****

# 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** | **6/22/25** | **Richard Adler** |  |

## 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

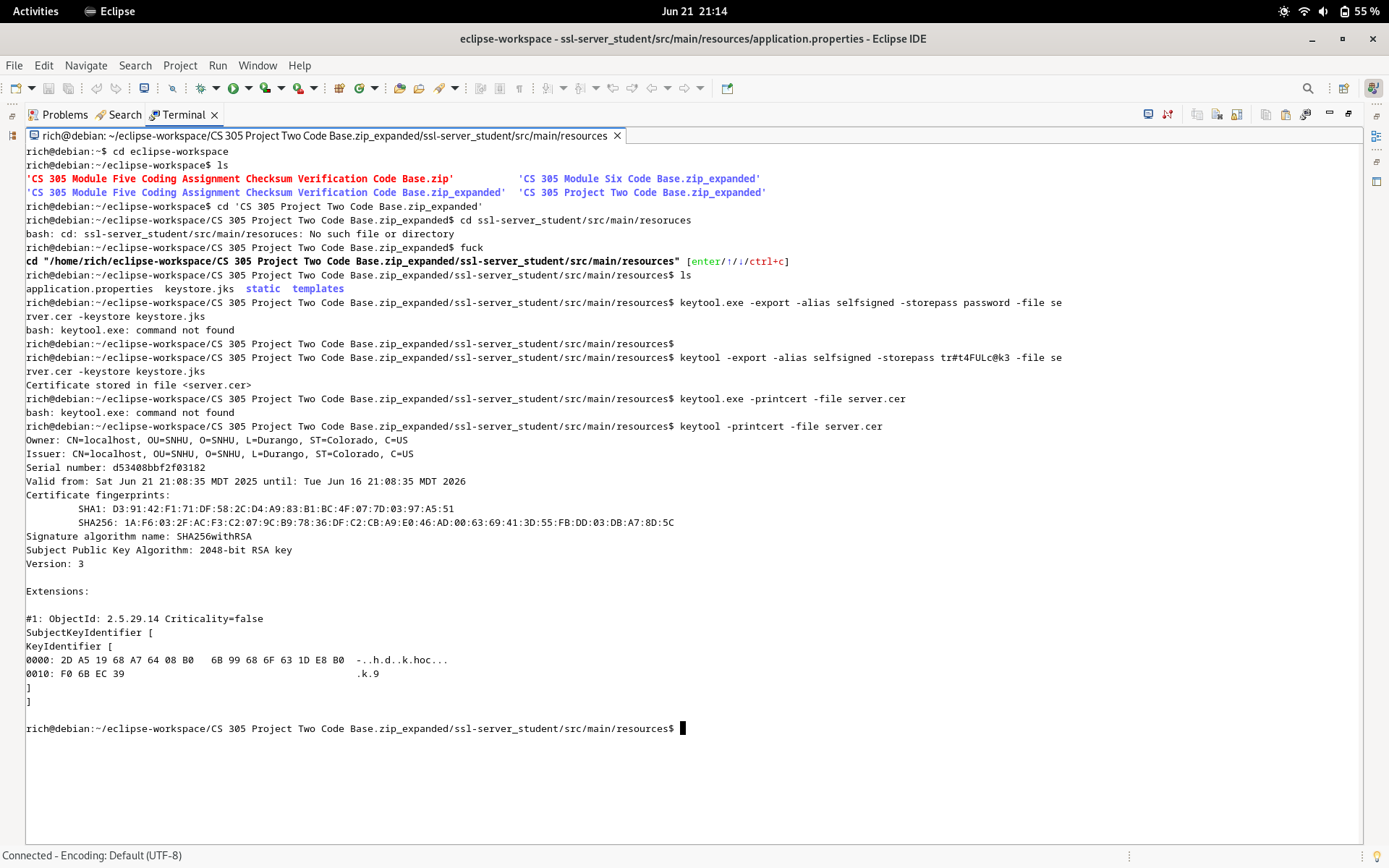
Richard Adler

## Algorithm Cipher

As a financial institution, cyber security is at the forefront of any online endeavors that Artimes Financial undertakes. Financial data is sensitive, subject to government regulations, and if leaked, poses massive consumer trust issues with the business. I have chosen the SHA-256 encryption to fortify the transfer of data because it is highly collision resistant, generates a fixed size hash making decryption extremely difficult and impractical without a valid key, and is designed to be irreversible, maintaining data integrity even if the hash is intercepted. The large 256-bit size hashes make collisions, when two pieces of data share the same hash value, uncommon enough to be practically impossible, meaning that once a piece of data is encrypted, it can be compared to the expected value for its encryption, and be verified as authentic. It is widely used for its robust security and impenetrability. Hash functions utilize a form of randomness generated from a fixed key and obfuscate data to a point where it is no longer readable. Random numbers are essential in cryptography to generate strong encryption keys that cannot easily be guessed or brute forced. Although ciphers have existed for a long time in Caesar ciphers and other forms of text encryption, modern day computing uses complex algorithms to heighten this perceived randomness into data that is so illegible after encryption that it may as well be random data without being to check it against an expected value or decrypt it. Initially, the US government established initial encryption standards before modern day algorithms such as AES and SHA-256 dominated the field for their impregnability.

