****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_heading=h.30j0zll)

[**Client 3**](#_heading=h.1fob9te)

[**Instructions 3**](#_heading=h.3znysh7)

[**Developer 4**](#_heading=h.2et92p0)

[**1. Algorithm Cipher 4**](#_heading=h.tyjcwt)

[**2. Certificate Generation 4**](#_heading=h.3dy6vkm)

[**3. Deploy Cipher 4**](#_heading=h.1t3h5sf)

[**4. Secure Communications 4**](#_heading=h.4d34og8)

[**5. Secondary Testing 4**](#_heading=h.2s8eyo1)

[**6. Functional Testing 4**](#_heading=h.17dp8vu)

[**7. Summary 4**](#_heading=h.3rdcrjn)

[**8. Industry Standard Best Practices 4**](#_heading=h.26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **08/20/23** | **Jeremy King** |  |

## 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

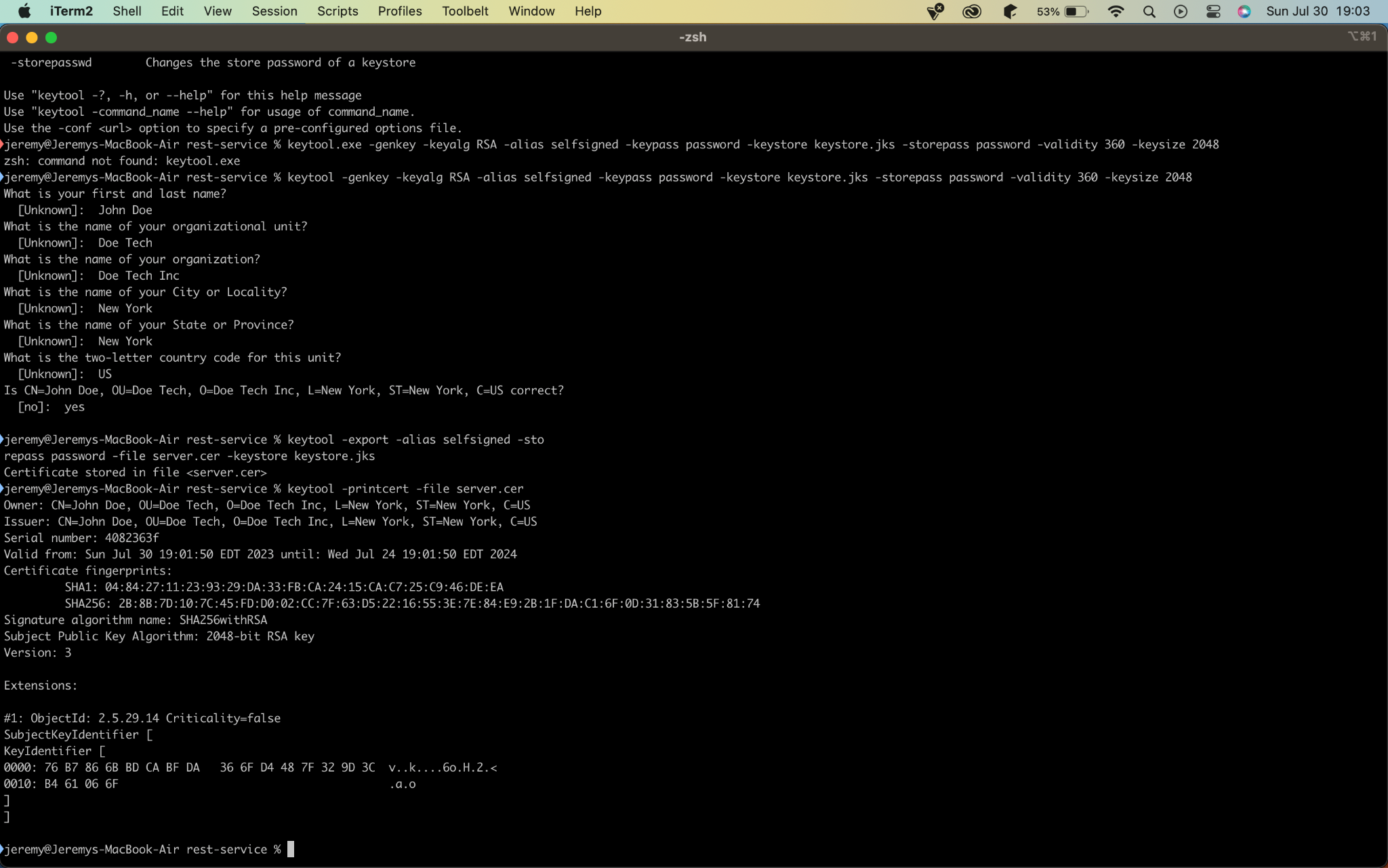
Jeremy King

## Algorithm Cipher

SHA 256 is a modern and secure hashing algorithm. It is fast and has not been broken as of yet. I believe it is adequate for generating data hashes for verification. There has not been a demonstration of a collision using this algorithm. Collisions happen when two strings put into the algorithm generate the same hash, if this happens then the algorithm is ”broken” and a malicious attack can be made by modifying data and then tricking the user by showing the hash of the unmodified data.

