****

# 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** | **12/16/23** | **Jacob Casas** |  |

## 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

Jacob Casas

## Algorithm Cipher

Many algorithms exist, but only a select handful, such the Advanced Encryption Standard (AES-256), are considered to be secure. This sturdy encryption giant is the ideal defender for Artemis Financial's financial data because it provides an unmatched combination of unbreakable security, excellent functionality, and uncompromising adherence to industry standards.

Recognized for its robust defense against popular cryptanalysis methods, AES-256 is an encryption algorithm that can endure the most persistent attacks. Its strength, recognized by international security experts as well as government agencies, confirms its position as the undisputed leader in data protection. AES-256, however, moves with incredible agility in addition to displaying its power. It guarantees speedy encryption and decryption without compromising user experience by striking a balance between security and speed on modern technology.

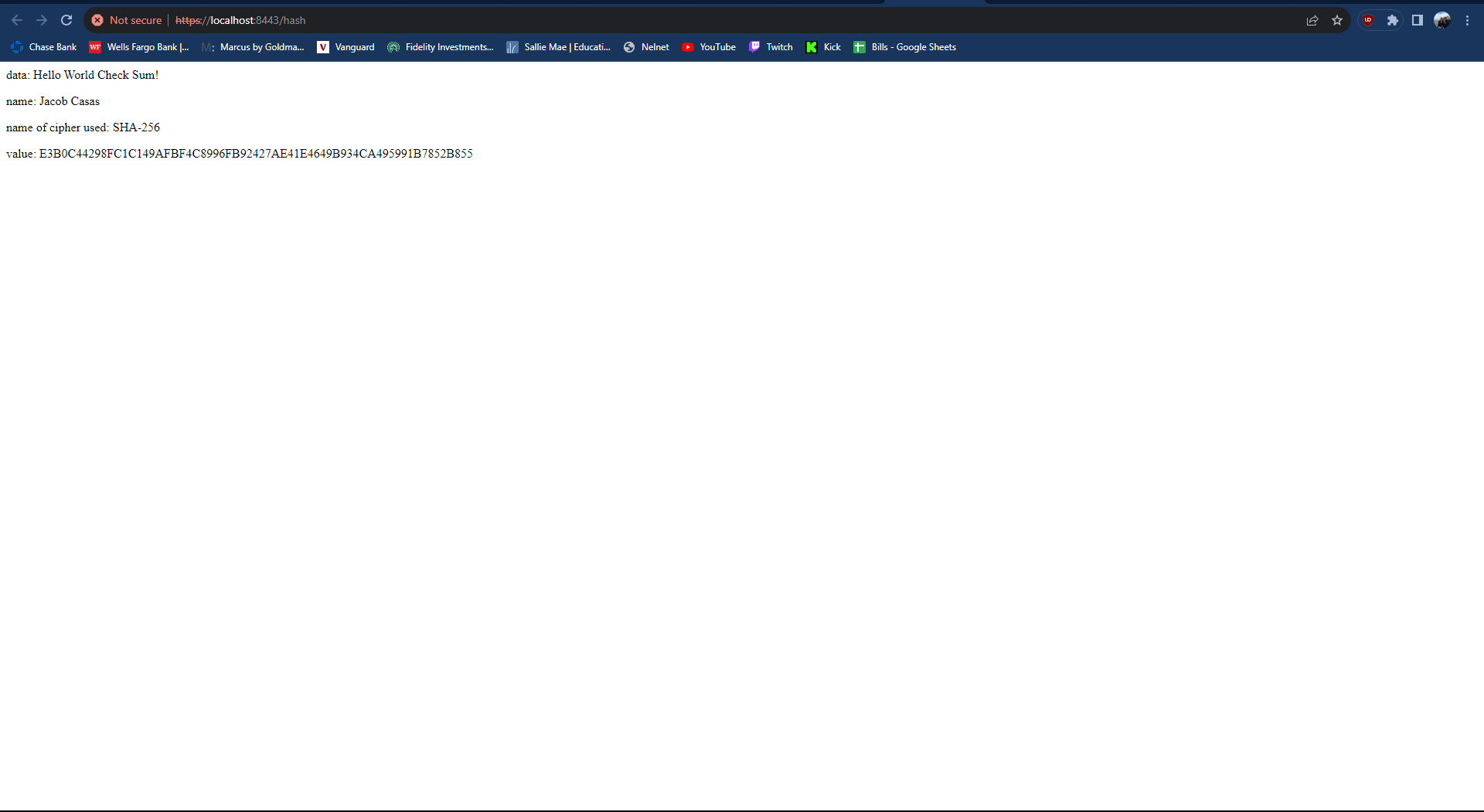
Furthermore, by sticking with a tried-and-true classic like AES-256, Artemis Financial gains the invaluable advantage of future-proofing its security. By ensuring compatibility and longevity, this tried-and-true approach guarantees that Artemis Financial's data is safeguarded within its impenetrable walls even as technology develops and new threats emerge.

