****

# 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** | **04/17/2021** | **Stephanie Hoffman-Kuszmaul** |  |

## 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

Stephanie Hoffman-Kuszmaul

## 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.

Artemis Financial has requested that an encryption algorithm be recommended that can be used to allow them to encrypt and archive long term files. It will be assumed that attacks will more than likely come from hackers that are trying to gain access to the associated files. Encrypting them will help to protect the information if the files are stolen because the information contained within them will be useless. Since the files are not being moved to another location Asymmetric keys will not need to be used. By using the SHA-256 algorithm that has a 256-bit keys that will encrypt the files, will be the most efficient way to utilize the encryption. Artemis Financial will be the sole company that will have access the related encryption keys. SHA-256 efficiently use Javas random number generator, this ensures that the files being encrypted are as secure as it is currently possible to secure them. The random number generator will allow the cipher to create a secure nonreversible checksum that will still allow the verification and authentication of the files. The hash function is used to verify the files that will use the SHA-256 cipher which creates the checksum signature for the message.

## 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.

![Text

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.

* 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

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.

![Text

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

//**FIXME**: Add route to enable check sum return of static data example: String data = "Hello World Check Sum!";

@RestController

**class** ServerContoller{

@RequestMapping("/hash")

**public** String myHash() **throws** NoSuchAlgorithmException{

String data="Hello World CheckSum!Author: Stephanie HoffmanKuszmaul";

String checksum;

MessageDigest md=MessageDigest.*getInstance*("SHA-256");

**byte**[] hash = md.digest(data.getBytes(StandardCharsets.***UTF\_8***));

checksum=*bytesToHex*(hash);

**return** "<p>data: "+data+"\n"+"<p>Name of Cipher Algorithm Used: AES CheckSum Value:"+checksum;

}

**private** **static** String bytesToHex(**byte**[] hash) {

StringBuilder hexString=**new** StringBuilder (2\*hash.length);

**for** (**int** i=0; i<hash.length; i++) {

String hex=Integer.*toHexString*(0xff &hash[i]);

**if** (hex.length()==1)

hexString.append('0');

hexString.append(hex);

}

**return** hexString.toString();

}

}

![Graphical user interface, text, application, email

Description automatically generated]()

![Graphical user interface, text, application, email

Description automatically generated]()



## 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.

**package** com.snhu.sslserver;

**import** java.security.NoSuchAlgorithmException;

**import** java.nio.charset.StandardCharsets;

**import** java.security.MessageDigest;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.web.bind.annotation.RequestMapping;

**import** org.springframework.web.bind.annotation.RestController;

@SpringBootApplication

**public** **class** SslServerApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SslServerApplication.**class**, args);

}

}

@RestController

**class** ServerContoller{

@RequestMapping("/hash")

**public** String myHash() **throws** NoSuchAlgorithmException{

String data="Hello World CheckSum!Author: Stephanie HoffmanKuszmaul";

String checksum;

MessageDigest md=MessageDigest.*getInstance*("SHA-256");

**byte**[] hash = md.digest(data.getBytes(StandardCharsets.***UTF\_8***));

checksum=*bytesToHex*(hash);

**return** "<p>data: "+data+"\n"+"<p>Name of Cipher Algorithm Used: AES CheckSum Value:"+checksum;

}

**private** **static** String bytesToHex(**byte**[] hash) {

StringBuilder hexString=**new** StringBuilder (2\*hash.length);

**for** (**int** i=0; i<hash.length; i++) {

String hex=Integer.*toHexString*(0xff &hash[i]);

**if** (hex.length()==1)

hexString.append('0');

hexString.append(hex);

}

**return** hexString.toString();

}

}

Text

Description automatically generated

## 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.

When recfacting my code I had to add a Rest Controller.java file , this helps