****

# Practices for Secure Software Report

**Table of Contents**

Practices for Secure Software Report 1

Document Revision History 3

Client 3

Instructions 3

Developer 4

1. Algorithm Cipher 4
2. Certificate Generation 4
3. Deploy Cipher 5
4. Secure Communications 5
5. Secondary Testing 6
6. Functional Testing 7
7. Summary 8
8. Industry Standard Best Practices 8

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **10/18/2025** | **Kimani Muhammad** | **Added TLS security and performed OWASP dependency check with mitigation.** |

## 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.

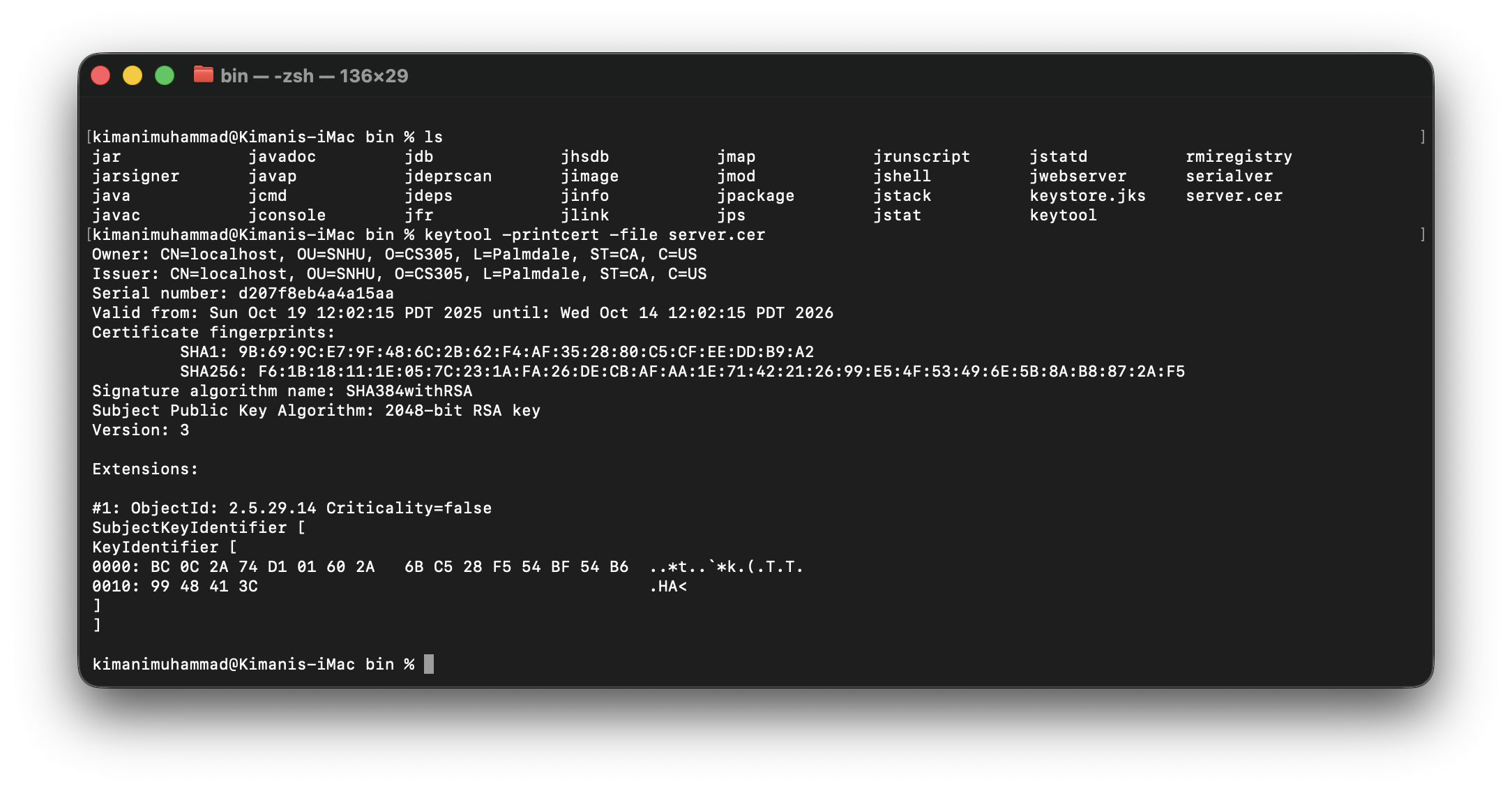
