

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/10/23** | **Trent Hesler** |  |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Trent Hesler

## Algorithm Cipher

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

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

Artemis Financial is looking for a solution to secure communications between their clients and their public web interface. To meet these security needs, I determined that it was best to use the SHA-256 encryption cipher. Data encryption is the process of securing data by encoding or “scrambling” it in such a way that it can later be decoded using a password or key. With strong enough encryption, only authorized parties can decode and read the encrypted data.

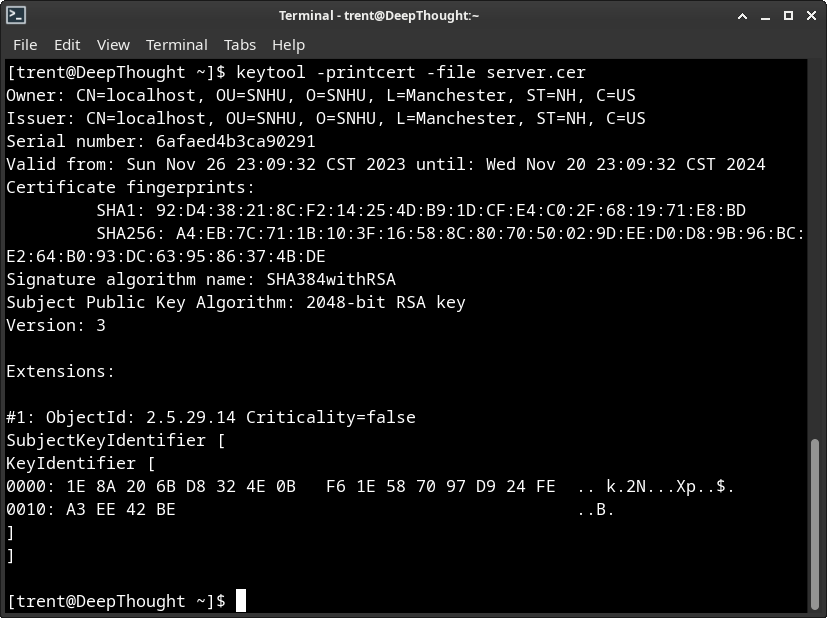
Securing communications to and from the public web interface is necessary because, as data is transferred over a network, it is susceptible to interception by potentially-malicious third parties. Unless this data is securely encrypted, unscrupulous individuals may be able to read and/or manipulate it. Encryption ensures that only the sender and recipient are able to understand the messages being passed to one another over the network, and that the messages’ contents have not been altered in transit. The SHA-256 encryption algorithm makes use of Java’s built-in random number generator to securely create checksums. These checksums are non-reversible representations that can be used to verify the integrity of data passed over the network.

SHA-256 is an asymmetric 256-bit encryption cipher. “Asymmetric” means that it uses a pairing of one or more public keys to encrypt and one private key to decrypt information. This is in contrast to what are known as “symmetric” encryption schemes, which use the one key for both encryption and decryption of data. The “256-bit” aspect of this cipher means that it uses a key that is 256 characters (letter, numbers, and symbols) long. This length is desirable because it renders it extremely difficult for a would-be attacker to guess the key.

An asymmetric cipher is ideal for this application because it allows for the issuance and revocation of public keys as needed. Authorized individuals and applications can be issued public keys to allow them to submit encrypted data to Artemis Financial. When it is no longer necessary or prudent for these key-holders to submit encrypted data (i.e. an employee leaves the company or an application is retired), their access can be cut off by revoking their public key. Further, an asymmetric encryption cipher ensures that only those with access to the singular private key can decrypt the data and read its contents.

