****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **Jeffrey McGinty** |  |

## Client



## Developer

Jeffrey McGinty

## Algorithm Cipher

I am recommending the SHA3-256 cipher algorithm. Currently, the SHA-256 algorithm is the most widely used, but the newer SHA3 is considered to be more secure while maintaining a similar level of computational performance. Since the processing power needed to user either algorithm is similar and both are approved for US Governmental use, I believe it is most prudent to go with the newer, more secure choice. At the very least, this will remove any need to update systems later when SHA3 inevitably takes over as the more commonly used cipher.

**What is a hashing cipher and how does it work?**

Following is a brief explanation of exactly what this cipher is and how it is used. SHA3-256 is what is known as a cryptographic hash function. It takes an input (any data of arbitrary size) and produces a fixed-size output of 256 bits that is unique to that input. If even one character in the data is changed, it will produce a completely different 256-bit hash when run through the algorithm. It is typically used for verifying the integrity of data and ensuring that it has not been tampered with. Although this is a cryptographic function, it is important to note that this type of algorithm is used to verify data, not encrypt it for secure transport.

Symmetric key algorithms, on the other hand, are used for encrypting and decrypting data. These algorithms require a shared secret key that is known to both the sender and the recipient. So, while SHA3-256 and symmetric key algorithms are both important components of modern cryptography, they serve different purposes and are not interchangeable.

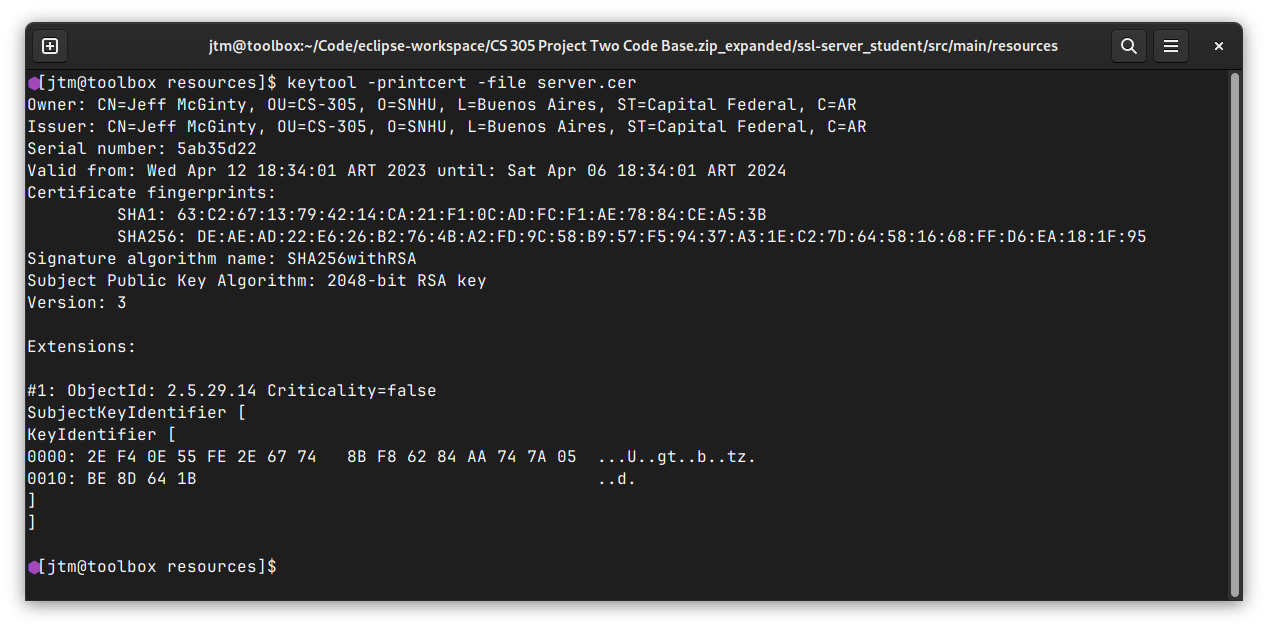
**The past present and future of encryption**

Up until the 20th century, most encryption techniques involved simple symbol substitution or re-ordering to hide underlying information. For example, one might replace each letter in the alphabet by shifting everything three or four letters over, such as replacing A with D, B with E, and so on. Other methods might reverse or otherwise alter the order of letters to produce scrambled phrases that could be decoded by a recipient who knew which technique had been used to create the message.

Increasingly complicated symmetric and asymmetric key algorithms and hash functions have been developed throughout the 20th century and continue to evolve in the 21st. While algorithms like SHA-256 and SHA3-256 are considered strong enough to make it computationally infeasible to try to reverse the encryption process, advances in quantum computing threaten to change that. Now, the goal is to devise even stronger ciphers that can withstand the powerful brute force cracking methods that will inevitably surface as quantum computers continue to evolve.

