

# 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** | **6/15/2023** | **Meharban Singh** |  |

## 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

Meharban Singh

## Algorithm Cipher

Given the sensitive nature of Artemis Financial's data and the regulations governing the protection of consumer financial information, prioritizing data security is crucial. To achieve this, I recommend implementing the AES cipher, a widely accepted encryption algorithm that remains unbroken and is considered the industry standard. AES, or Advanced Encryption Standard, utilizes a 256-bit cipher and is trusted by various government agencies handling sensitive data. Its compliance with regulatory requirements is endorsed by the National Institute of Standards and Technology.

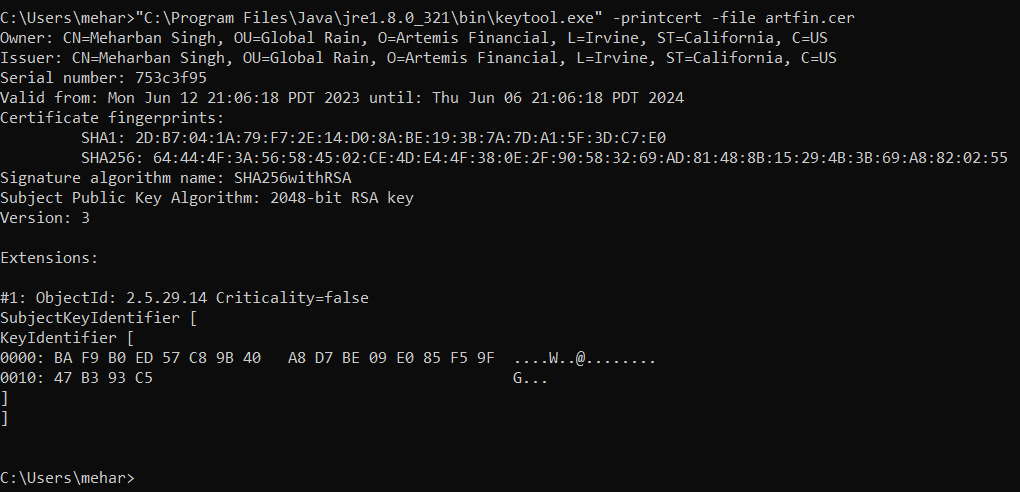
Data integrity is maintained through the use of hash functions, which transform data into a unique hash value that changes if any modifications occur. By checking for unauthorized changes, we can ensure the security of our information. The length of the hash value, measured in bits, determines the level of difficulty for hackers attempting to compromise data integrity. SHA is commonly used for hashing data and offers various bit levels based on desired security.

For encryption, symmetric key encryption is a more straightforward and efficient approach, where the same key is used for both encryption and decryption. Asymmetric encryption, on the other hand, employs separate public and private keys and utilizes mathematical algorithms involving prime numbers to resist brute force attacks. While asymmetric encryption provides higher security, it is less efficient due to computational requirements.

AES is an example of the Data Encryption Standard (DES), which was initially introduced in the 1970s and evolved as computing technology advanced. DES served as the national standard until it was cracked in 1997. Subsequently, AES was established as the new national standard in May 2002 by NIST, and it remains unbroken and widely used today. The SHA-256 cipher was created a year prior to AES. It was also created as an improvement over its predecessor, the SHA-1 cipher, which was found to have some key irrevocable vulnerabilities and had to be retired. The SHA-256 was developed by the NSA as an improvement upon SHA-1 and, like AES, has become the standard encryption algorithm used to create hashes.