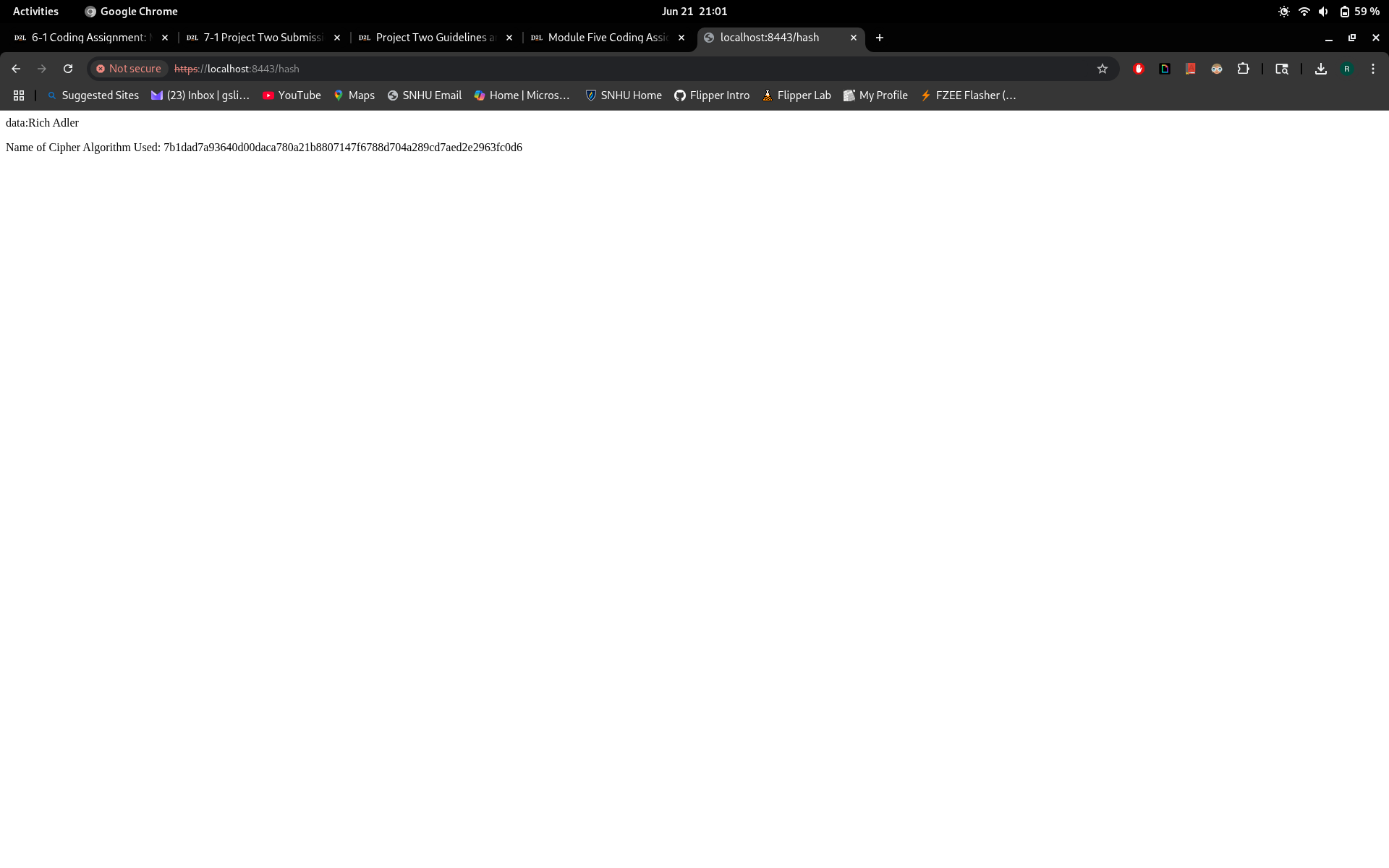
## Certificate Generation

Insert a screenshot below of the CER file.



