

# 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]** | **[Your Name]** |  |

## 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

Mehdi Salhi

## Algorithm Cipher

#### **A. Overview and Choice of Algorithm**

For Artemis Financials' secure web application, I chose RSA-2048 for encryption. RSA is an asymmetric algorithm that works with two keys, a public key that can be shared with anyone, and a private key that must stay secret. The 2048-bit key length is currently the industry standard for strong security. RSA is great for protecting data in transit, especially during the HTTPS handshake, because it allows the client and server to safely exchange keys and verify each other’s identity.

#### **B. Hashing and Bit Levels**

To make sure data hasn’t been changed or tampered with, we use hash functions like SHA-256. This algorithm generates a 256-bit hash, a long, unique string that represents the original data. With 2²⁵⁶ possible combinations, it’s practically impossible for two different inputs to have the same hash value (a collision). That makes SHA-256 highly secure and resistant to brute-force attacks.

**Comparison of Common Hash Algorithms:**

* MD5 (128-bit): Broken and insecure
* SHA-1 (160-bit): Deprecated and vulnerable
* SHA-256 (256-bit): Secure and widely recommended
* SHA-512 (512-bit): Even stronger, but slower

I went with 256-bit because it’s the perfect balance between performance and security, and it’s what NIST currently recommends.

For RSA, a 2048-bit key is the baseline for strong encryption. Anything shorter, like 1024-bit, is outdated, and while 4096-bit is more secure, it’s also slower and less practical for most cases.

#### **C. Random Numbers and Key Types**

Randomness plays a huge role in cryptography. It’s used to generate keys, salts, and initialization vectors (IVs), all of which ensure that even if you encrypt the same message twice, you’ll get two completely different results. In this project, Java Keytool uses a secure random number generator to create the RSA key pair.

Here’s a quick breakdown of encryption types:

* Symmetric encryption (like AES) uses the same key for encryption and decryption. It’s very fast and efficient for large amounts of data but requires a secure way to share the key.
* Asymmetric encryption (like RSA) uses a public/private key pair, solving the key-sharing problem but running slower, which makes it less ideal for encrypting large files directly.

Modern systems combine both methods in what’s called a hybrid approach, RSA is used for exchanging a temporary session key, and AES encrypts the actual data. This is exactly how HTTPS works today.

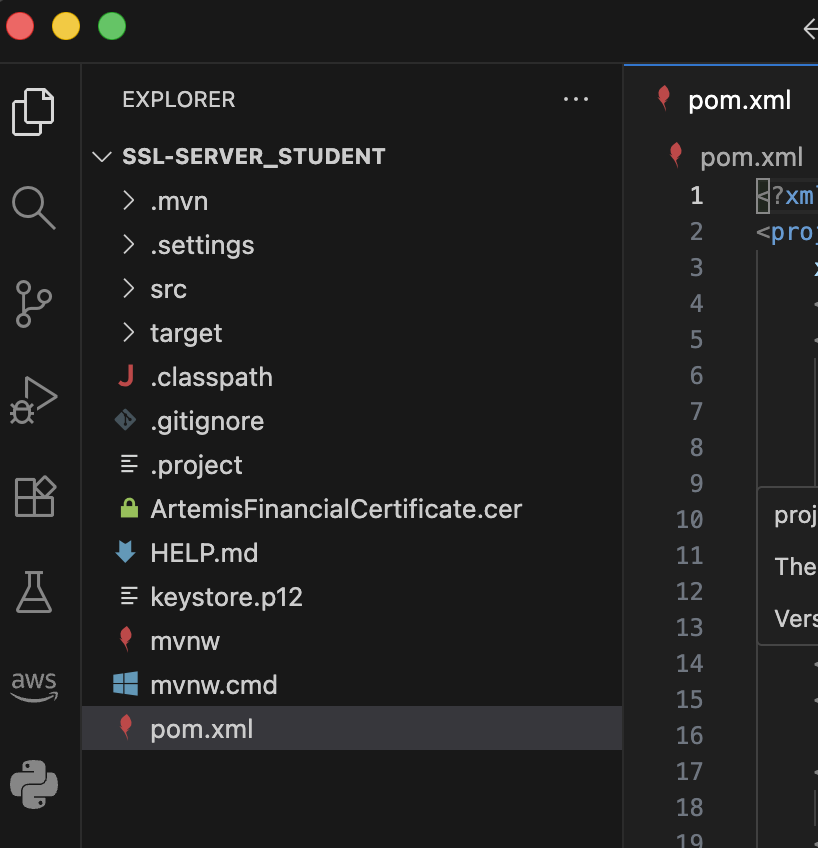
#### **D. History and Evolution of Encryption Algorithms**

Encryption has come a long way.

