

# 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** | **October 19, 2025** | **Dat Nguyen** | **Initial creation of the secure software report.** |

## 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

Dat Nguyen

## Algorithm Cipher

For Artemis Financial, a company handling sensitive financial plans, a multi-layered cryptographic strategy is essential. I recommend implementing the **SHA-256** algorithm to generate checksums for file verification, ensuring data integrity during transfers. For securing the web application's communications, the server should be configured to use a modern, robust TLS cipher suite, such as **TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384**, to protect all data in transit. This dual approach addresses the client's specific requirements for both file verification and overall secure communication, aligning with current industry best practices.

The Secure Hash Algorithm 256, or SHA-256, is a cryptographic hash function that creates a unique, fixed-size 256-bit digital fingerprint for any given piece of data. This process is a one-way function, making it computationally infeasible to reverse-engineer the original data from its hash. For Artemis Financial, this means that when a file is transferred, a SHA-256 checksum can be generated and sent along with it. The recipient can then compute a new hash on the received file and compare it with the original; if they match, it provides strong assurance that the file has not been altered or corrupted. This method relies on the hash function itself and does not require keys.

In contrast, securing the web channel with HTTPS involves encryption, which uses keys. The recommended cipher suite employs both symmetric encryption (AES), which uses a single secret key to encrypt and decrypt the data exchanged between the client and server, and asymmetric encryption (RSA and ECDHE), which uses a public/private key pair to securely establish that shared secret key at the beginning of the session. The generation of these keys relies heavily on secure random numbers to ensure they are unpredictable and cannot be guessed by an attacker.

The field of cryptography evolves continuously as new vulnerabilities are discovered in older algorithms. Early hash functions like MD5 and SHA-1 are no longer considered secure for most applications, as weaknesses have been found that could allow for malicious data to generate the same hash value as a legitimate file. This led to the development and adoption of the SHA-2 family, of which SHA-256 is a prominent and widely trusted member. While a newer standard, SHA-3, exists, SHA-256 remains the industry standard, offering a proven and robust level of security appropriate for financial applications. By implementing SHA-256 for data integrity checks and a strong, modern TLS cipher suite for channel encryption, Artemis Financial will be deploying a defense-in-depth strategy that mitigates the risk of data tampering and eavesdropping, aligning its security posture with current best practices.

## Certificate Generation

Insert a screenshot below of the CER file.



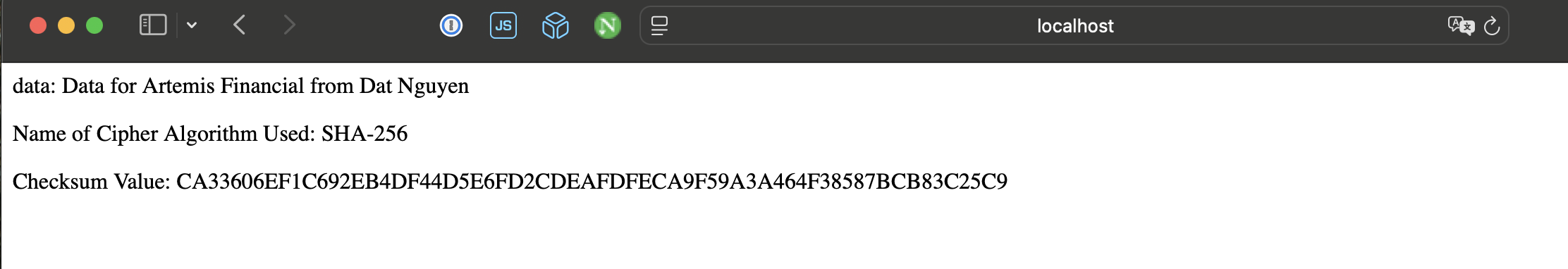
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