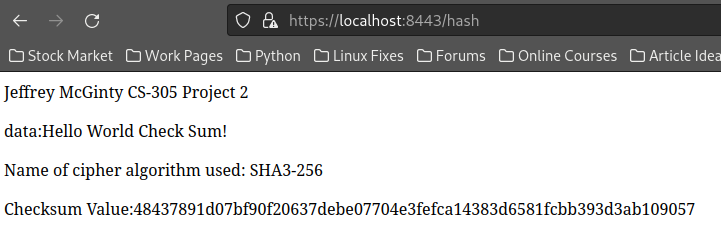
## Certificate Generation

Following is a screen capture of the security certificate generated for testing the web application system and cipher:



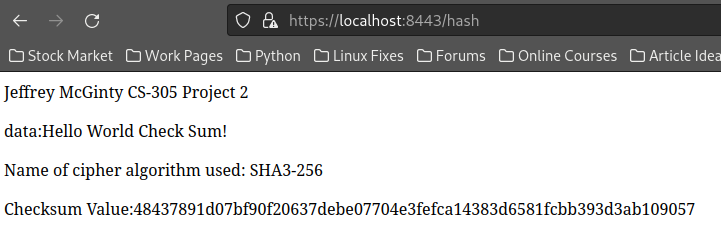
## Deploy Cipher

Following is a screenshot of the output of the SHA3-256 hash function with example data:



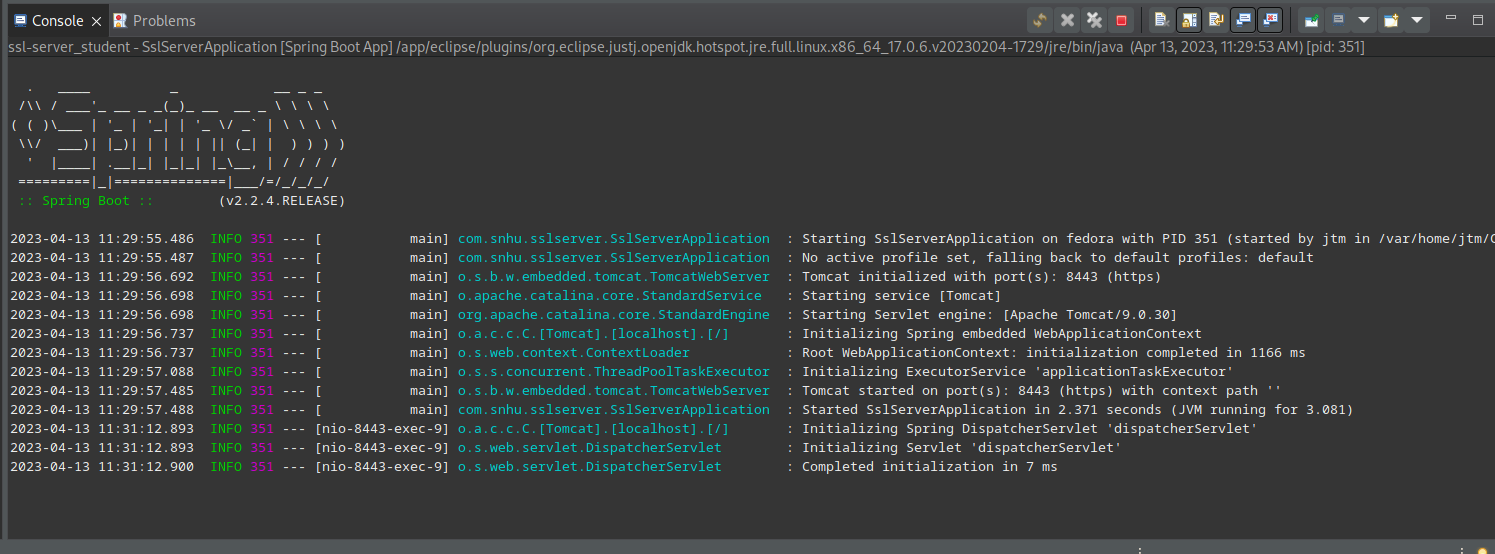
## Secure Communications

The following screenshot also shows that the application is using the secure HTTPS protocol. Note the https:// prefix on the URL:

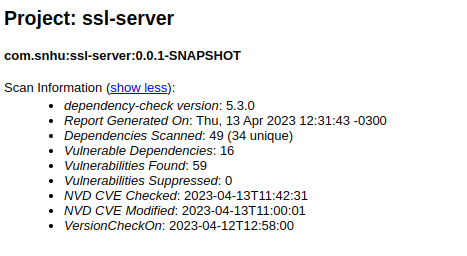


## Secondary Testing

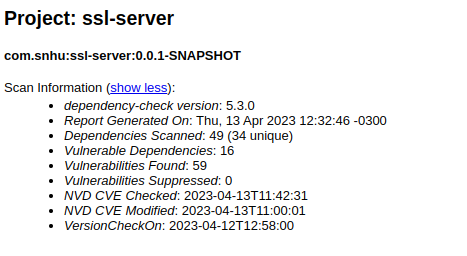
Following is a screenshot of the updated system code compiling and running without error:



In addition, this is shot of the dependency-check vulnerability report before any code was altered:

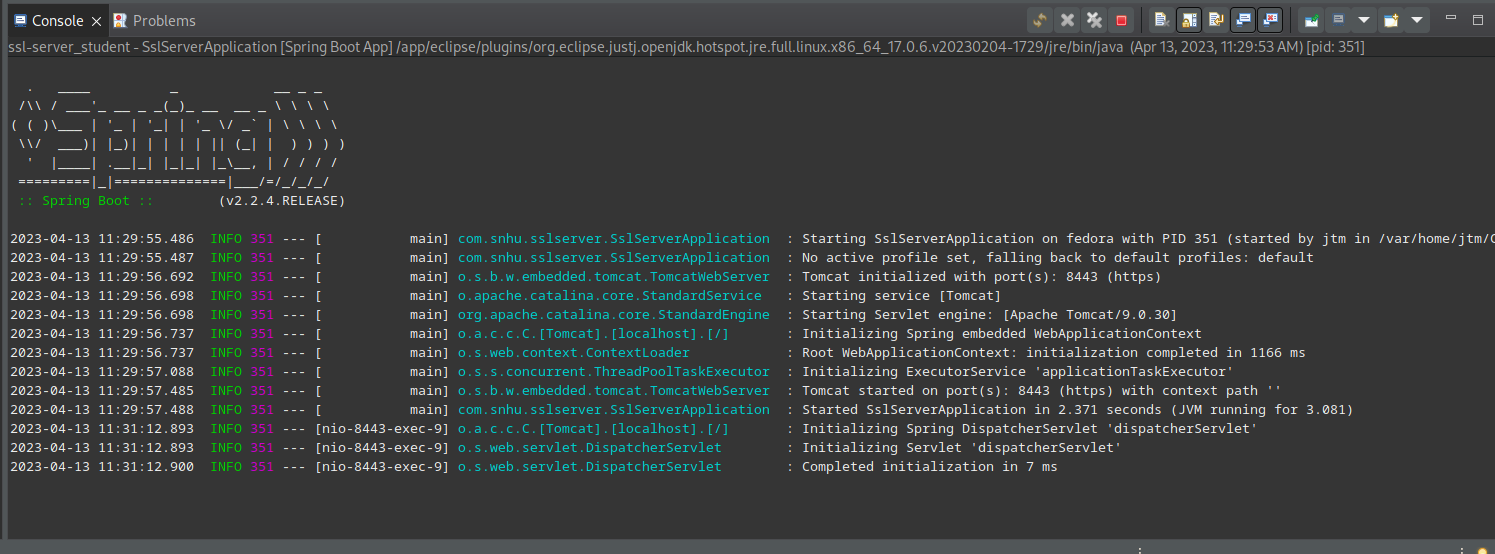


And finally, the following is a shot of the vulnerability report after the code was updated, showing that no new vulnerabilities have been introduced into the system:



## Functional Testing

Following is a screenshot of the updated code compiling and running without error after functional testing was completed:



In its current form, the code written outside of the framework dependencies does not exhibit any obvious security problems. The system will operate over the secure HTTPS protocol and does not use any form of user generated input to function. Trying to enter URLs other than the /hash URL results in 404 errors and does not expose any system information.

The vulnerabilities present in the system involve the dependencies for the Spring Framework used to build the web application. The resolution for these vulnerabilities is to simply update to the latest version of the framework. Currently reported vulnerabilities are known and have been remedied in later releases of Spring Framework.

## Summary

The supplied code was refactored to include a secure API access point (/hash), enable a cryptographic hash function to be used to validate data, and enable secure communications (HTTPS) between the server and clients using the API.

Each of the enabled features was added in an iterative process. As one feature was added, tested, and determined secure, the next was then added in the same fashion. Following this methodology, we could be sure that no new vulnerabilities were inadvertently introduced into the system as it was altered.

## Industry Standard Best Practices

While there is no singular exhaustive list of best practices for coding web-enabled applications of this type, the following recommendations will help to ensure that as project development continues, the product will be as secure as possible at any given point.

* Ensure that all user input is validated and sanitized to prevent malicious code injection. Use proper validation techniques to avoid buffer overflows, SQL injection, and other attacks.
* Use encryption techniques to protect sensitive data, such as passwords and credit card information. Use strong encryption algorithms such as AES or RSA.
* Implement robust authentication and authorization mechanisms to ensure that users have access to only the data they need. Use secure protocols like HTTPS and implement multi-factor authentication.
* Protect data at rest by using secure storage mechanisms such as encryption, hashing, and salting.
* Keep software up to date with the latest security patches and updates to ensure that it is protected against the latest security threats.
* Establish secure coding guidelines and ensure that all developers follow them. This helps to ensure consistency in coding practices and reduces the likelihood of security vulnerabilities.
* Conduct code reviews and testing to identify and fix security vulnerabilities.

Consciously incorporating security-minded best practices into the entire development process results in several key benefits. The overall number of security flaws found will be reduced. Flaws that are discovered will be found earlier in the process, reducing or eliminating costs associated with fixing them. Flaws exposed to the public in production environments will be minimized, if not eliminated. And finally, practice, as they say, makes perfect. Developers will inevitably and automatically become better at constructing secure code right from the start as security becomes a regular part of their mindset as they work.