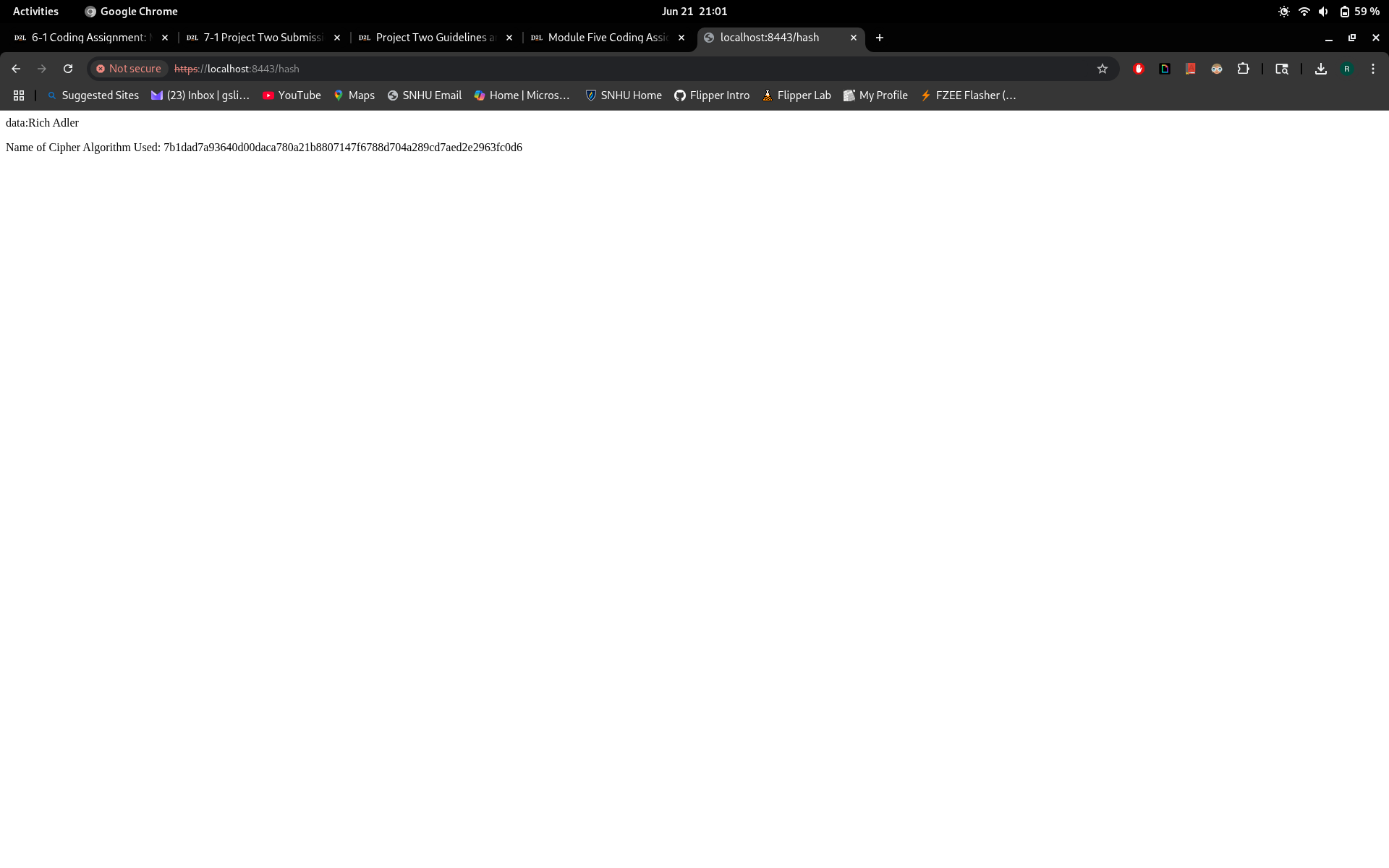
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

