****

# 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 | 12-11-2022 | Allen White | draft |

## Client



## Developer

Allen White

## Algorithm Cipher

Algorithmic ciphers are instructions which computers use to convert plain-text information to secure encrypted text through a series of translations and transformations. Many hash functions (algorithmic ciphers) are block ciphers which break input text into words of a specific size (of a specific bit size or length), and perform mathematical operations to them incorporating an initial secret key and pseudorandom numbers until they reach a specific form which is virtually impossible to reverse engineer (Block Cipher - Glossary | CSRC, n.d.). The pseudorandom numbers that are introduced during the operation of the algorithm are never truly random as they are pulled from internal readings and other sources of randomness which the computer can access, these “seed” values are then transformed through a separate algorithm to produce a unique value used inside of the cipher. While it is important to encrypt data through strong cipher algorithms, the resulting data must be reversible to a plain-text state again once received by the intended recipient. To do this, there are two common approaches – a symmetric and an asymmetric key pair approach.

In a symmetric key approach (less secure), a specific “key” or initial value is used by the sender to encrypt the file, which corresponds to the same key held by the recipient to reverse the encryption. The problem is that anyone can intercept the key to decrypt the message. In an asymmetric key approach, two keys are held by both the sender and receiver of the message, one that is “public” and one which is “private” and highly sensitive. The sender and receiver exchange public keys, then use the public key of the other to encrypt messages intended for the other. During transit, even if someone had stolen the public key of the receiver, they could encrypt messages, but could not decrypt the message as only the private key of the receiver can encrypt the message. Upon receipt, the private key is used to decrypt the message.

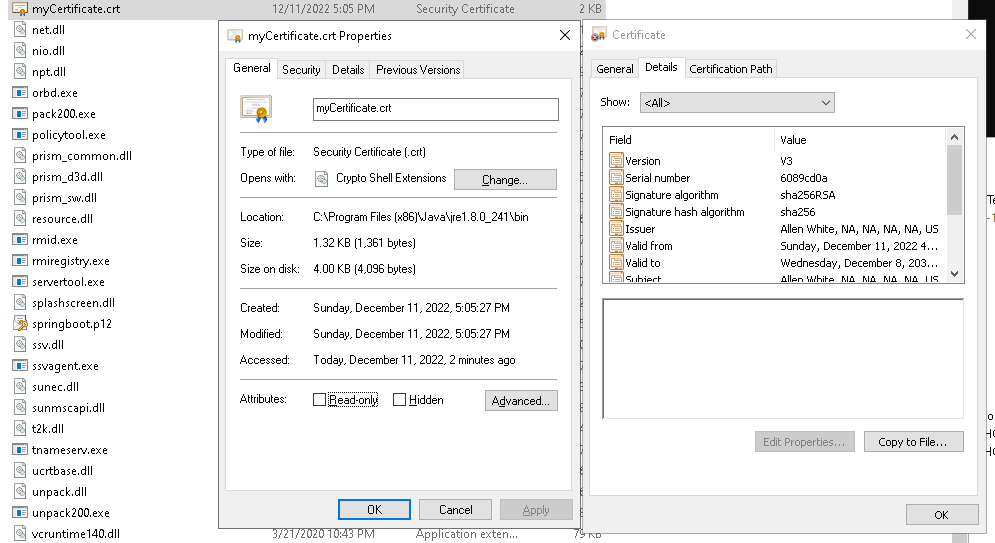
HTTPS (Hypertext Transfer Protocol Secure), which we have enabled for Artemis Financial, leverages the asymmetric key scheme described above to ensure that the transmissions to clients are done securely over TLS 1.3 (Transport Layer Security). We have generated a local certificate and key to use for demonstration through the Java Keytool utility, but in production we would need a Public Certificate Authority to issue a certificate telling customer browsers to trust our sites and transmissions. We do also have the option of hosting our own in production.

The encryption algorithm cipher which was deployed in this work for the purpose of ensuring data integrity (data sent is the same as data received) is the 256 bit Secure Hash Algorithm 3 (SHA3-256 ). While other algorithms would have been sufficient, such as the SHA-256 of SHA-512, the SHA3-256 algorithm is more resistant to collisions and has , in essence meaning that it is more secure. These algorithms are part of the SHA-3family which replaces the SHA-2 families in 2015, and are approved for by The National Institute of Standards and Technology for use by federal agencies, stating, “ Currently only the four fixed-length SHA-3 algorithms are approved hash algorithms, providing alternatives to the SHA-2 family of hash functions.” (Hash Functions | CSRC, n.d.-b).

