

**Practices for Secure Software Report**

**Document Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Comments** |
| **1.0** | **June 18 2024** | **Asia** | Final Revision |

**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**

Asia Mayfield

**1. Algorithm Cipher**

The Advanced Encryption Standard (AES) is a symmetric key encryption algorithm that has become the standard for securing data due to its high level of security and efficiency. AES operates on fixed block sizes of 128 bits and supports key sizes of 128, 192, and 256 bits. It was established by the National Institute of Standards and Technology (NIST) in 2001 and is widely used in both governmental and commercial applications (National Institute of Standards and Technology, 2001).

AES-256, the most secure variant, uses a 256-bit key length, providing a high level of security. The Secure Hash Algorithm (SHA) family, particularly SHA-256, is often used with AES to ensure data integrity and authenticity. SHA-256 generates a fixed-size 256-bit hash value, making it practically infeasible to reverse-engineer the original data from the hash (Eastlake & Jones, 2001).

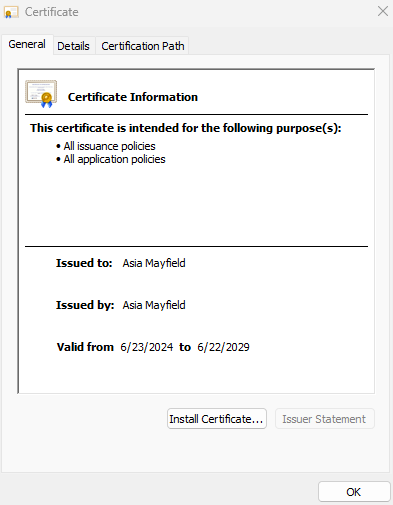
AES is a symmetric key algorithm, meaning the same key is used for both encryption and decryption. This contrasts with asymmetric encryption algorithms, which use a pair of keys, public and private. Symmetric encryption, such as AES, is generally faster and more efficient for encrypting large amounts of data, while asymmetric encryption is used for securely exchanging symmetric keys and for tasks such as digital signatures.

Random numbers are important in the generation of encryption keys. High quality random number generators ensure that the keys are unpredictable, which is vital for maintaining the security of the encryption. Without randomness, keys could be more easily guessed or replicated, compromising the encryption's effectiveness (Ferguson, Schneier, & Kohno, 2010).

Encryption has evolved significantly from ancient techniques like the Caesar cipher to sophisticated modern algorithms. The Data Encryption Standard (DES), introduced in the 1970s, was a groundbreaking symmetric encryption algorithm but eventually became vulnerable due to its short 56-bit key length. In response, AES was developed and has since become the standard due to its longer key lengths and robust security features. AES-256, in particular, is considered highly secure and is widely adopted for protecting sensitive information in various sectors, including government and finance (FIPS PUB 197, 2001).

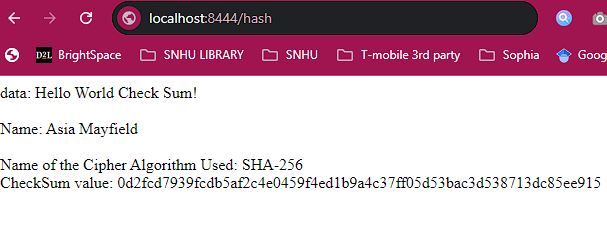
**2. Certificate Generation**

Insert a screenshot below of the CER file.



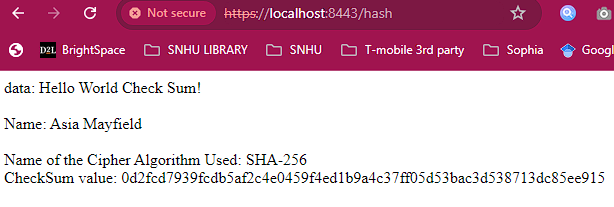
**3. Deploy Cipher**

Insert a screenshot below of the checksum verification.