* Ancient times: Simple ciphers like Caesar and Vigenère relied on substitution and secrecy, which are easy to break now.
* Mechanical era: Machines like the Enigma were used during WWII but were eventually cracked.
* Early computer era: DES (56-bit) became the first standard but was later proven weak. The invention of RSA in 1977 introduced public-key cryptography and changed everything.
* Modern era: AES (introduced in 2001) became the global symmetric encryption standard, offering 128-, 192-, and 256-bit options. SHA-2 (including SHA-256) became the standard for hashing. RSA key sizes increased to 2048 bits and higher.
* Today: Algorithms like SHA-3 and Elliptic Curve Cryptography (ECC) offer even stronger security and better performance. Researchers are now focusing on quantum-resistant algorithms to prepare for future threats.

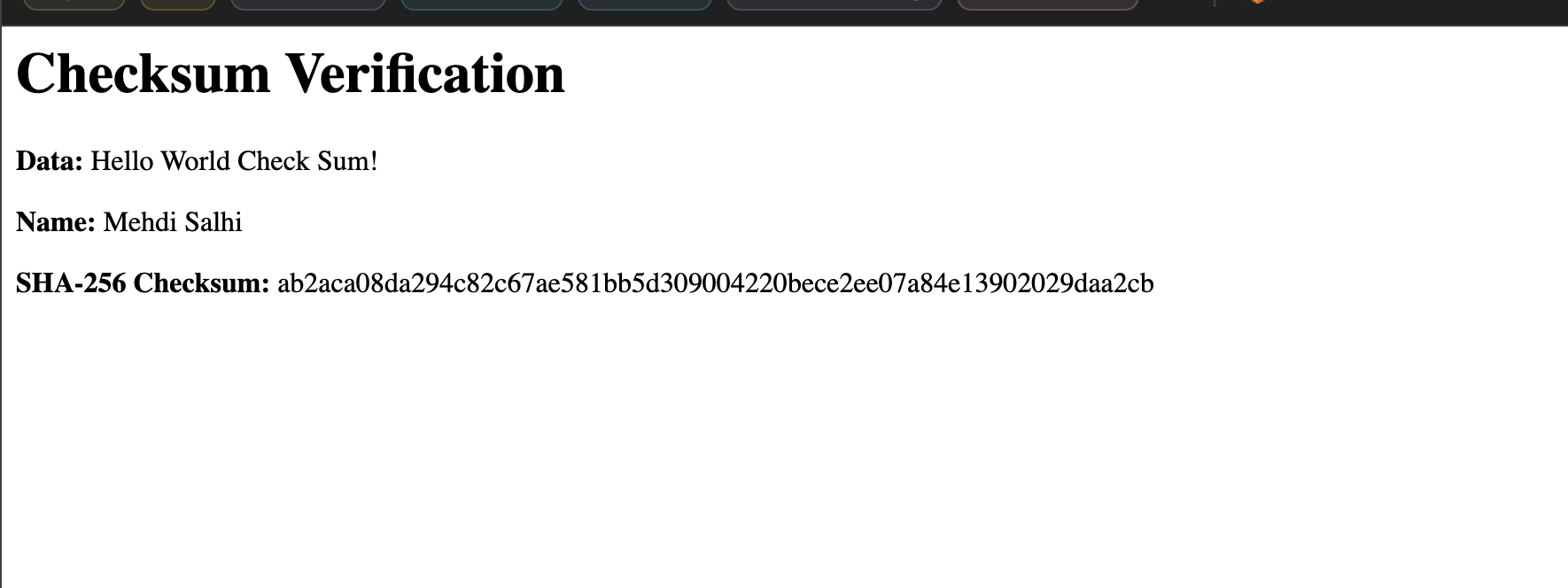
## Certificate Generation

Insert a screenshot below of the CER file.



