****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **14 Aug 22** | **Zachary Wright** |  |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Zachary Wright

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

I recommend Secure Hash Algorithm (SHA-256) for Artemis Financial. SHA-256 is used for encryption where each user still gets their own individual identification token upon creation of their account or message, but it is very limited to who even gets the ability to decode the encrypted data.SHA-256 uses symmetric encryption, meaning that the only people who can decode the encryption is those that receive the message or data along with the creator. Compared to asymmetric encryption, it does not use a public and private key, so it is more secure, since the keys are given on a need-to-know basis.

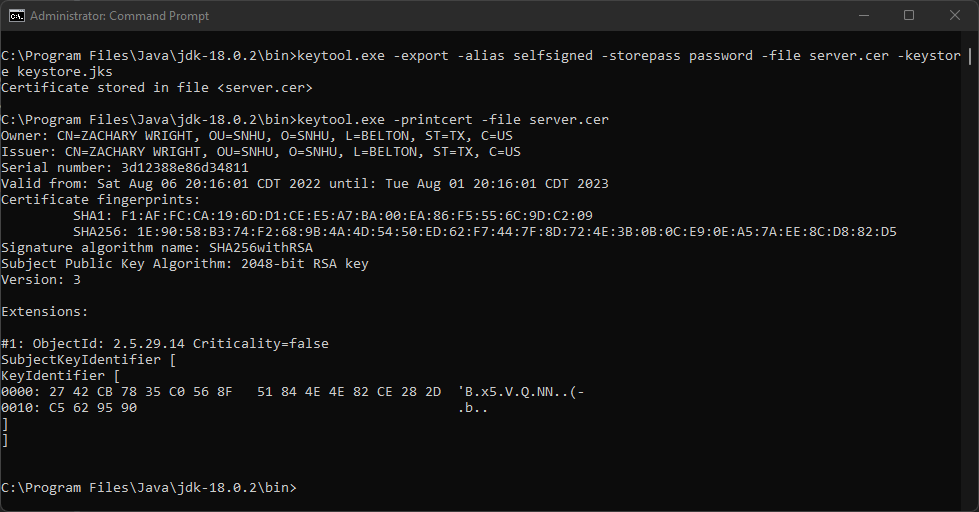
When using SHA-256, you create a 256 bit hash that is often just as long when decode which makes this that much harder to break into. SHA-256 is yet to be cracked and is one of the most widely used hash algorithms for cryptography. Hash functions encrypt the data to a point where you cannot read it and the only way to read it at this point is with the key.

There is a long history with encryption as there is always some information that is too sensitive to be floating around, like our social security numbers. The military has been using encryption for years while the civilian sector was able to start using it in 1971 after IBM created the Lucifer encryption. At this time 56-bit encryption was far more common but today the standard is 256-bit encryption with a maximum size of 2048-bits.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