## Developer

Kimani Muhammad

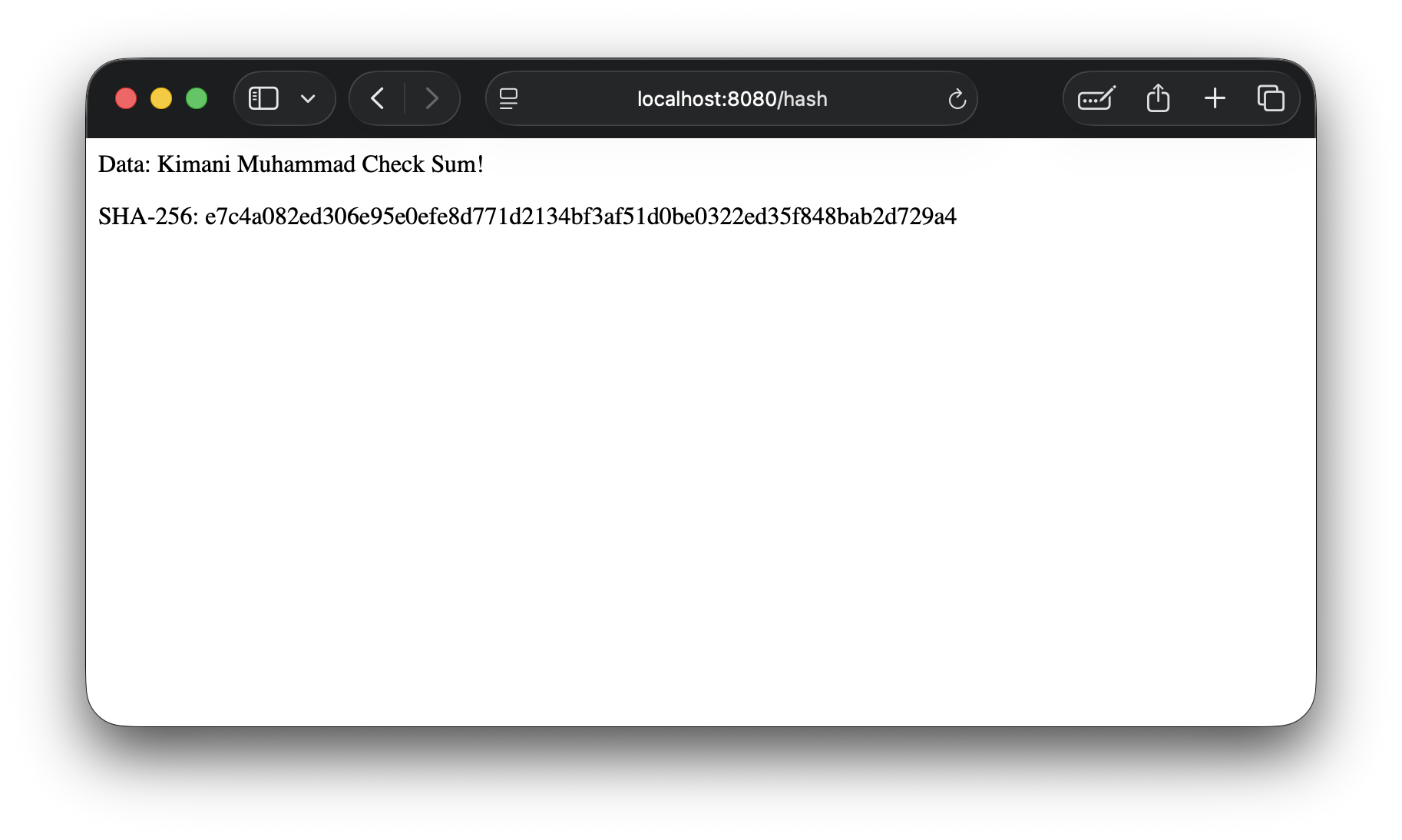
## Algorithm Cipher

In order to verify secure communication with the web application, we’ll use the AES-SHA-256 encryption algorithm. This algorithm uses strong, 256-bit, hashing to verify the file hasn’t been compromised and altered, while keeping the file data private with AES. This is also a symmetric algorithm, so it will use the same keys for encryption and decryption. This is the most secure and recommended standard in use today, recommended by the National Institute of Standards and Technology.

