****

# 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** | **8/10/2022** | **Nicholas Ciesla** | **Initial** |

## 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

Nicholas Ciesla

## 1. Algorithm Cipher

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

**I recommend the SHA-512 encryption algorithm.**

* 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.

**I chose this algorithm because it is one of the most common cryptographic hash functions and has yet to have a collision. It’s little brother, SHA-256, is estimated to take a brute force “birthday” attack comprising of 4038 hashes to cause a collision (Ellis, 2018). A 512-bit digest is certainly overkill, but it is also one of the best options for security. If more performance is desired, the hash could safely be downgraded to 256-bit.**

**The algorithm works by breaking the message into blocks of 1’s and 0’s. The first block is then put through a compression function, then the next block is put through the compression function along with the previous compressed block. This goes on until the end of the message. Then, a padding block is added with information about the message. This block is then put through a compression function with the second to last compressed block. The output of this final compression is the hash returned to the user (Ellis, 2018).**

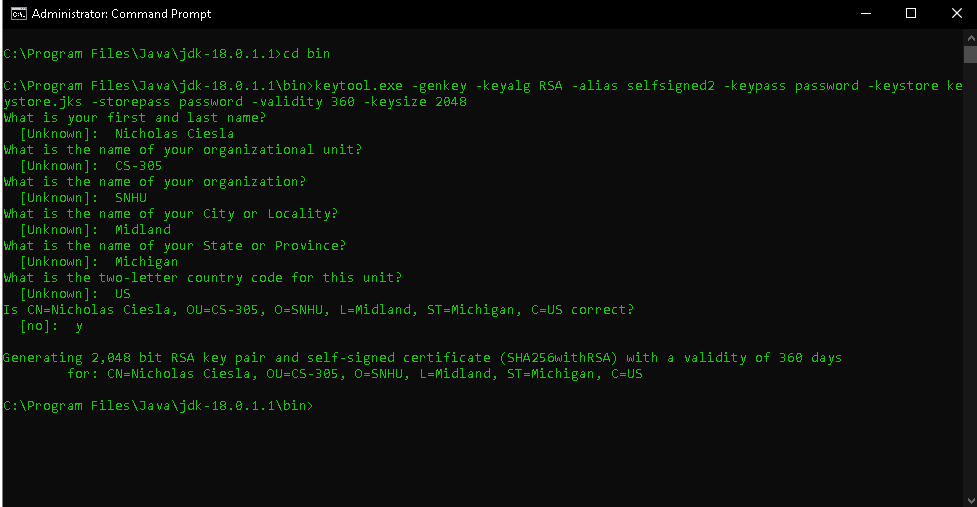
**Hash functions are what is used to encrypt the data. A hash function is a mathematical function that converts readable data into jumbled random characters using the key to convert the data. “Random” number generators are often used to generate these keys (true random in computing is not quite possible). These keys can also be symmetric, meaning the same key encrypts and decrypts, or asymmetric meaning that two different keys together, one private and one public, are needed to encrypt and decrypt the information. The history of encryption is much longer than you may expect. Things started as soon as we put pen to paper, or even chisel to stone, with things like the Caesar Cipher. Things continued in the Middle Ages with mathematicians flexing their prowess. Further on, the enigma machine in WW2 hid Axis communications. Post Caesar Cipher, things have been like hash functions. However, the future holds even stronger encryption with the advent of Quantum Encryption (Arcserve, 2022).**

## 2. Certificate Generation

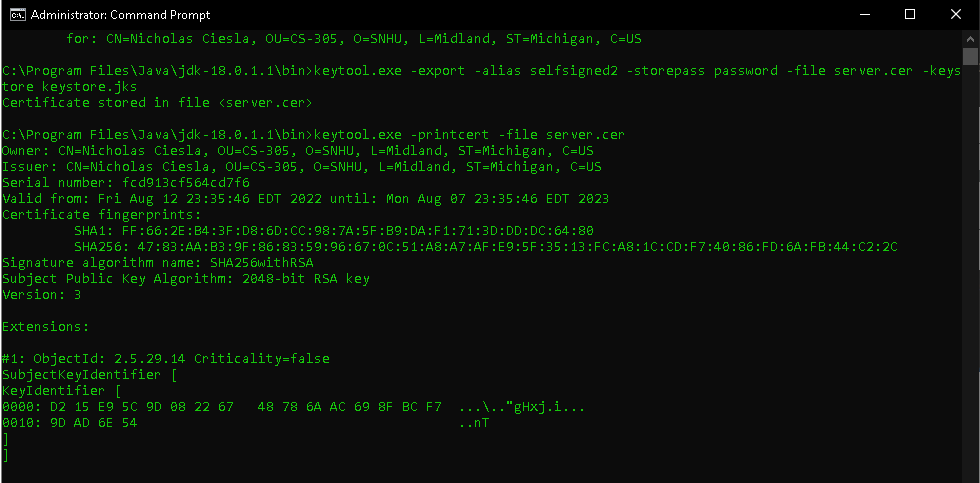
Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

