

# 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** | **2/19/2025** | **Cooper David** |  |

## 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

Cooper David

## Algorithm Cipher

I will be using SHA-256 for the algorithm cipher for this project.

SHA-256 is a cryptographic hash function that generates a 256-bit hash value (32 bytes) from any input. It's commonly used to ensure data integrity and verify authenticity, but it doesn't encrypt or decrypt data. Being a one-way function, it’s nearly impossible to reverse the hash to retrieve the original input.  
  
- Hash Functions: SHA-256 produces a 256-bit hash, offering robust security and resistance to collisions.  
- Bit Levels: The 256-bit output makes it tough against brute-force attacks and helps maintain data integrity.  
  
While SHA-256 doesn’t use keys on its own, it can be paired with them in protocols like HMAC-SHA-256 (using symmetric keys) or digital signatures (with asymmetric keys) to boost security.  
  
SHA-256 is part of the SHA-2 family and was launched in 2001. It remains widely utilized in various security protocols, such as SSL/TLS and cryptocurrencies.

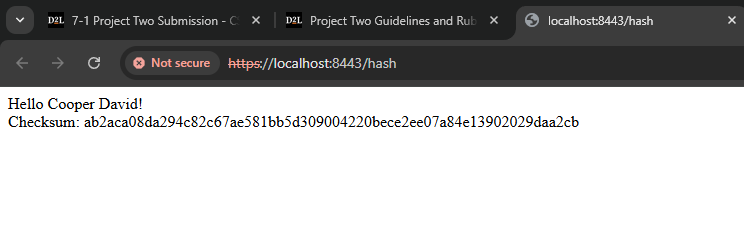
## Certificate Generation

Insert a screenshot below of the CER file.



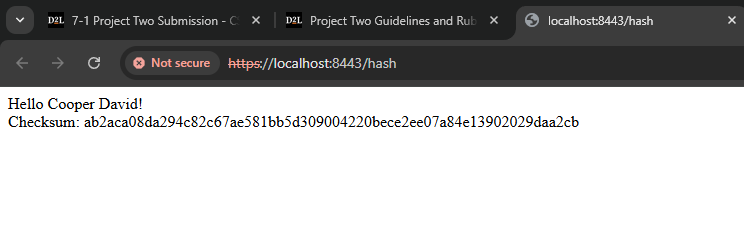
## Deploy Cipher

Insert a screenshot below of the checksum verification.