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

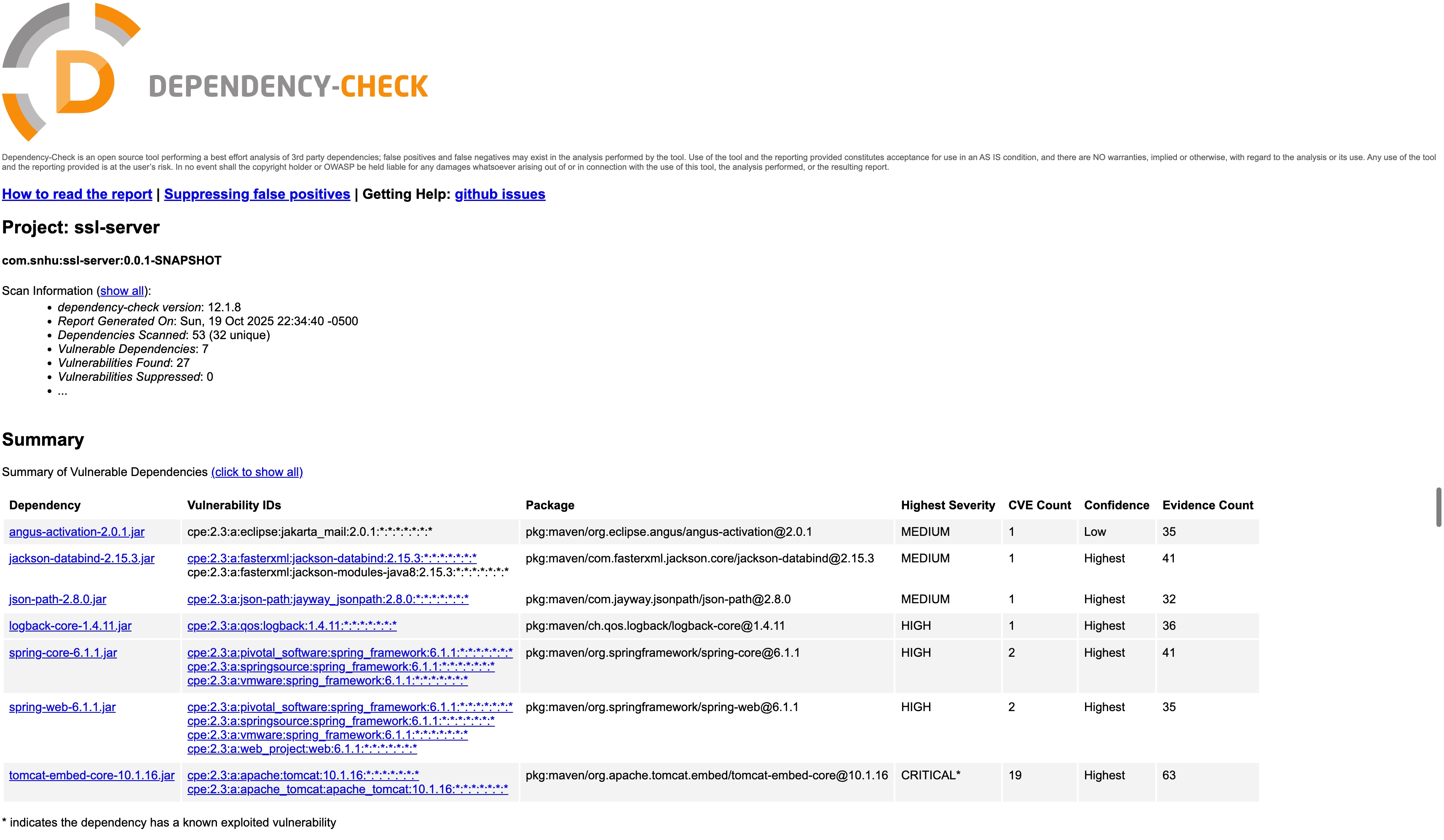


## Secondary Testing

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



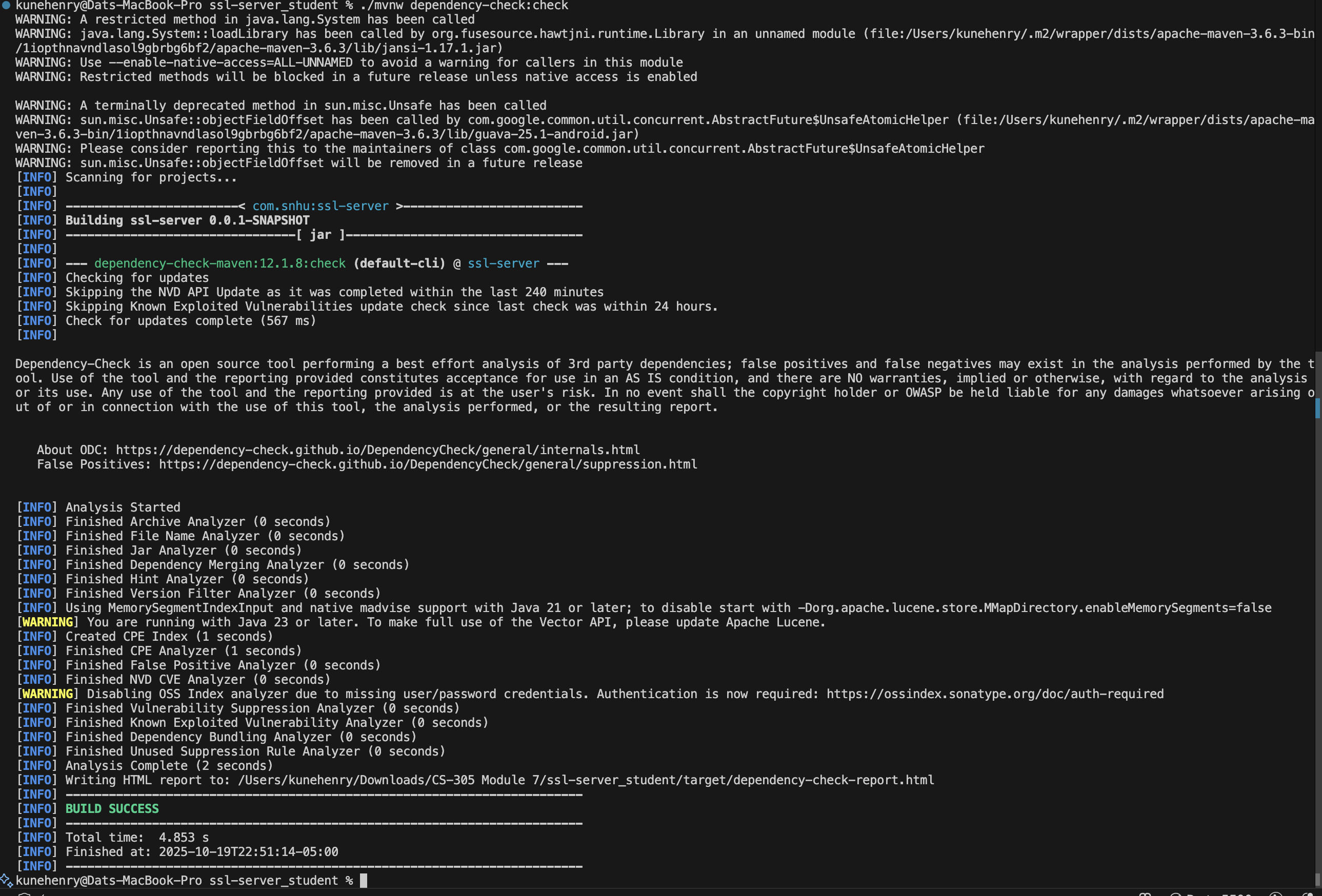


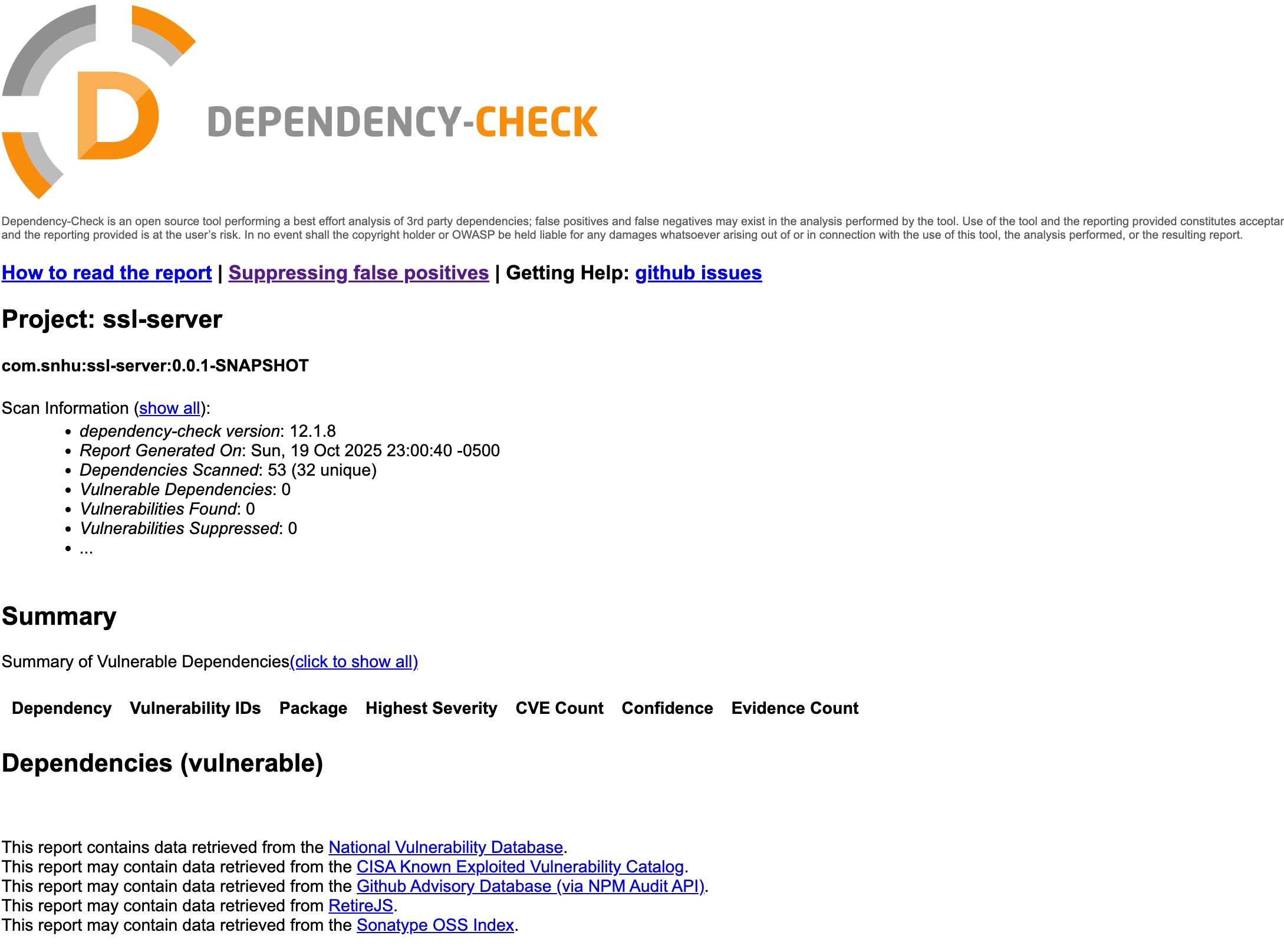


## Functional Testing

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







## Summary

In this project, I successfully enhanced the security posture of Artemis Financial's software application by implementing multiple layers of security. Referencing the Vulnerability Assessment Process Flow Diagram, the refactoring efforts specifically addressed the **Cryptography** and **Client/Server** components of the software development lifecycle. For Cryptography, I deployed a strong SHA-256 hash algorithm to serve as a checksum for data integrity verification, which is a critical control for ensuring that financial data is not altered in transit. For Client/Server security, I configured the application to enforce secure, encrypted communications by converting the protocol from HTTP to HTTPS using a self-signed TLS/SSL certificate, thereby protecting the confidentiality of the data being exchanged.

My process for adding these layers of security followed a structured approach. First, I addressed the specific requirement for a data verification step by implementing the SHA-256 checksum functionality within the Java code. Second, I secured the entire communication channel by generating a certificate with