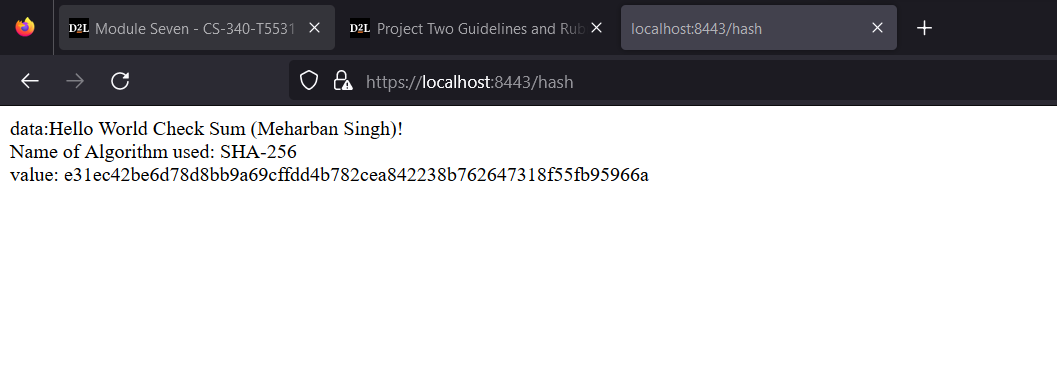
## Certificate Generation

Insert a screenshot below of the CER file.



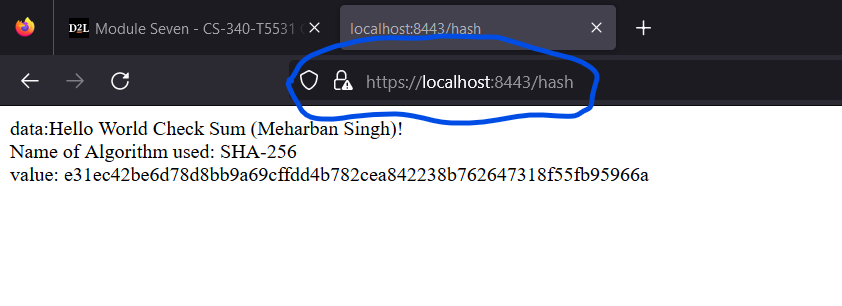
## Deploy Cipher

Insert a screenshot below of the checksum verification.