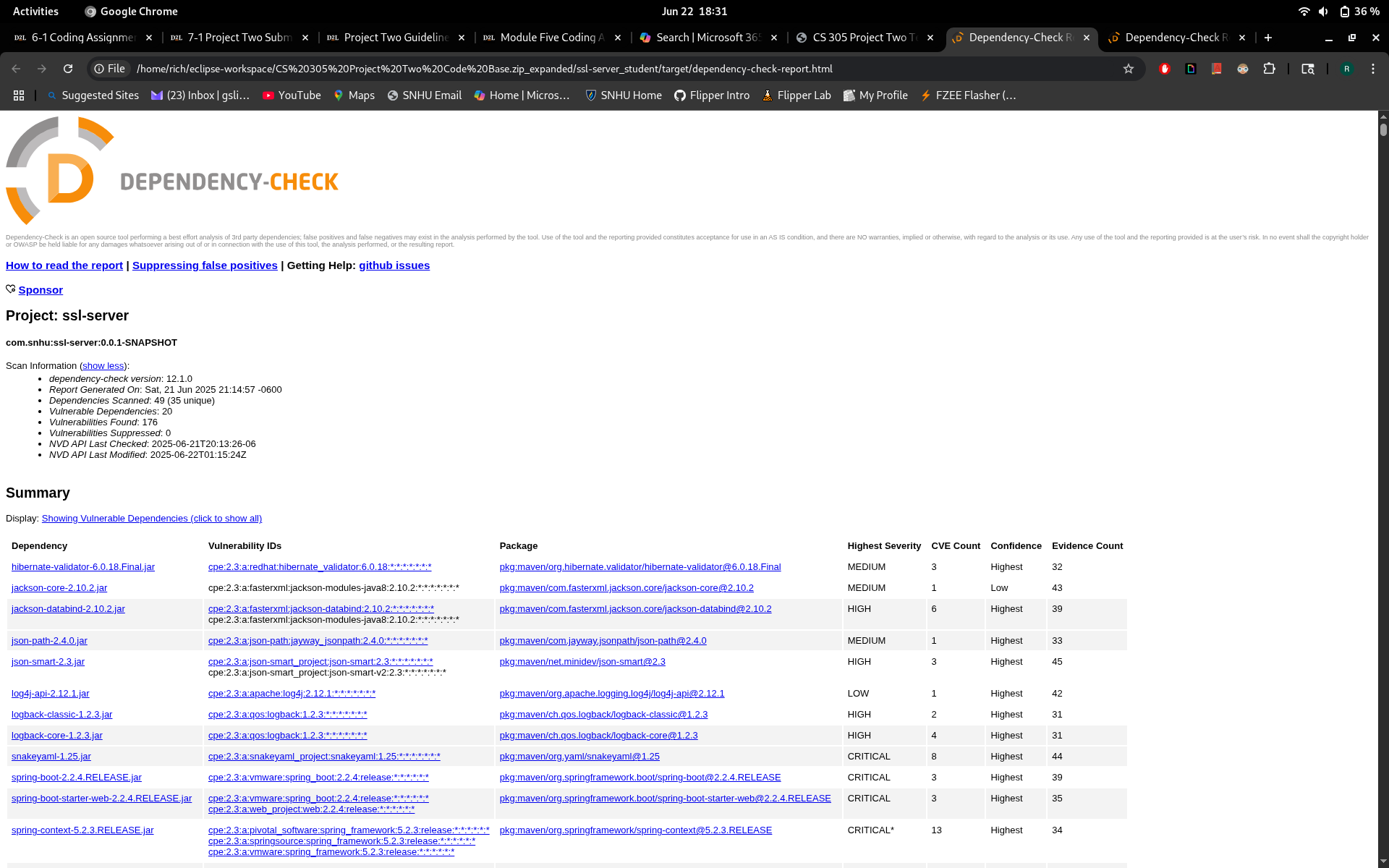
Insert a screenshot below of the web browser that shows a secure webpage.