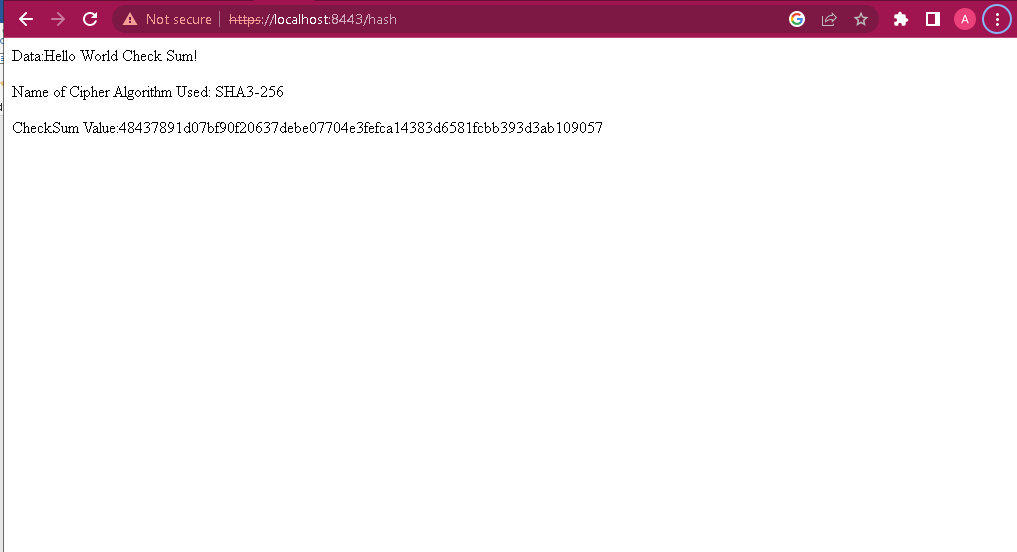
While the cryptographic techniques which are described above are adequate for the secure transmissions for Artemis Financial, the race between secure programming and hackers is constant. The majority of these algorithms and approaches work because the time and complexity of decrypting transmissions is extremely resource intensive (time, cost, and complexity). With the introduction of new types of computers such as quantum computers, the time complexity required to decrypt transmissions and stored secrets may become trivial, which has spurred research into quantum cryptography (Post-Quantum Cryptography, 2022). While on the horizon for most, quantum computers are a reality now, and as they are refined will pose a much greater risk to the security of organizations like Artemis Financial which will require constant vigilance.