the Java Keytool and then refactoring the application.properties file to enable SSL, which is a foundational practice for protecting data in transit. Finally, to ensure these changes did not introduce new risks, I complied with security testing protocols by integrating the OWASP dependency-check tool into the Maven build process. I ran a full static analysis scan, which verified that the project's third-party libraries were free from any known, publicly disclosed vulnerabilities. This comprehensive process of implementing cryptographic controls, securing the transport layer, and verifying dependencies ensures the refactored code is robust and secure.

## Industry Standard Best Practices

Throughout this project, I applied several industry-standard best practices for secure coding to mitigate known security vulnerabilities and maintain the application's existing security. First and foremost, by implementing HTTPS, I ensured that all data in transit between the client and the Artemis Financial server is protected by Transport Layer Security (TLS). This is the fundamental industry standard for preventing eavesdropping and man-in-the-middle attacks, safeguarding the confidentiality and integrity of sensitive financial data. Second, the choice of **SHA-256** for the checksum aligns with current cryptographic standards recommended for data integrity, consciously avoiding older, deprecated algorithms like MD5 or SHA-1, which are known to be vulnerable. Finally, I integrated a static analysis security testing (SAST) tool, the OWASP dependency-check, directly into the build process. This practice is a key component of a modern Secure Software Development Lifecycle (SSDLC), allowing for the automatic detection of known vulnerabilities within third-party libraries before they can be exploited in a production environment.

The value of applying these industry-standard best practices to a company like Artemis Financial is immense and multifaceted. Primarily, it provides robust protection for their most critical asset: their clients' sensitive financial data. A security breach could result in direct economic losses, reputational damage, and significant legal and regulatory penalties. By building security into the application from the ground up, the company demonstrates a strong commitment to protecting its customers, which is a crucial factor in building and maintaining client trust—a priceless commodity in the financial services industry. Furthermore, adopting a secure coding culture and integrating automated security testing is far more cost-effective in the long run. It is significantly cheaper and more efficient to identify and fix security flaws during the development phase than it is to perform emergency remediation after a costly security breach has occurred. Ultimately, these practices contribute directly to the company's overall well-being by reducing risk, fostering customer loyalty, and protecting the bottom line.