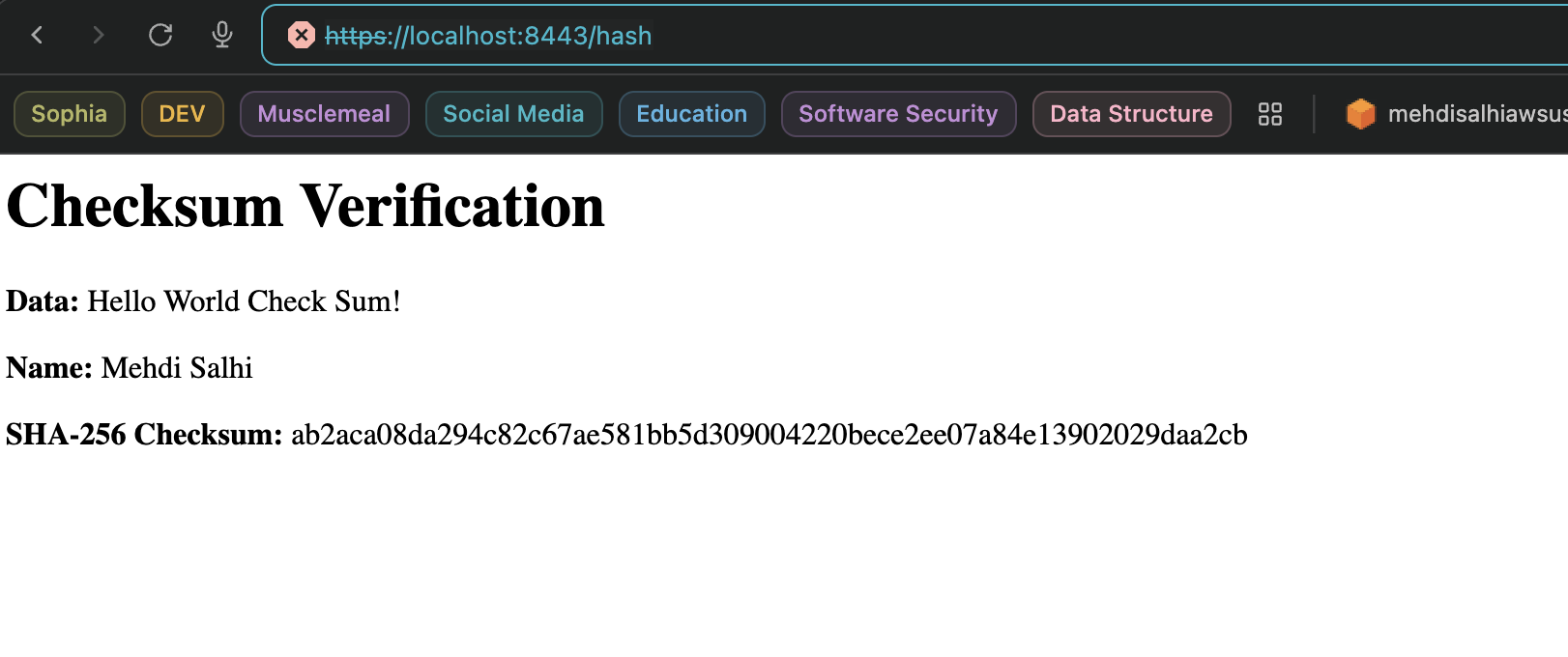
## Deploy Cipher

Insert a screenshot below of the checksum verification.