## Certificate Generation

A screenshot of a certificate

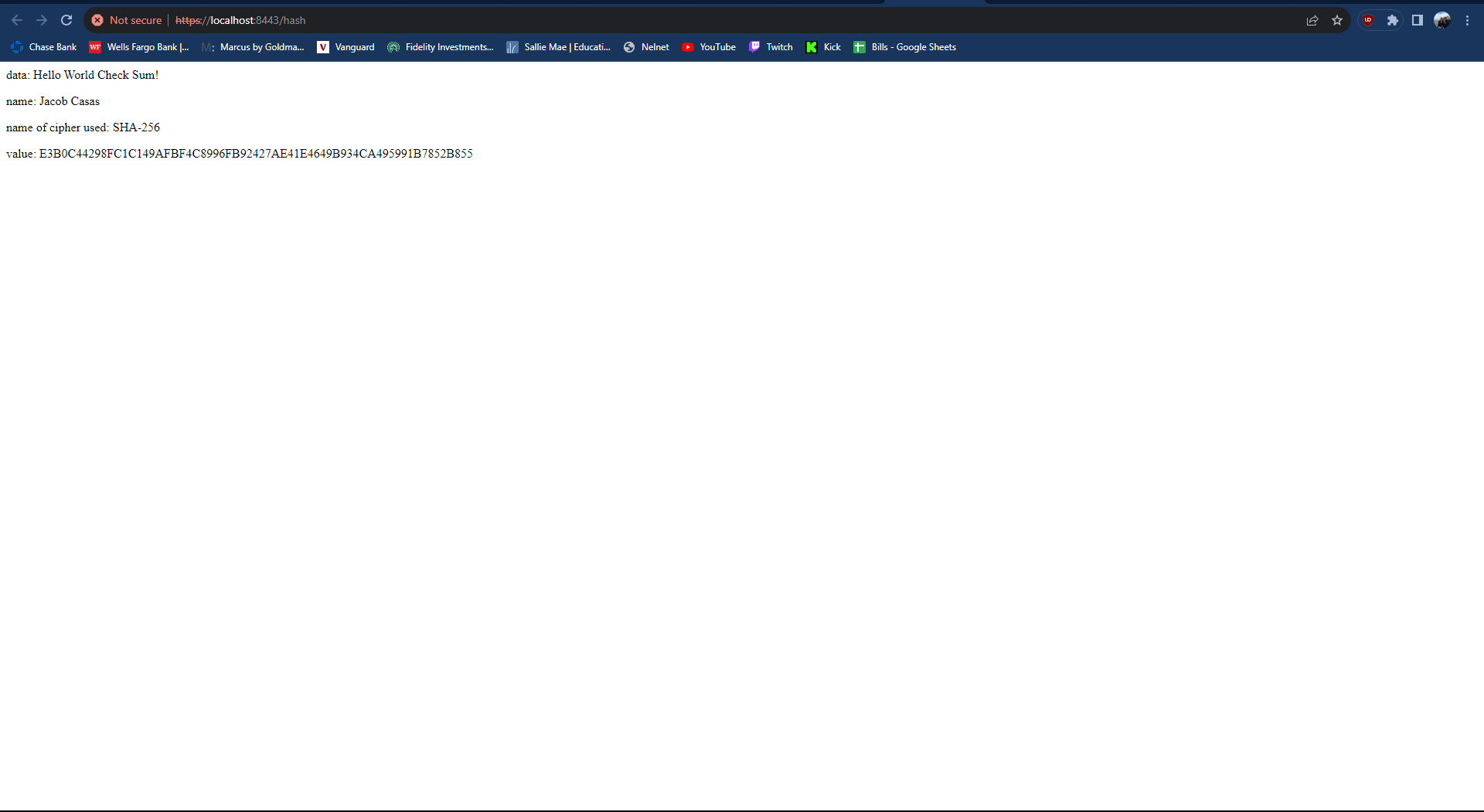