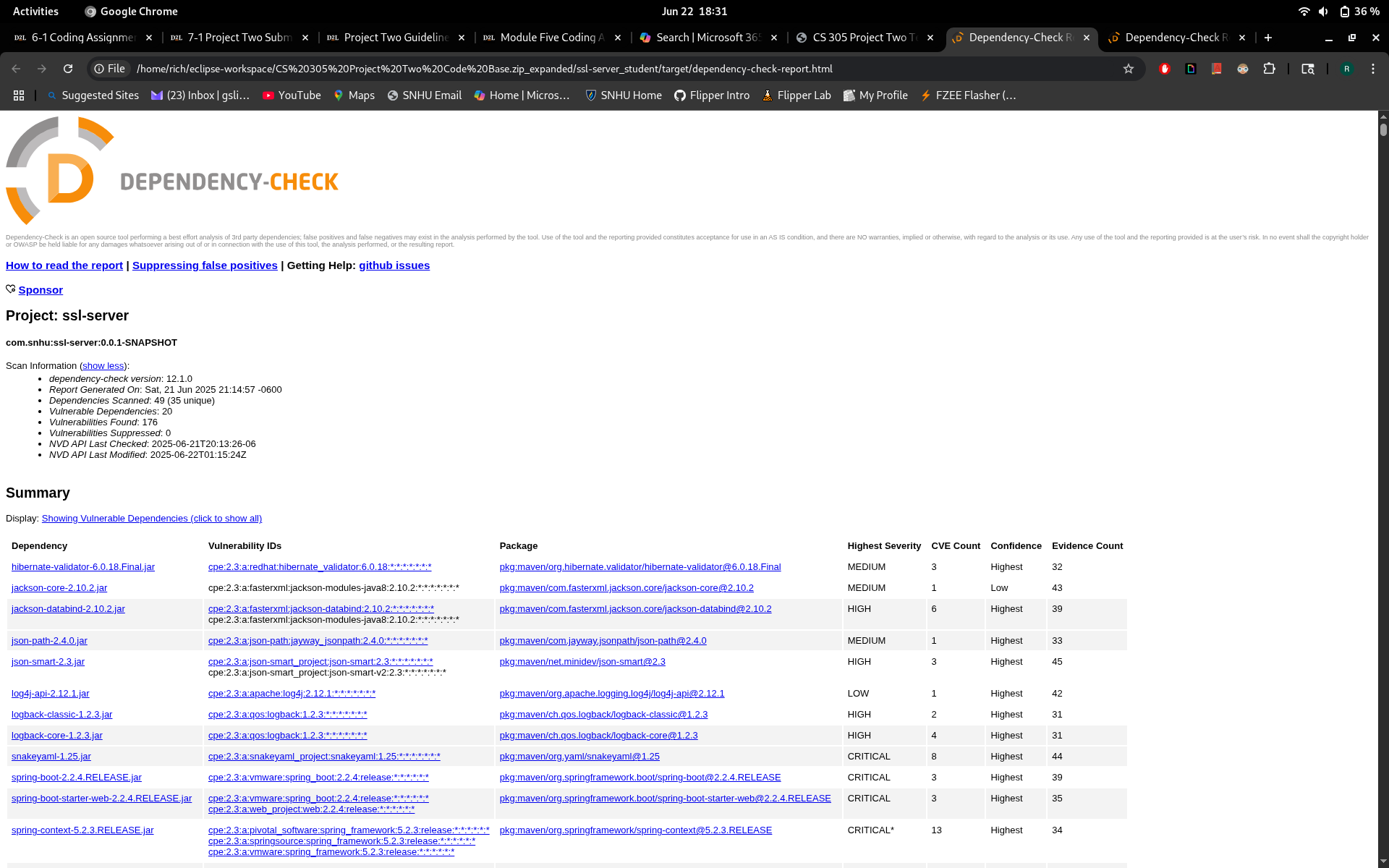


## Secondary Testing

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

