****

# 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]** | **[Your Name]** |  |

## 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

Nicholas Boyer

## Algorithm Cipher

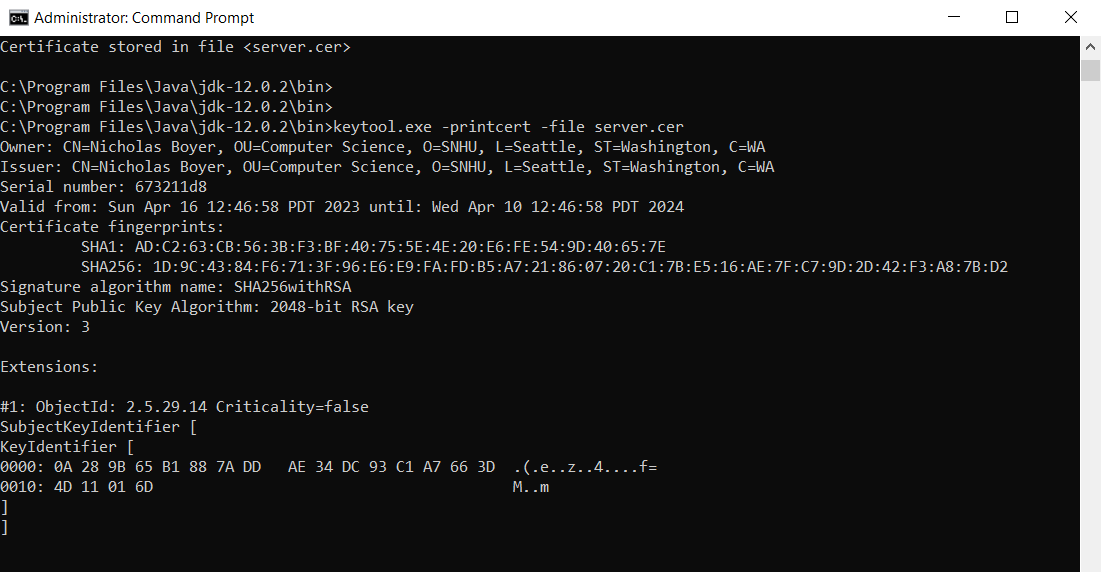
We will use the SHA-256 hashing algorithm. The SHA-256 algorithm divides a ciphertext into a series of blocks and applies padding so that each block is 256 bits long. The hashing algorithm applies a series of transformations on the blocks so that we get a 256-bit hash value as a result.

A symmetric cipher uses one key for encryption as well as decryption. An asymmetric cipher utilizes a private key for signing or encrypting data and a public key for verification of the data. It is important to use random numbers when generating a public or private key, because if predictable numbers are used, the keys may be predicted by a hacker which would give them access to the encrypted data and defeat the purpose of the cipher.

Ciphers have been used since ancient times to encrypt messages, but they were not very secure. Since the advent of the Enigma machine during World War II, computer-assisted cryptography has vastly improved the quality of cryptograph algorithms. DES was developed in the 1970s and was used for a few decades, though now it is rarely used because it can be broken with ease. An asymmetric cryptographic algorithm named RSA was developed in the 70s and is still used to this day. AES was developed in the 90s to succeed the much weaker DES algorithm, and AES is still used to this day. To date, no AES-128 encrypted message has ever been decrypted by brute force.

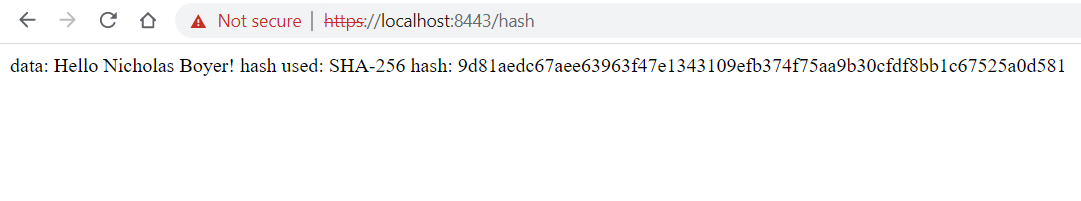
## Certificate Generation

Insert a screenshot below of the CER file.



