****

# 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** | **4/12/2021** | **Kyle Pavlack** |  |

## 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

[insert name here]

## 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.

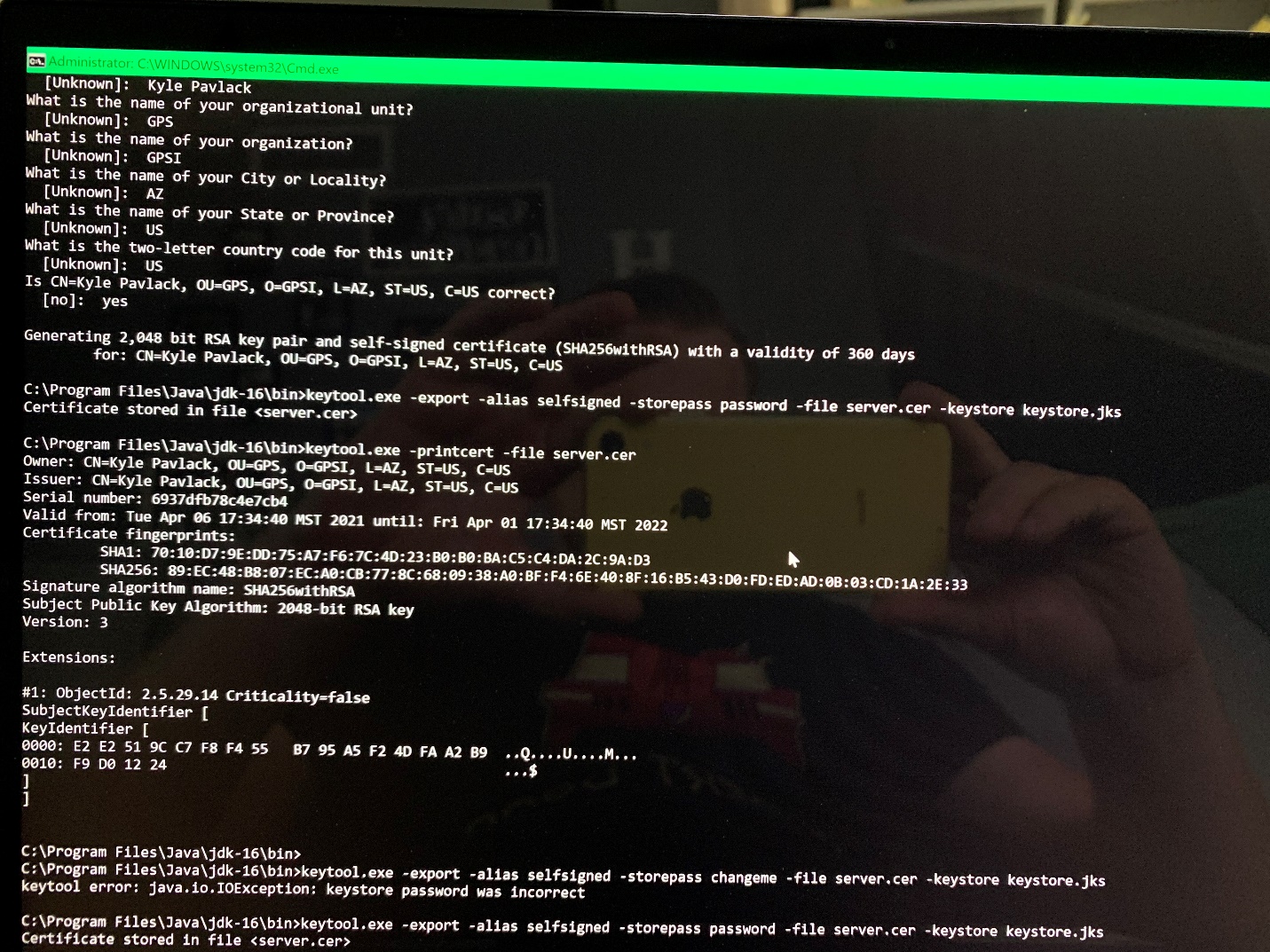
When looking at encryption algorithm ciphers, we can say the best to use the AES cipher as it supports a variety of key sizes and presents one of the best standards that is in place as of today. There are 2 options as the 128-bit and the 256-bit which offers the same level of security but on different levels. The 256-bit makes the application very difficult for attackers to crack as it creates symmetric keys that allows data to be encrypted and delivers keys to the clients for proper communication.

The use of symmetric and non-symmetric keys varies on what the application would need to accomplish. Symmetric keys are basically shared between the sever and the client where non-symmetric keys include public and private keys with public keys being known to the public and private keys being known specifically to the client. These keys are used to encrypt data and de-crypt only when the correct key is used. A downside to this is if the key to any encrypted data is lost, the data is pretty much lost as well since there will be no way to decrypt that data without the proper key. We can use a random number generator which creates unique identifier to transactions which might help with identifying certain events like data transfer or communications that took place along the way.

