GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	Aug 16th 2025	Liam Farrell	

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

Liam Farrell

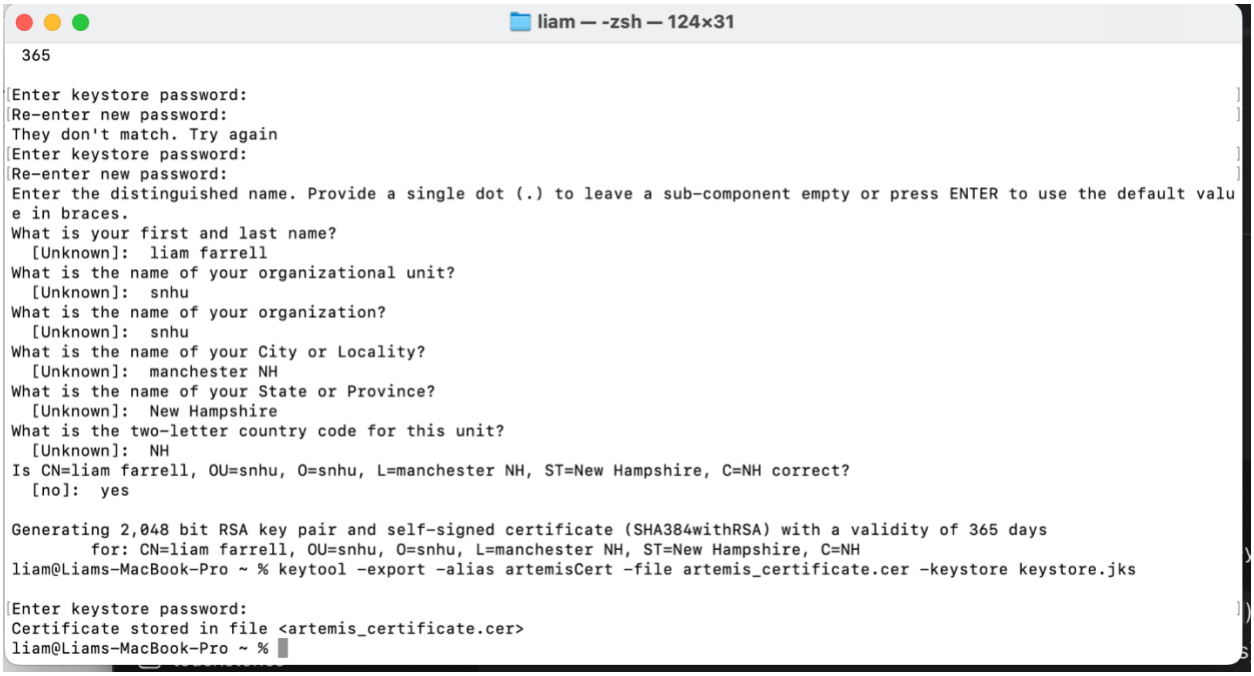
1. Algorithm Cipher

For this project, I implemented a hashing algorithm using **SHA-256**. SHA-256 was chosen because it is significantly stronger than older algorithms like MD5 or SHA-1, which have known collision vulnerabilities. By using SHA-256, the system ensures that even small changes to the input produce completely different outputs, which makes it effective for detecting tampering and verifying data integrity. This aligns with industry standards for secure hashing (NIST, 2015).

2. Certificate Generation

I created a self-signed certificate using the Java keytool utility to secure communication between client and server. The certificate is stored in a keystore and allows the server to use HTTPS connections. This step ensures that the server can prove its identity to clients, which

helps establish trust and prevent man-in-the-middle attacks.