## Certificate Generation

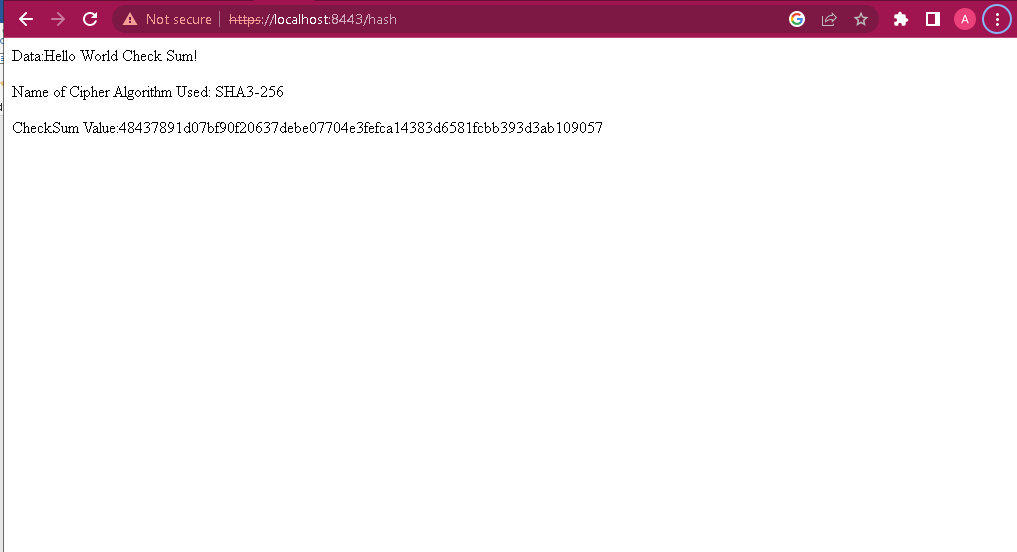


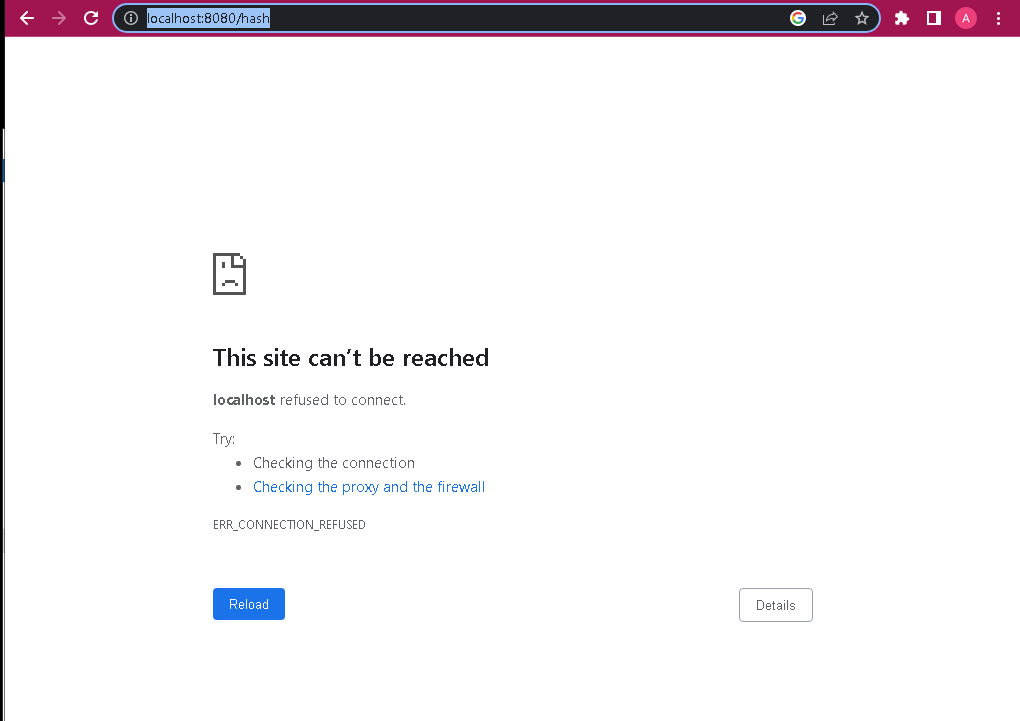