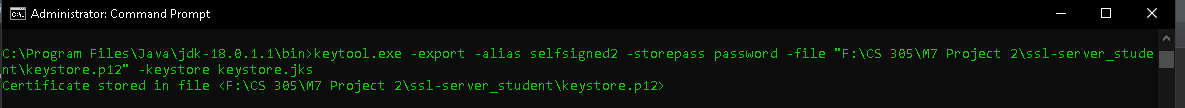
Generation:



Confirmation:



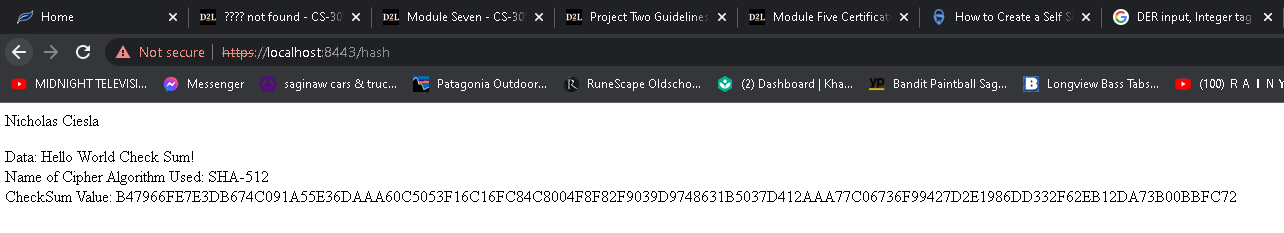
Export:



**I was unable to use this keystore because of a “DER input, Integer tag error” and had to use the keystore from M5. I tried using jks and cer files instead to no avail. I have stored my generated certificate in “keystore1.p12”. Any attempt to use it on my machine resulted in that error.**

## 3. Deploy Cipher

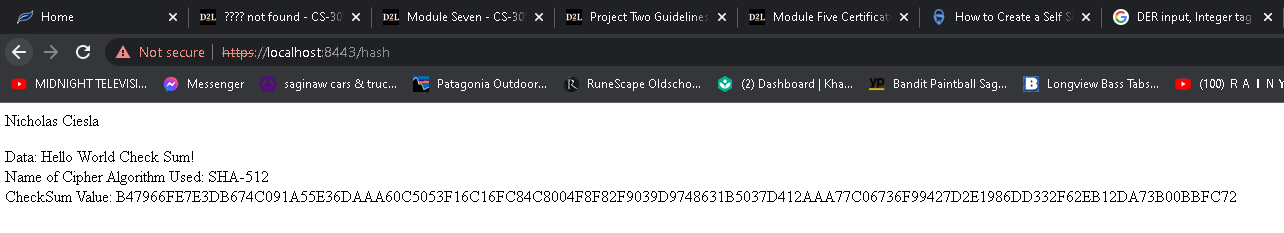
Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.