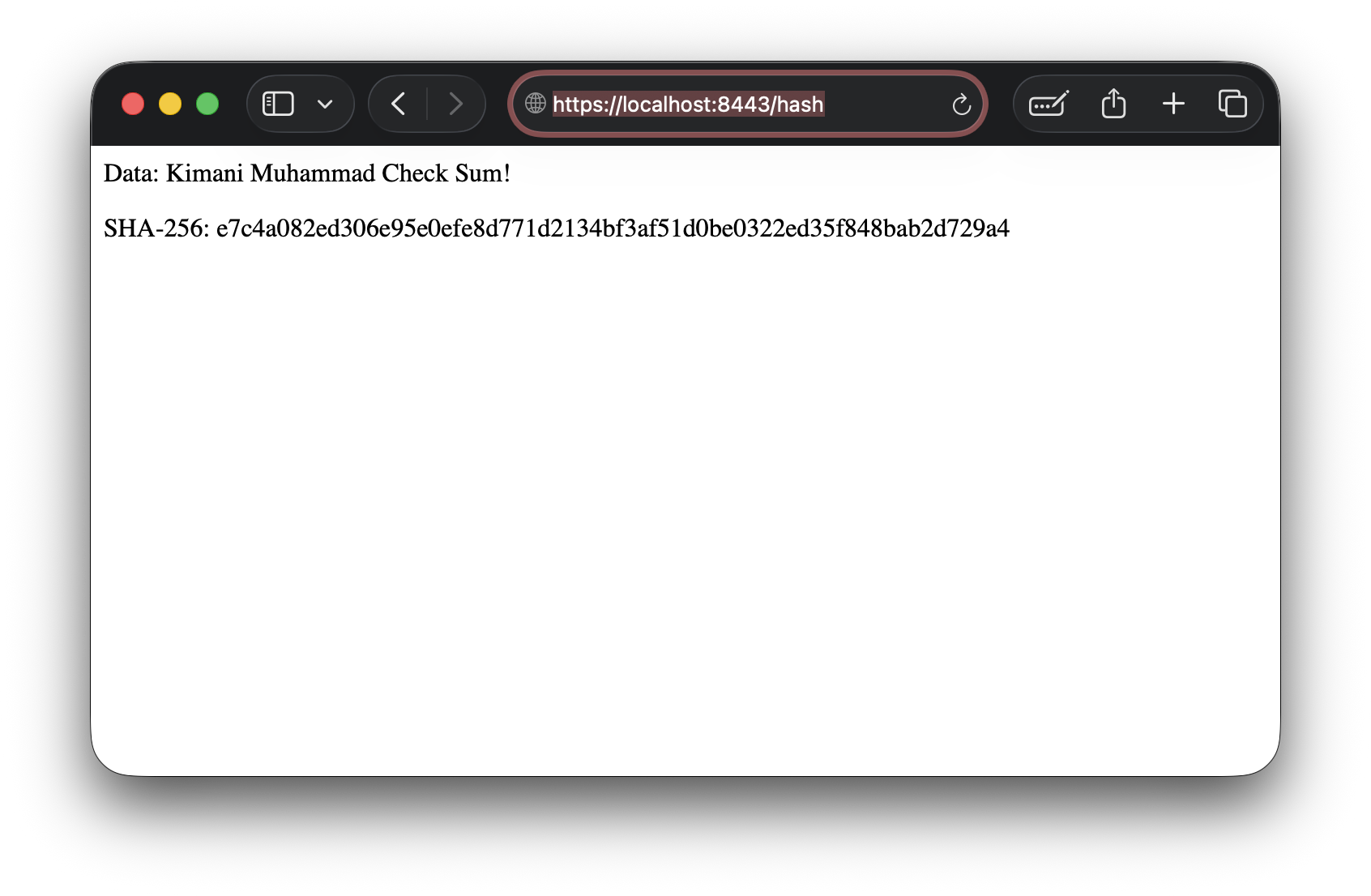
## Certificate Generation

Insert a screenshot below of the CER file.

