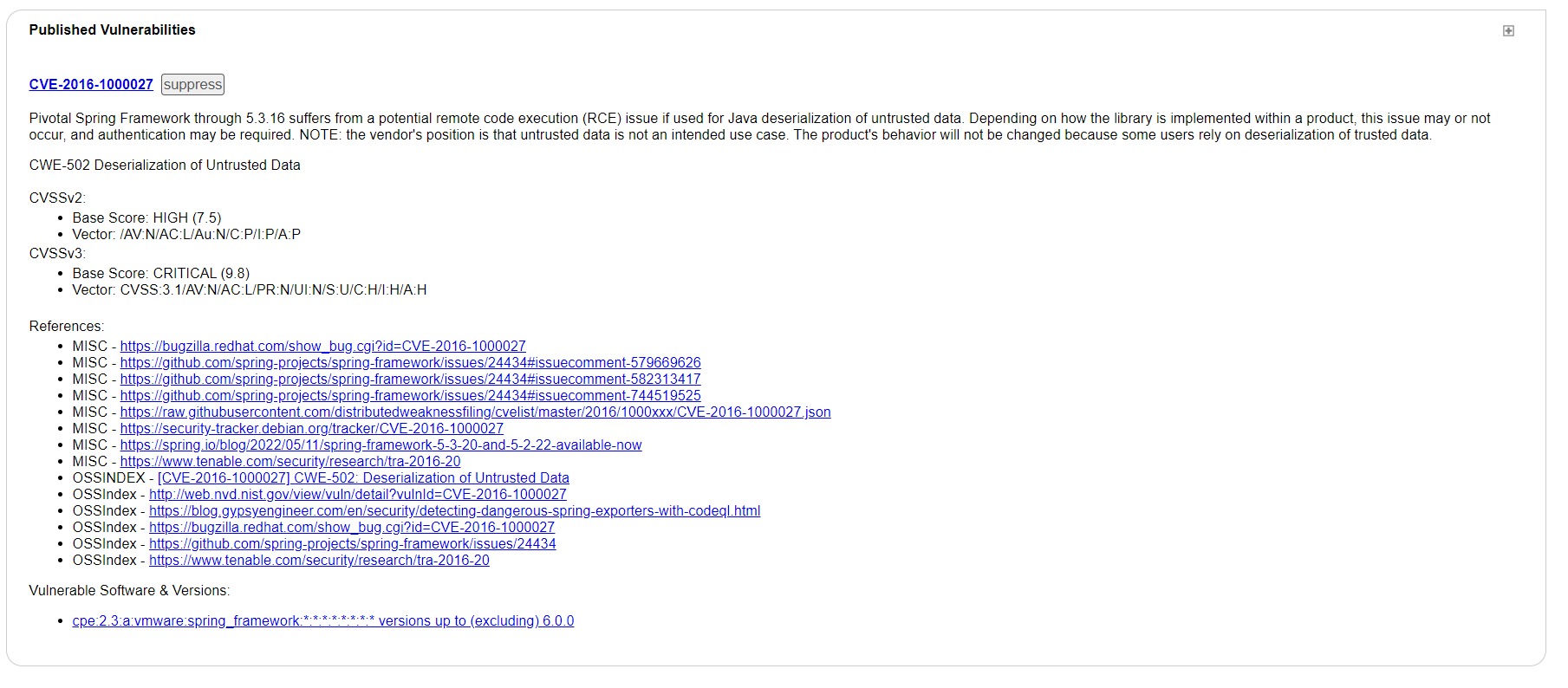
****

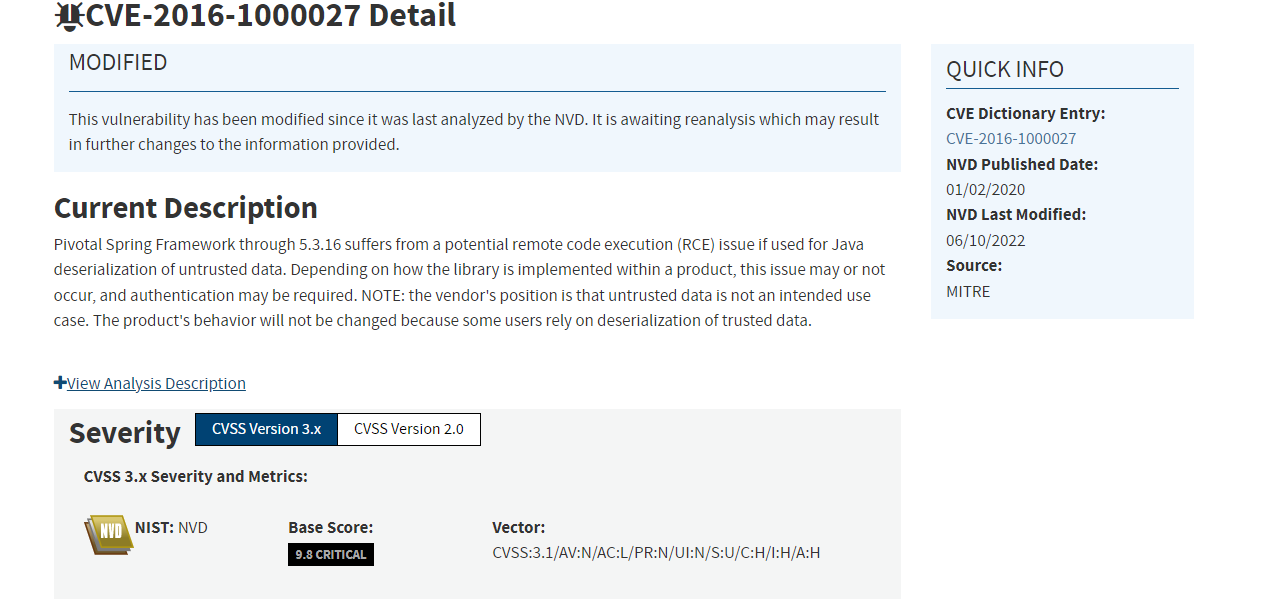
****

**After Refactor:** I updated pom.xml with the latest version of OWASP, and this eliminated all but one vulnerability. Upon researching this vulnerability, I learned on the NIST that this vulnerability is undergoing reanalysis. The vendor has also decided that the product’s behavior will not be changed since some users rely on deserialization of trusted data (per the dependency check report for CVE-2016-1000027).





