****

# 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** | **2025/08/11** | **Tanner Hunt** |  |

## 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

Tanner Hunt

## Algorithm Cipher

Artemis Financial manages sensitive user data and has the legal and ethical responsibility to protect that data. To protect this data, it is obscured through a hash function that makes it illegible to third parties while it’s in transit, protected with a checksum to verify the data wasn’t modified, and signed by a certificate authority to verify the source of the data. The following encryption strategies are recommended for Artemis Financial:

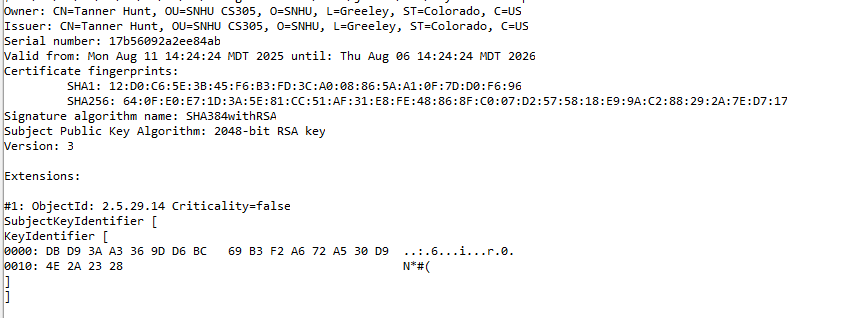
* ECDSA (eliptic curve digital signature algorithm) as the default certificate-signing operation
* DSA SHA-256-bit encryption for signing certificates that require faster turnover times than ECDSA
* AES-128 bit for data encryption

Each encryption strategy has its own use case. ECSDA is the most secure certificate signing algorithm and offers forward security, meaning data is still secure after a breach, but is slower (Manica & Detlefsen, 2014). DSA is a faster certificate signing algorithm and has support from the NIST, but is less secure than ECDSA (Nohe, 2017). AES is a standard and fast encryption algorithm recommended by the NIST (Nohe, 2017). The best cypher for the job is the one that maximizes security under the performance restraints set by the system.

A hash is a mathematical algorithm that obscures data. The data is difficult to read without the necessary key. Hashed data may use data from an organizations certificate, which verifies the source of the data, and help websites determine if data was tampered with in transit. The size of the cypher key, called it’s bit level, can vary. It is recommended keys are greater than 128-bits. Keys should also be generated using a cryptographic pseudorandom number generator for additional security. Both ECDSA and DSA are asymmetric encryption algorithms, which means an additional secret hash key is stored on the companies server to generate certificates, while users access a public key to decrypt them. That way, only Artemis financial can generate certificates. AES uses symmetric key cryptography, where both the servers and the users agree on a key to use for encryption and decryption. This makes the algorithm better for passing data between the two users. Encryption algorithms are constantly evolving to respond to newer technologies and cyber-attacks. The Data Encryption Standard (DES) was first certified by the National Institute of Standards in 1975. As computers became more powerful, the DES became inadequate and evolved into the advanced encryption standard (AES) – which enforced longer encryption keys (Schneider, 2024). This highlights the continuous need to monitor and evolve the companies security strategies.

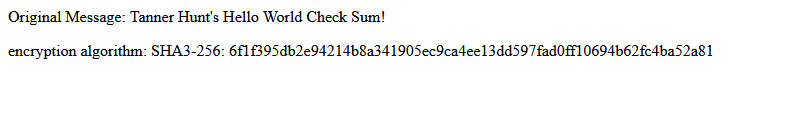
## Certificate Generation

Insert a screenshot below of the CER file.



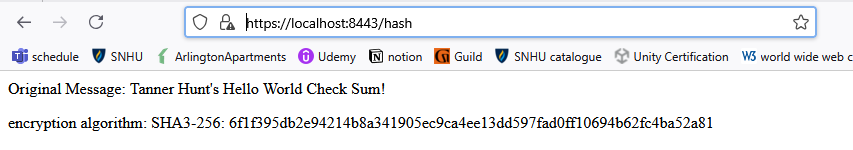
## Deploy Cipher

Insert a screenshot below of the checksum verification.



