****

# 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** | **22 August 2021** | **Gabe Balicki** |  |

## 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

Gabe Balicki

## 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.

The AES cipher is the best one to use because it offers a good number of key sizes, as well as having a reputation for being one of the best ciphers being used in modern applications. Data is encrypted as needed with a128-bit or 256-bit encryption. This type of encryption is highly secure, making it extremely challenging for hackers to break the security system.

For this application, the symmetric key creation is used to transfer sensitive data. Symmetric keys work by transferring data between the server and client. Non-symmetric keys include public and private keys. With the non-symmetric key, public keys are given to the public and private keys are exclusive to the client only. Keys are used to encrypt and decrypt data only when the proper key is used. One downside to using key encryption is that if the key is forgotten or lost, the data will likely be lost forever.

With the current AES, hackers have almost no chance of cracking encryptions because the variety of different variables generated with a 256-bit encryption is so great. For the application in this project, a 128-bit encryption is used, which is still quite secure.

## 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

## 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.

Text

Description automatically generated with low confidence

## 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.

Graphical user interface, text, application

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

Graphical user interface, text, application

Description automatically generated

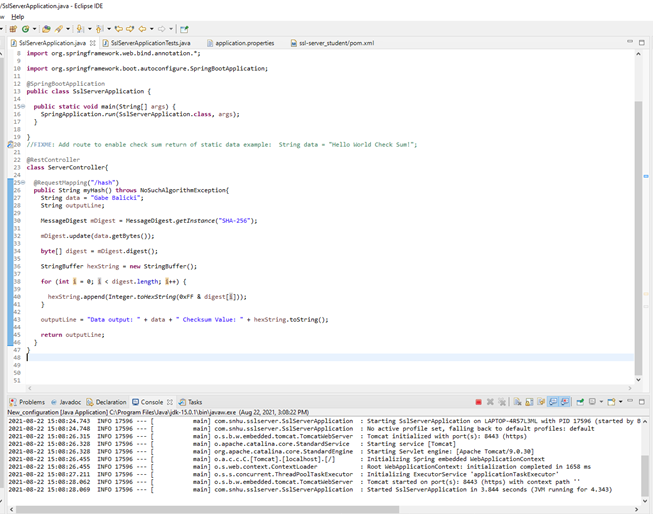
Graphical user interface, text, application, email

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.



## 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.

API’s, Cryptography, client/server, and syntax/code quality were all taken into consideration when refactoring the code to be more secure. We connected to the 218-bit encryption by creating a self-signed certificate and generating keys that helped us connect the server to the client. This was done to ensure that hackers would be unable to access sensitive data that was being communicated. Code was checked for vulnerabilities at all stages of refactoring to ensure that there were no vulnerabilities available for hackers to exploit. If vulnerabilities are ever found that do not have a current solution, it is important to determine what might work best for those vulnerabilities and to understand their impacts on the application. Developers should suppress and vulnerabilities found in code that do not impact any elements used by the code. For vulnerabilities that developers find that do impact elements used by the code, those vulnerabilities should be analyzed, available updates implemented, and solutions found.