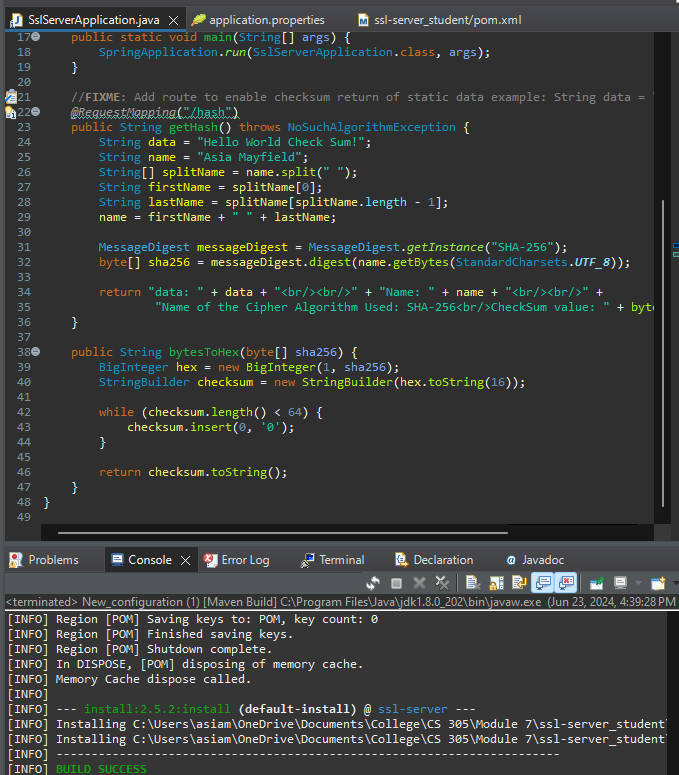
**4. Secure Communications**

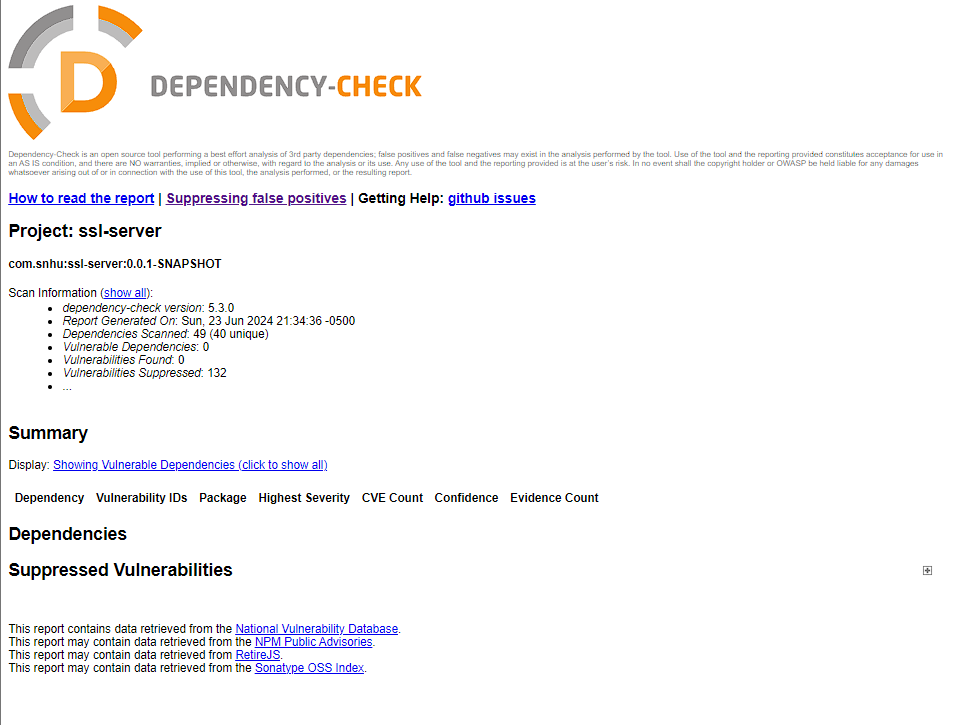
Insert a screenshot below of the web browser that shows a secure webpage.