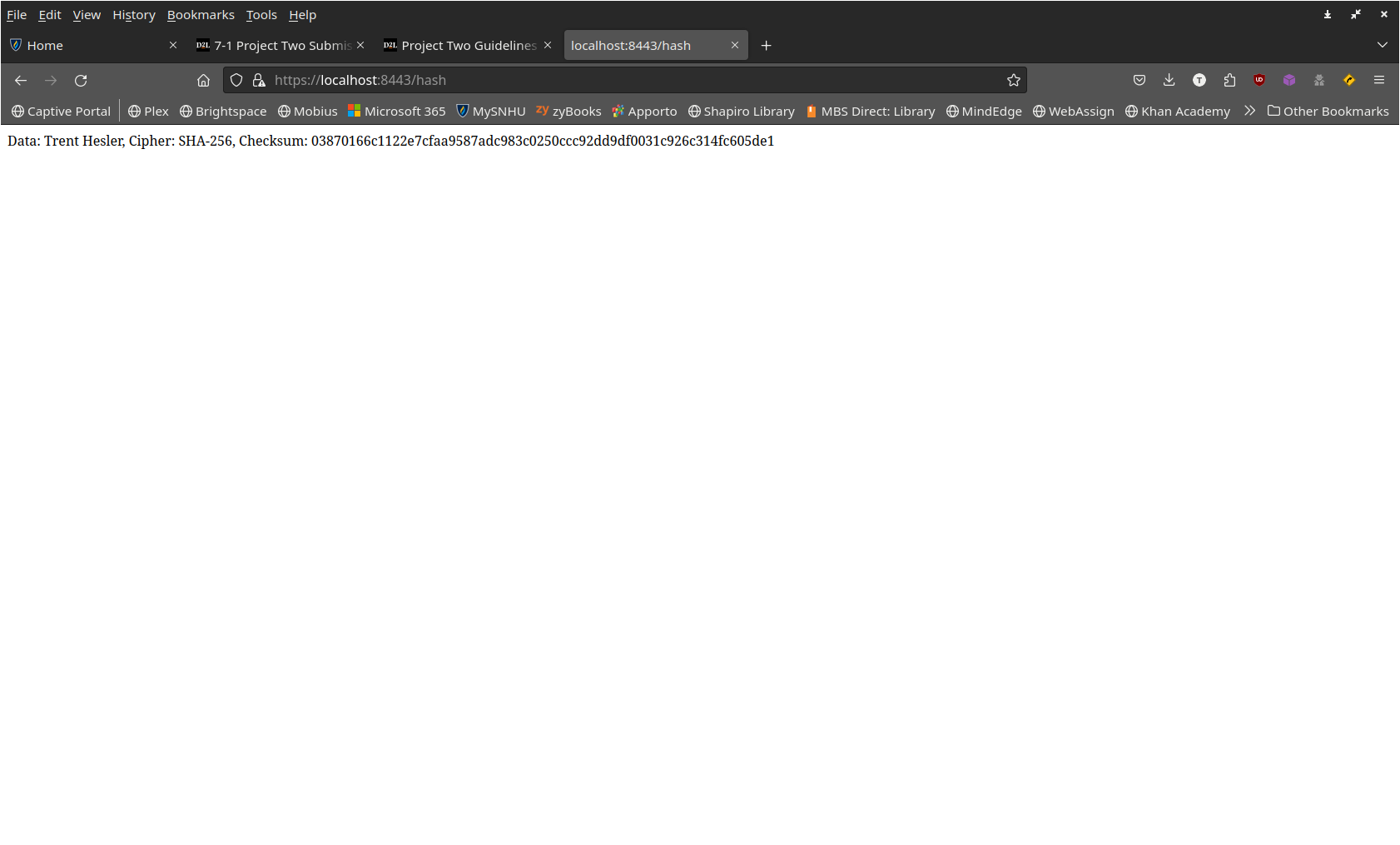
## Certificate Generation

Insert a screenshot below of the CER file.