## Deploy Cipher

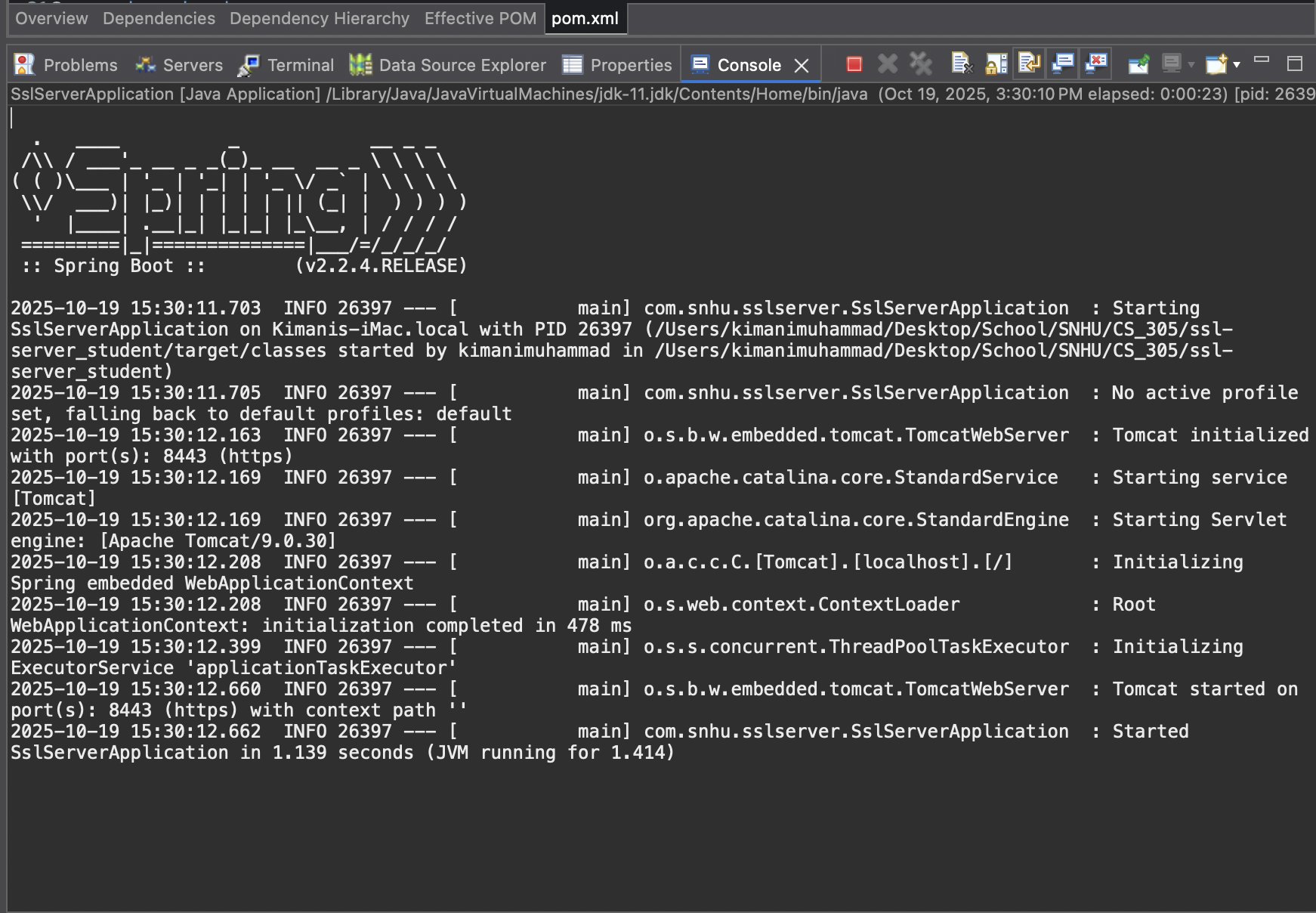