## 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.

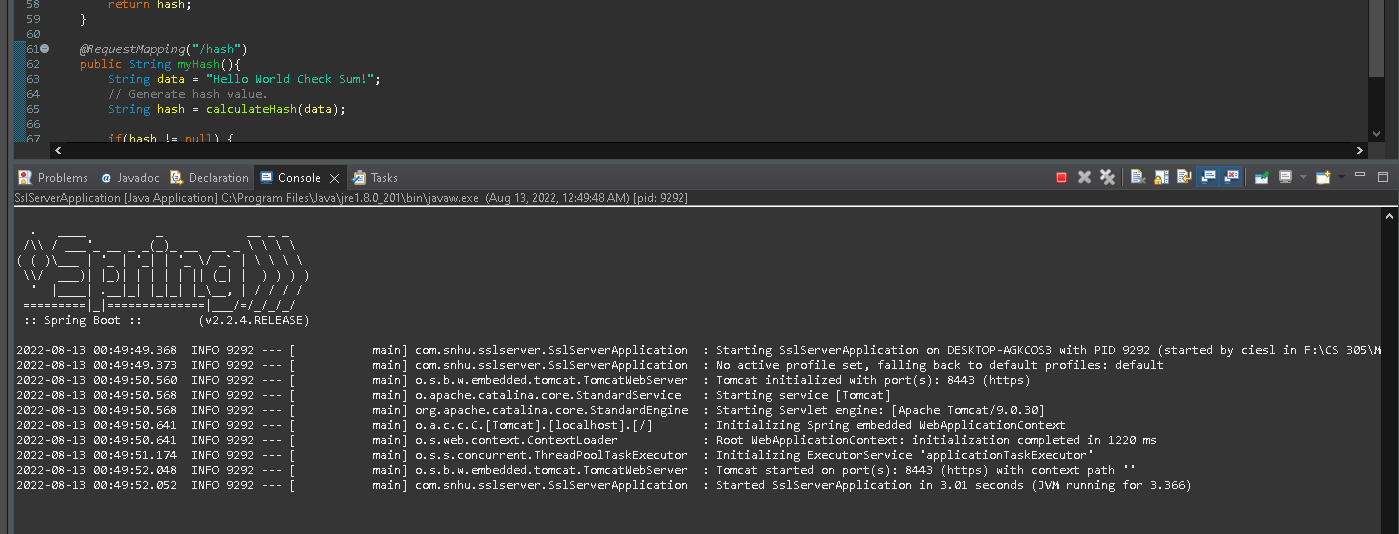


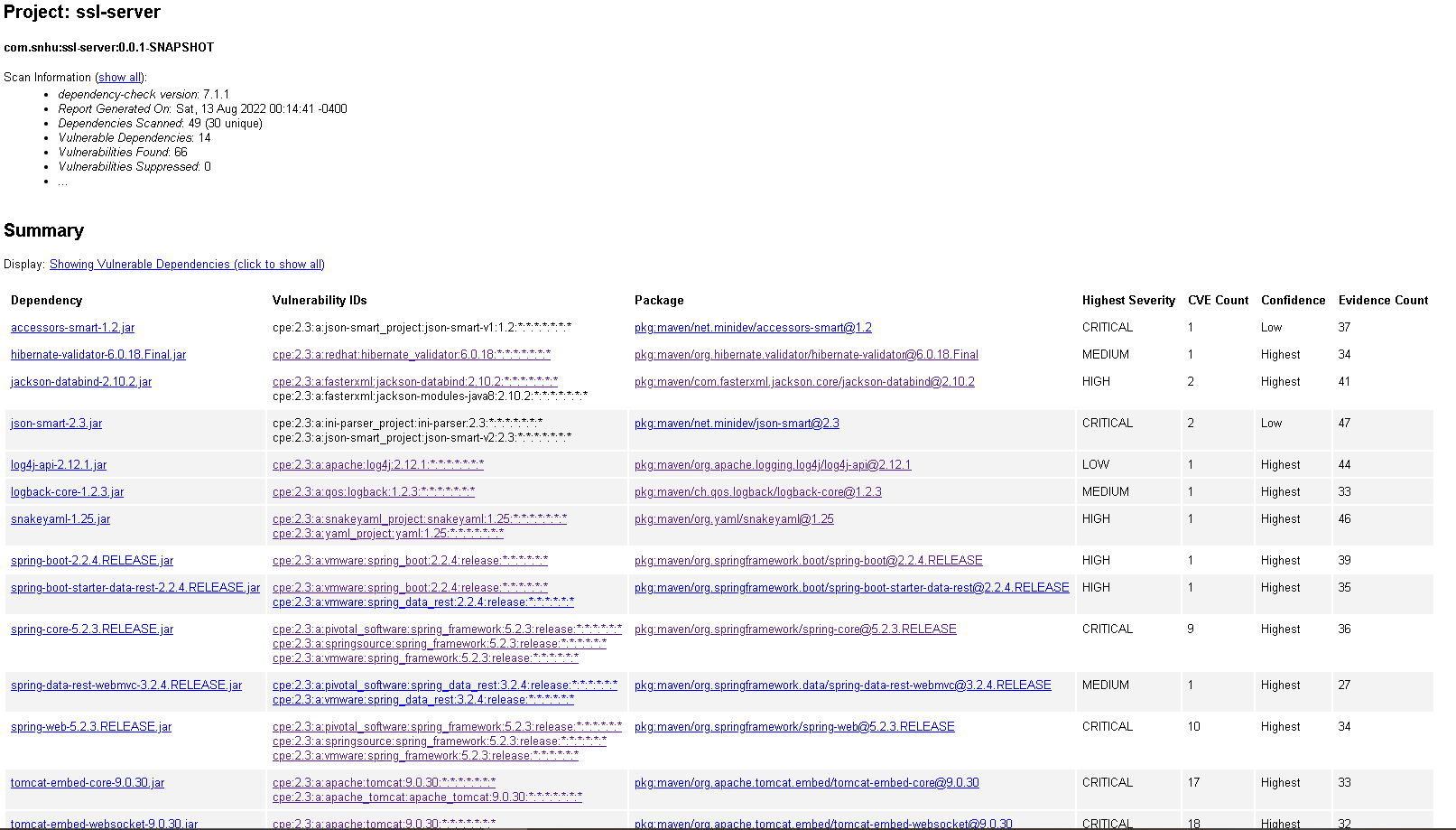
**Because the cert is a self-signed sample cert, it is not seen as valid by chrome. However, HTTPS is being used.**

