****

# 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 6](#_Toc102040760)

[4. Secure Communications 7](#_Toc102040761)

[5. Secondary Testing 7](#_Toc102040762)

[6. Functional Testing 8](#_Toc102040763)

[7. Summary 9](#_Toc102040764)

[8. Industry Standard Best Practices 1](#_Toc102040765)0

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **8/12/23** | **Kayleigh Kinsey** |  |

## 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

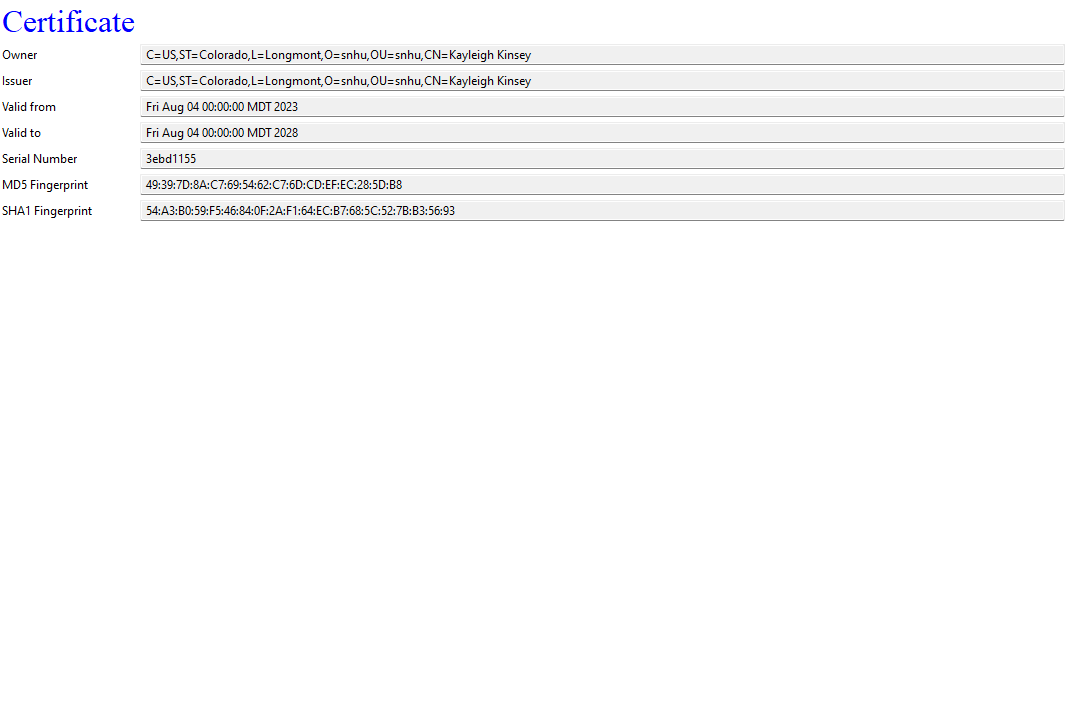
Kayleigh Kinsey

## Algorithm Cipher

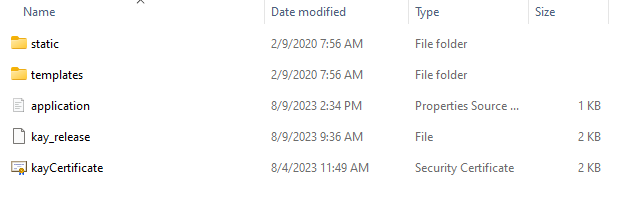
The Secure Hash Algorithm-3 (SHA-3) was created as the next generation of secure hash algorithm as a response to increased efficiency in breaking SHA-2 (Graunke et al, 2009). The final algorithm was determined via an international competition and chosen from 64 total submissions (Boutin, 2015). Currently, SHA-3 is the standard for cryptographic hash algorithms, with many derived functions to choose from, such as SHA3-256 and SHA3-384. The difference between these and most other SHA-3 hash functions (with some exceptions) is the size of the function's output. Larger outputs are more secure, but take up more space, so the best SHA-3 function to use will depend on the needs of a project. For this assignment, I have chosen to use SHA3-384, which is the second largest of the SHA-3 functions. I think it's a well-rounded choice that will be safe without sacrificing too much space. SHA3-384 has a collision resistance strength of 192 bits, a preimage resistance strength of 384 bits, and a second preimage resistance strength of 384 bits (Dworkin, 2023).

