****

# 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]** | **Nicholas Wyrwas** | **N/A** |

## 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 Wyrwas

## Algorithm Cipher

I recommend SHA-256 for Artemis Financial. It is a member of the SHA-2 family, which is chosen often as a method for ensuring the integrity of a piece of data against external threats. It is a strong, non-reversible irreversible hash, which has been paramount in privacy and data security for a number of reasons. The most obvious is the physical impossibility of reverse-engineering the original data based on the hash, which reduces the ability to compromise the individual privacy of the data. It is also extremely directed, which makes it incredibly difficult to breach algorithmically, even with a very high level of computing power. SHA-256 is an irreversible hash and compression function, which means it is useful in data integrity but not for encryption like simpler hashes, including email spam filtering.

In the case of an encryption requirement, I recommend AES-256 in cipher block chaining (CBC) mode because the design goal must be to lock yourself out of your own code, assuming an attack. It is a symmetric encryption which utilizes the same key for encrypting and decrypting data, using 256-bit key length for the highest strength and combination space. The biggest benefit of this type of encryption is the heightened resistance of any brute force attack, which makes the process a little “slower”. Brute force attacks tries every possible key combinations for the input, which has considerably more combinations than, say, a 100-bit number. Symmetric hashing functions tend to be the fastest approach to encryption (but not as fast as plaintext data). Another encryption consideration is using asymmetric encryption (RSA), but the need for this encryption is high. The most common use of asymmetric encryption is for public key encryption, which is used to securely communicate in a public forum. It utilizes a pair of keys for high security, with one user choosing a small parameter which he shares via a public forum, and then computes an asymmetric key (each user has their own unique key). Each user then uses each other’s unique key to communicate back to each other. Asymmetric encryption tends to be slower than its symmetric counterparts, but it is considerably more secure.

## Certificate Generation

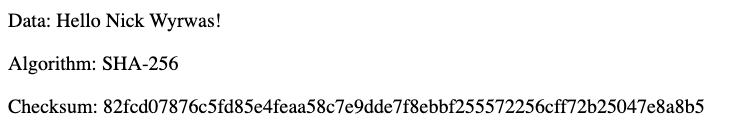
Insert a screenshot below of the CER file.

A screenshot of a computer

Description automatically generated

## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

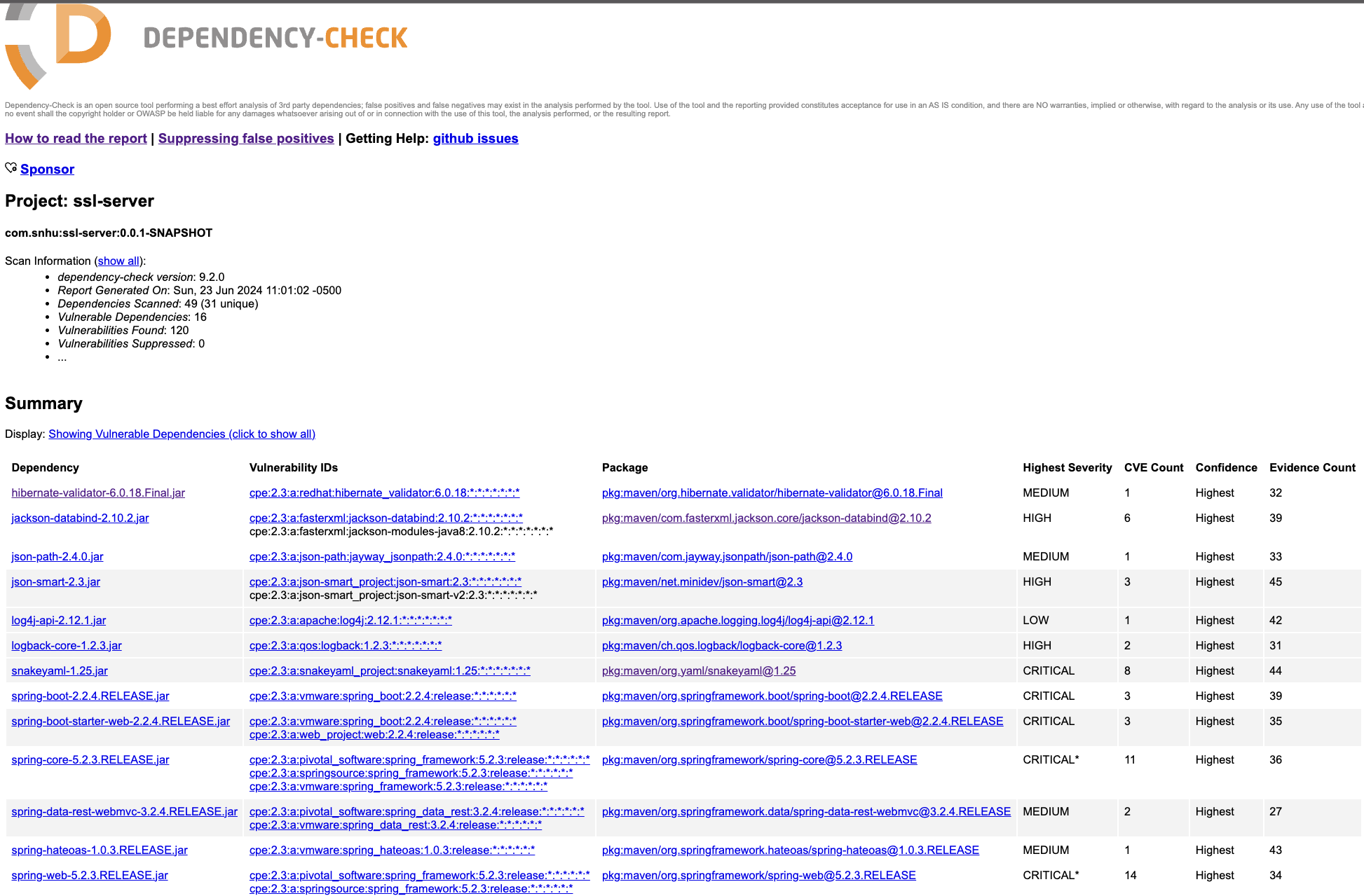
Insert a screenshot below of the web browser that shows a secure webpage.

A screenshot of a computer

Description automatically generated

## Secondary Testing

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

A screenshot of a computer

Description automatically generated

## Functional Testing

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

A screen shot of a computer

Description automatically generated

## Summary

The code was refactored, as well as passed security testing that requires self-signed certificates to set up an HTTPS application. I've also updated pom.xml to create the checksum for issues that needed fixing during the refactoring process. The verification of these secure certificates is essential to make sure that the website created is fully functional and secured. That way the user who visits the page can trust that the information that this application provides is authentic. An important step is securing the hash function. When a hash function is configured to be checked into a checksum, the data are safe and secured. The process of scrambling customer data during the database access makes it harder to get back or harder to obtain and therefore harder to attack. Finally, the hash function was configured to be processed with SHA-256, which maximizes the strength in the security measures.

## Industry Standard Best Practices

I used industry standard practices by reviewing every bit of code within the application to ensure that everything is correct and up to date. This would allow me to correct any issues or mistakes made. As well as, by following correct standard industry practices, it would allow the company to maintain a level of security for its customers, add increased security against future attacks, and finally maintain integrity. With their motto “Security is everyone’s responsibility”.