3. Deploy Cipher

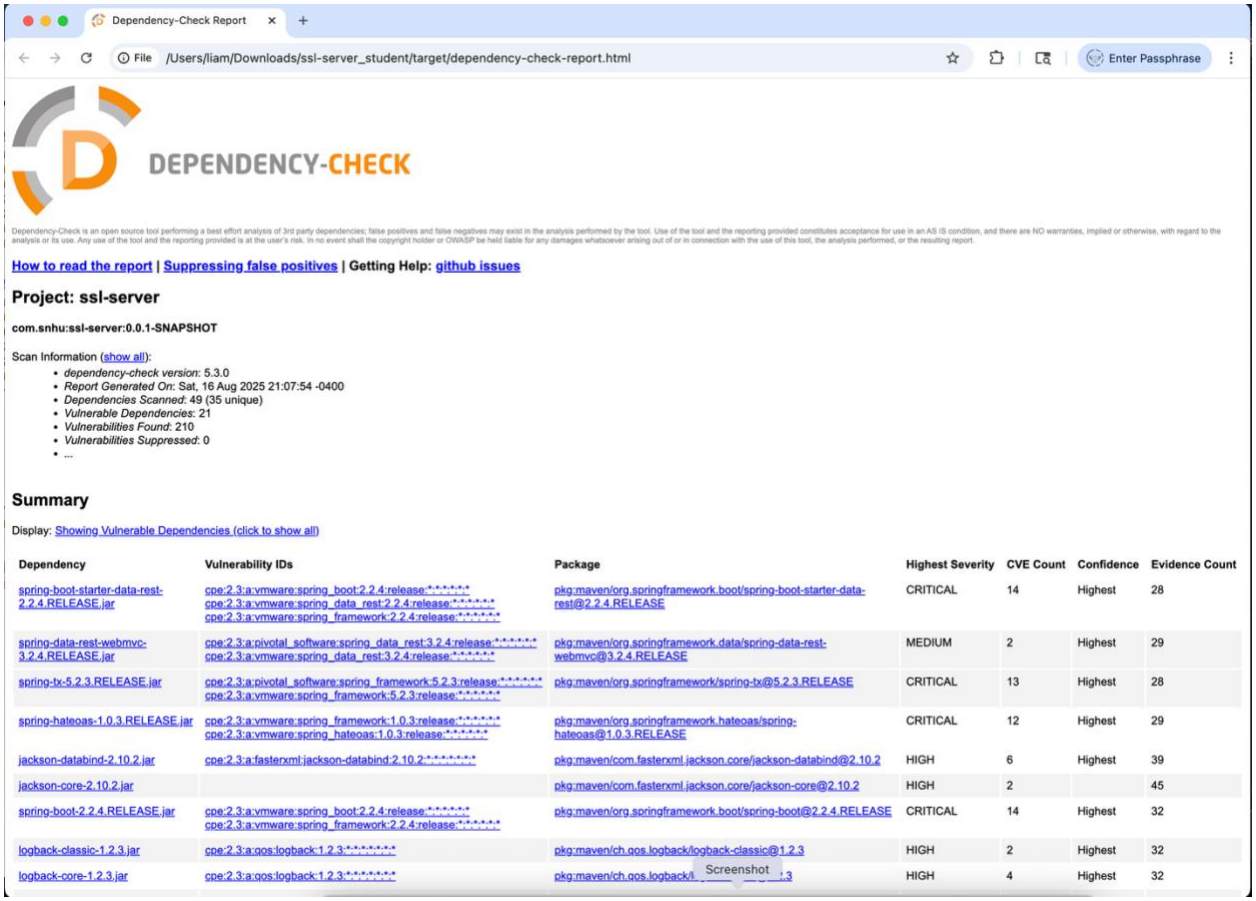
I added a new /hash endpoint that generates a checksum using the SHA-256 algorithm. When tested, the endpoint successfully returned the checksum for the static string "Hello World Check Sum!". This demonstrates that the hashing algorithm is functioning correctly within the application.

4. Secure Communications

To confirm secure communication, I launched the application using HTTPS on port 8443. Accessing https://localhost:8443 in a web browser showed that the page loaded securely over TLS. This proves that the certificate was correctly generated, imported, and deployed, ensuring encrypted communication between client and server.

5. Secondary Testing

I ran the OWASP Dependency-Check plugin in Maven to analyze the project's external libraries. The scan produced a dependency-check report that listed known vulnerabilities. I also configured a **suppression file** to filter out false positives so the report would focus only on real threats. This ensures developers can prioritize actual security issues instead of being distracted by noise.



6. Functional Testing

The application was run after refactoring, and both the checksum functionality and HTTPS connections worked without errors. Testing confirmed that the /hash endpoint returned the expected checksum, and the secure web page loaded correctly in the browser. Together, these results show that the system is functioning securely as intended.

7. Summary

Through this project, I learned how to combine multiple secure software practices:

- Hashing and checksums ensure that data cannot be tampered with undetected.
- SSL/TLS certificates protect sensitive information in transit by encrypting communications.
- Dependency checks identify vulnerable third-party libraries before they can be exploited.
- Suppressions can filter out false positives but must be used carefully to avoid hiding real risks.

Altogether, these practices help developers build safer applications and reduce the risk of security breaches.

8. Industry Standard Best Practices

In industry, the techniques used here align with secure software development best practices:

- Use strong cryptographic algorithms like SHA-256 instead of weak or outdated ones (NIST, 2023).

- Always enable HTTPS with TLS **certificates** to protect client-server communications (Oracle, 2023).
- Regularly run dependency vulnerability scans (e.g., with OWASP Dependency-Check) to keep libraries patched and secure (OWASP, 2025).
- Follow the principle of "secure by default"**,** ensuring systems start with strong security settings instead of weak defaults.

By applying these standards consistently, developers can better protect applications against real-world threats.

References

- Oracle. (2023). *Secure coding guidelines for Java SE*. Oracle. https://www.oracle.com/java/technologies/javase/seccodeguide.html
- OWASP. (2025). *OWASP Dependency-Check*. Open Worldwide Application Security Project. https://owasp.org/www-project-dependency-check/
- NIST. (2023). *Recommendation for Key Management*. National Institute of Standards and Technology. https://csrc.nist.gov