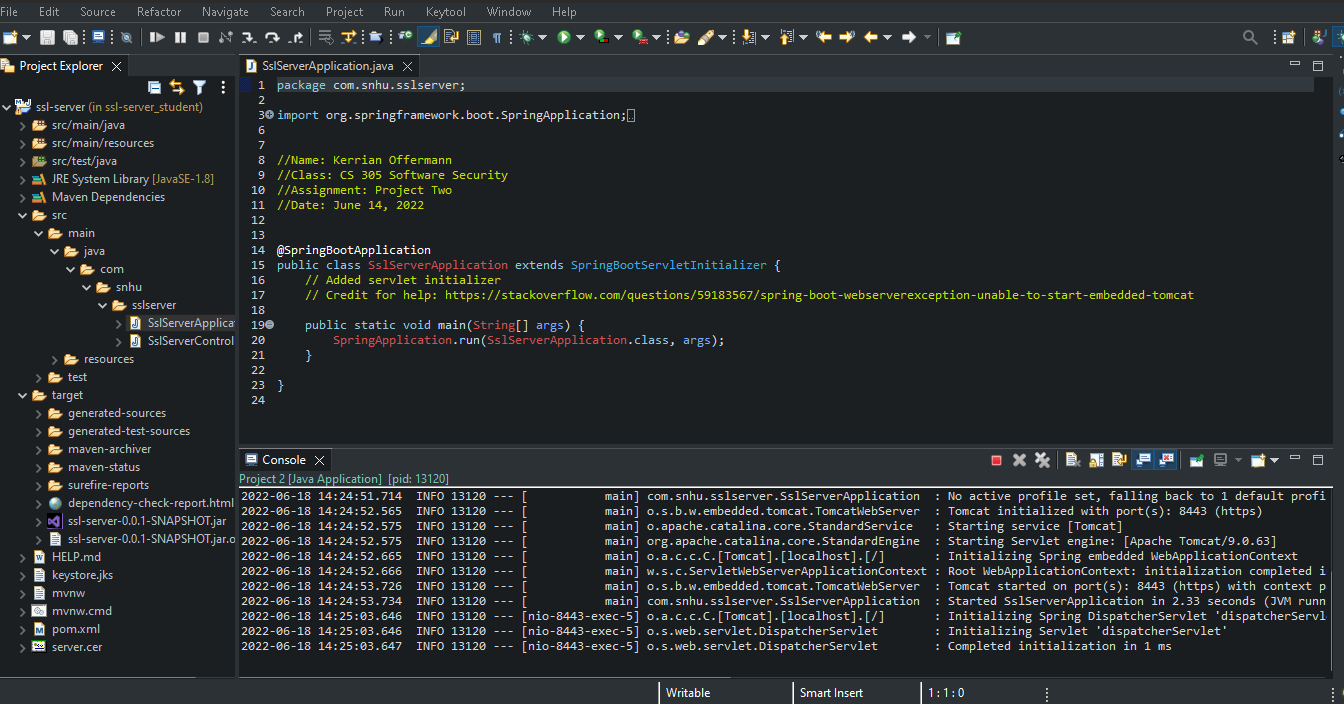


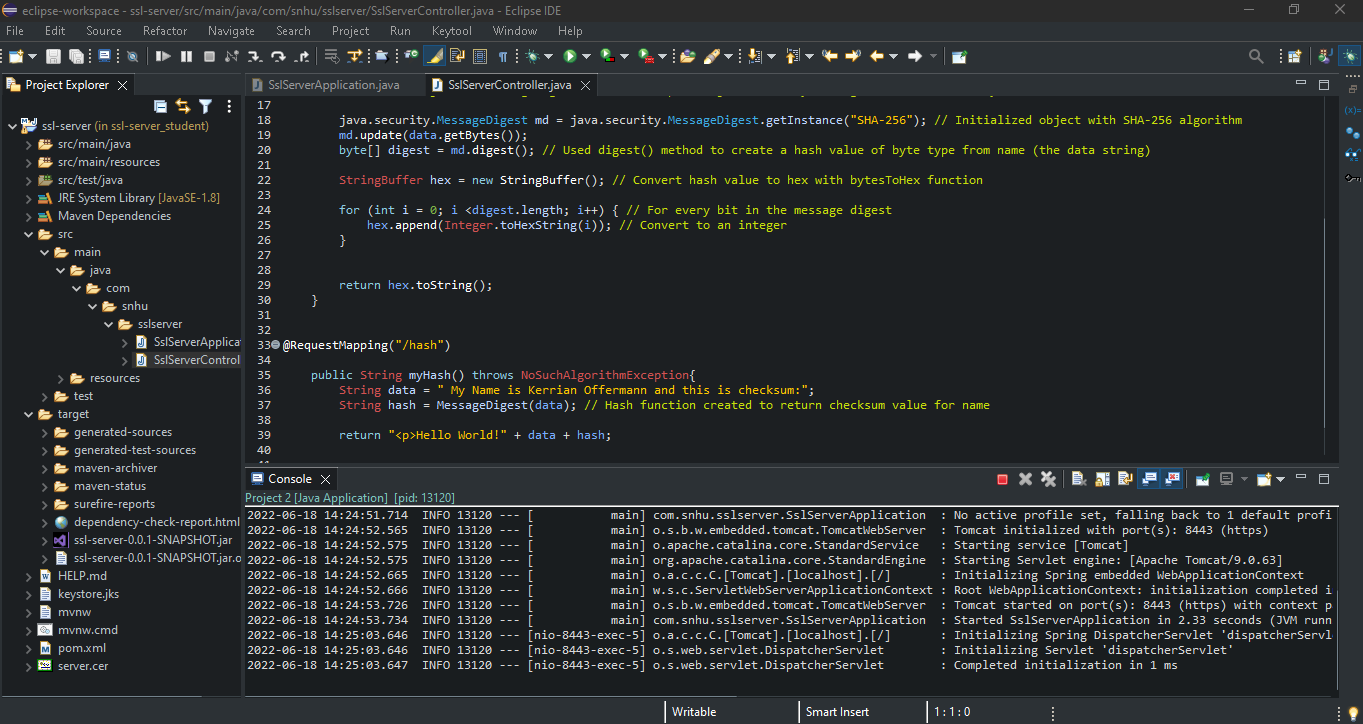


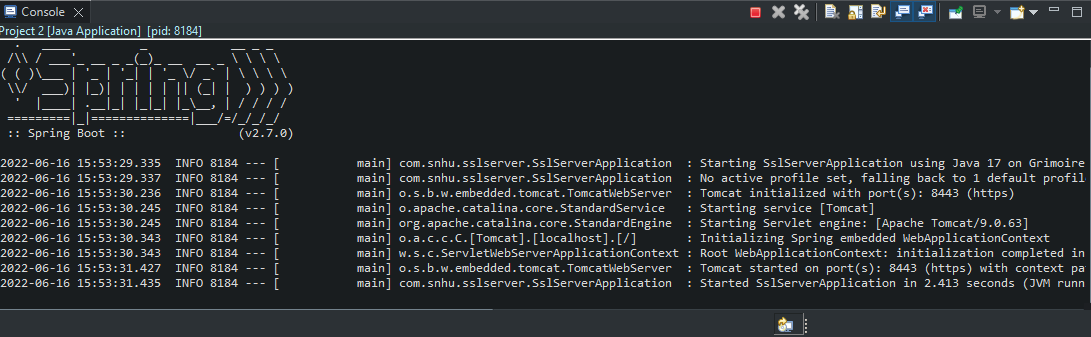
## 

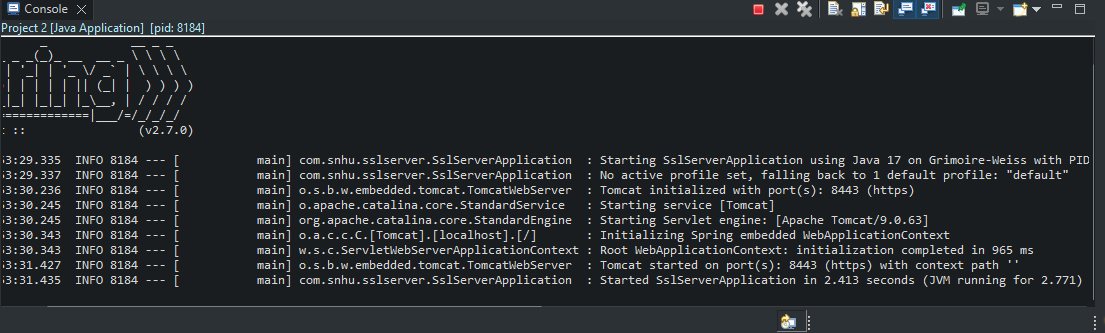
## 6. Functional Testing

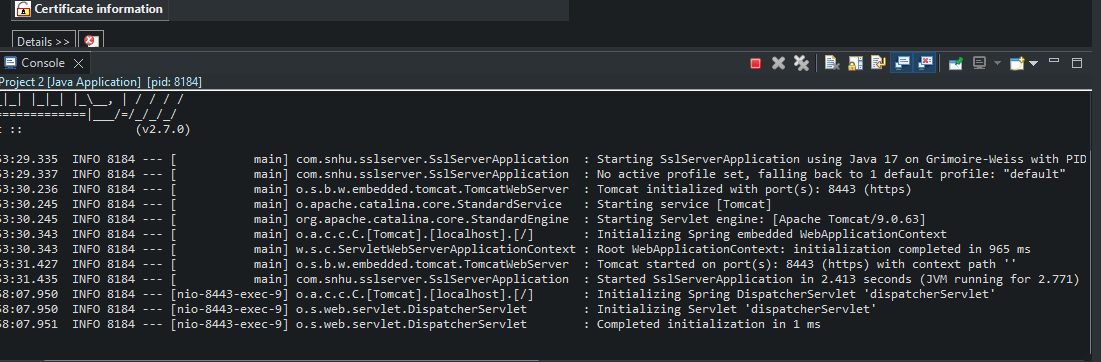
Below is the code running successfully with a connection to the server. Before this was accomplished, the code was inspected for syntactical, logical, and security vulnerabilities. More information on this is available after the screenshots:

