****

# 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** | **February 2022** | **Carlie White** |  |

## 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

Carlie White

## 1. Algorithm Cipher

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

Artemis Financial transmits and holds sensitive information. They want to be able to secure the transfer of data to their web application. Given that this company needs a very high level of security, I believe that it would be best to use AES-256. According to the NIST, AES "specifies a FIPS-approved cryptographic algorithm that can be used to protect electronic data" and is approved and maintained by the Department of Commerce, NIST, and ITL. Furthermore, the NIST also allows this standard to be adopted by non-federal government organizations and "is encouraged when it provides the desired security for commercial and private organizations." Prior to AES, federal agencies used DES as their main encryption algorithm, and it was successful for over 20 years but due to Moore's Law it began to become obsolete and the government needed to find a stronger, uncrackable algorithm, and that's where AES comes in.

AES also utilizes SHA, what take data input as plaintext and creates a cipher that is nearly impossible to decrypt. Further, the bit level (128, 192, or 256) signifies that the attacker would have to enter 2^bit-level combinations in order to get in, which is nearly impossible. Since the highest level of AES is 256, that means that it is also the most secure and that is why I suggest using it since it would take billions of years to crack (N-able).

AES is a symmetric key encryption cipher, meaning it uses the same key to encrypt and decrypt. Symmetric encryption allows for ease of use because it requires the user to perform and remember less actions. Even though the same key is used both ways, that doesn't make it less safe because the length of the encryption is what keeps it safe in this instance.

## 2. Certificate Generation

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

Graphical user interface, application

Description automatically generatedGraphical user interface, application

Description automatically generated

Text

Description automatically generated

## 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.

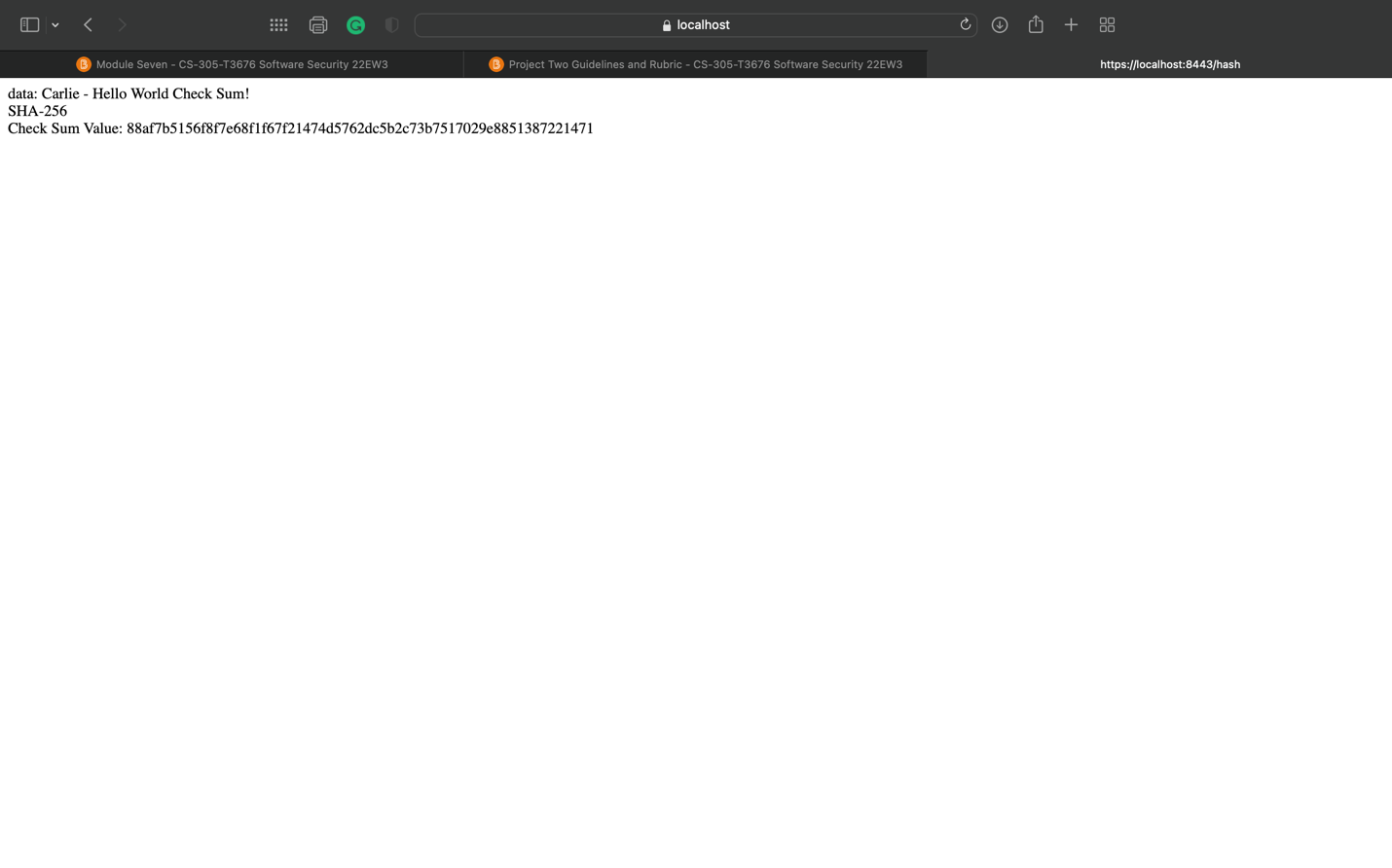
Graphical user interface, text, application

Description automatically generatedText

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.



## 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. A screenshot of a computer

Description automatically generatedGraphical user interface, text, email

Description automatically generated

## 

## 6. Functional Testing

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

A screenshot of a computer

Description automatically generated

Here is the refactored code. As you can see, there are no errors which shows that there are no obvious insecurities.

## 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 project allowed me to explore cryptography, server/client, and code quality. The first part of this project was to create an encryption. For this, I decided to implement an SHA-256 bit algorithm in order to protect the data. The second part of the project was implementing client/server by generating a certificate to use a TLS connection to the web application. The third part was code quality. After the project was complete, I decided to inspect the code to find any errors. I didn't find any errors in the refactored code therefore the code quality aspect was pretty short.

For this project, there are essentially three layers of security: the SSL certificate, the SHA-256 encryption, and the HTTPS implementation. The SSL certificate ensures that the application is secure by encrypting the communication. The SHA-256 encryption adds another layer of security by directly encrypting the data that is being communicated, that was the data, and the communication are both encrypted. Lastly, the HTTPS implementation which ensures that if the user is accessing the application, it must be an encrypted session.

The best practices for maintaining this level of security is by keeping up with current advancements and performing regular system integration tests. If anything is added or changes, they need to perform these tests to ensure that everything still works properly and there's no leaks.