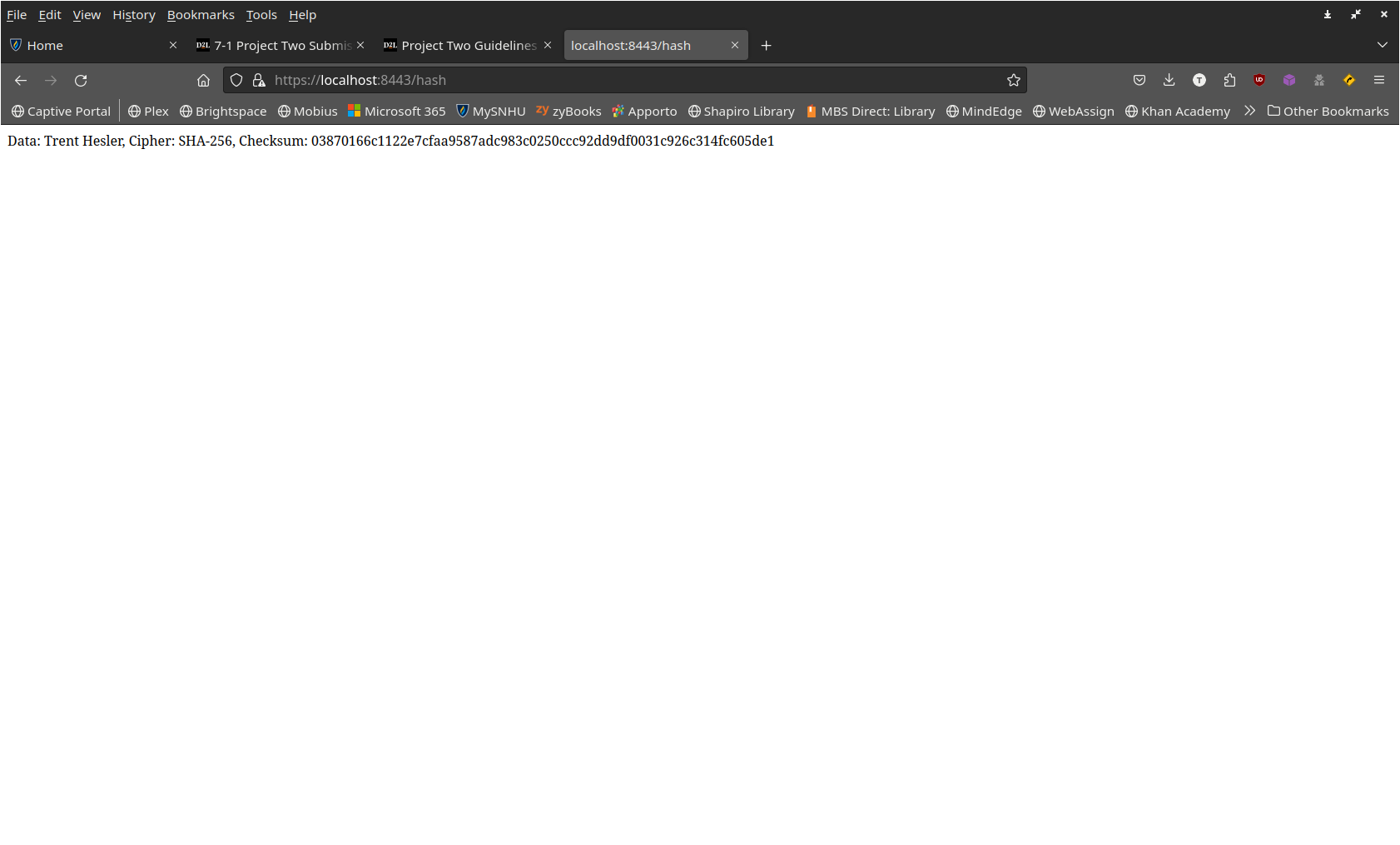