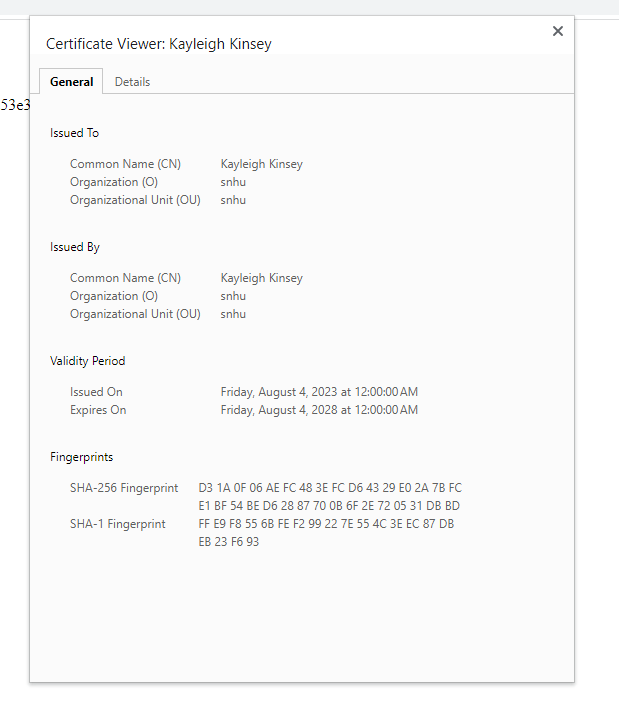
## Certificate Generation

Here is the certificate:



Here is the keystore and certificate in my project’s resources:



Here is the certificate as viewed in the web browser:

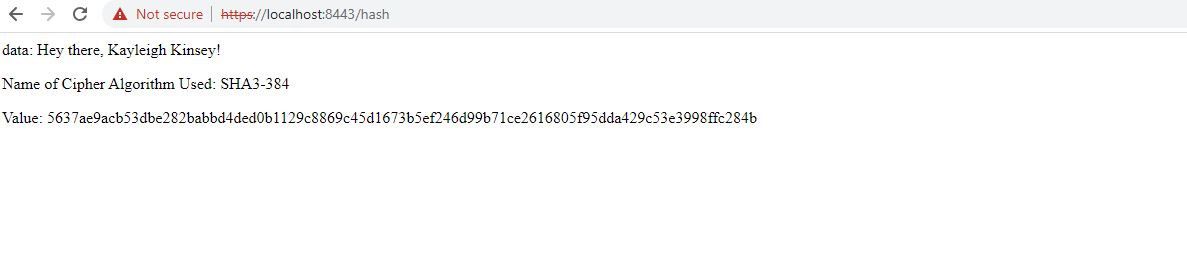
## Deploy Cipher

Here is the code for my checksum verification:



## Secure Communications

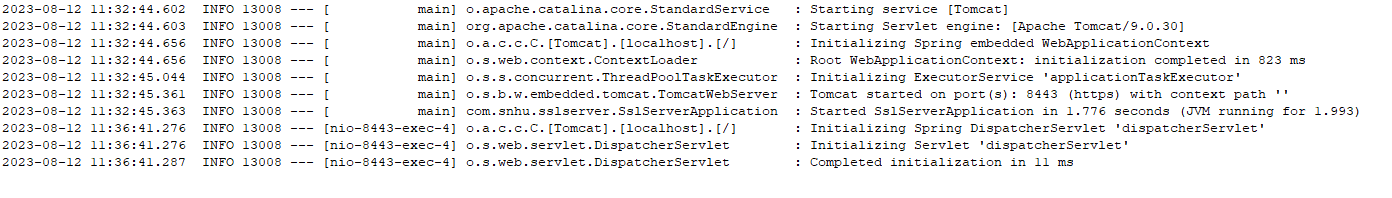
## Here is the webpage:



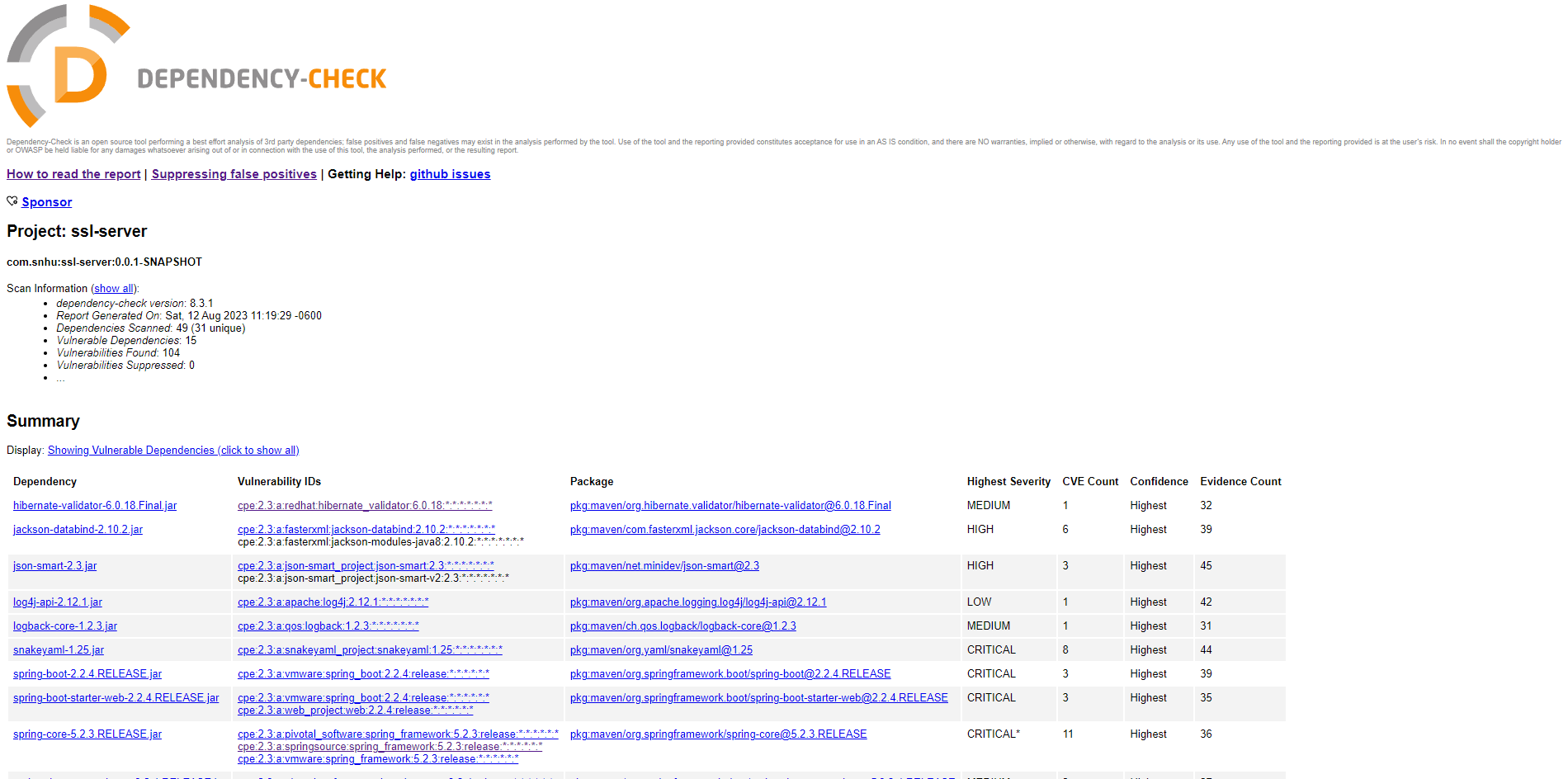
## Secondary Testing

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

Here is the output of the code executing without errors:

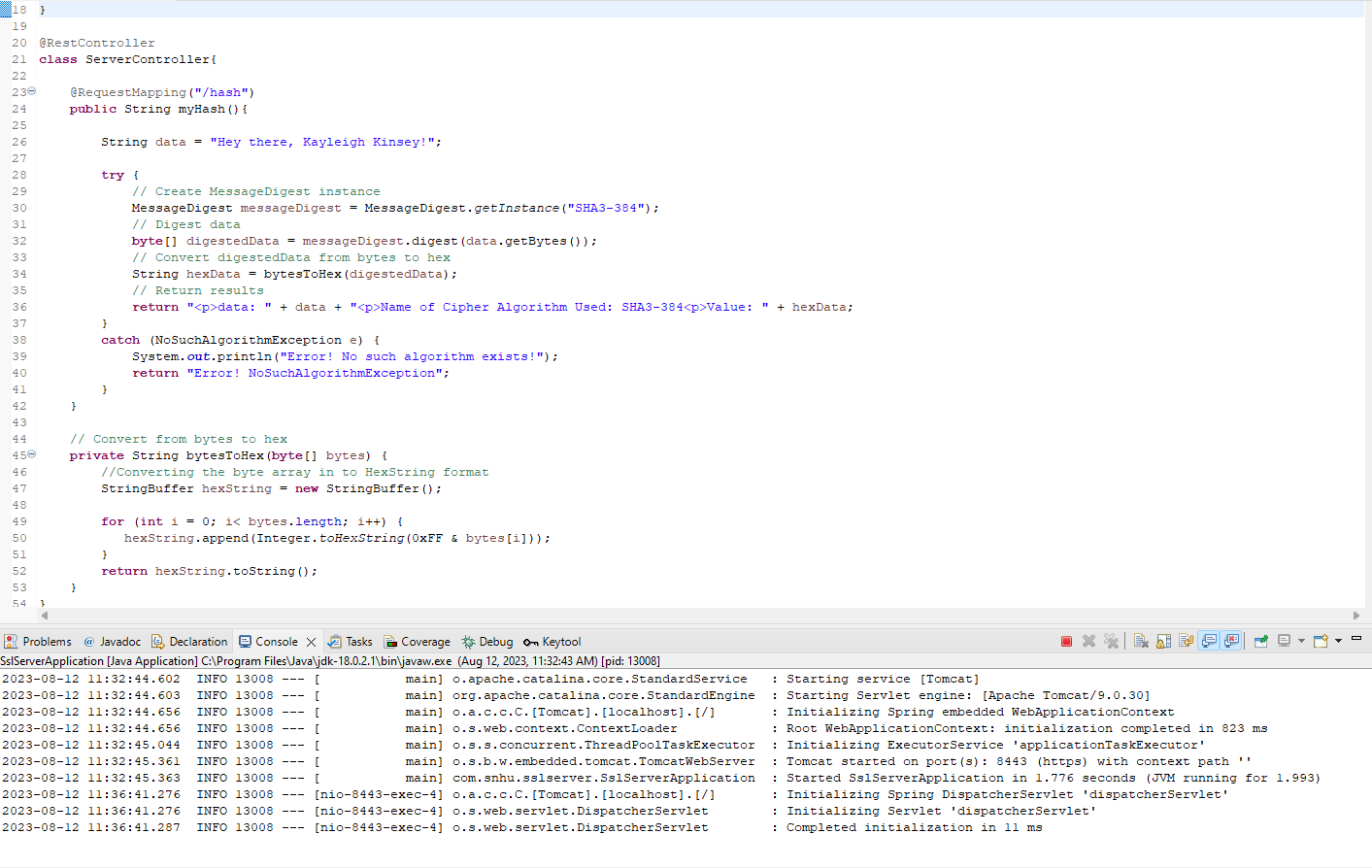


And here is the dependency check report:



## Functional Testing

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



## Summary

The main areas of security I addressed with my code were Cryptography and Client/Server. The assignment focused on certificate generation and hash algorithms, which are both important to establishing a secure client/server connection, and use encryption to ensure third parties cannot tamper with the information that travels across the connection.

I also addressed the Code Error and Code Quality areas of security with my code. These security areas apply to every project, and always need to be kept in mind. I made sure that my program had no errors or warnings, and that expected runtime errors were caught. I maintained quality in my code by adhering to best practices for format, naming conventions, and encapsulation. My code is as clean and straightforward as I could manage, and comments are used to further explain what the code does.

My process for adding layers of security to the code was pretty simple: I focused on one area of security at a time. First, I set up my certificate and ensured its information and location were correct. There were a few roadblocks throughout the process. Namely, that I couldn’t get the program to access the certificate. Eventually, I realized I hadn’t changed the application.properties file, and was able to configure it to use my self-generated certificate. Next, I added the checksum verification code. Most of it had already been done in the Module 5 assignment, so I reused it and made some adjustments. This was helpful in dealing with later issues with my program because I knew the code had worked on the previous assignment. I spent days trying to get my program to execute. For some reason, it would tell me it started the application, but when I went to localhost:8443/hash, I got nothing. Eventually, I was able to open the application web page, but on localhost:8080/hash. This implied to me that the socket was not being assigned correctly. I tried manually configuring the run settings to include the application.properties file in several ways, but none of them worked. Eventually, I noticed an error in the application.properties file itself, and once it was removed, the application ran on the correct socket at last. Despite all the trouble, I did not end up introducing any new vulnerabilities to the application; also, the final program is free of compilation errors and runs as intended.

## Industry Standard Best Practices

I applied industry standard best practices in my checksum by selecting the current best standard hashing algorithm: SHA-3. The specific algorithm I used has one of the largest outputs of the SHA-3 functions, and thus is one of the most secure (because it has high collision resistance). I did not directly use any user input in my code, and only implemented the RequestMapping annotation to obtain input. In my application.properties file, I made sure to enable ssl to establish a secure connection.