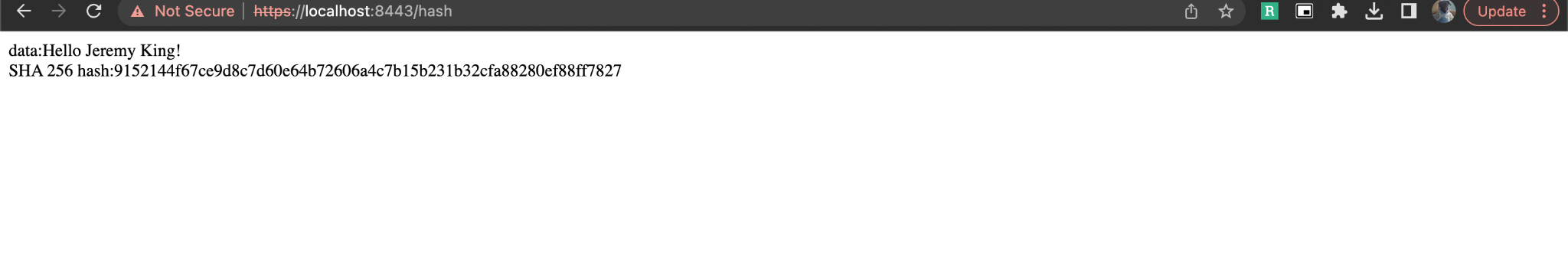
## Certificate Generation

Insert a screenshot below of the CER file.