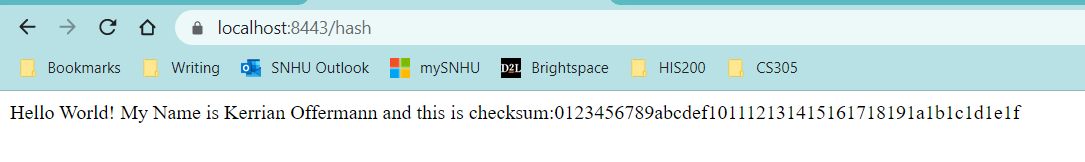




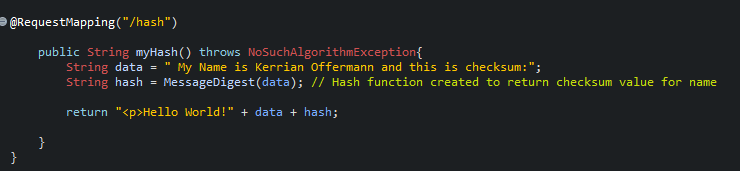








Upon examining the code further, there are a few vulnerabilities that should be mentioned in the software application. The first is the aforementioned vulnerability in the dependency check report. Although the vendor does not intend to change this dependency due to other developers needing it to function as it is, there still a problem with the untrusted dependency. At the moment NIST is reanalyzing the vulnerability; therefore, it would be ideal to keep checking for updates to this issue until there is a solution. One recommendation for developers is to suppress the vulnerability until such a solution comes along so it does not show up again in future dependency reports (Offermann, 2022). Another vulnerability to keep in mind when the web application starts to require users to log in is that lack of input validation. At the moment, only generic data is being used as input for the hash algorithm, so there is no sensitive information there:



However, when private usernames and passwords are being requested later on, input validation needs to be put in place to ensure that hackers cannot use these fields for untrusted data. The best example of this are SQL injections. With SQL injections, hackers can enter codes in the input fields that will give them access to the Artemis Financial application, design, and/or database (Manico & Detlefsen, 2014, p. 180). Security measures such as restricting character lengths in these fields, whitelisting accepting characters (ex: alphanumeric only), and blacklisting unacceptable characters (ex: symbols) can hinder hackers who want to use SQL injections to attack Artemis Financial (Manico & Detlefsen, 2014, p. 16-17).

## 

## 7. Summary

In order to refactor the code, I first attempted to set up RESTful API in order to successfully display data and the checksum using a localhost connection. This process involves a focusing on the security of APIs, client/server, cryptography, and code quality. I added a layer of security for the API by TLS/SSL and HTTP protocol (Manico & Detlefsen, 2014, p. 7). By doing this, a secure communication channel is established between the client and the server. We want Artemis Financial to know that when their patrons connect to their website and transfer files that it is not being watched by a hacker wishing to steal valuable information. Another layer that was added to the security was the certificate. Upon the secure connection being made, the certificate is used to confirm to visitors of the site that the communication between their endpoint and the other is protected via an encryption (Manico & Detlefsen, 2014, p. 140-141). This was accomplished by setting localhost as the only URL acceptable for a connection within the Subject Alternative Name (SAN). It will be comforting for Artemis Financial’s patrons to have confirmation in the form of a certificate because this allows them to read details about the connection’s security by merely selecting the lock next to the web address. This leads to the next layer of security which is cryptography. The certificate and the communication between client/server are encrypted using complex algorithms composed of a breakable level of randomized bits. Using encryption algorithms that include keys and hash functions makes it difficult for hackers to hijack a connection by guessing the unique encryption bits assigned to it. Finally, the last but probably most important layer of security is code quality. By assessing existing vulnerabilities in the code and performing manual reviews to ensure that the code is doing what it is meant to do, the web application is able to operate more securely.

As time progresses, these security layers will need to be maintained in order to protect against future threats and vulnerabilities. One of the best, most reliable ways to maintain security is to update regularly. Each day the National Institute of Standard Technology and other companies like OWASP, the MITRE Corporation, and Oracle bring to the forefront new vulnerabilities in software, applications, etc. These sites should be monitored frequently to keep the security team knowledgeable about new threats, vulnerabilities, and how to fix them. Another thing that needs to be maintained are certificates. Certificates contain an expiration date, so it is important to stay on top of them or users will be notified upon accessing the web application that their connection is no longer secure.

**References**

Dierks, T, & Rescorla E. (2008, August). *The transport layer security (TLS) protocol version 1.2*. Oracle. <https://www.rfc-editor.org/rfc/rfc5246.html>

Manico, J., & Detlefsen, A. (2014). *Iron-clad java: building secure web applications*. McGraw-Hill Education.

Menezes, A., & Stebila, D. (2021). The advanced encryption standard: 20 years later. *ieee security & privacy*, *19*(6), 98–102. <https://doi-org.ezproxy.snhu.edu/10.1109/MSEC.2021.3107078>

Offermann, K. (2022, June 17). *Solutions to project problems*.

Oracle. (2017). *Java security standard algorithm names*. Oracle. <https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names>

Reddy, N. (2022, April 27). *Cryptographic hash function in java*. GeeksforGeeks. <https://www.geeksforgeeks.org/cryptographic-hash-function-in-java/>

The OWASP Foundation. (2010, August). *OWASP secure coding practices quick reference guide*. OWASP. <https://owasp.org/www-pdf-archive/OWASP_SCP_Quick_Reference_Guide_v1.pdf>