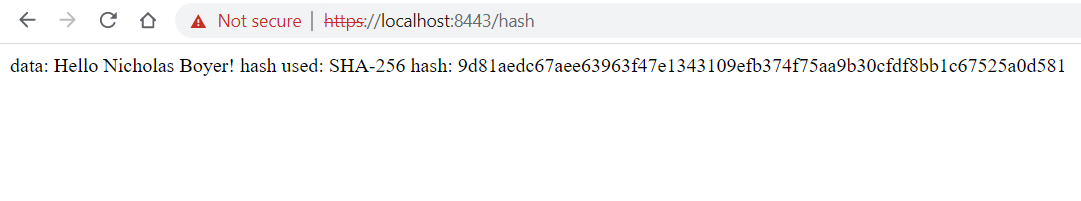
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

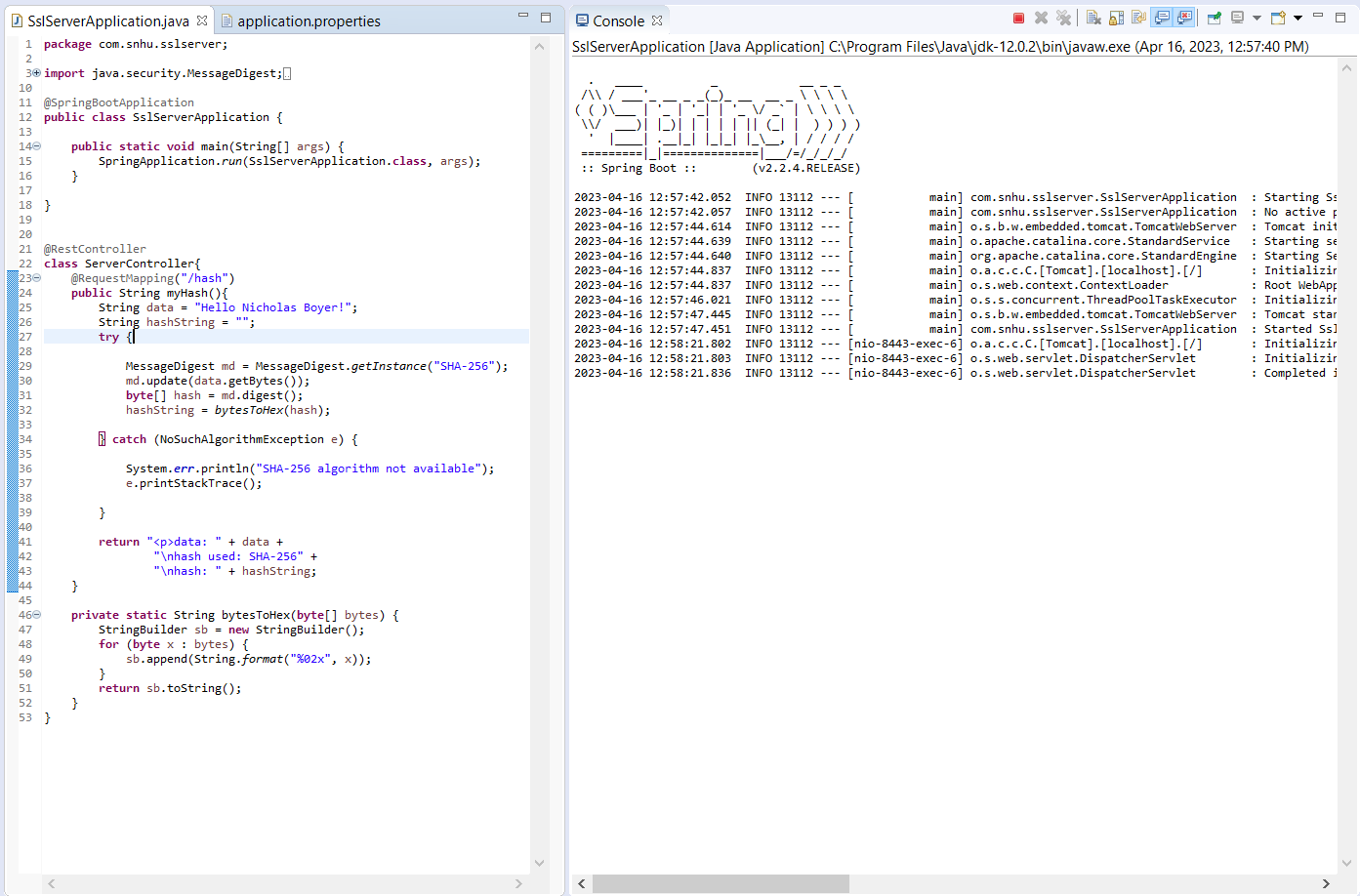
Insert a screenshot below of the web browser that shows a secure webpage.