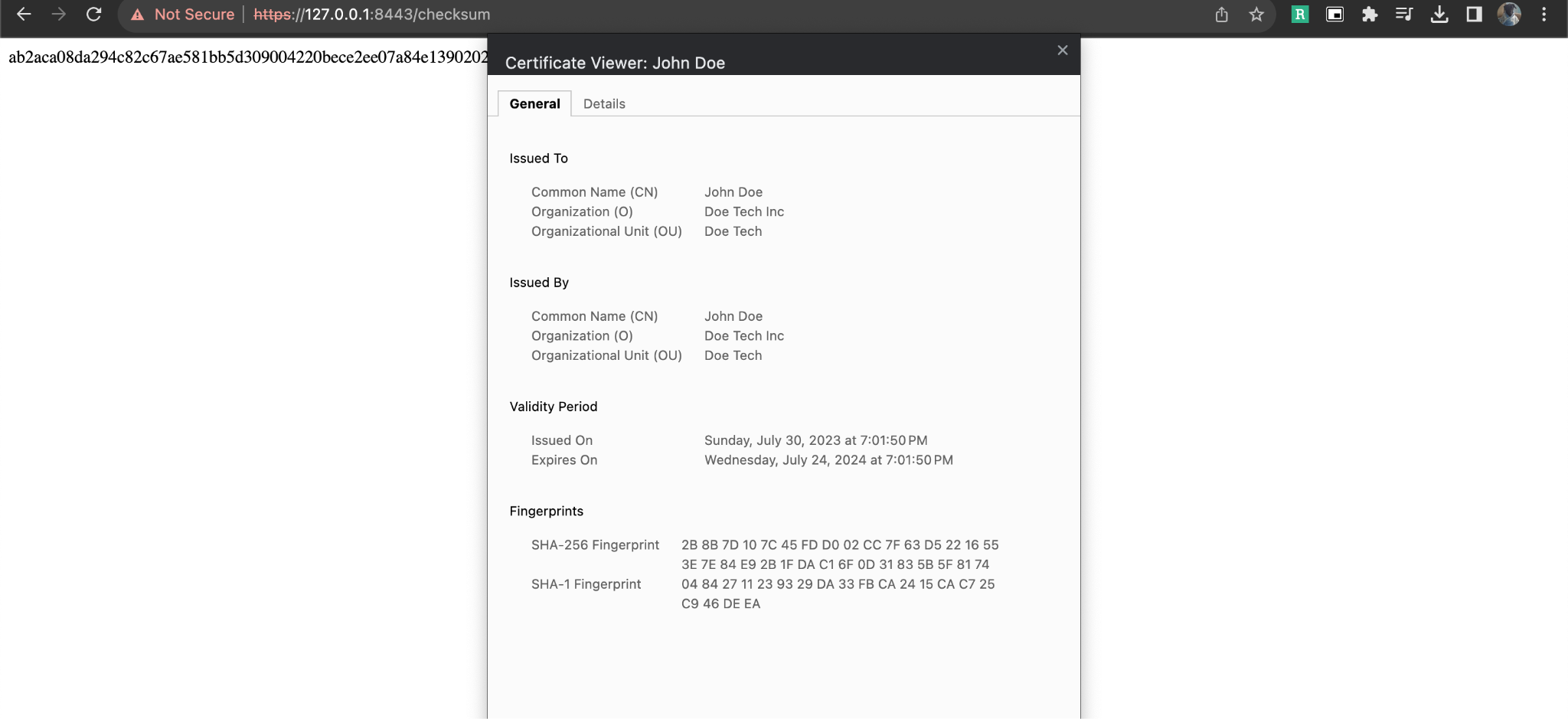
## Deploy Cipher

Insert a screenshot below of the checksum verification.