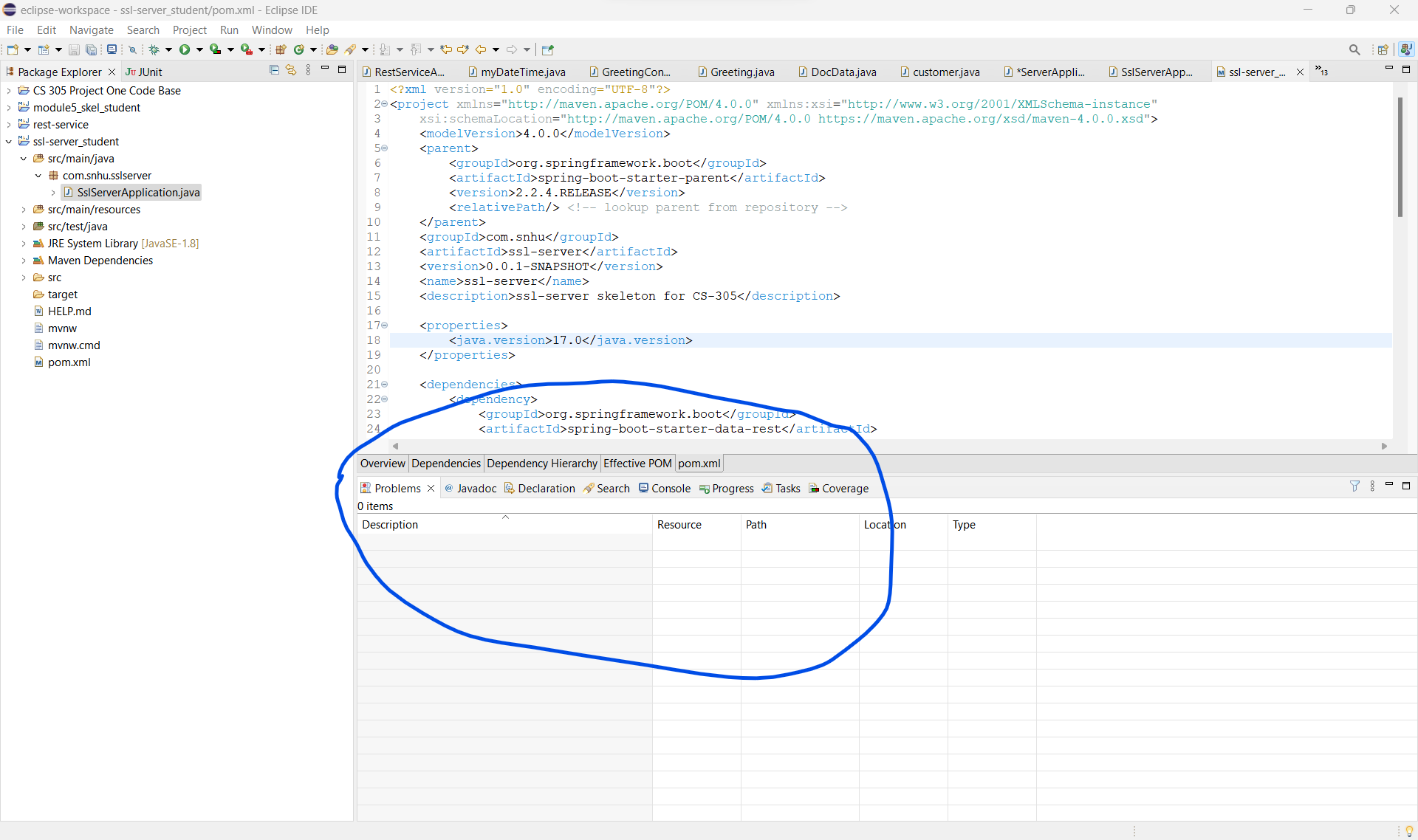
## Secure Communications

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



## Secondary Testing

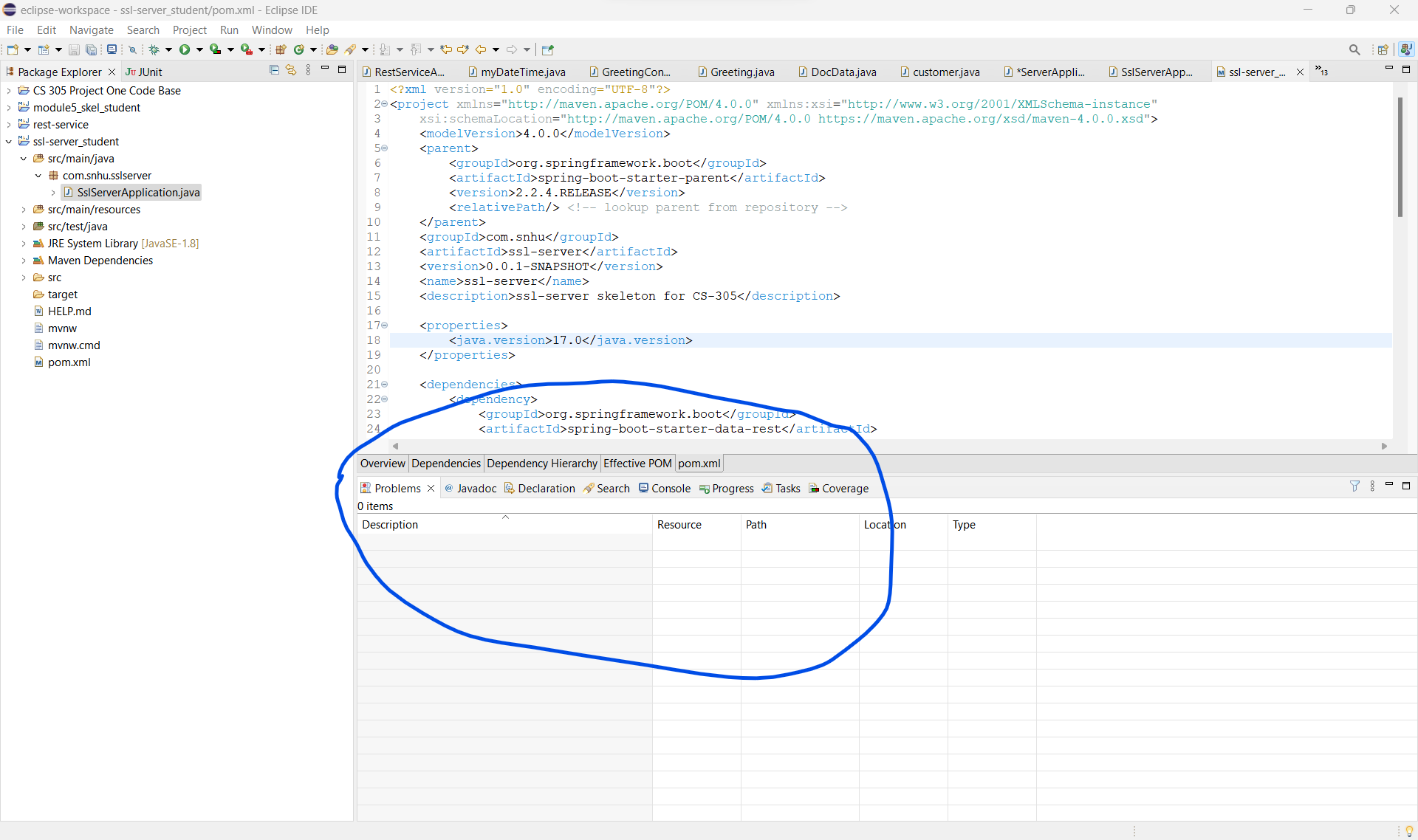
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