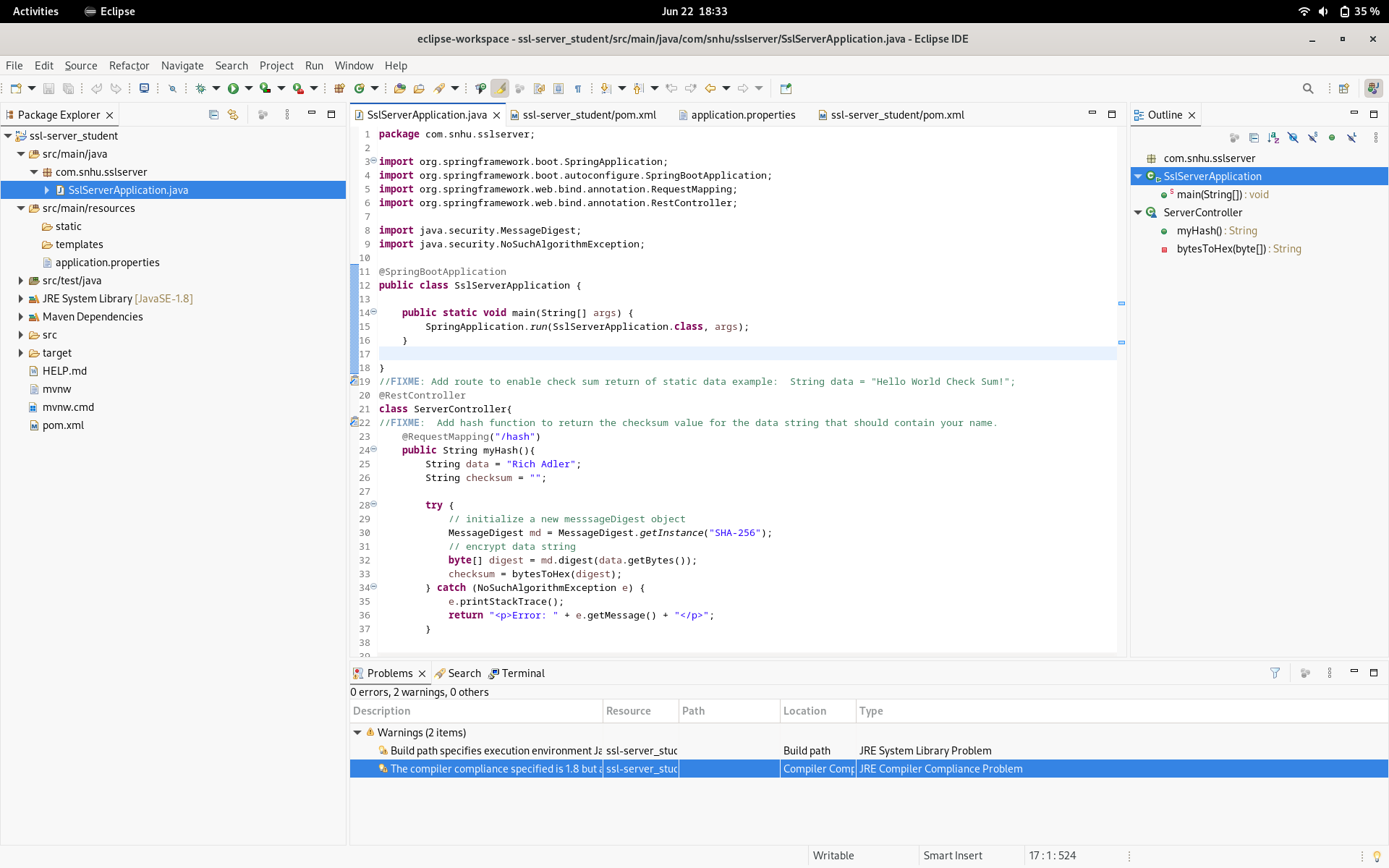
Dependency check before making any changes to the codebase