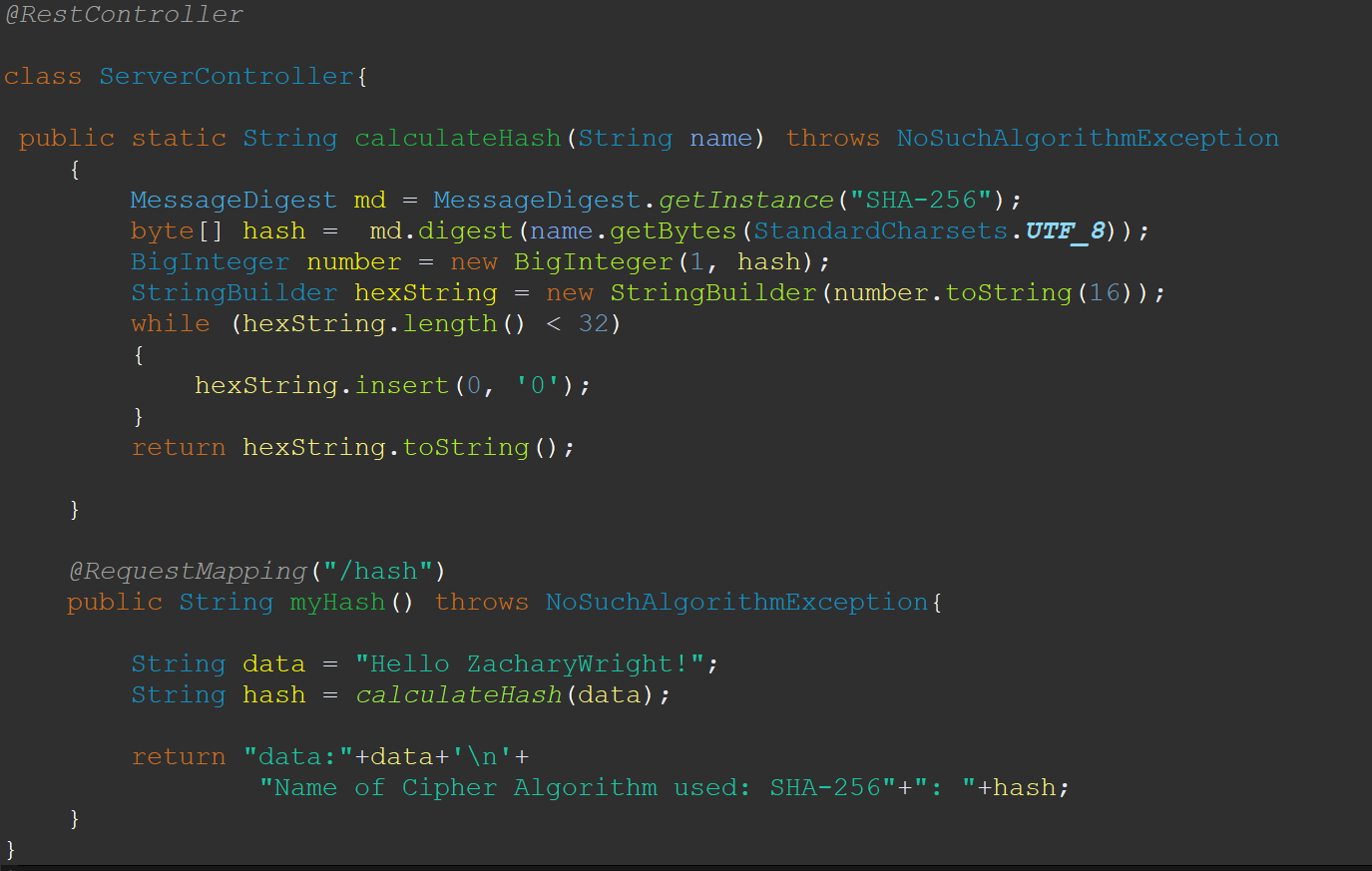
## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.

Graphical user interface, text, application, chat or text message

Description automatically generated



## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

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

The redline is through https due to it being a self signed certificate, google does not recognize it.

Graphical user interface, text, application, chat or text message

Description automatically generated

Http unable to run at this pointA screenshot of a computer

Description automatically generated

## 5. Secondary Testing

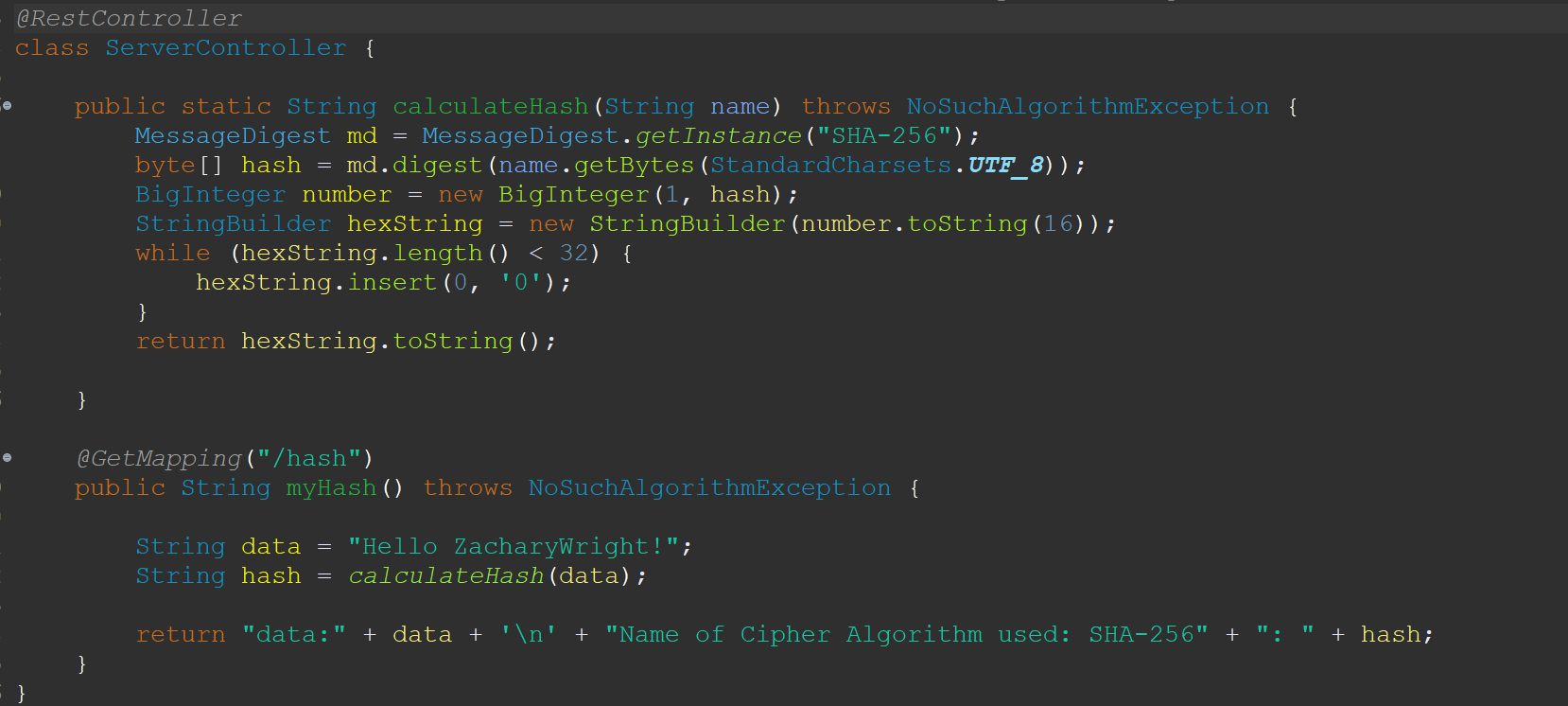
Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