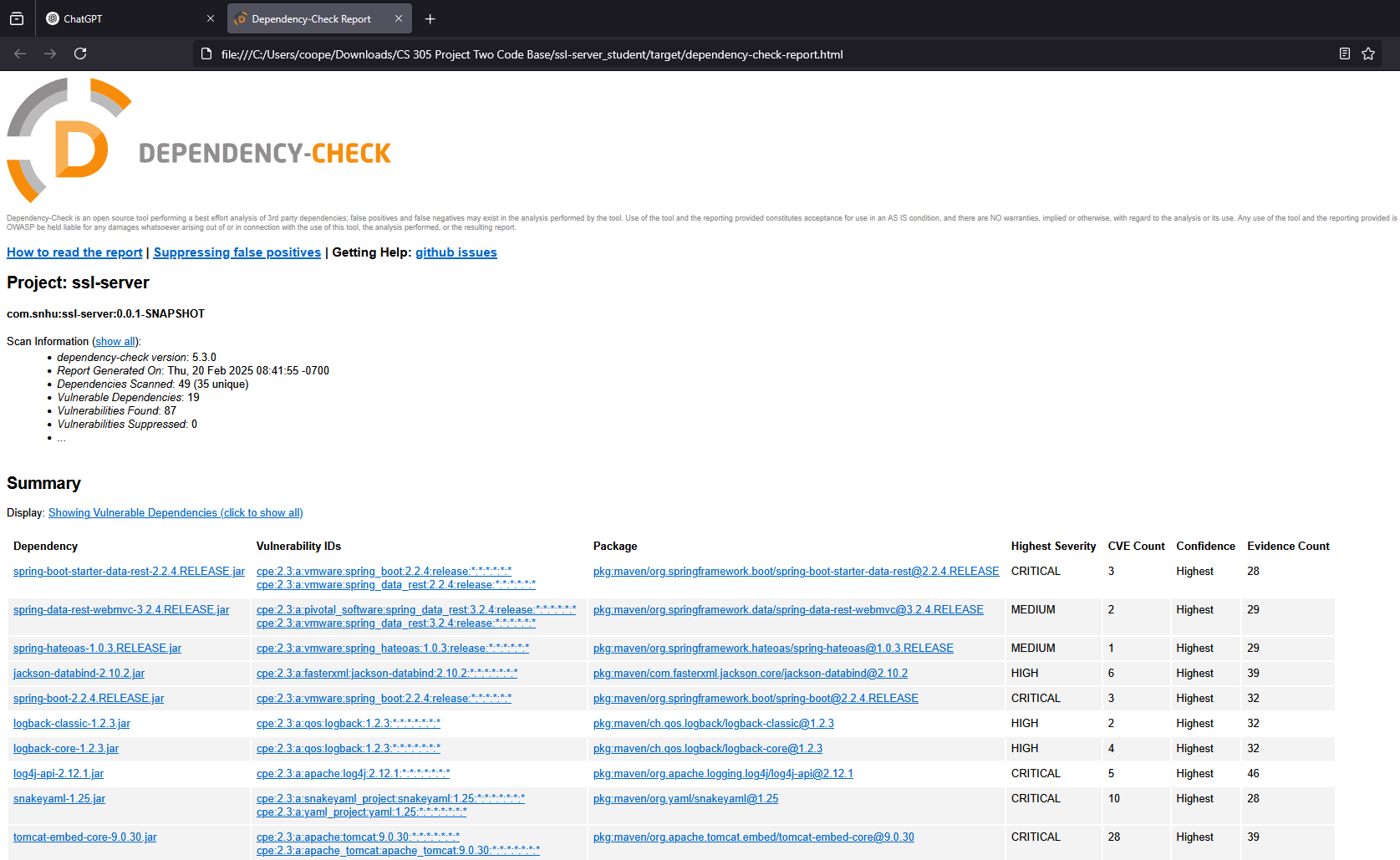
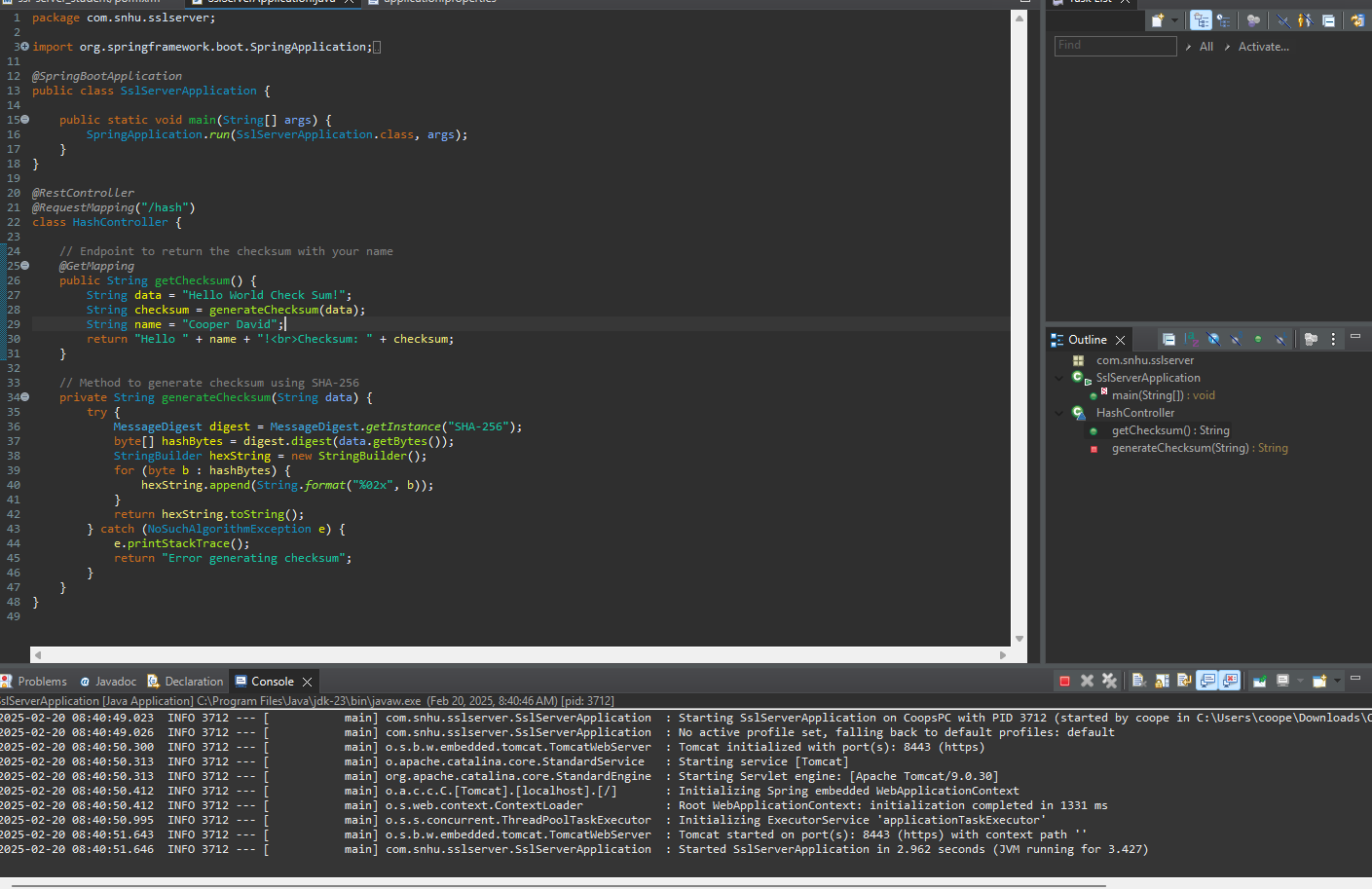
## Secure Communications