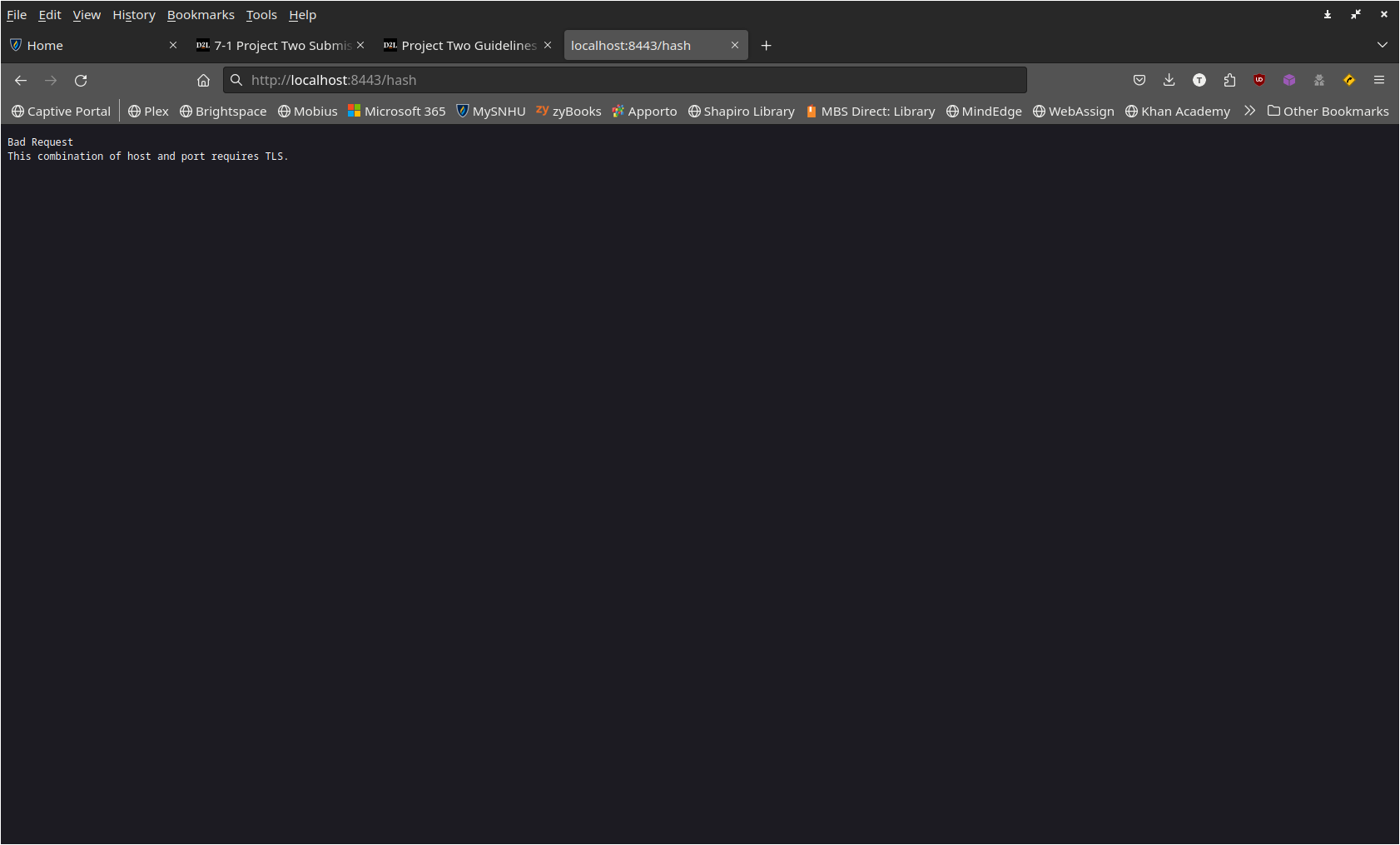
## Deploy Cipher