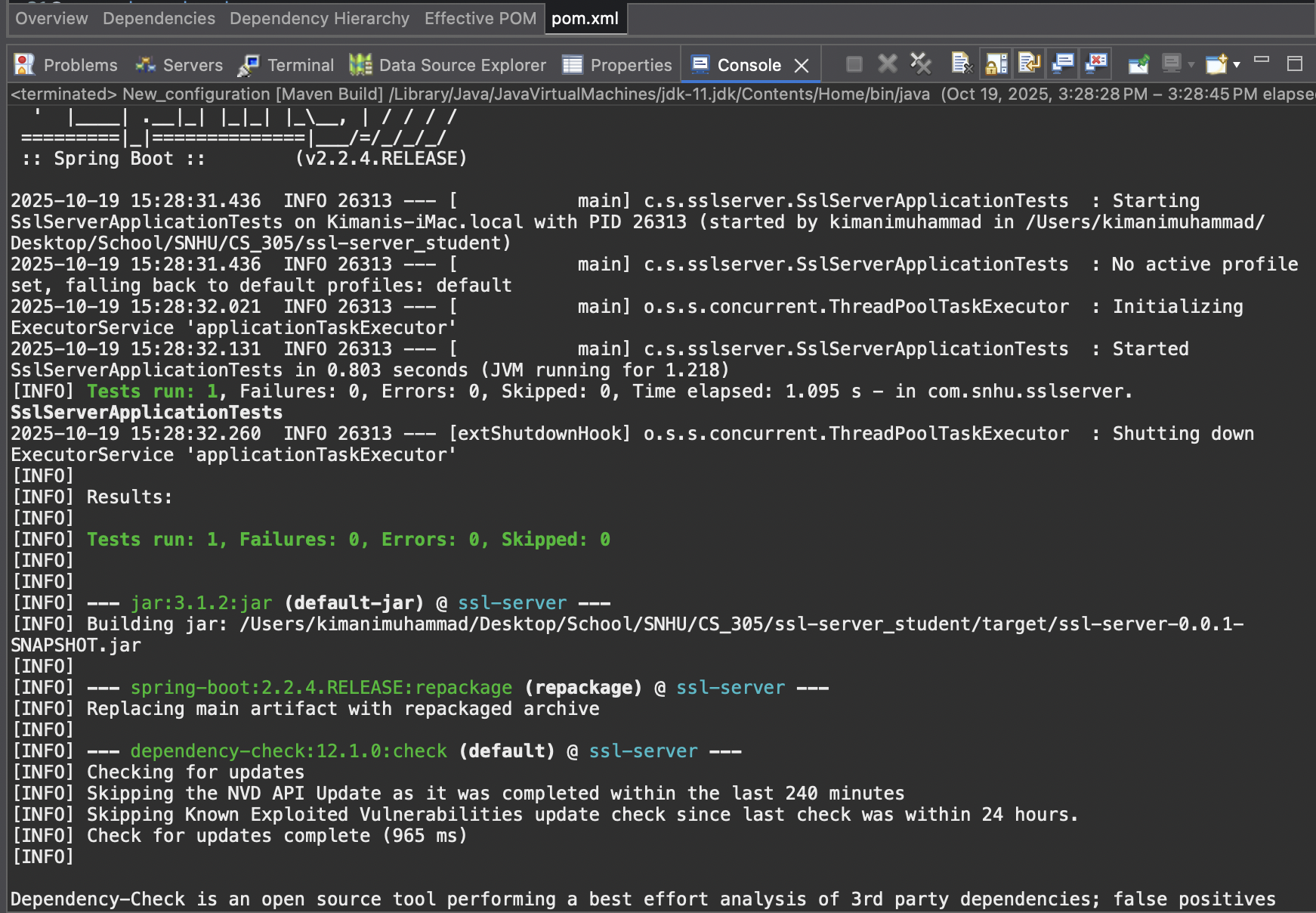
Insert a screenshot below of the checksum verification.