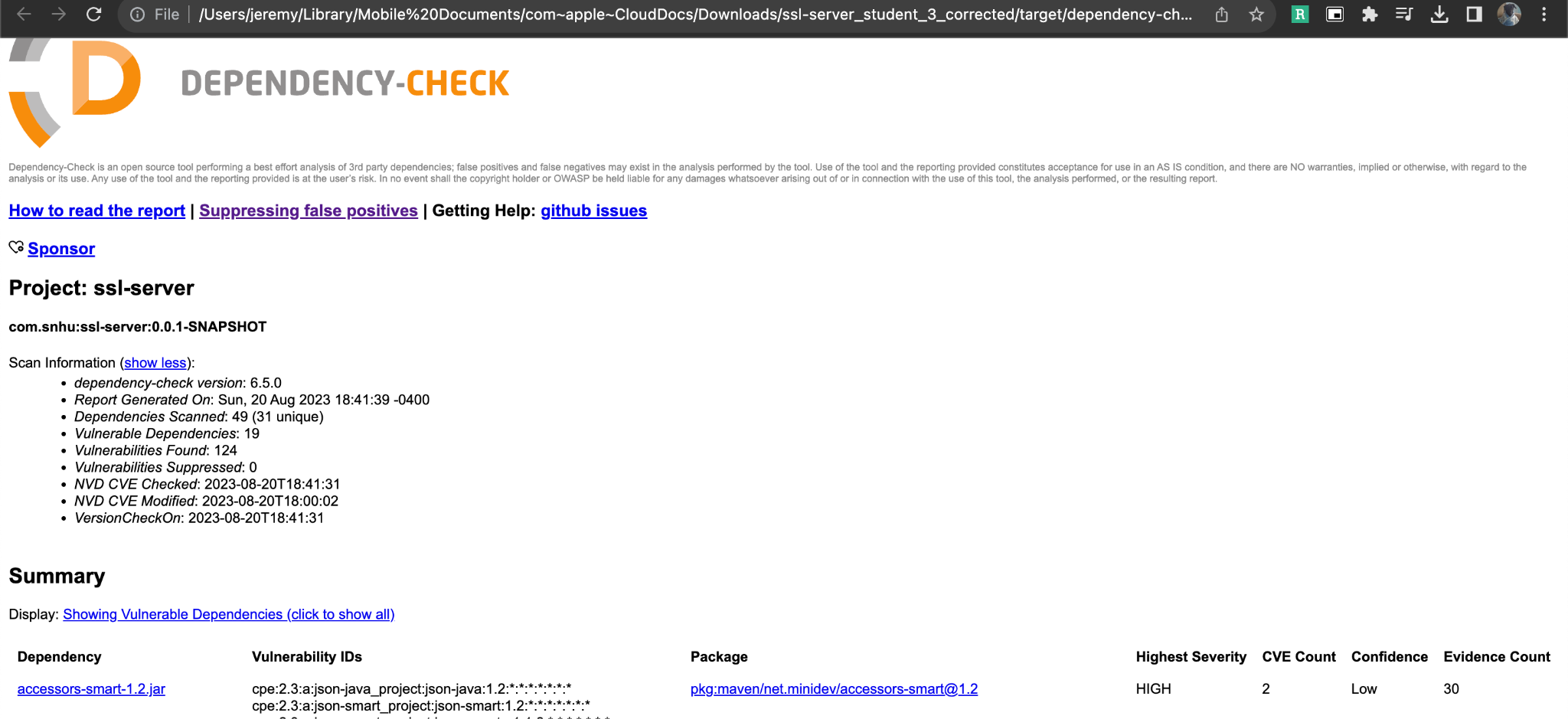
## Secure Communications

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



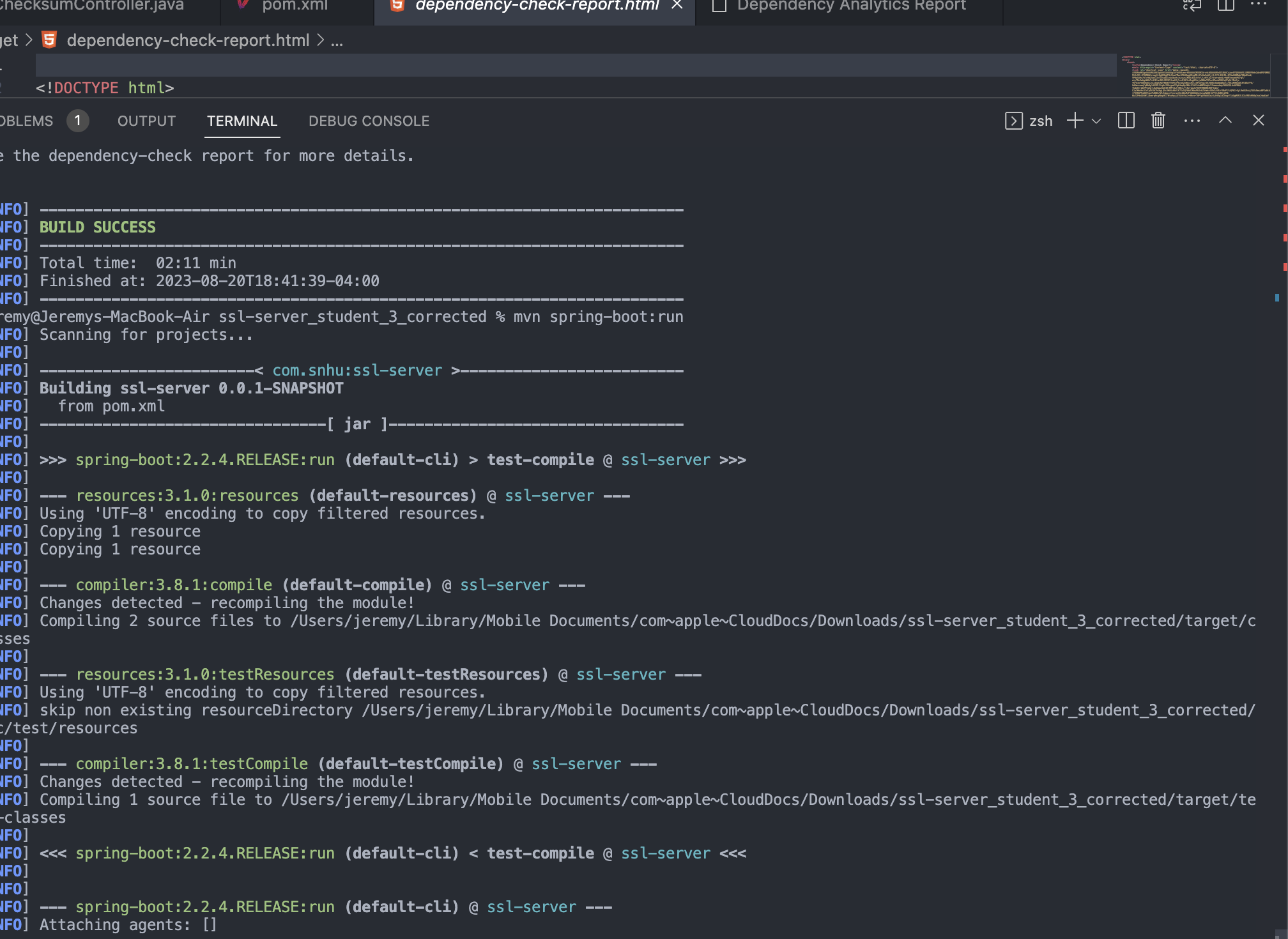
## Secondary Testing

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

* I created a ChecksumController class.
* I added the @RestController annotation to the class, marking it as a Spring Web component. This ensures Spring's component scan recognizes and manages it.
* To specify the base URI for the endpoints within this controller, I introduced the @RequestMapping("/checksum") annotation to the class.

## Industry Standard Best Practices

**HTTPS & SSL Integration**: All data transmissions are encrypted by configuring the application to run over HTTPS.

**Dependency Scanning:** By integrating the OWASP Dependency-Check plugin, I can now detect vulnerabilities in the project's dependencies, ensuring I'm using secure libraries.

**Selective Dependency Inclusion:** I've deliberately excluded potentially vulnerable components like junit-vintage-engine from the dependencies.

**Consistency with Conventions:** This project adheres to the standard Spring Boot structure, making the code more maintainable and predictable.

**Annotation-based Configuration:** I've used annotations to clearly define exposed endpoints, minimizing the risk of unintended data exposure.