Insert a screenshot below of the checksum verification.



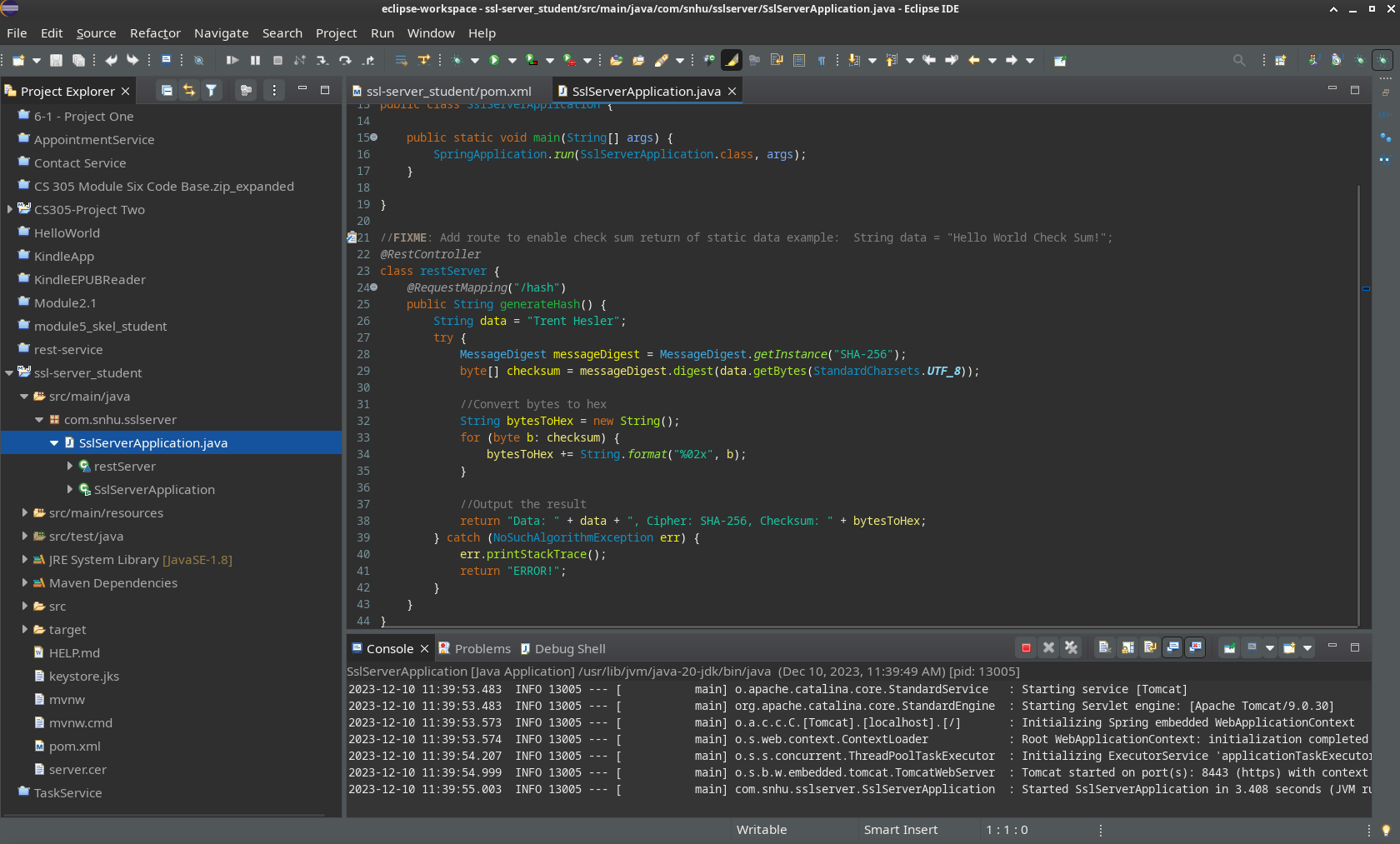
## Secure Communications

