****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_heading=h.30j0zll)

[**Client 3**](#_heading=h.1fob9te)

[**Instructions 3**](#_heading=h.3znysh7)

[**Developer 4**](#_heading=h.2et92p0)

[**1. Algorithm Cipher 4**](#_heading=h.tyjcwt)

[**2. Certificate Generation 4**](#_heading=h.3dy6vkm)

[**3. Deploy Cipher 4**](#_heading=h.1t3h5sf)

[**4. Secure Communications 4**](#_heading=h.4d34og8)

[**5. Secondary Testing 4**](#_heading=h.2s8eyo1)

[**6. Functional Testing 4**](#_heading=h.17dp8vu)

[**7. Summary 4**](#_heading=h.3rdcrjn)

[**8. Industry Standard Best Practices 4**](#_heading=h.26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/11/2022** | **Becky Scott** | **Initial draft** |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Becky Scott

## Algorithm Cipher

The Hash algorithm cipher I chose from the Java Security Standards page was SHA-256. I chose SHA-256 due to its large checksum size of 256 bytes, which produces an incredibly large checksum space. This is important in choosing a hashing algorithm because the larger the checksum space, the less likely the chance of collisions. Collisions occur when two different inputs in a hash function create the same output. This is especially important with checksums, as attackers can use this to spoof a checksum and get users to download malicious packages instead of the expected package (Walker et al., 2009). It is infeasible with the current hardware to try and find a collision in SHA-256 by brute force.

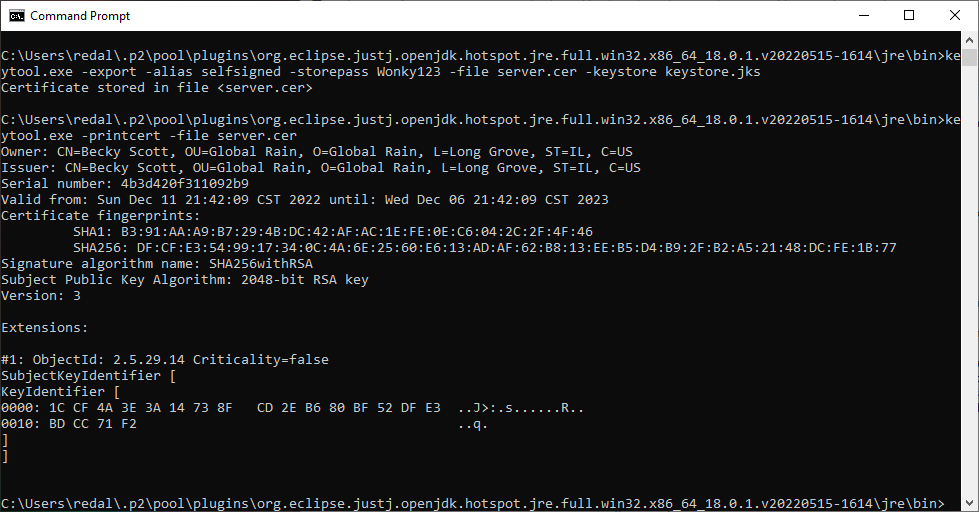
SHA-256, like other hash ciphers, takes the bit values of some data and outputs some distinct value after passing it through its specific function. A difference of one bit between two inputs will yield extremely different outputs. Because of this, hash functions are valuable ways to demonstrate to a user that your software hasn’t been changed by an attacker. If an attacker were to pretend to be you and publish a virus, for example, the checksum wouldn’t match the checksum you generated when you deployed the code, and your user would know that the software isn’t genuine.

