****

# 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** | **February 17, 2023** | **Tyler Primas** | **Completed all sections providing screenshots to show examples of what was created and how it is secure** |

## 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

Tyler Primas

## Algorithm Cipher

The best algorithm to avoid collisions is SHA-256. The data is transformed into a 256-bit string and secured through that algorithm. SHA-256 is an incredibly secure algorithm and has virtually no collisions due to the number of possible hash values that can be created. The number of total possible hash values is 2^256, so finding two of the same hash values is near impossible.

The AES cipher will be used to protect data submitted by the user. The user inputs information, the cipher takes that information and encrypts it then sends it to wherever it needs to go. When the information needs to be read by either the same user or another verified user, the information is then decrypted and displayed back to them as the original information.

The purpose of the hash function is to create a public key that makes a secure connection. The bit levels involved in AES are either 128, 192, or 256 bits (Manico & Detlefsen, 2015). More bits is more secure and will take longer to crack the cipher.

Random numbers can be used to create keys or random data to secure the system. For symmetric cryptography, the same key is used for encrypting and decrypting. The key has to be kept a secret in order for the data to be secure. For asymmetric cryptography, there are two different keys. One key is kept private, and the other is public. There is no way to retrieve the private key from the public one, so it is generally seen as a more secure method (Manico & Detlefsen, 2015).

There are many different kinds of encryption algorithms, and many have come and gone since they have been created. Some have gone outdated or been figured out by hackers, so they were no longer secure to use. The best algorithms have stayed up to date on their security and added extra layers to be sure that the cipher cannot be cracked. With technology advancing as fast as it does, cracking algorithms is almost guaranteed in the future. As computers get more advanced, their processing power can be utilized to brute force many different options and find out the keys and encryptions.

## Certificate Generation

Insert a screenshot below of the CER file.

Text

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.