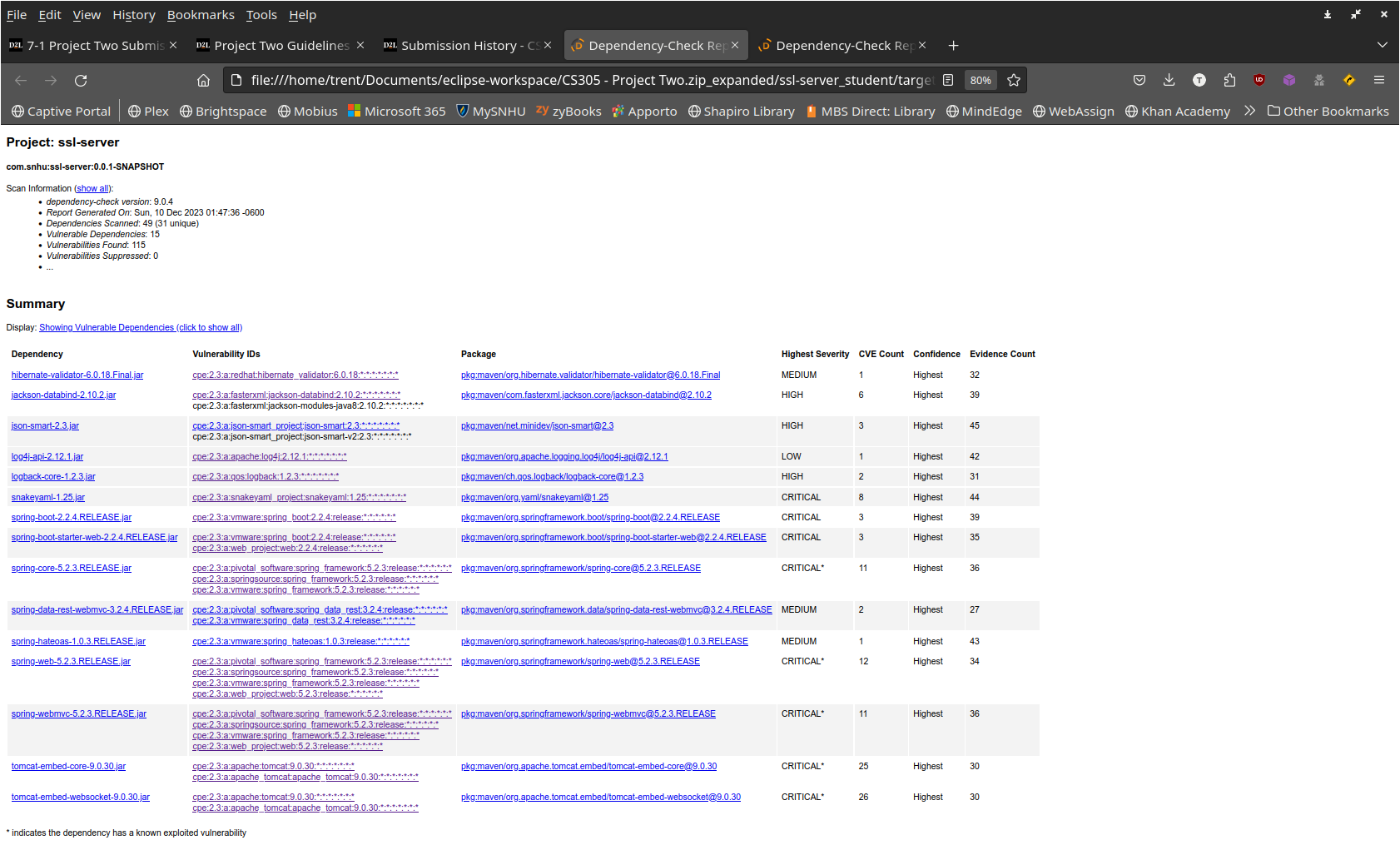
Insert a screenshot below of the web browser that shows a secure webpage.