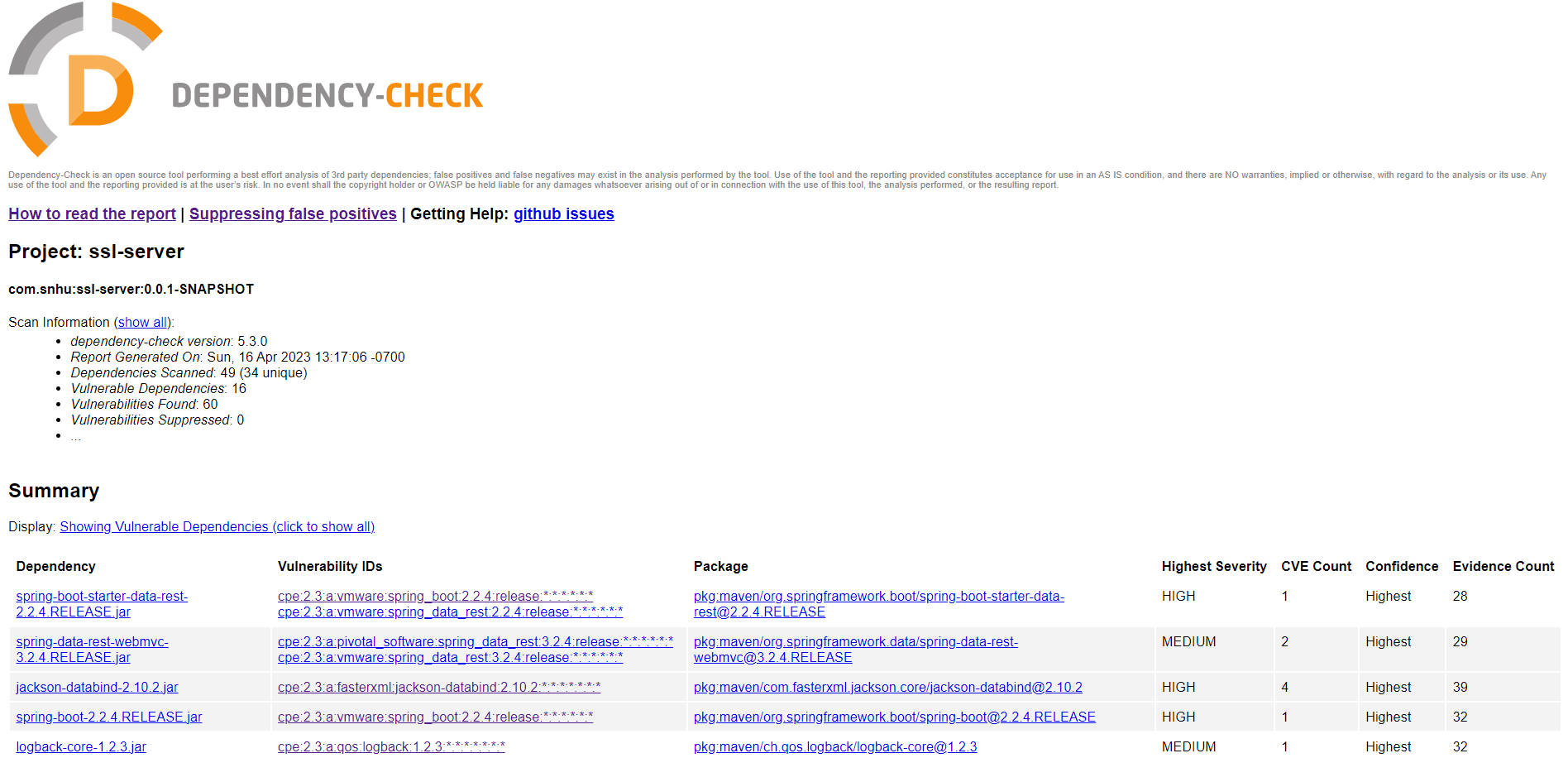


On the Chrome browser, https is red and crossed out only because we used a self-generated certificate. However, be rest-assured