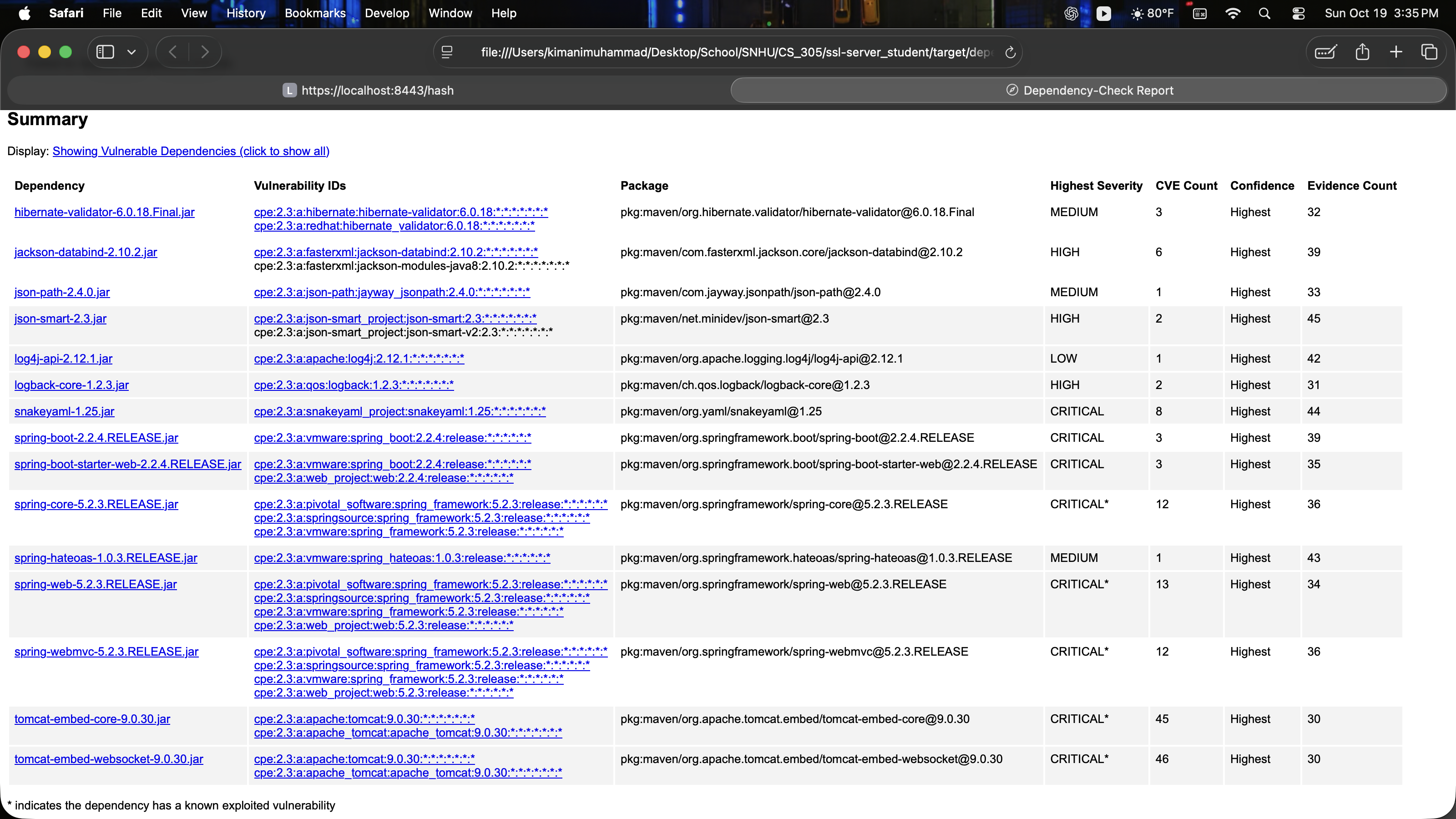
## Secure Communications

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

## Secondary Testing