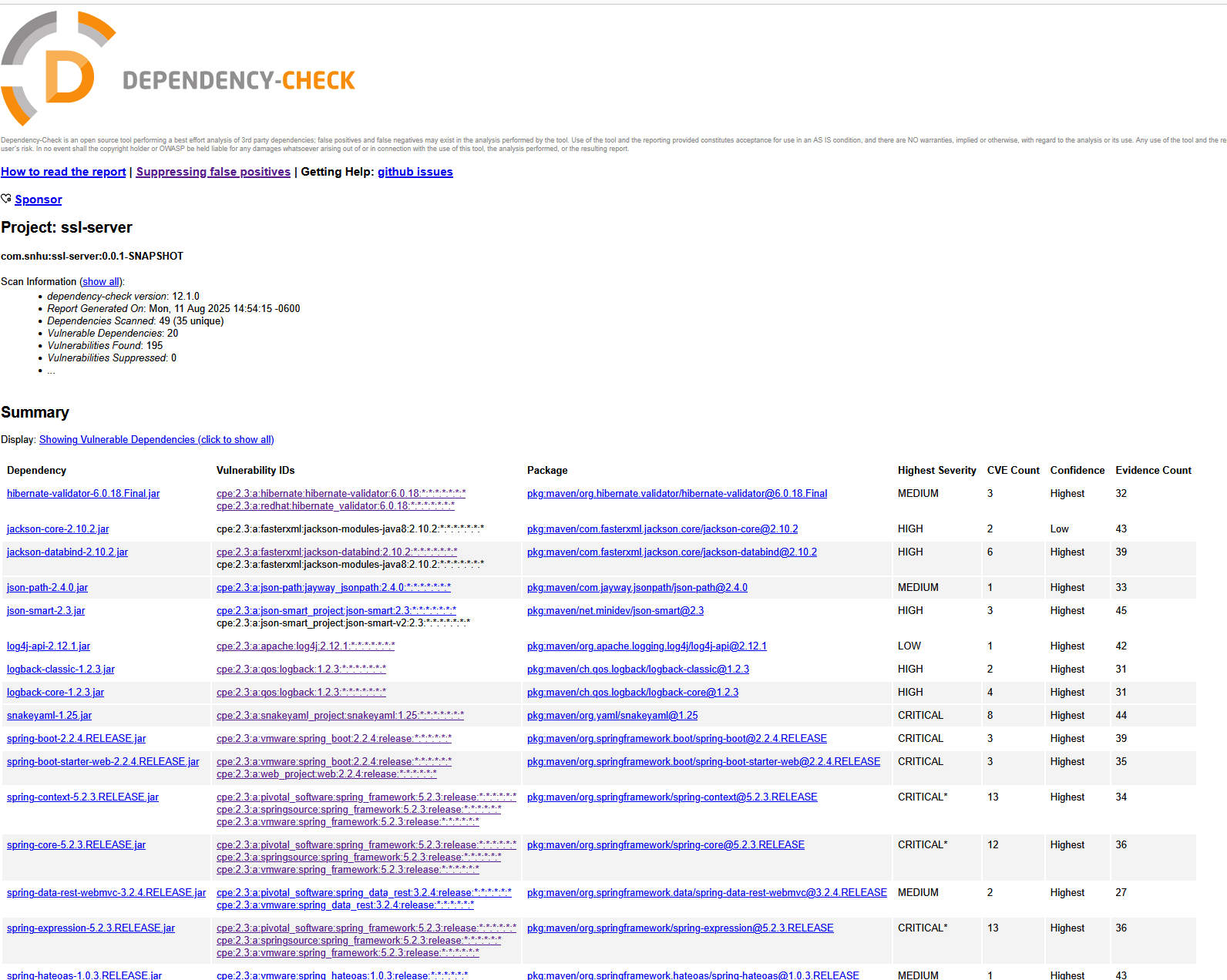
## Secure Communications

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



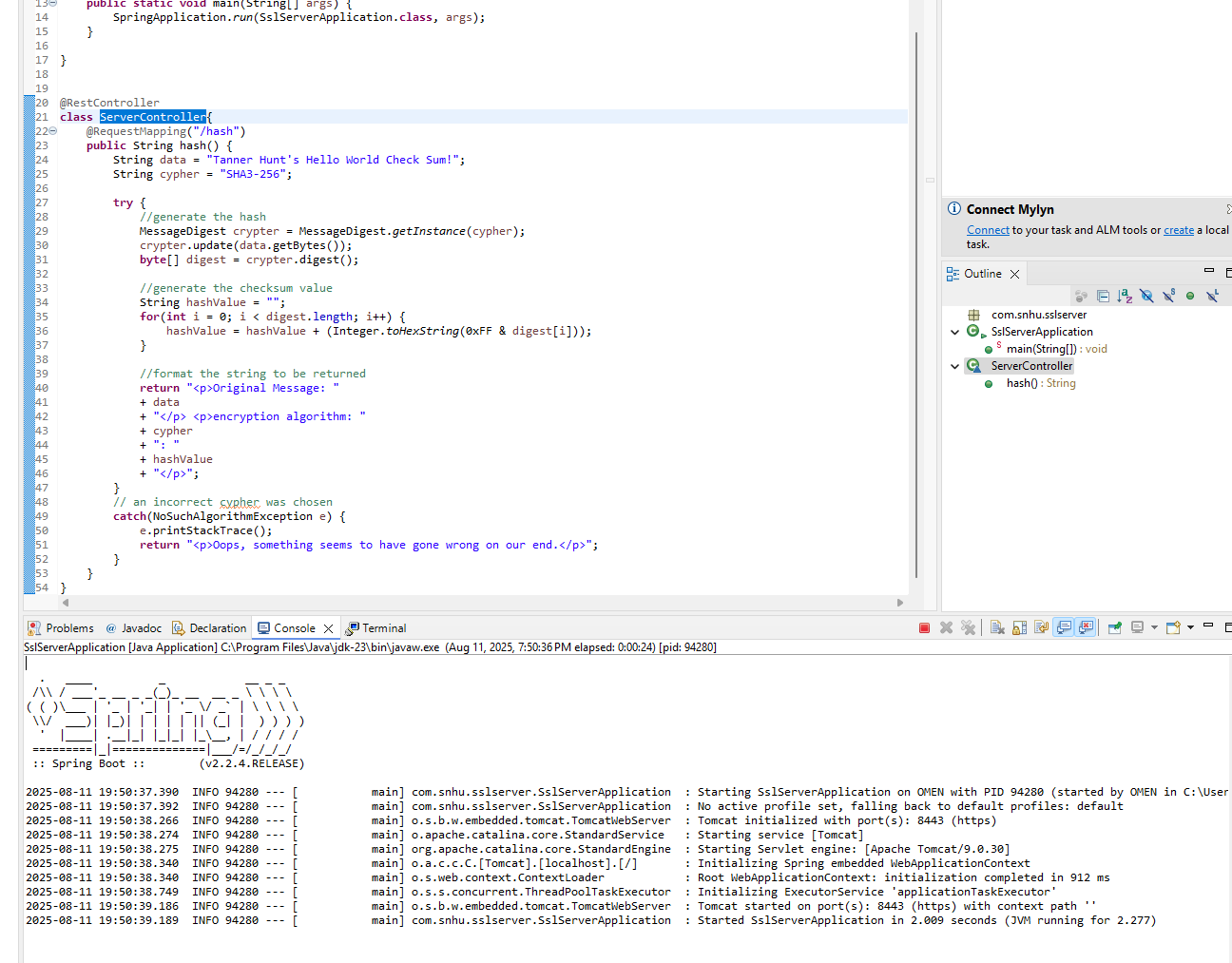
## Secondary Testing

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 code has been refactored to hash data into a checksum before sending it to users. This code complies with the following vulnerability domains:

* Cryptography: This code uses 128-bit or greater encryption to fulfill NIST standards. Data integrity is verified with a certificate and checksum
* Client/ Server: Requiring HTTPS encryption ensures data is secure before it is sent by users or the company
* Code Error: Any changes to the code that may result in an error that doesn’t encrypt the data will display an error message instead of sending unencrypted data

## Industry Standard Best Practices

I used a few industry standards to extend the functionality of this code. I checked and handled errors in the hashing algorithm I implemented. I also used existing, reliable libraries to implement the security protocols. Finally, I tested the code to make sure the API properly encrypted the data. Following industry best practices produces code that is more resilient to security defects – which is especially important as the complexity of a project grows.

**References**

Manica, J. Detlefsen, A. (2014). Iron-Clad Java. McGraw Hill Computing.

Nohe, P. (2017). What are NIST Encryption Standards? <https://www.thesslstore.com/blog/what-are-nist-encryption-standards/>

Shneider, J. (2024). A brief history of cryptography: sending secret messages throughout time. <https://www.ibm.com/think/topics/cryptography-history>