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



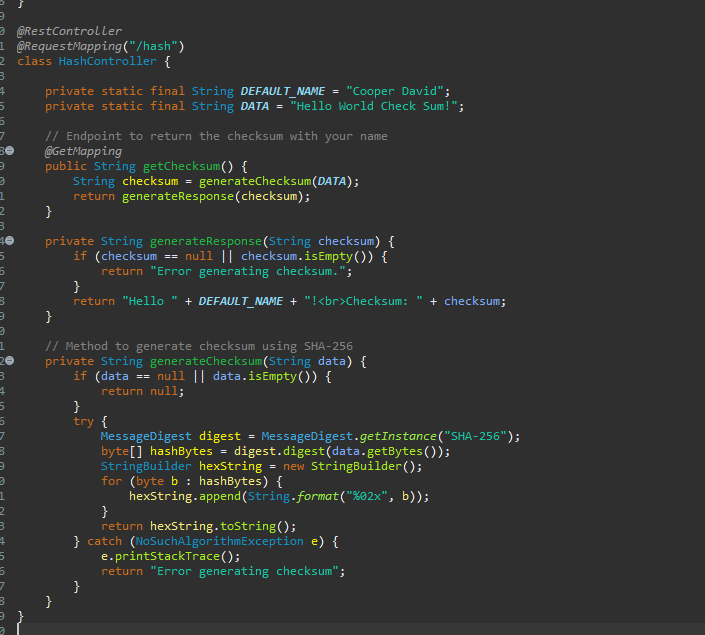
## Secondary Testing

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 updated code tackles security issues by improving input validation, error management, and cryptographic methods. The main updates include:  
  
- Input Validation: Implemented checks to confirm that data isn't null or empty before creating the checksum, which helps avoid problems with invalid inputs.  
- Error Management: Established smooth error handling during checksum creation to prevent revealing sensitive error information.  
- Secure Hashing: Continued use of SHA-256 for checksum generation, ensuring robust cryptographic security.  
- HTTPS/SSL Setup: It’s important to enable HTTPS for secure data transmission.  
  
Security Approach:  
  
- Input Validation: Checking input before processing helps stop attacks like injection or buffer overflow.  
- Error Management: Protects sensitive information from being disclosed in error messages.  
- Cryptographic Security: Utilizing SHA-256 for secure checksum generation guarantees data integrity.  
- HTTPS: Securing communications with HTTPS is vital for safeguarding data in transit.  
- Maintainability: Refactoring for dynamic data management enhances scalability and minimizes hardcoded values.

## Industry Standard Best Practices

- Input Validation and Sanitization: This step is crucial for safe data handling, as it checks inputs to avoid security problems.  
 - Secure Hashing: Using SHA-256 is the go-to method for secure hashing in the industry.  
 - Error Handling & Logging: Good error management keeps your internal processes safe and prevents data leaks.  
 - Secure Communication: Setting up SSL/TLS (HTTPS) is key to protecting data during transmission.  
 - Configuration Management: For better security, keep sensitive information in configuration files or environment variables.  
  
Advantages of Following Best Practices:  
  
 - Lower Risk of Breaches: Solid security measures help minimize the chances of vulnerabilities and data leaks.  
 - Boosted Trust: Clients are more likely to trust systems that adhere to secure coding practices.  
 - Compliance with Regulations: Following secure coding guidelines helps meet industry regulations.  
 - Easier Maintenance & Scalability: Well-organized and secure code is simpler to maintain and expand.