Graphical user interface, text, website

Description automatically generated

## Secure Communications

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

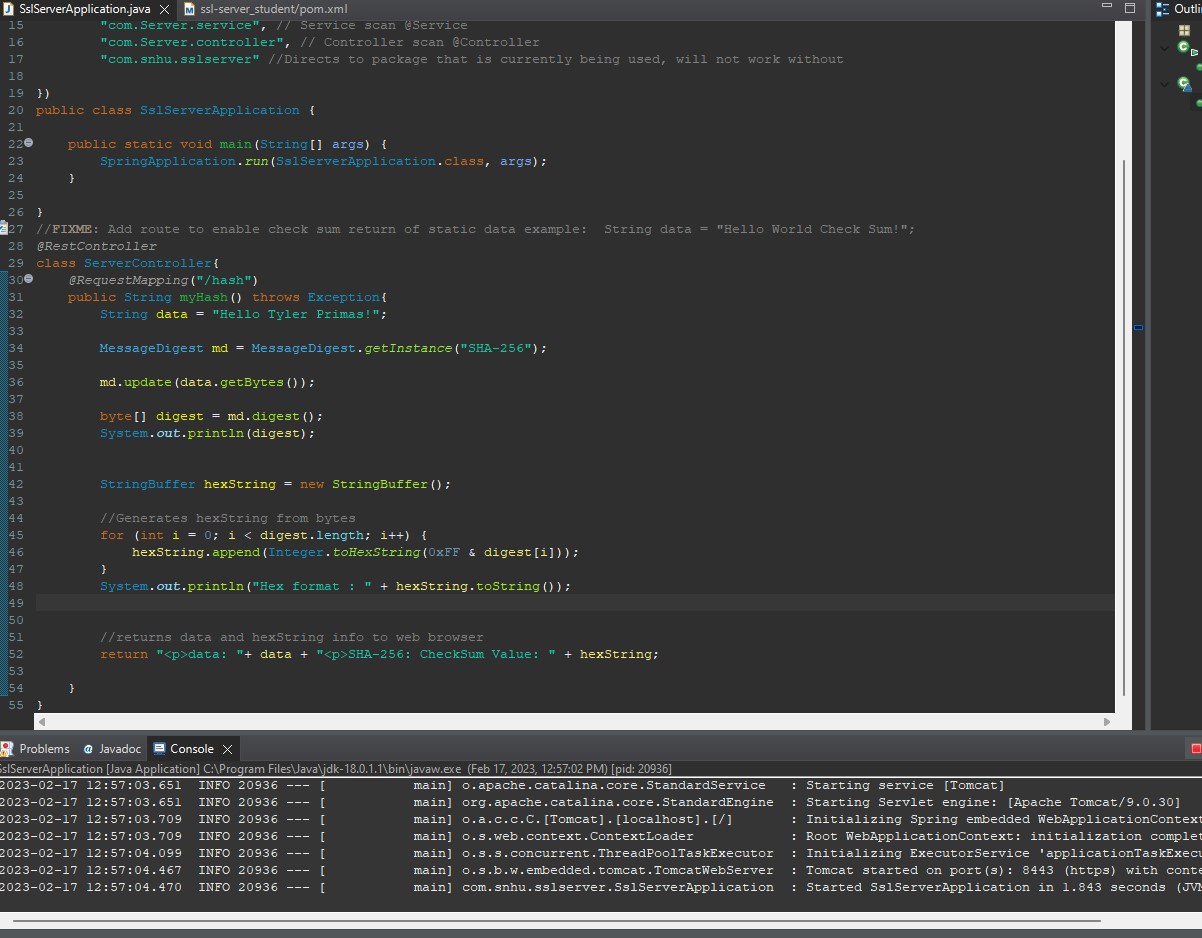
Graphical user interface, application

Description automatically generated

The certificate information shows for the site that was created. It shows all of the details from the when I set it up using the java keytool.

## Secondary Testing

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



Graphical user interface, text

Description automatically generated

The code executes without errors and going to the localhost on a browser properly displays the information.

Graphical user interface, text, application

Description automatically generated

Running the dependency check shows there are 55 vulnerabilities in the code, but searching through them shows that updating should fix many of those vulnerabilities. I will update what is needed to fix vulnerabilities.

## Functional Testing

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

Text

Description automatically generated

Code executed with no errors.

Graphical user interface, text, application, email

Description automatically generated

After updating the springframework version to the latest version, the vulnerabilities dropped from 55 to 3. The only remaining vulnerabilities are in the snakeyaml dependency. As far as I’m aware, the java file we are running is not working with yaml files currently, so those vulnerabilities should not reduce the overall security of the project.

## Summary

The main area of security that was addressed from refactoring the code was cryptography. The main purpose of the code is to display the string that is typed into the code and then show what it is encrypted into. Cryptography also focuses on vulnerabilities, so going through and updating the outdated dependencies to reduce the overall vulnerabilities is a major part of that. Another area of security was Client/Server. I had to create a secure site by creating a security key that would register as secure for the client to use when accessing the site.

To add layers of security to the application, I had to create a certificate using the java keytool. This was the main step for the client to be able to access a secure and trusted website. The code also uses encryption to encrypt the message that is sent to make sure that it can only be understood by someone with the key to decrypt it. The final layer of security was making sure the dependencies were not vulnerable. Having vulnerable dependencies creates an easy way for an attacker to access information or harm the site. Updating the dependencies solved many of the problems in this application.

## Industry Standard Best Practices

I used industry standard best practices to maintain the software applications current security by making sure everything was as secure as possible and everything was up to date and working as intended. The application is secure because it uses SHA-256 to encrypt the data and keep it secure from potential attackers. By making sure there were no bugs in the code, it creates a more secure application as well that will provide a better user experience.

It is important to apply industry standard best practices for secure coding to maintain the overall wellbeing of the company. Making sure the code is secure and best practices are used creates an overall positive view of the company. A company that has constant security issues or bugs in the code is not viewed as a trusted or reliable company. This can prevent users from wanting to use the company’s software and any investors from wanting to fund the company. A company that always makes sure to secure the data of its users and make the experience as pleasant as possible will always be seen as a company more favorable to use by the public.

**References**

Manico, J., & Detlefsen, A. (2015). Iron-clad java: Building secure web applications. McGraw-Hill Education.

Nechvatal, J., Barker, E., Bassham, L., Burr, W., Dworkin, M., Foti, J., & Roback, E. (2001). Report on the Development of the Advanced Encryption Standard (AES). *Journal of research of the National Institute of Standards and Technology*, *106*(3), 511–577. https://doi.org/10.6028/jres.106.023