****

# 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** | **6/21/25** | **Alex Chadburn** |  |

## 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

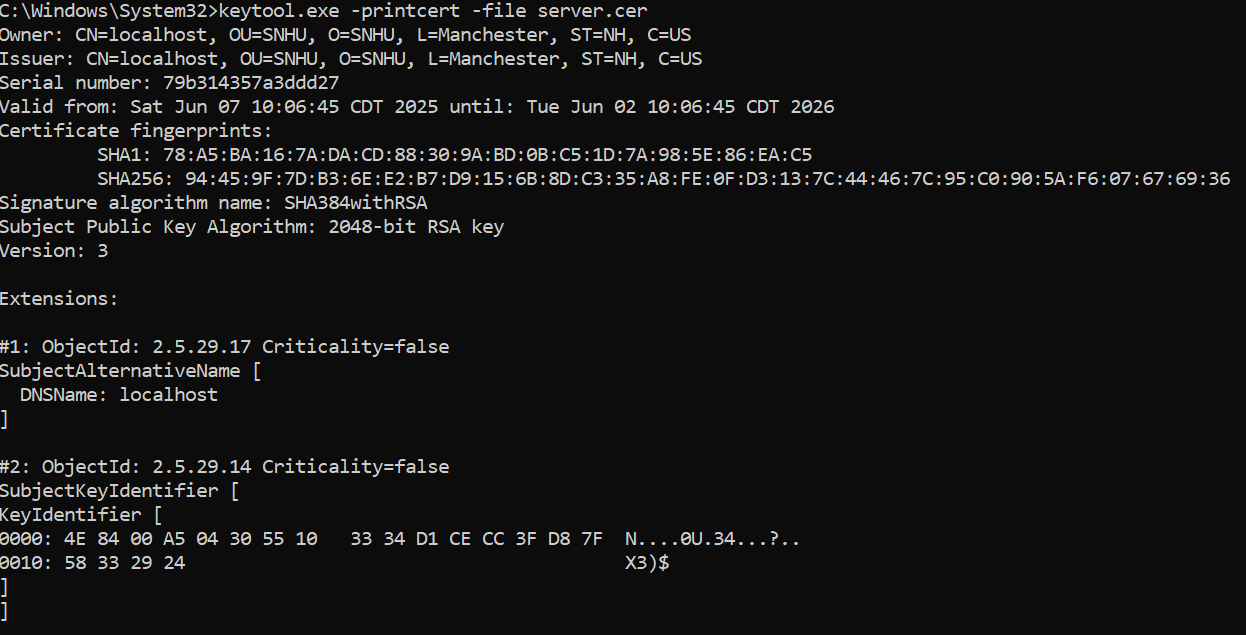
Alex Chadburn

## Algorithm Cipher

I have chosen SHA-256, a secure, one-way 256-bit hash function from the SHA-2 family. It’s widely used for validating data integrity and digital signatures due to its resistance to collisions and brute-force attacks, thanks to its exponential hash space of possible values. SHA-256 does not use encryption keys but plays a key role in secure systems by producing consistent, irreversible outputs from variable input.

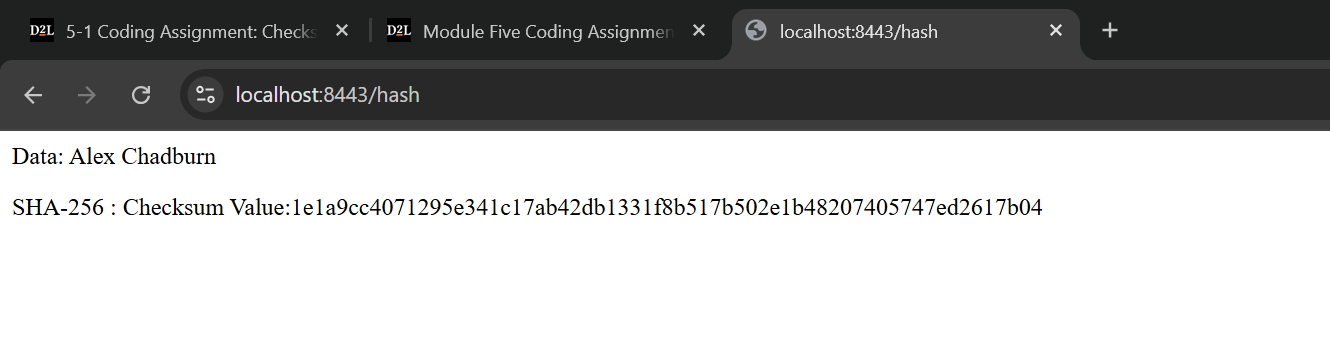
## Certificate Generation

Insert a screenshot below of the CER file.