While hash functions and encryption ciphers mostly rely on the same underlying principles, there is one key difference: hash functions are one directional and the original message cannot be generated from a hash. Encryption functions can convert ciphertext, which is text that has been passed through the algorithm, back into the plaintext by using a particular key. In symmetric key cryptography, the same key is used to decrypt as was used to encrypt. With symmetric key cryptography, the encryptor needs to figure out how to distribute the key to the decryptor to prevent any attackers from stealing the key. Alternatively, developers can use asymmetric key cryptography, which uses 2 different keys to encrypt and decrypt. Each party has a private key, and a public key is generated based on the private key. The private key cannot be generated using the public key. A person can then use a public key to encrypt a message, and that message can only be decrypted by using a private key, which the intended target should only have access to.

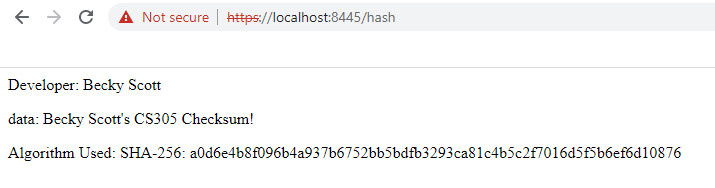
Encryption has been around for much of human history. As long as there has been a need to keep messages secret, there have been encryption ciphers. One of the earliest known ciphers is the Caesar cipher which was used by Julius Caesar. The Caesar cipher is extremely basic compared to today’s encryption ciphers, which encrypted by replacing the letters in the plaintext with a substitute letter shifted a set distance down in the alphabet. Modern day ciphers, such as AES, take a block of bytes of the plaintext and applies it through several mathematical transformations using the private key.

## Certificate Generation

I generated a new certificate using the keytool.exe application within Eclipse.