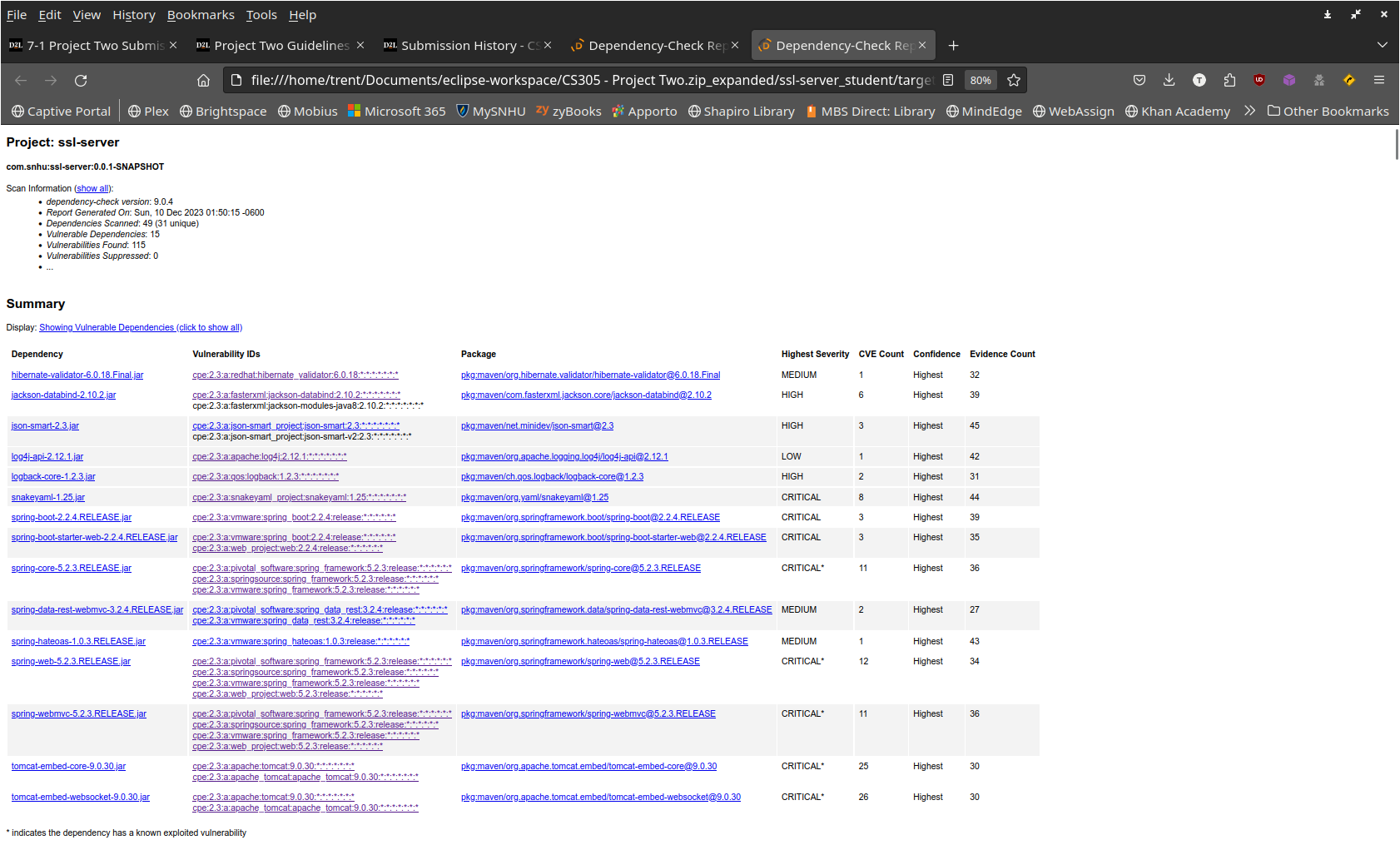


## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.