Dependency check from after refactor

I understand that these look the same, but no new vulnerabilities were added during refactoring, I included both generated dependency checks in the zip file.

## Functional Testing

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



## Summary

RESTful APIs are considered secure for their ease of use and prevention of any bad client-side requests by forcing interactions through very specific requests, waiting to process each request as it comes in. SHA-256 is a strong encryption algorithm that is highly resistant to collisions and only encrypts one way. Data is irrecoverable after encryption, and because collisions are practically impossible, there is little risk of authenticating data without a previous hash to check against. The code has no errors and follows best practices. The application uses HTTPS via a self-signed certificate, providing a secure connection to our clients and reenforcing their trust in the application. With appropriate certifications, clients can be sure they are dealing with Artimes Financial and be confident in using it. Ensuring the hashing algorithm is working correctly via checksum verification was the next step. This allows Artimes Financial to be confident that their system can protect consumer data without issue. Although outside the scope of my role, I feel it necessary to mention that the overall system vulnerabilities are due to many deprecated dependencies and an outdated Java version. I encourage Artimes Finincial to invest resources in a total system update, as many dependencies are no longer publicly supported and places strain on their backend to mitigate these security concerns. Whatever cost is incurred in the update will pay off in dividends in the future, as updating will mitigate many, if not all these concerns without requiring enormous effort on the backend day over day, and I would be happy to consult on this issue following this project to ensure server side infrastructure is secure.

## Industry Standard Best Practices

By utilizing best industry practices such as ensuring that the encryption algorithm was working correctly, grooming code to be clean, legible, and functional, converting the application to utilize HTTPS and keeping as up to date with modern security offerings as I can in my role, and ensuring that I offer my opinion on matters of security concern outside of my role where it is relevant, I can confidently say that I have made Artimes Finacial's application as secure as possible in my role. Best practices are considered best because they lead to the desired outcome, robust security with as few vulnerabilities as possible. By following these practices, double checking code for any common errors or unsecure practices and ensuring that the program takes advantage of as many new security developments as possible, for example, using HTTPS instead of HTTP, secure software is developed from the beginning. This saves countless hours later in attempting to mitigate vulnerabilities and avoids some protentional future problems of losing user trust and exposing the company to the risk of fines and legal action.