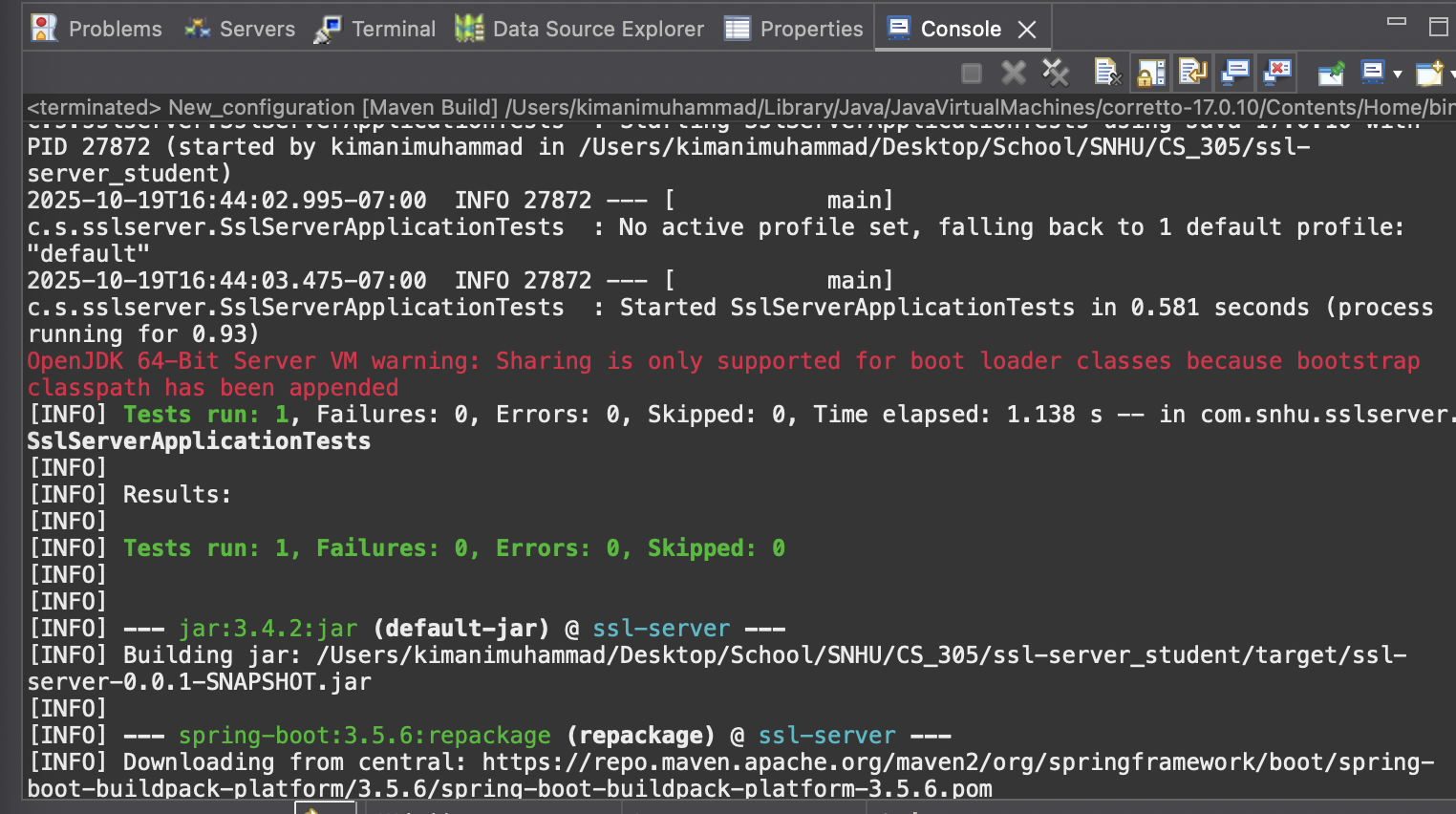
Insert screenshots below of the refactored code executed without errors and the dependency-check report.