that we are using HTTPS and the communications are safe.

## Secondary Testing

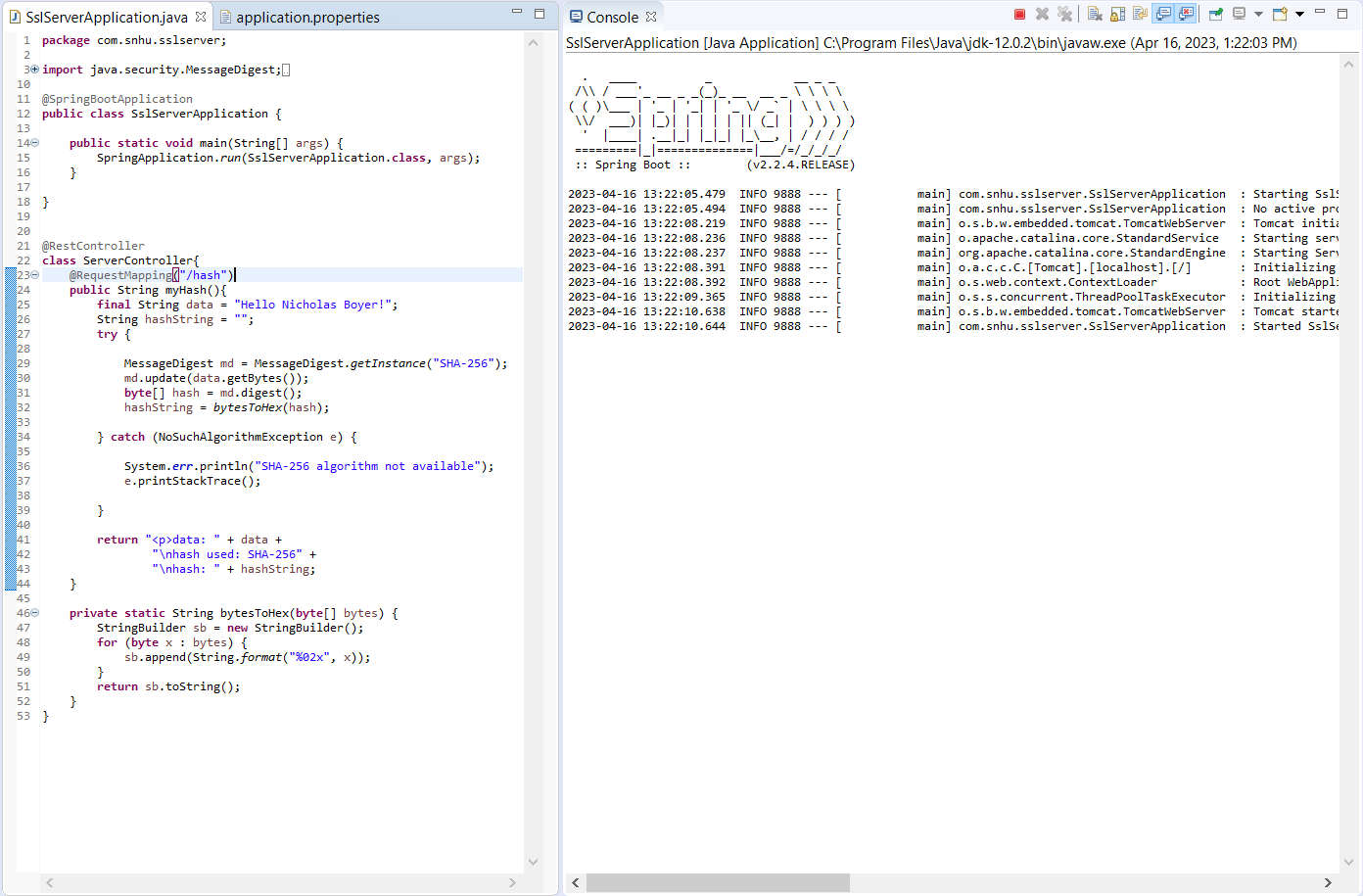
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