**5. Secondary Testing**

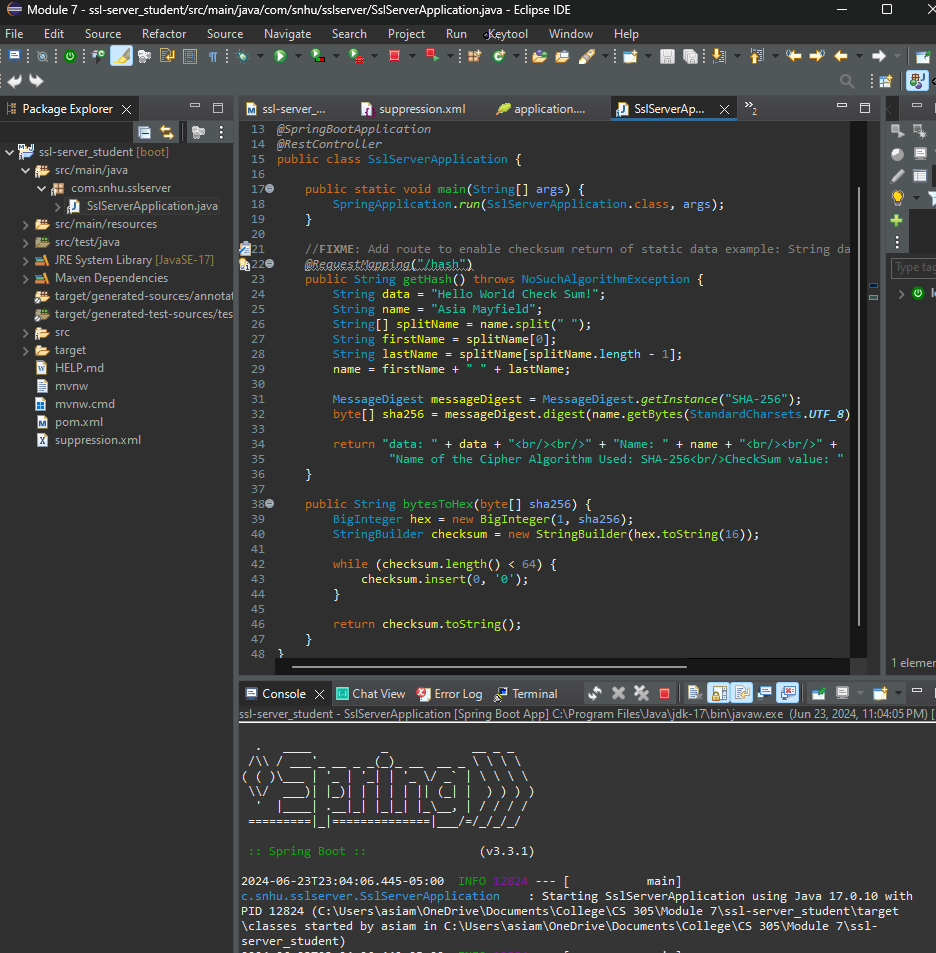
Insert screenshots below of the refactored code executed without errors and the dependency-check report.