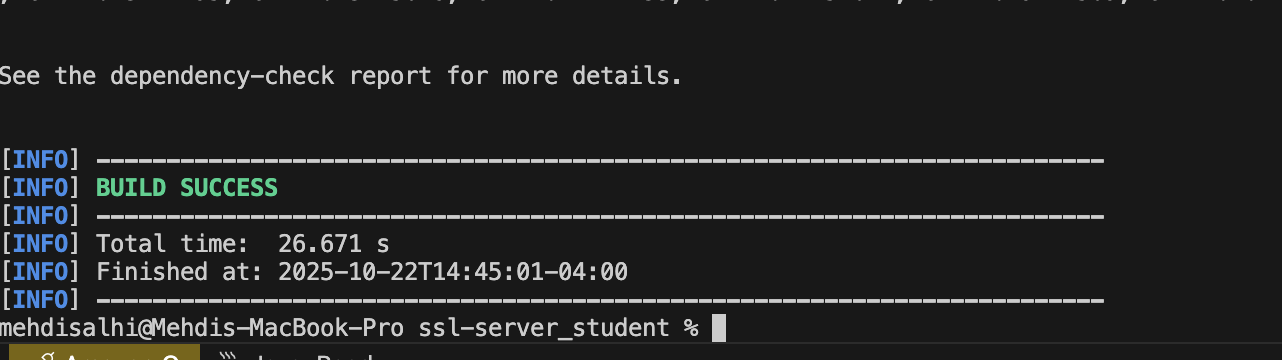
## Secure Communications

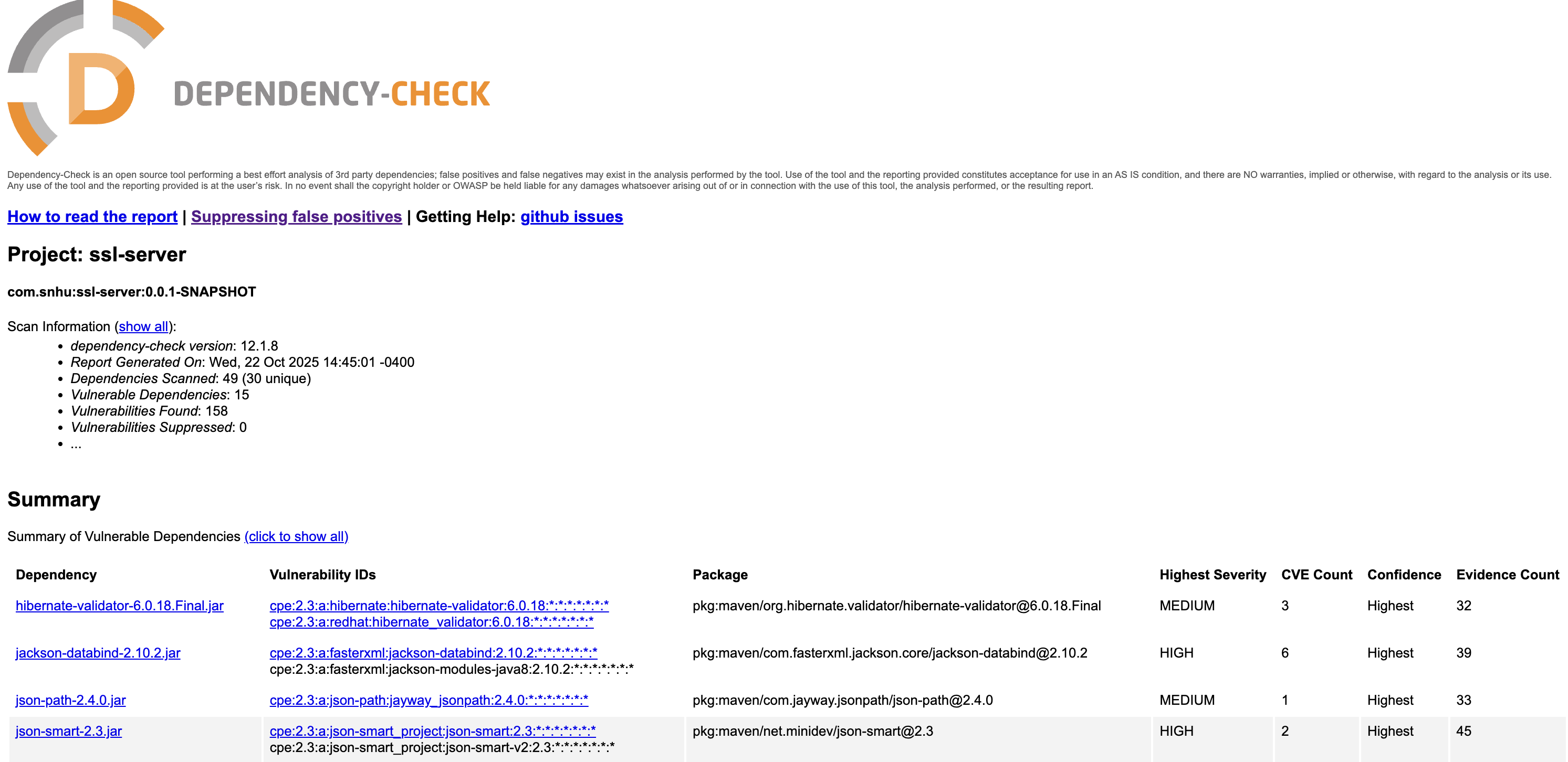
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