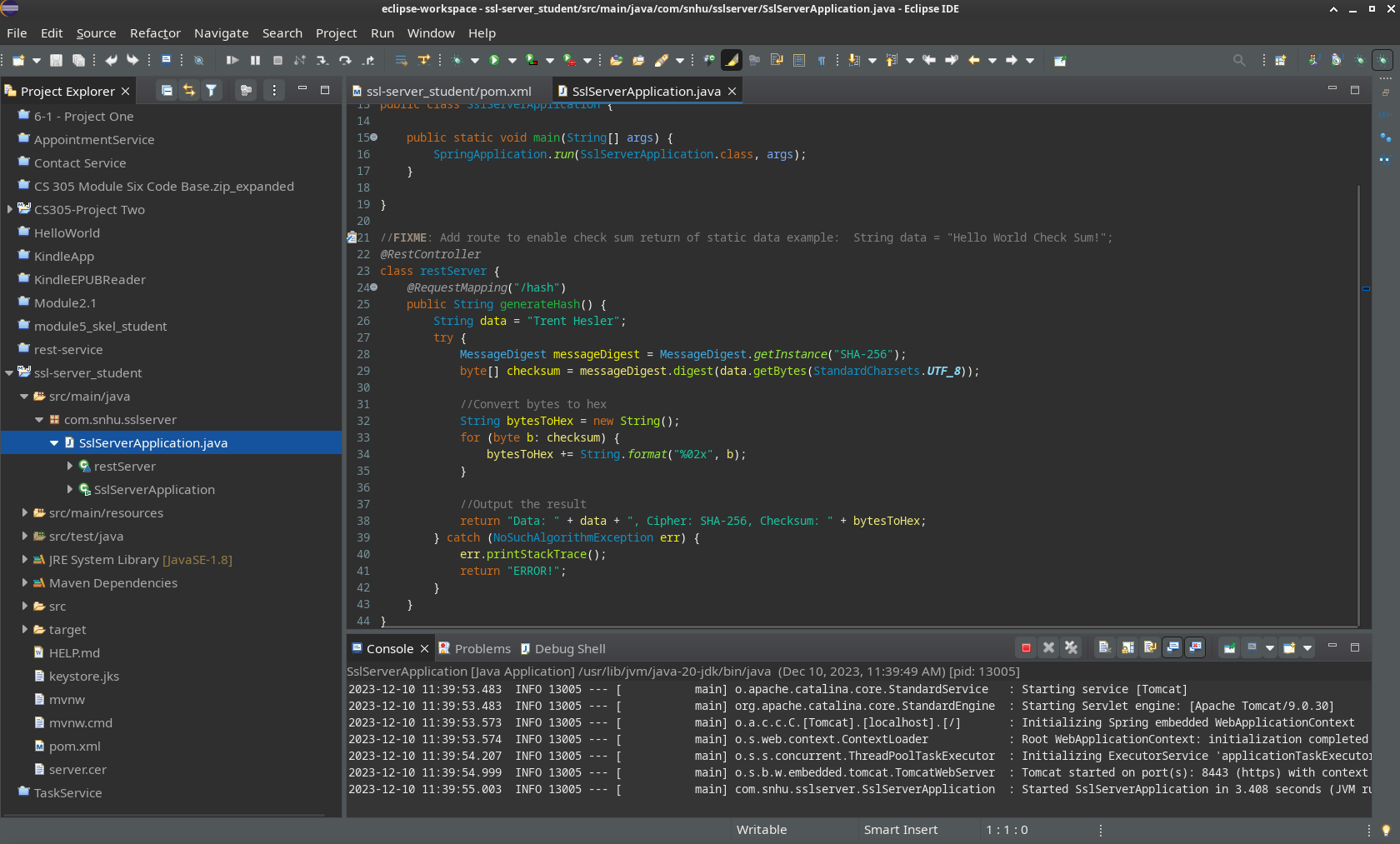






## Functional Testing

Insert a screenshot below of the refactored code executed without errors.



## Summary

For code refactoring, I focused on creating a secure RESTful API for the server application. This API generates an SHA-256 hash of a given data string and provides a web interface for the user to view the hashed data. This web interface uses SSL encryption to keep the user session secure and private, in accordance with industry standard best practices. SHA-256 is a very secure encryption algorithm, as there is only an infinitesimal chance of collisions (two data strings producing the same checksum). SHA-256 is also considered to be secure against brute force decryption attacks performed by quantum computers, which will likely become a greater threat as this emerging technology progresses (O’Shea, 2022).

## Industry Standard Best Practices

As described above, I was careful to stick to industry standard best practices when refactoring the web application code. The application now employs strong SSL encryption to secure user sessions and data can be verified using the SHA-256 checksum. I also ran the application through the OWASP dependency vulnerability check before and after my refactoring to ensure that no vulnerabilities were introduced. While there were some vulnerabilities already present before I added my RESTful API, no additional vulnerabilities were present following its addition. Going forward, I would recommend that the main development team ensure that all project dependencies are as up-to-date as possible. I would also recommend regularly performing additional dependency vulnerability checks to identify potential issues. Once vulnerabilities are identified, the team should immediately work to mitigate these risks.

**References**

O’Shea, D. (2022, April 29). *AES-256 joins the quantum resistance*. Fierce Electronics. <https://www.fierceelectronics.com/electronics/aes-256-joins-quantum-resistance>