## Summary

The areas of security that I focused on for this project were: cryptography, client/server, and code quality.

* **Cryptography**: To ensure secure handling of data within the application, I implemented the SHA-256 cipher to hash the data and generate a checksum. This cipher guarantees the integrity of our customer information and reduces the likelihood of a data breach since any alterations to the data will result in a changed hash. Hashing data does not encrypt it but employs cryptography to prevent tampering. Additionally, I used HTTPS to encrypt the data transmitted between the client and the server, ensuring that it remains confidential and inaccessible to third parties. I will discuss the use of HTTPS further in the client/server section.
* **Client/Server**: To prevent unauthorized access to data transmitted between the client and the server, I utilized HTTPS. This encrypts the data during transit, enabling secure communication between the two. By employing HTTPS instead of HTTP, we add an extra layer of security to our customers' data while it is in transit, significantly reducing the chance of interception and unauthorized access to sensitive information such as credit card details, bank accounts, and social security numbers.
* **Code Quality**: It is crucial for the code to be free of syntactical and logical errors to prevent vulnerabilities in the application. Additionally, we need to be aware of any vulnerabilities present in the dependencies and libraries used in our code base. To address the first issue, I thoroughly reviewed the refactored code to ensure that my changes did not introduce any new vulnerabilities. Furthermore, I executed the code multiple times to confirm its error-free execution. Although a few warnings were raised, I resolved them by updating the JRE version specified in the build path and removing an unused library import. I also incorporated an OWASP dependency check report into the code base and have included a screenshot of the report. This dependency check compares our code base against a database of known vulnerabilities and exploits, providing a detailed report highlighting any outdated or potentially vulnerable dependencies. We can utilize this list to systematically update and secure our code base, further improving the quality of our code.

## Industry Standard Best Practices

As mentioned above in the summary section, I implemented several industry best practices to secure our code. First, I employed SHA-256 to hash our data. SHA-256 is currently the most widely implemented cipher used for this purpose and remains unbroken. This is why many companies adopt this cipher as the industry standard for data security. Additionally, I utilized the OWASP dependency check, which cross-references our dependencies with the vulnerability list provided by OWASP Foundation. OWASP is an open-source service that is constantly updated and widely used in the industry to ensure the security of code. Concerning the code itself, I diligently commented the refactored code and adhered to standard naming conventions to improve readability. This allows for easy identification of changes in the event of a security breach. By ensuring that my code can be properly read and understood, any associated security risks can be promptly addressed without causing confusion.

SEE ATTACHMENT BELOW TO REVIEW CODE FILES

Sources:

1. OWASP. (n.d.). Dependency Check. Retrieved from https://owasp.org/www-project-dependency-check/
2. 2.Mell, P., & Grance, T. (2011). The NIST definition of cloud computing. National Institute of Standards and Technology, 53(6), 50. Retrieved from https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
3. Chen, P., Zhang, S., & Zhang, H. (2020). Best Practices for Secure Software Development: A Systematic Literature Review. Information and Software Technology, 122, Article 106282. https://doi.org/10.1016/j.infsof.2020.106282