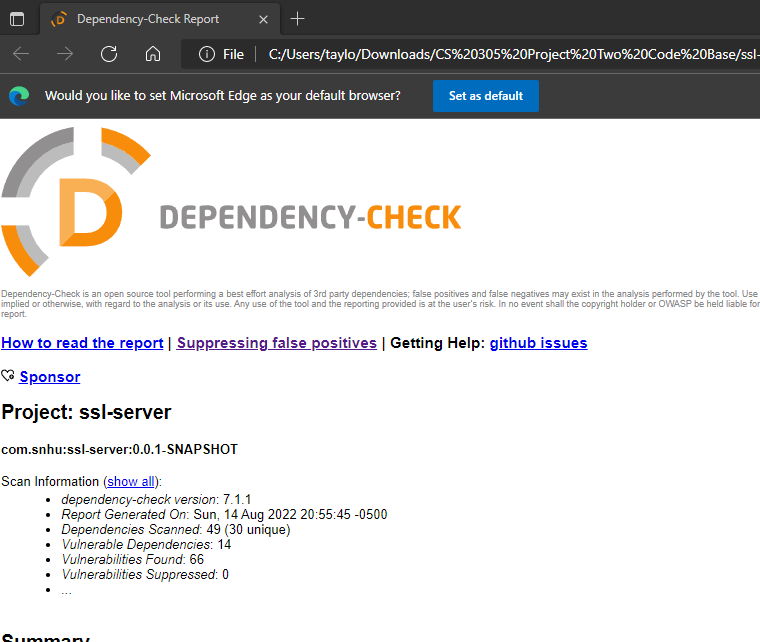
* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

No errors in code:

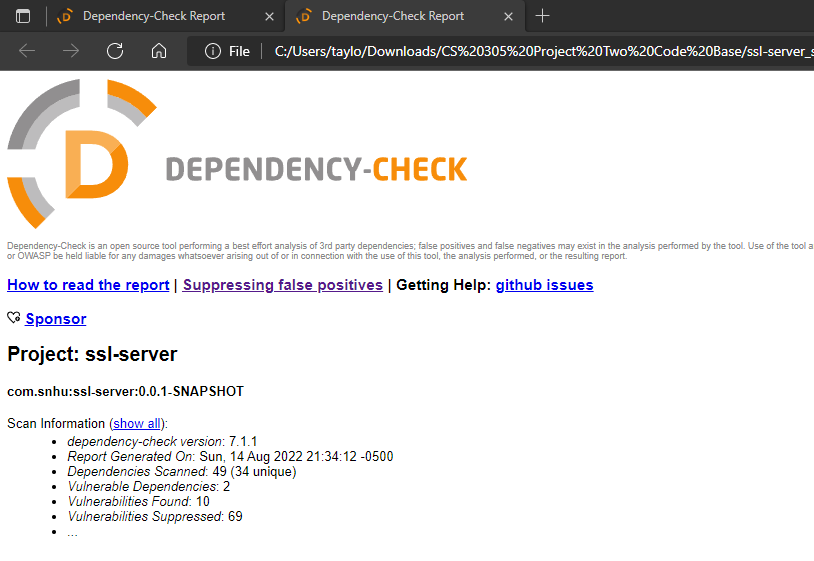
Text

Description automatically generated



Befor suppression

After suppression

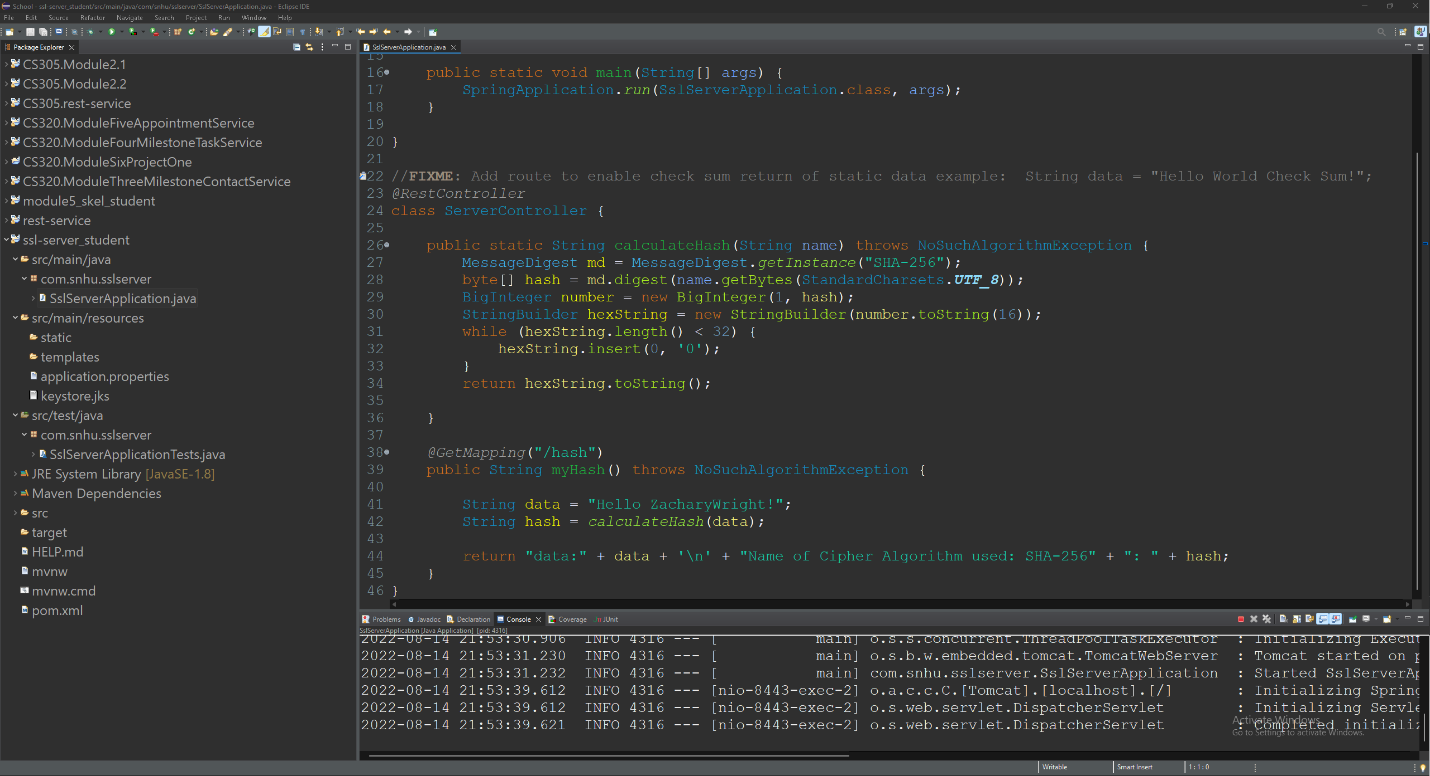


\*Added the suppression because in the general discussions it was mentioned that for this section we must present a before suppression and after suppression screenshot\*

## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.



## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

This was project we had to address cryptography for Artemis Financial while using client and server application keeping it within the industry standard. I implemented the SHA-256 hashing algorithm to protect the sensitive data that could be used by this financial institution. We made their website even more secure but enforcing HTTPS use of the website. I finished the project of with a code review for vulnerabilities caused by my dependencies.

First Layer of security I used was an SSL certificate that ensures the site is secure. This allows this site to become a trusted site and therefore customers can input personal data within and trust it will be secure. Second layer of security added was SHA-256 encryption, making it harder for those who force their way into the database cannot access the sensitive data the customers have trusted the website to store. The final layer of security is HTTPS enforcement. This ensures that a user is using an encrypted session to protect the server and the client.

Best practices for maintaining current security of the software involves regular checkups on the software. This includes regularly conducting a dependency check to see if there are any new vulnerabilities. We can also keep our software secure by using best practices in our software like input validation, proper structure and use of REST APIs and mapping so specific functionality is limited to having its own page, like updating contact information would be in the update account page. Finally, we can conduct regular code reviews to see where we can improve and if any changes have been made to the software, and if there were changes that it does not hinder the security or functionality of the software.,