Currently, AES can encrypt using up to 256-bit which is virtually immune to being cracked as the number of different values that can be created is large. For this application, we utilized 128-bit encryption which should provide enough of a secure communication between the server and client.

## 2. Certificate Generation

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



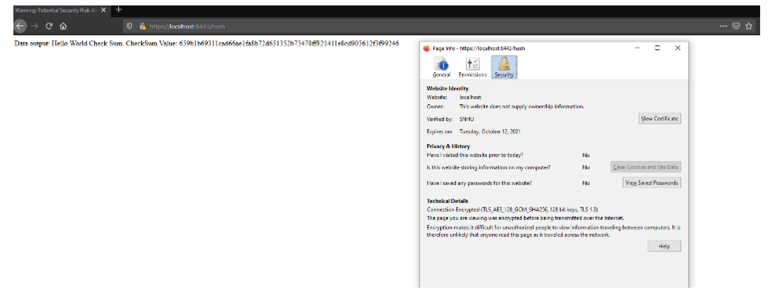
## 3. Deploy Cipher

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.



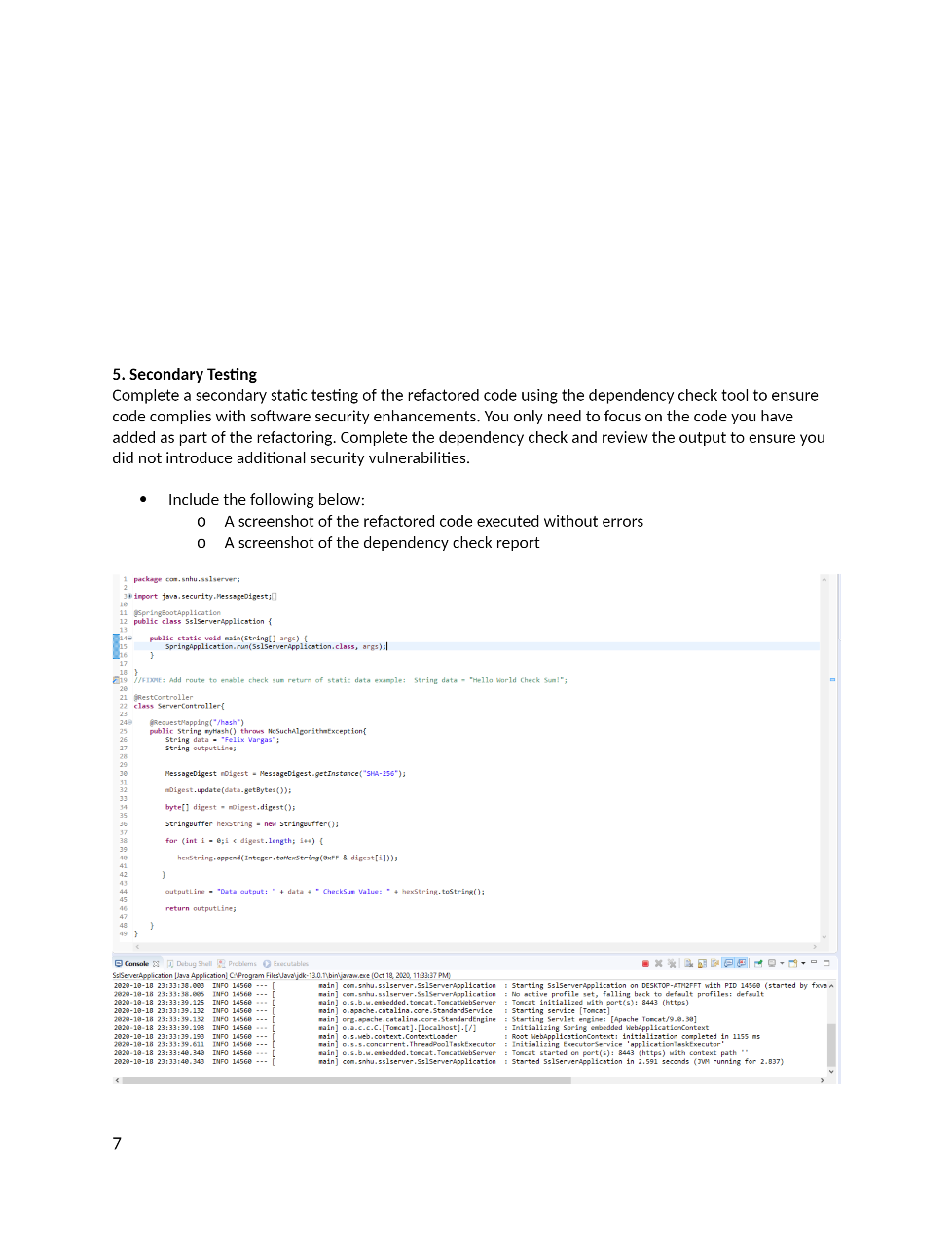
## 4. Secure Communications

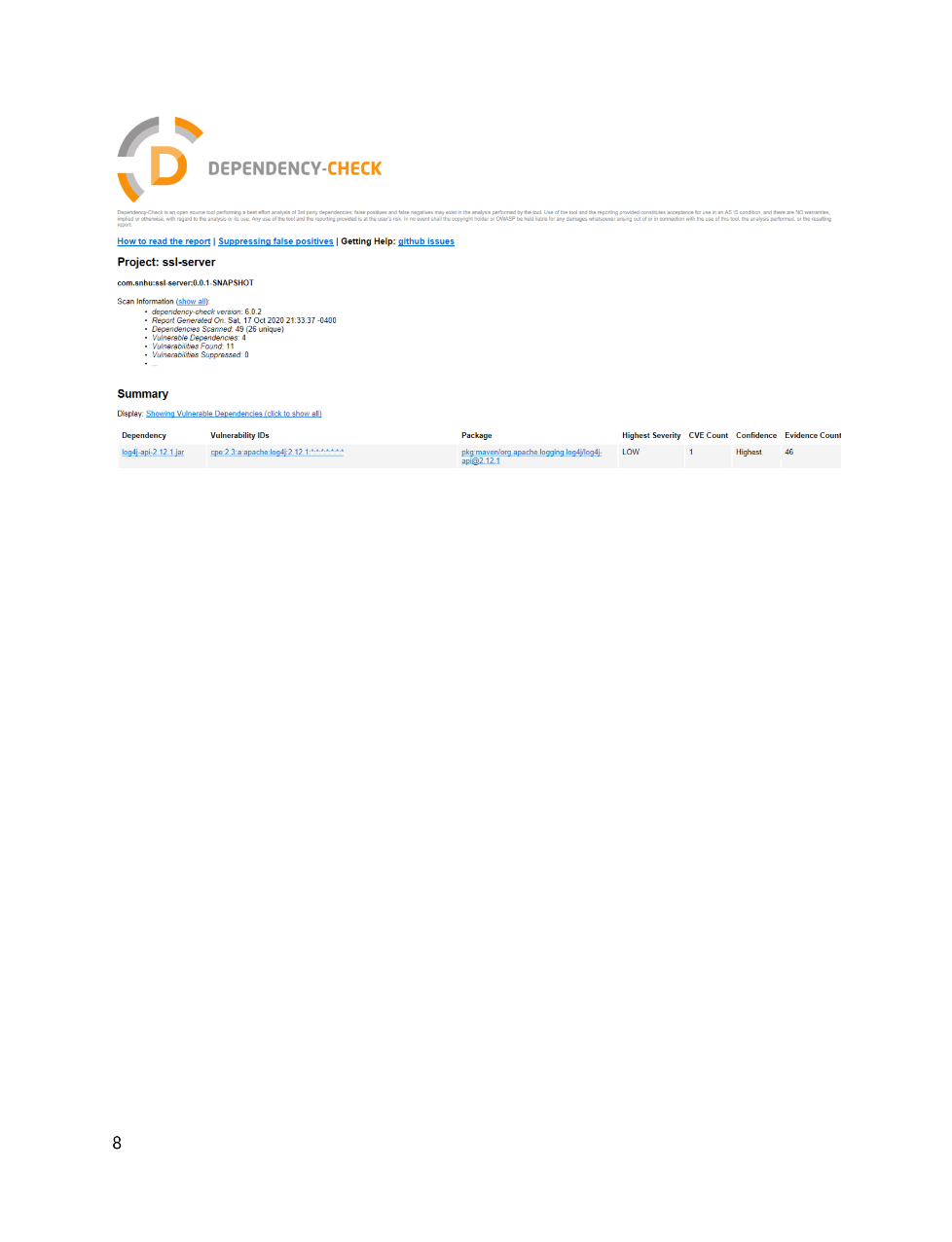
Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.