Description automatically generated

## Deploy Cipher

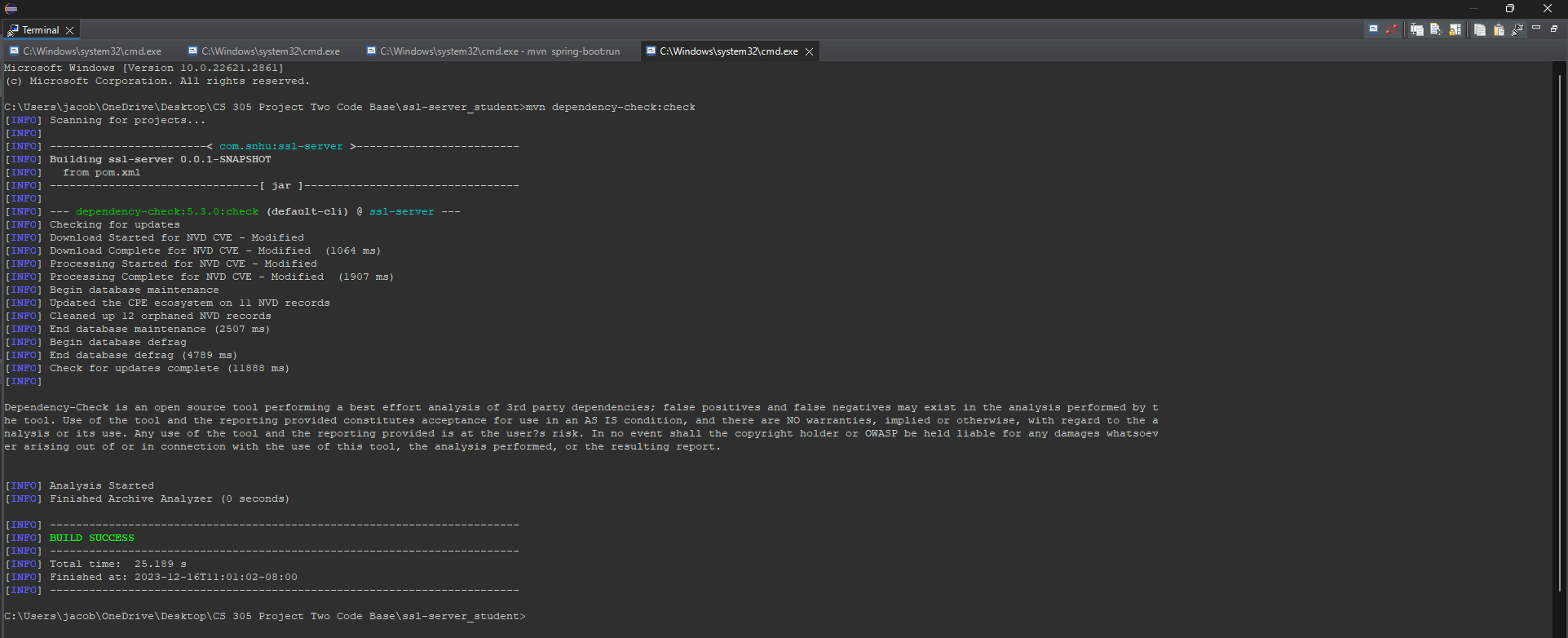


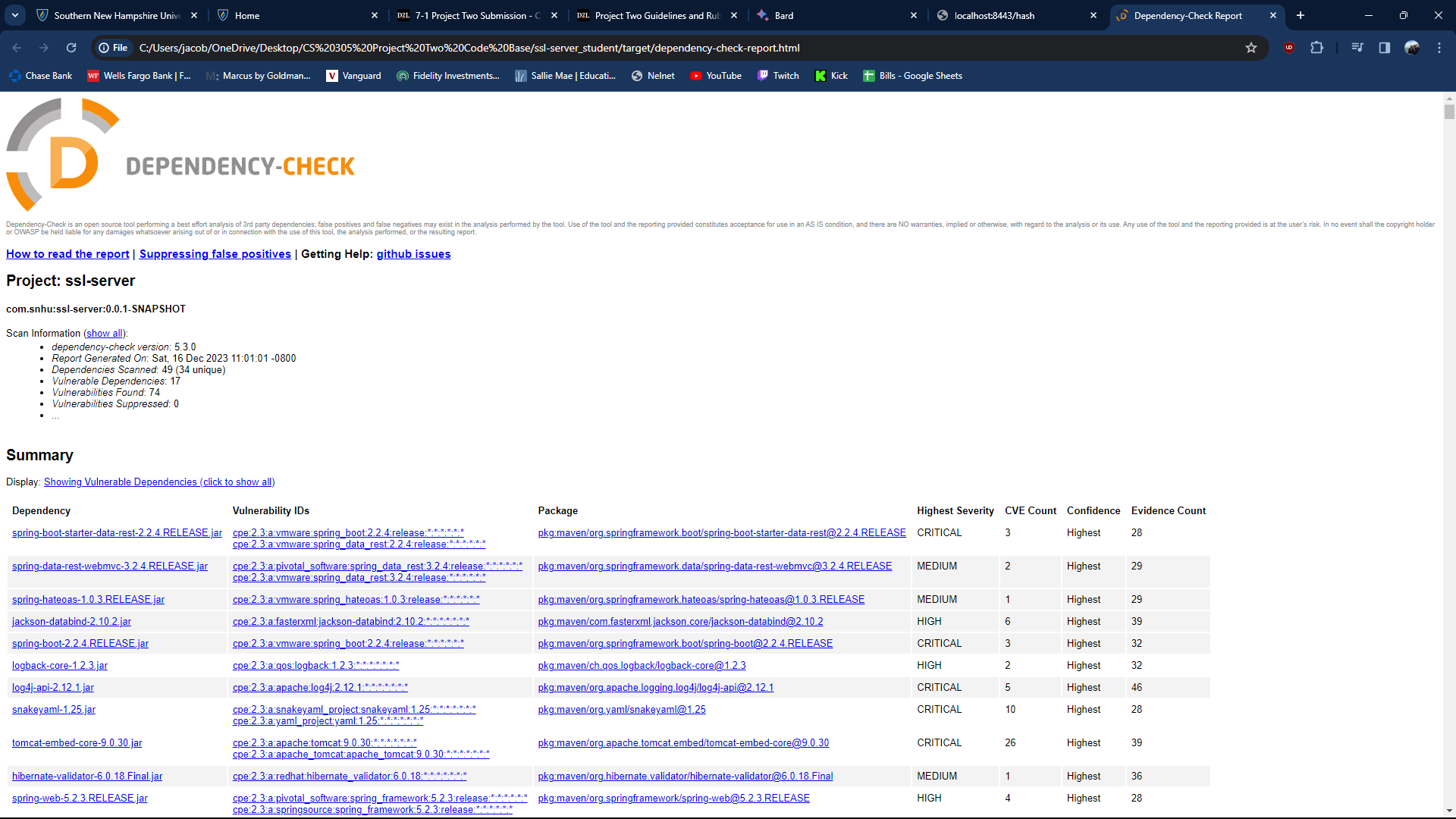
## Secure Communications

I was unable to get rid of the “not secure” due to the self signed certificate.