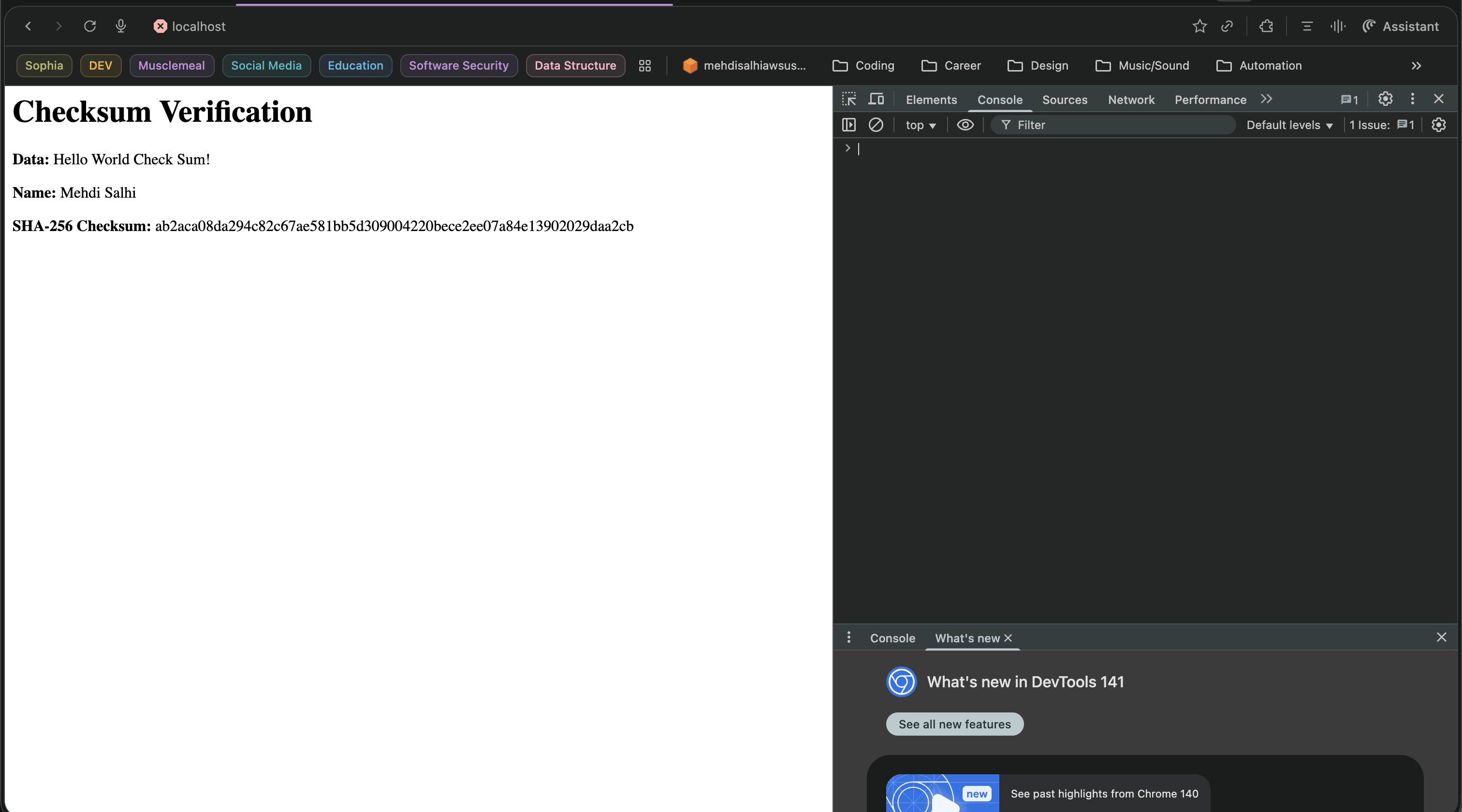
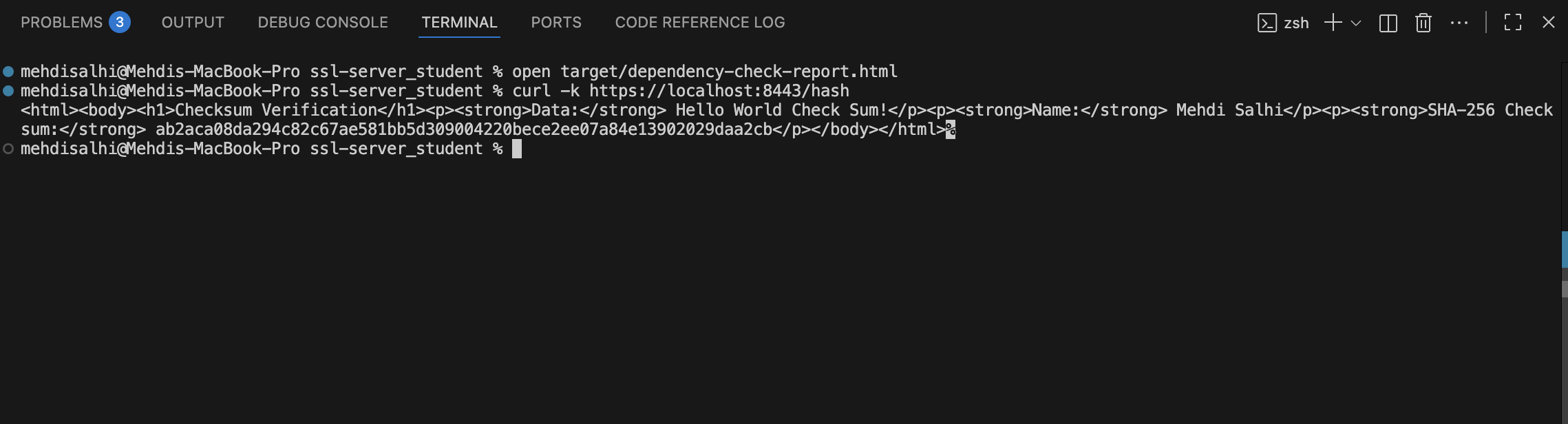
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