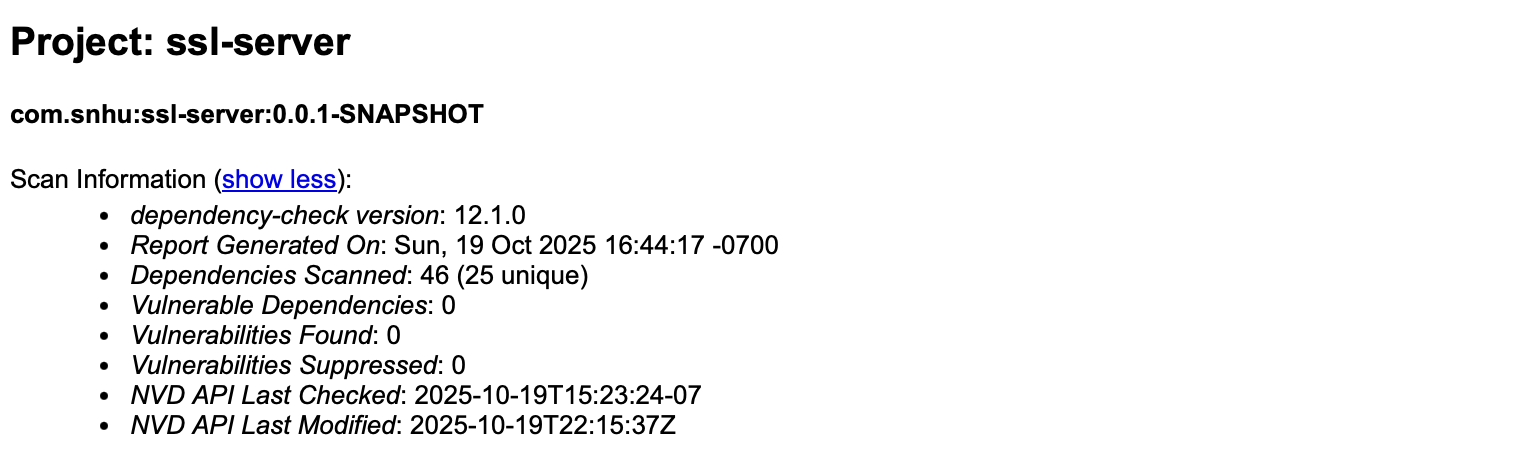




## Functional Testing

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





## Summary

Areas of security that were addressed by refactoring the code included Cryptography, Client/Server Security, Code Quality, and Code Errors. I first generated a self-signed private server certificate using SHA-256 encryption and exported it to the application’s keystore to enable transport layer security (TLS). This ensured that client-server communication is always encrypted and verified for authentication before trusting, addressing the Cryptography and Client/Server security layers. Then, I addressed the Code Quality and Code Error layers by updating Spring Boot to its latest version, 3.5.6, and following secure coding patterns. This mitigated the known security vulnerabilities introduced to the application by updating other outdated dependencies within the framework. Finally, I conducted a manual code review to develop a mitigation plan and validate the refactored code still safely builds.

## Industry Standard Best Practices

I used industry-standard best practices for secure coding by setting up transport layer security encryption and keystore authentication in accordance with OWASP guidelines. I also used industry-standard best practices for secure coding by updating the dependencies to the latest stable versions, since it mitigates known vulnerabilities. The AES-SHA-256 encryption algorithm I used is even recommended for use by the National Institute of Standards and Technology (NIST). Abiding by these secure coding standards will ensure known vulnerabilities are patched in the future and reduce the cost of maintaining the application over time.