## 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





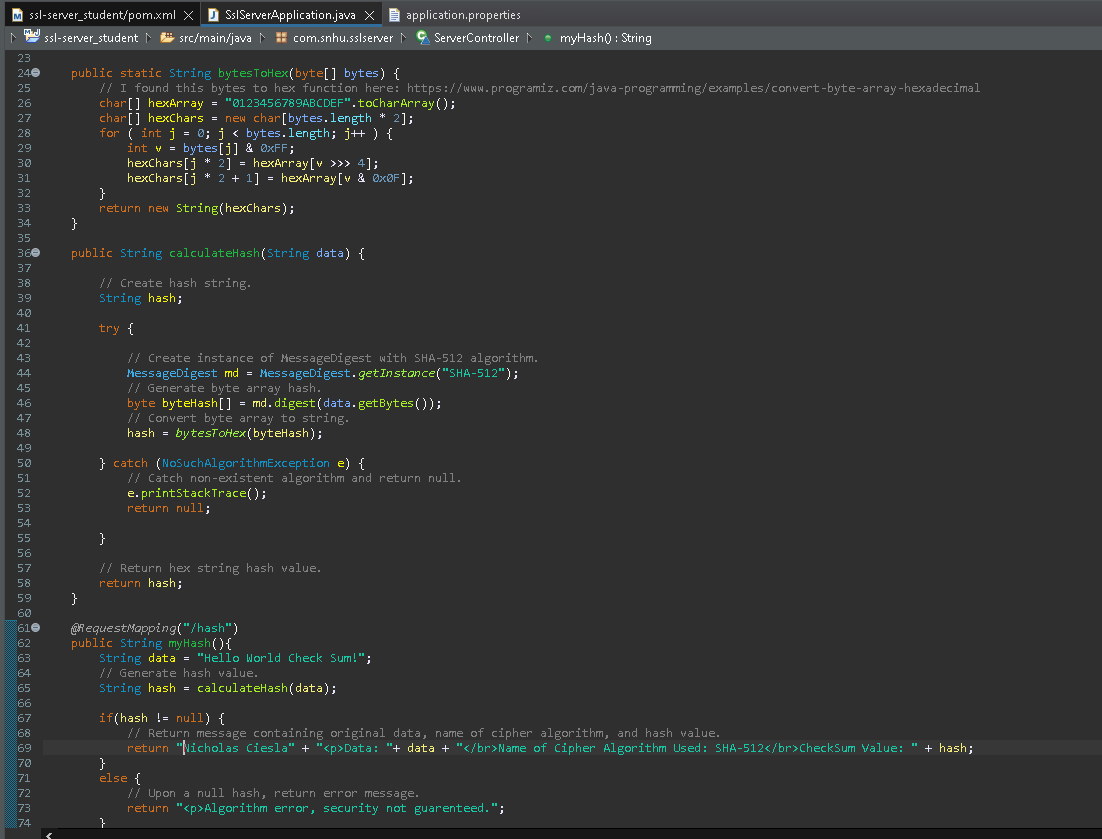
**No new dependency vulnerabilities were found after implementation.**