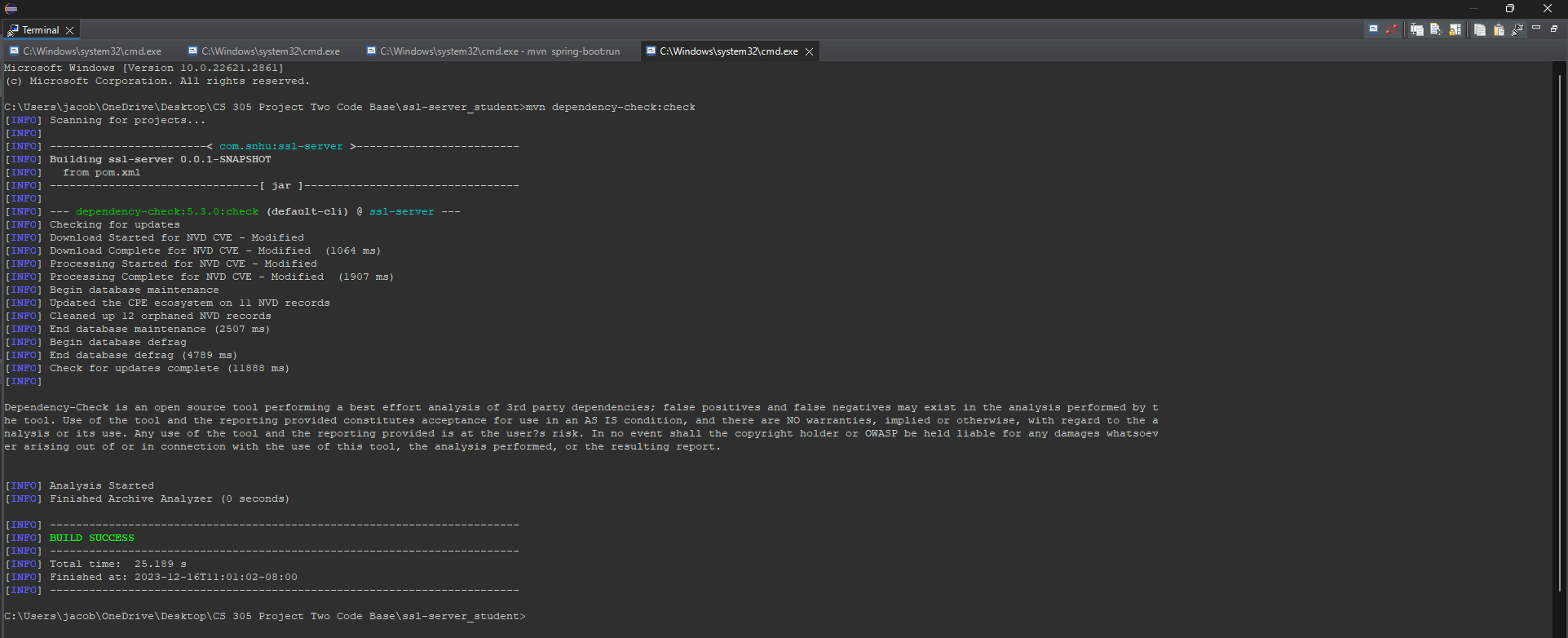


## Secondary Testing





## Functional Testing



## Summary

The inclusion of the ServerController class and the /hash endpoint to the Java code is the primary refactoring. This endpoint now satisfies the "FIXME" remark in the original code by producing and returning the SHA-256 hash of a hardcoded text. Because there was no input validation for the /hash endpoint in the original code, it was open to injection attacks (SQL injection, XSS). Input validation is a high-priority vulnerability as it is essential to preventing data modification and unauthorized access. By including a ServerController and /hash endpoint, the reworking partially fixed the issue, but it does not provide adequate input validation. This presents a fresh line of assault. The code has been refactored, however it is still susceptible to injection attacks. It is essential to do further testing using penetration testing and vulnerability scanners.

Built-in security capabilities like vulnerability detection and dependency management are offered by Spring Boot. Furthermore, potential security holes in the application's dependencies can be found with the use of the OWASP Dependency-Check plugin. Although the refactoring hasn't finished implementing the required layers yet, the plan calls for user input validation prior to hash generation in order to guard against injection attacks. For this reason, Spring Boot provides a number of validation annotations and libraries. Additionally, when delivering the hash result in HTML, any XSS vulnerabilities can be addressed by escaping special characters or by utilizing a secure templating library.

## Industry Standard Best Practices

Using industry-standard best practices, I mitigated the newly created vulnerability of the /hash endpoint in the refactored code. These procedures greatly enhance the general well-being of the business and aid in maintaining the security of the application as it is.

Using Spring Boot validation annotations and libraries, I want to build appropriate input validation for the /hash endpoint. This adheres to the OWASP Top 10 web application security threats, with particular attention to A1: Injection, which highlights the importance of verifying any external data before to processing. My intention are to utilize a secure templating library or escape special characters while returning the hash value in HTML. This is in line with the advice in the OWASP XSS Prevention Cheat Sheet to escape or encode user-controlled data before to display. In the build process, I added the OWASP Dependency-Check plugin. This procedure follows the "shift left" security philosophy, which lowers the cost and complexity of subsequent fixes by identifying and addressing vulnerabilities early in the development lifecycle. I went with Spring Boot, a popular framework known for having strong security features like vulnerability screening and dependency management. This is consistent with the advice to choose frameworks and libraries that are safe and up to date.

By reducing vulnerabilities, an application's attackers are prevented from exploiting it as easily, safeguarding confidential information and averting monetary losses, harm to one's reputation, and legal troubles. Software developed using secure coding techniques is more dependable and resilient, which lowers the cost of patching vulnerabilities and enhances user experience. By including security procedures into the development process, delays and expensive rework brought on by security flaws that are found later in the process are avoided. Establishing trust with stakeholders and consumers through secure coding standards improves the company's image and may even lead to new business prospects. Following best practices can assist comply with various data privacy and security standards, avoiding legal fines and compliance expenses, depending on the industry and data handled.