## Summary

In this project, I used RSA-2048 for secure key exchange and SHA-256 for data integrity, both in line with modern best practices. This combination shows how cryptography has evolved from simple substitution ciphers to powerful mathematical systems that protect nearly every aspect of today’s digital communication.

## Industry Standard Best Practices

A. Secure Coding Standards Followed

For this project, I used several secure coding standards to help protect the application:

* Input Validation: I made sure to check and validate any data that comes from users before it’s processed, which helps prevent attacks like SQL injection or unexpected behavior.
* Avoiding Hard-Coded Keys: I kept sensitive values like keys and passwords out of the source code, using configuration files or secure storage instead. This way, keys aren’t accidentally exposed.
* Strong Encryption: I chose modern encryption algorithms like RSA-2048 for key exchange and SHA-256 for hashing, which are recognized industry standards and considered secure.

B. Why Secure Coding Is Important

Writing secure code is important to protect both the company and its clients:

* Building Trust: When software is secure, users can trust the company with their personal data and business information.
* Compliance: Many laws and industry guidelines require companies to use encryption and other security practices, so following these helps us stay compliant and avoid penalties.
* Avoiding Breaches: Secure coding reduces the risk of hackers breaking in, stealing data, or causing damage. This protects the company’s reputation and prevents financial losses for both the business and its customers.