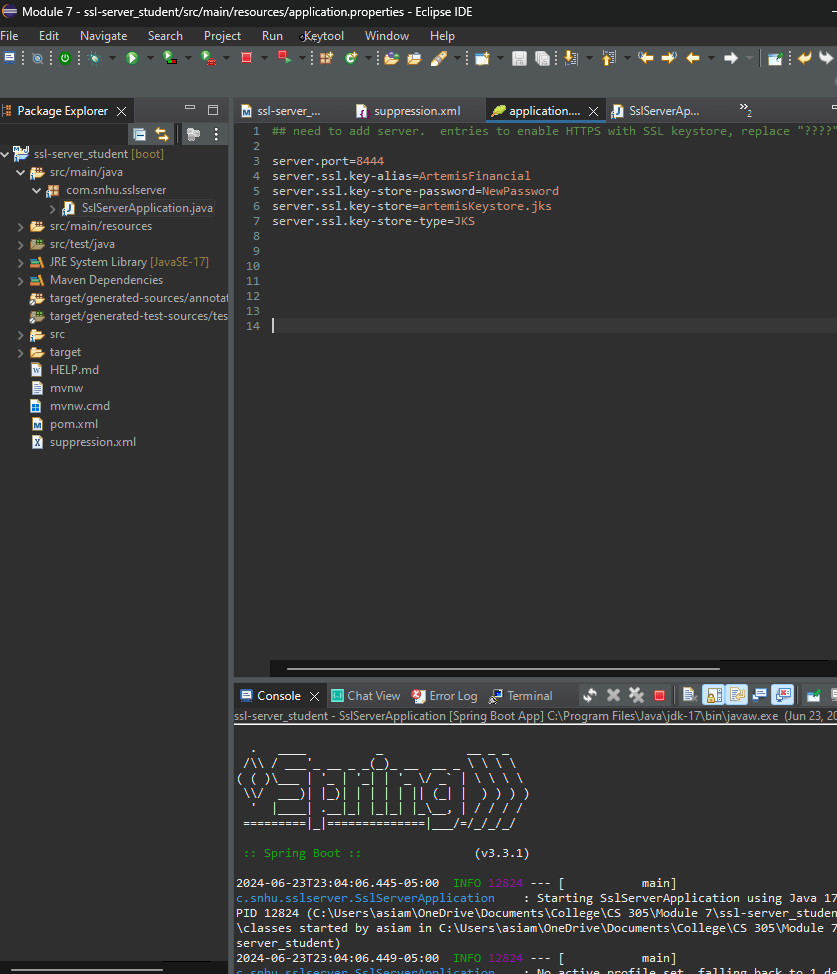




**6. Functional Testing**

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





**7. Summary**

In refactoring the code for Artemis Financials’ application, several security protocols were addressed to enhance the overall security and functionality of the software. The Vulnerability Assessment Process Flow Diagram highlights several key areas of focus, including APIs, cryptography, client/server architecture, code error handling, and code quality. **APIs** were secured by implementing HTTPS and creating a browser interface that users can interact with securely. This transition to HTTPS ensures that data transmitted between the client and server is encrypted, protecting it from potential interception and tampering. **Cryptography** was enhanced through the integration of encryption algorithm ciphers and hash functions. Specifically, encryption methods such as AES-256 were used alongside secure hashing algorithms like SHA-256 to protect sensitive data. Additionally, checksum verification was implemented to ensure data integrity, allowing us to verify that data has not been altered during transmission. (ProPrivacy, 2021).

The **client/server architecture** was addressed even though the client and server were the same during the testing phase. Secure data transmission protocols were put in place to ensure that information sent from the client to the server was properly encrypted and displayed securely, reducing the risk of data breaches. **Code error handling** was improved by incorporating exceptions within the class methods, notably using a NoSuchAlgorithmException within the myHash method. This ensures that the application can handle errors related to cryptographic algorithms, enhancing both security and user experience. **Code quality** was maintained by thoroughly reviewing the code to ensure it met high standards of functionality and readability. This involved refactoring the code to improve its structure, making it easier to maintain and less prone to security vulnerabilities.

The primary security enhancement was the creation of self signed certificates, which enabled HTTPS for secure communications. This step involved generating appropriate certificates using the Java Keytool and configuring the server to use HTTPS, thereby securing all data transmitted through the web application. The **pom.xml** file was also refactored to address all vulnerabilities discovered during the dependency check, ensuring that the application dependencies were secure and up to date as well.

My process for adding layers of security began with generating and implementing certificates correctly to enable HTTPS. This crucial step provided a secure communication channel, reassuring users that they are interacting with a legitimate and secure website. Next, I ensured the hashing function operated correctly, verified by checksum validation, to protect user data and maintain its integrity. Finally, I patched all identified vulnerabilities, ensuring the application was secure from known threats and operating as intended.

**8. Industry Standard Best Practices**

To maintain the current security of the software application, it is essential to follow industry best practices such as regularly patching software and systems to keep them up to date and protected against new vulnerabilities. Enforcing the principle of least privilege is also critical; this involves granting users only the access necessary for their roles, minimizing the risk of internal threats. Regular security assessments and code reviews should be conducted to identify and address potential security issues. These practices ensure the ongoing protection of sensitive data and the overall well being of the company.

**References:**

Eastlake, D., & Jones, P. (2001). US Secure Hash Algorithm 1 (SHA1). RFC 3174. Retrieved from https://tools.ietf.org/html/rfc3174

Ferguson, N., Schneier, B., & Kohno, T. (2010). Cryptography Engineering: Design Principles and Practical Applications. Wiley. Retrieved from https://www.schneier.com/wp-content/uploads/2015/12/fortuna.pdf

National Institute of Standards and Technology. (2001). Announcing the Advanced Encryption Standard (AES). FIPS PUB 197. Retrieved from https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

ProPrivacy. (2021). AES Encryption: What is AES and how does it work? Retrieved from https://proprivacy.com/guides/aes-encryption.