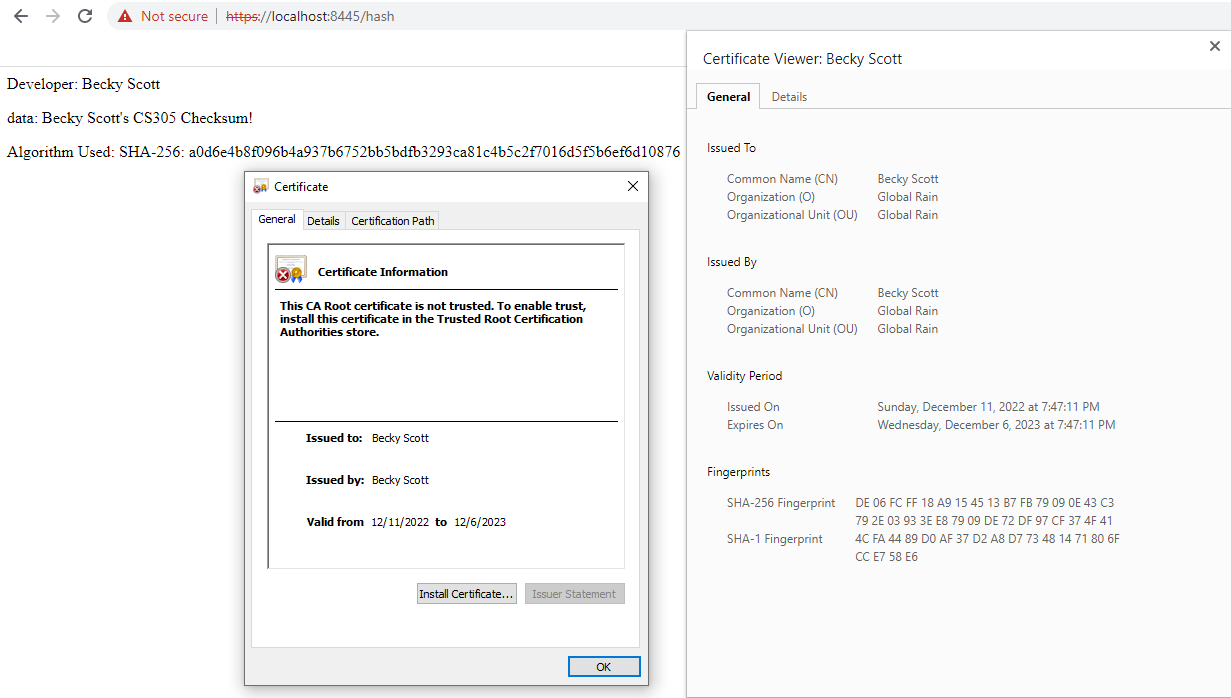


## Deploy Cipher

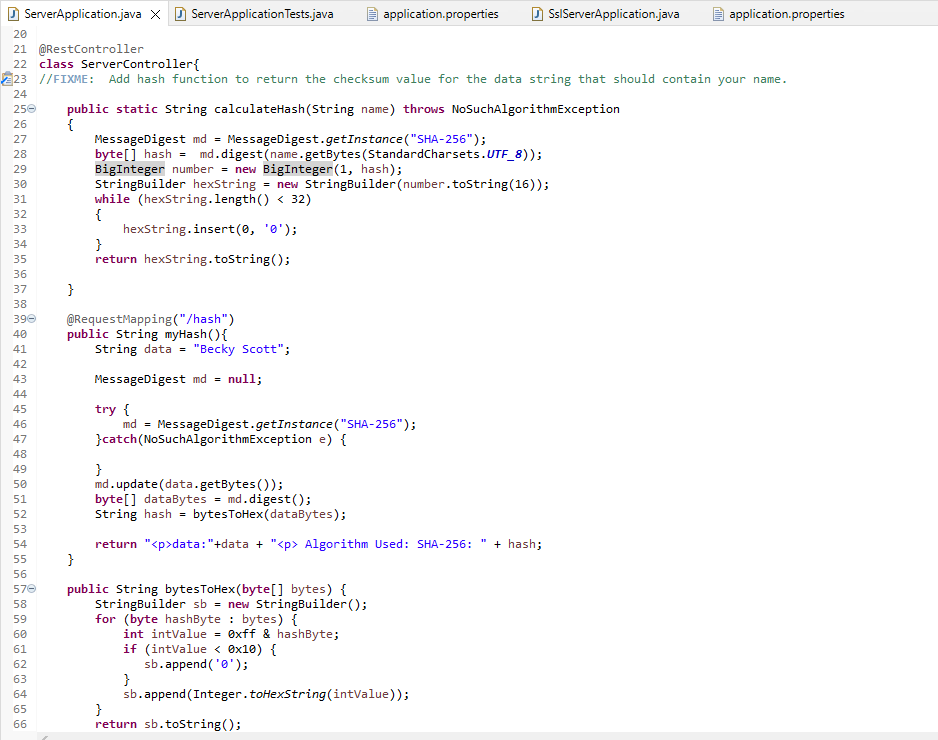


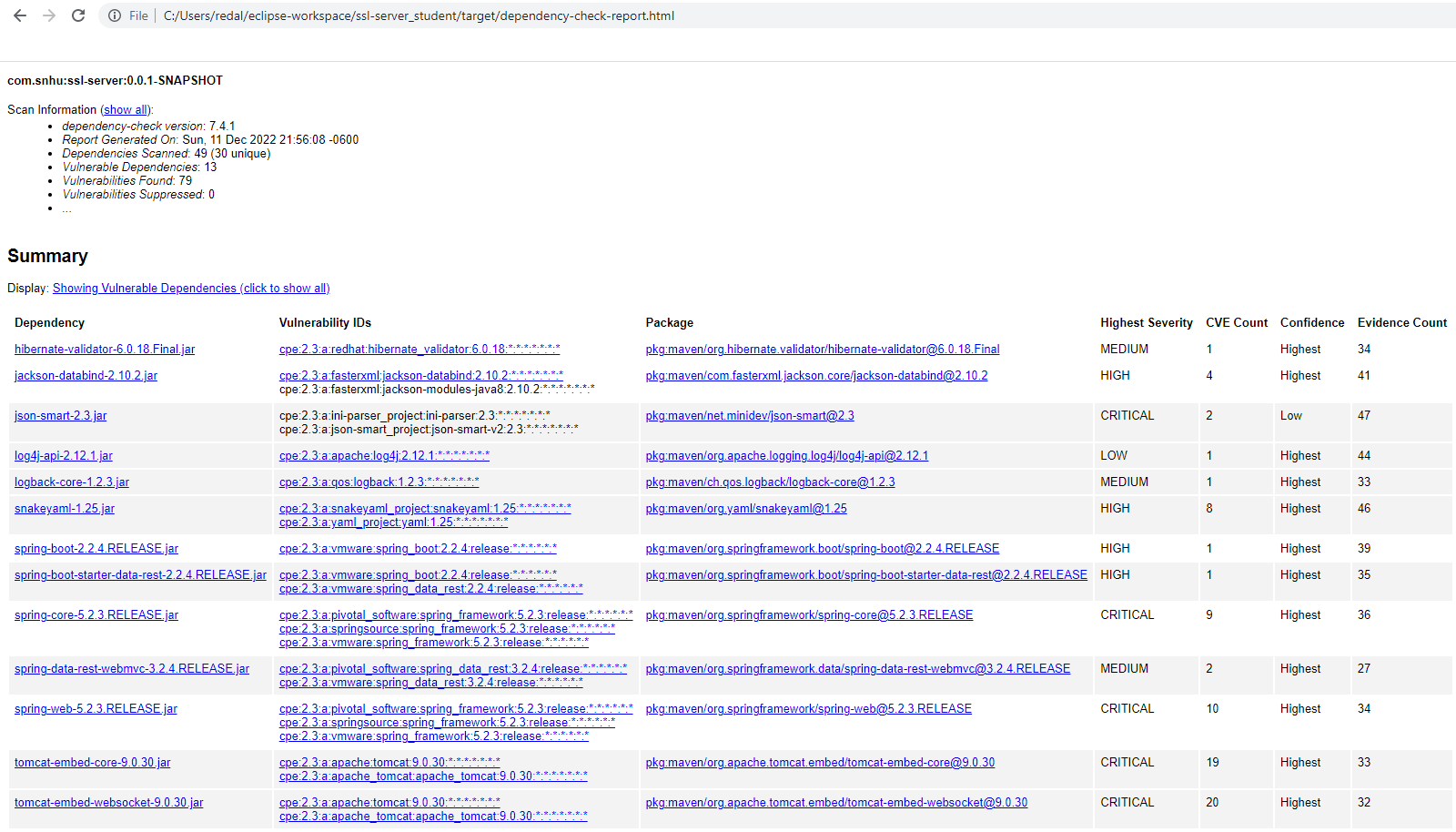
## Secure Communications

When navigating to the URL for our hash, it shows a certificate authority not valid error. This is due to the fact that I am using a self signed certificate. When showing the details of the certificate, I can see that my certificate was used.