Insert a screenshot below of the refactored code executed without errors.



## Summary

In the unfactored code, the string to be hashed was not final. This means that it could be modified after initialization, which would leave open an attack via input manipulation. To fix this, I changed the String variable data to final. This way, the string cannot be modified after initialization, meaning an attacker would not be allowed to change the string to some input that would open up a vulnerability. This revision of the code focuses on input validation and on code quality.

When it comes to adding layers of security to the application, it is best to focus on one thing at a time. Once a security vulnerability has been identified, it should be documented and edits to the code should be made to fix it promptly. Once one vulnerability has been fixed, we should move on to the next identified vulnerability. Working on one vulnerability at a time gives the developer the benefit of not becoming overwhelmed by multiple vulnerabilities at once. This is likely to improve the developer’s focus and the quality of their code.

## Industry Standard Best Practices

Applying standard best practices is vital to the wellbeing of the company. It assures that code is readable, maintainable, and that glaring security vulnerabilities are not left in the code. This benefits the company by reducing costs, improving efficiency, and strengthening security.

Some practices that we should always employ include making sure code is readable, testing code on a regular basis, and using up-to-date dependencies. Readable code helps developers understand how the codebase works and make edits to the code faster. If code is unreadable, developers will spend a lot of time scratching their heads to figure out how functions, classes, and so on actually operate. Testing code on a regular basis ensures that errors are found promptly and are dealt with before they become a big problem. It Is much better to test code often and fix bugs as they appear than write a bunch of code at once and deal with many errors after the first compilation. Using up-to-date dependencies prevents exploits from old versions of dependencies cropping up in the software. Often times, many vulnerabilities can be eliminated simply by updating the dependencies.

Other industry standards that we should always employ include using query parameterization, hashing passwords, and using certificate-based authentication. When using SQL to store and retrieve data, it is always best to use query parameterization. This technique virtually eliminates the possibility of an injection attack. When passwords are stored for user or admin credentials, they should be hashed instead of stored as plaintext. This way, if the server’s storage is compromised, the hackers do not have access to everyone’s passwords in plaintext. When signing users in, the password should be hashed and then compared to the list of hashed passwords for entry. Finally, using certificate-based authentication will improve the reputation of a company’s website. When a digital certificate is used, users can be assured that they are accessing the intended website instead of a fake.