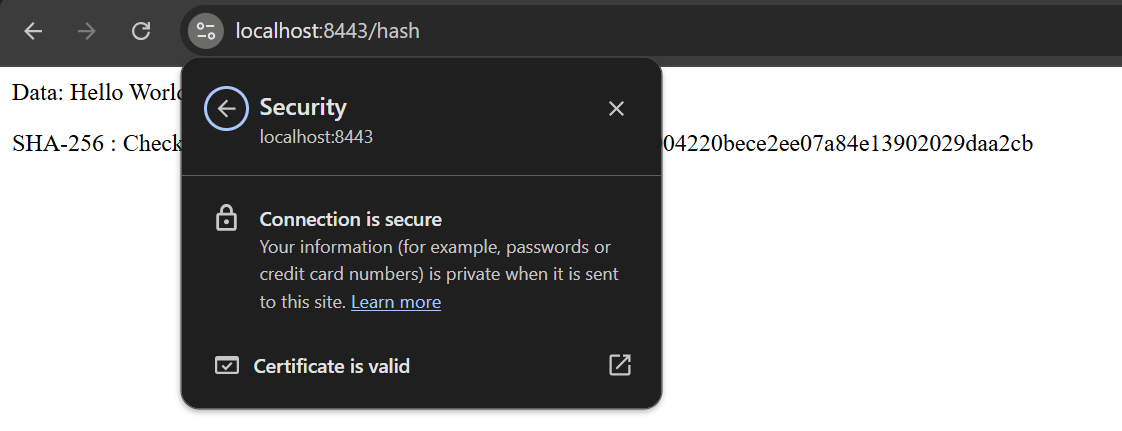
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

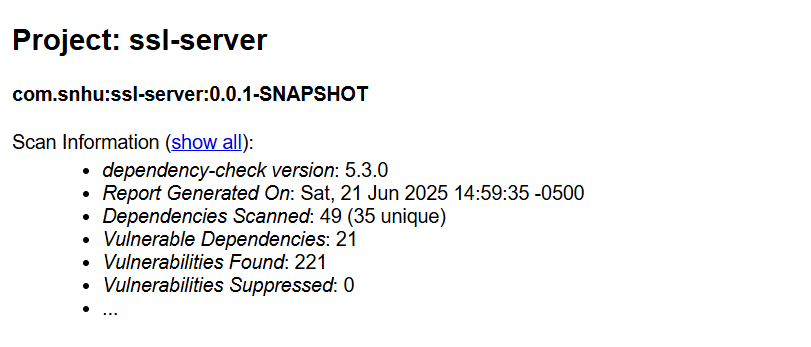
Insert a screenshot below of the web browser that shows a secure webpage.



## 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

The refactored code adheres to modern security protocols by integrating SHA-256 for integrity validation, securing communications through HTTPS enforcement, and implementing self-signed certificates for encrypted data exchange. The process followed the vulnerability assessment flow starting with threat identification, implementing mitigation strategies, testing for effectiveness, and confirming secure operation via static and functional testing.

Key layers of security added include:

* Transitioning HTTP to HTTPS to secure data in transit
* Refactoring code to include cryptographic hashing
* Running OWASP dependency checks to catch outdated or vulnerable libraries

## Industry Standard Best Practices

The original code provided a minimal Spring Boot setup that served only to launch the application. I added basic functionality by creating a controller that returns the SHA-256 hash of a static string through a /hash endpoint. The changes followed secure development practices from the start.

I used SHA-256 for its strong track record in producing consistent, irreversible hashes that are resistant to collisions. The data was encoded using UTF-8 to ensure reliability across platforms, and I wrapped the hashing logic in exception handling to manage potential errors during runtime. I also configured HTTPS and implemented a self-signed certificate using Java Keytool, establishing a secure connection between the server and the client.

These updates introduced real functionality to the project while following widely accepted security practices. Even though the application began with little to no risk surface, starting with secure habits like proper exception handling, HTTPS configuration, and the use of SHA-256 ensures that any future development begins on solid ground. This kind of early attention to security prevents small missteps from becoming larger issues down the line. It also reduces the chances of needing major revisions later, which helps support long-term reliability, easier maintenance, and a smoother development process overall.