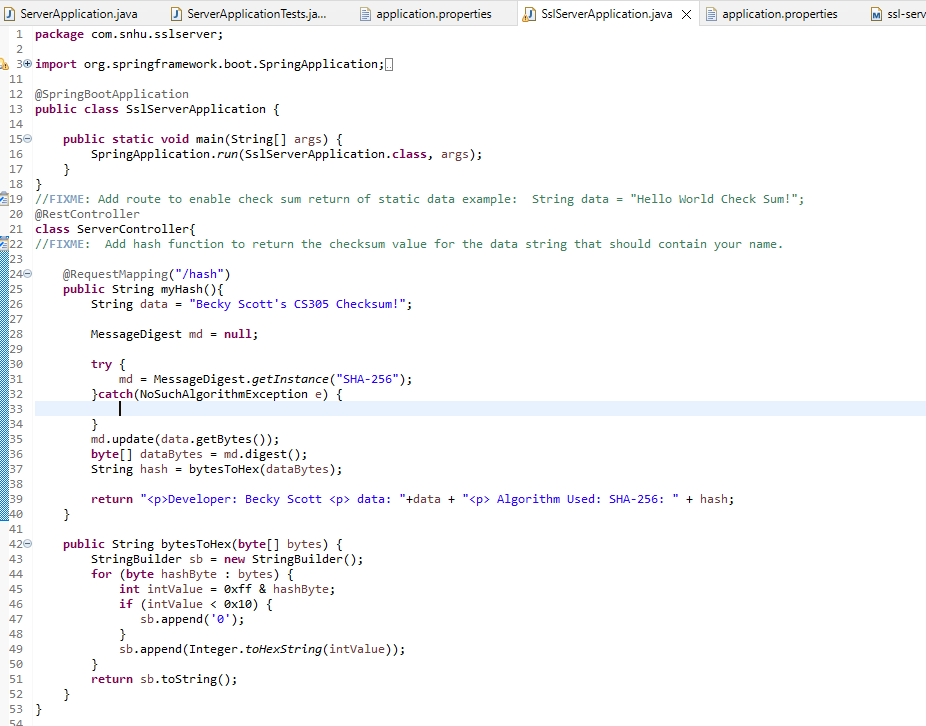
## Secondary Testing





## Functional Testing

I refactored my code by removing the superfluous code in the ServerController class.



## Summary

Originally I had included a function which did not contain explicit error checking in case an invalid algorithm was used. I updated this to use a try and catch statement to handle code errors gracefully. This corresponds to the code error section of the vulnerability assessment process flow. I also reviewed my code to see if there were any areas that did not adhere to the industry standard best practices for secure code design. Since my code did not have any user inputs, I did not have to worry about sanitizing user data or buffer overflow attacks. I considered the use of APIs in my code. My code does not publish any APIs for others to use. My code does rely on some APIs to dependencies, such as the RESTful API to capture HTTP data. I made sure I reviewed the documentation of the packages I was using to ensure I wasn’t coding this insecurely.

My code does utilize cryptography within the checksum generation function. I ensured that this was secure by selecting an appropriate protocol for my use case, SHA-256. SHA-256 is an industry standard and recommended for hashing functions. My code also uses a client / server architecture, so this is a relevant area for security for my code as well. I signed a certificate and enabled SSL to make use of HTTPS for my application. Including a certificate gives the users some validation that my code is legitimate. If I were deploying this code for real, I would use a trusted third party certificate authority to give my users comfort that my code is legitimate and hasn’t been modified. Using SSL and HTTPS provides encryption over user data as they interact with my site. This is a must if I was collecting any sort of sensitive user data such as passwords or PII.

Lastly, I reviewed my code to confirm that I didn’t use any insecure coding practices or data structures which can expose my application. I had some superfluous code in my application left over from the error checking fix. My review identified this code and removed it before it could be exploited.

## Industry Standard Best Practices

One best practice to perform on any code to keep it secure is to continuously check for code vulnerabilities and updates. The cyber threats are always evolving and changing each day and what is secure code today can be full of vulnerabilities tomorrow. The team will need to check for new vulnerabilities daily and when implementing new functions or changing existing ones. When the new vulnerabilities are found, the development team will need to work on mitigating these vulnerabilities and assess the scope and impact of any issues. If the team decides a new vulnerability does not affect their code, then they will need to suppress it. It is possible that there isn’t a solution currently to the vulnerability. A choice will need to be made on how best to handle the situation. If the vulnerability does affect elements in the application, they will need to be on the lookout for updates or solutions.

Staying on top of the most current threats and keeping your code secure helps prevent catastrophic failures. Businesses rely on keeping their systems up and running when they are needed and also rely on their clients/customers knowing their information is safeguarded. Failing to stay current in cyber security will cause damage to businesses productivity and reputation. There can also be legal ramifications if companies do not follow federal regulations on what is needed to operate in various online businesses to maintain a uniform cyber security, such as financial institutions.