## 5. Secondary Testing

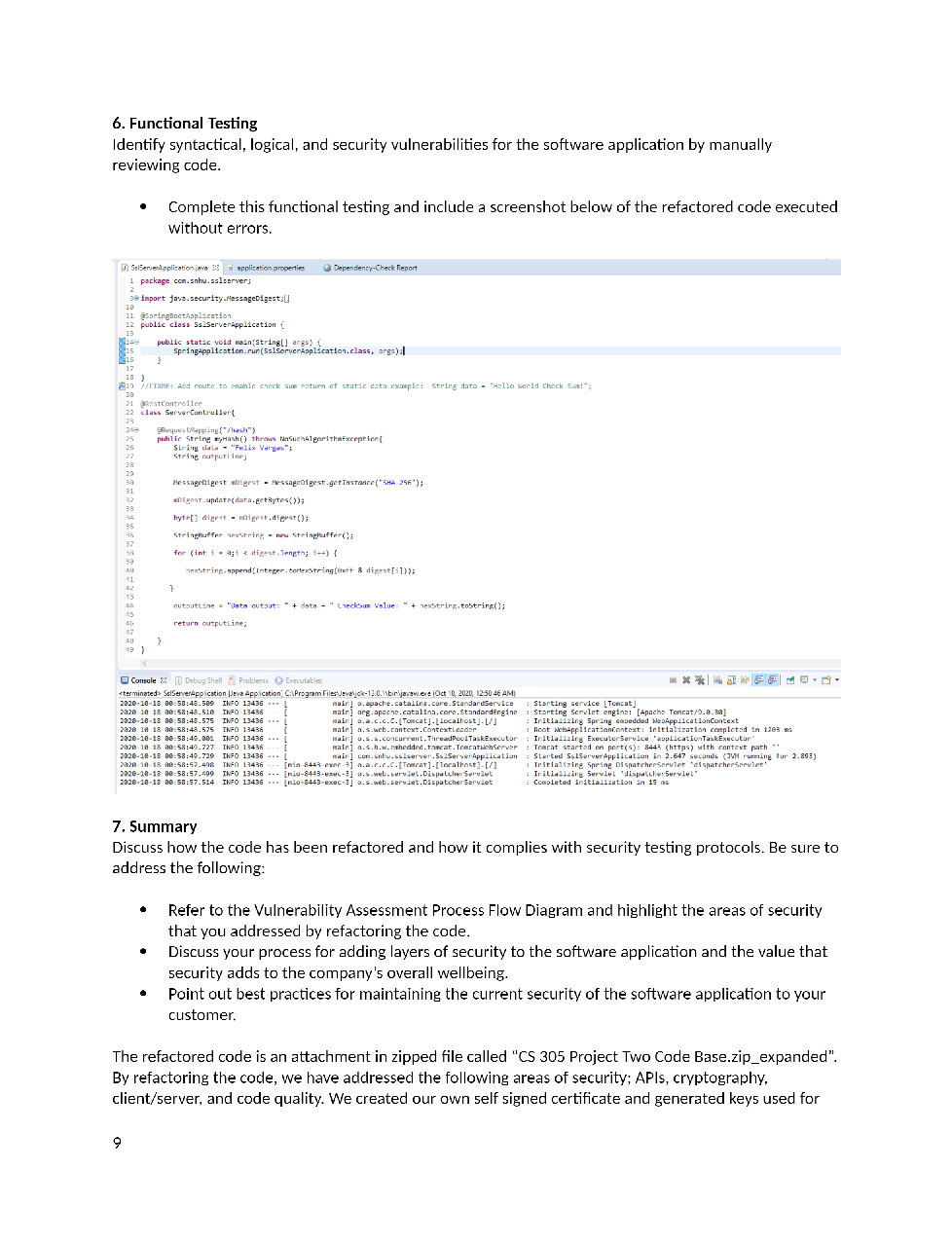
Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not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





## 6. Functional Testing

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



## 7. Summary

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

The refactored code is an attachment in zipped file called “CS 305 Project Two Code Base.zip”. By refactoring the code, we have addressed the following areas of security; APIs, cryptography, client/server, and code quality. We created our own self signed certificate through the command prompt which gave us a temporary generated key that is used for this application which allowed us to connect with the 128-bit AES encryption. This encryption helps make it so that only the intended recipients will be able to read the data on a secure level. Secure communications are very important to maintain in this application as unsecure communications can cause data to be leaked and available for an attacker to steal the data and compromise the information.

The results of a security breach can cause a loss of trust between the application’s owner and the consumer in which return can cause a financial loss due to fines incurred by governments requiring secure communications. Maintaining security in the application will protect the company’s products and assets and keep their clients trust and business at optimal performance. Some of the best practices to follow that maintain security is to frequently check the code for vulnerabilities, especially after implementing new functions or changing current ones and before publishing or making those changes over to the live code. If there are any new vulnerabilities found during the development functions, the development team will work to make the code quality more secure and remove the vulnerabilities. For any vulnerabilities that have no solutions at the time, they should decide what will work best and determine if those vulnerabilities impact any elements that the application is using. If the team learns of any vulnerabilities that exist that do not impact the code, the team can then suppress the solutions so they won’t be able to run through. From there the team will look for any updates that can be applied to the solutions to perform at the needed level.