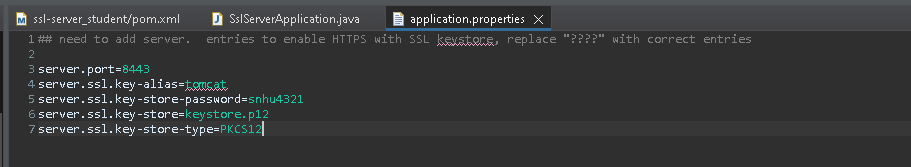
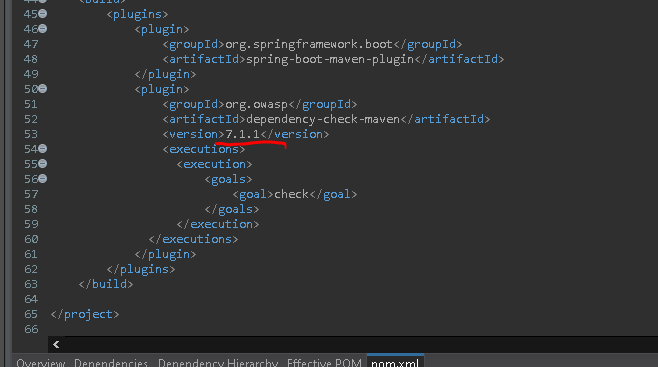
## 

## 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.



**I could only just fit the changes that I have added.**

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.

The refactoring that was done within this project focused primarily on secure client/server communications, code error, and secure API interactions. When it came to secure client/server communications, the changes that were made to the application.properties file upgraded the communications from HTTP to HTTPS. HTTPS is a more modern communication that uses a certificate to verify the identity of the server. Despite my troubles with implementing the certificate, the application now uses HTTPS for communication. Next came error handling. If the hash function was compromised, the function would return a null value. In this case, instead of sending the original message, I sent a message letting the user know that security could not be guaranteed. This warning may stop them from entering their personal information into the site in the event of encryption failure. Lastly, I updated the OWASP Dependency check to the latest version. This ensured that any possible vulnerabilities or issues in older version of the tool would not interfere with the application.

Software Security is something that modern companies should be taking very seriously. Especially in the financial sector, it is not just your data that malicious people are after, but your customer’s data as well. A data breach could cause thousands of cases of identity theft and drained bank accounts. This is something that would be extremely difficult to recover from, not only for your customers, but for your company. A data breach that causes your customers to lose money causes the public at large to lose trust in your company. A data breach large enough could wipe your company off the map forever in a single day. Sound security gives you and your customers piece of mind.

Going forward, it will be important to keep up on dependency checks. These checks alert you of emerging vulnerabilities within the tools that your system is using. These alerts come from a large community of red hat hackers (the good guys) and industry professionals that are seeking out these issues to make computing a safer activity for all of us. Additionally, I would strongly recommend updating all APIs to the latest available production version. This will make sure that they are as free from known vulnerabilities as possible.

Works cited

Ellis, S. (2018, December 13). *The beautiful hash algorithm*. Medium. Retrieved July 30, 2022, from https://steviecellis.medium.com/the-beautiful-hash-algorithm-f18d9d2b84fb#:~:text=Since%20executing%20a%20brute%2Dforce,resistant%2C%20for%20now%20at%20least.

*Java program to convert byte array to hexadecimal*. Programiz. (n.d.). Retrieved July 30, 2022, from https://www.programiz.com/java-programming/examples/convert-byte-array-hexadecimal

5 common encryption algorithms and the unbreakables of the future. Arcserve. (2022, May 24). Retrieved July 23, 2022, from https://www.arcserve.com/blog/5-common-encryption-algorithms-and-unbreakables-future