## Deploy Cipher

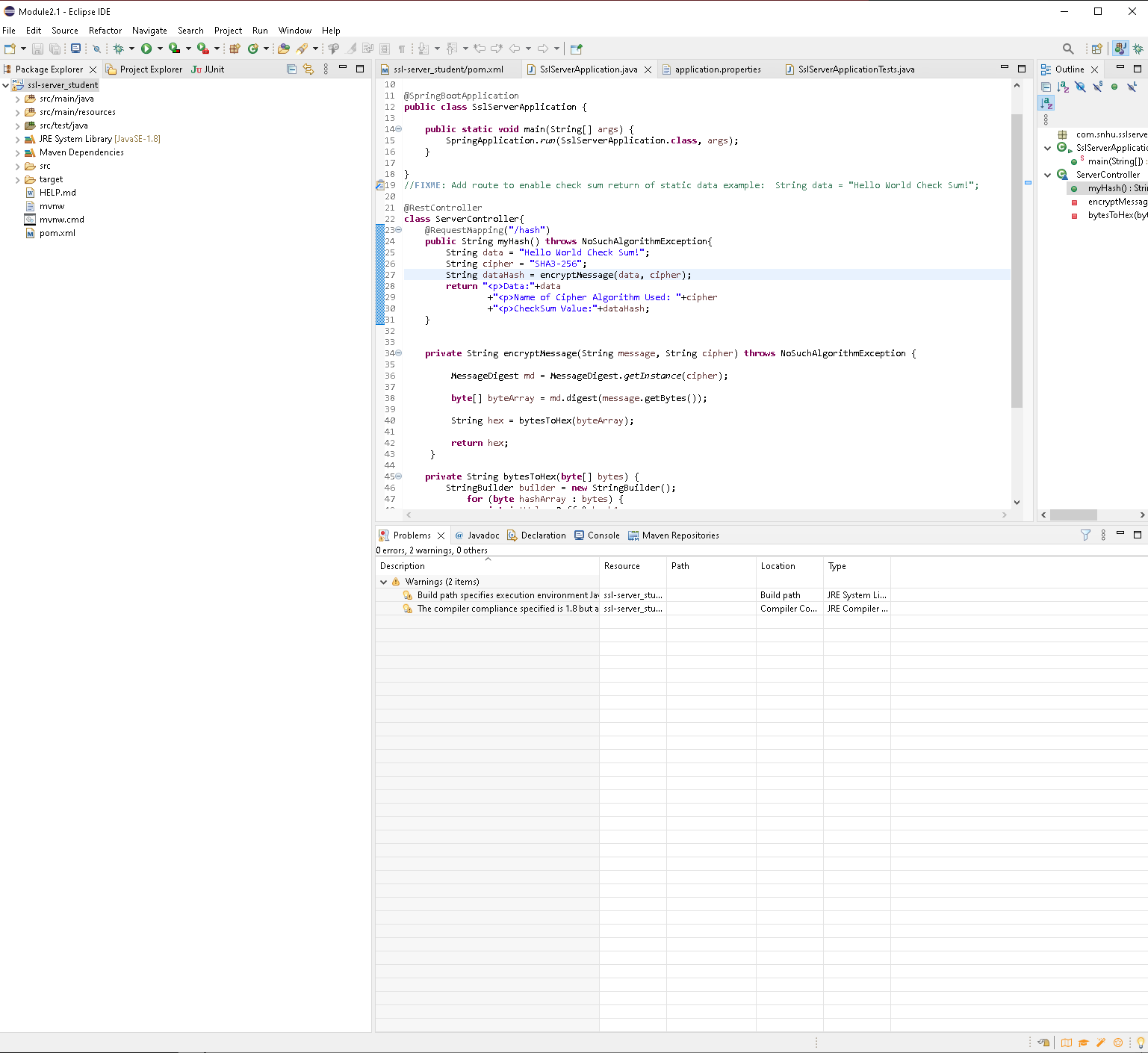


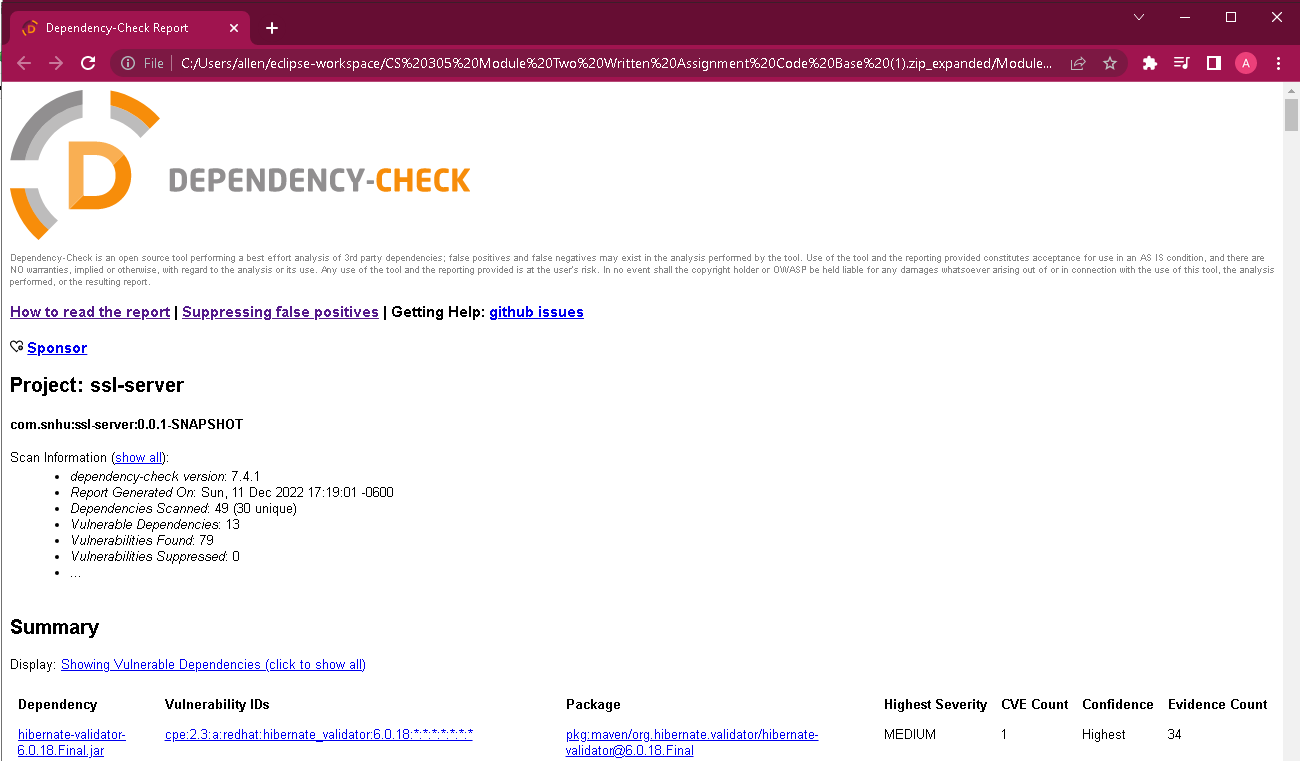
## Secure Communications



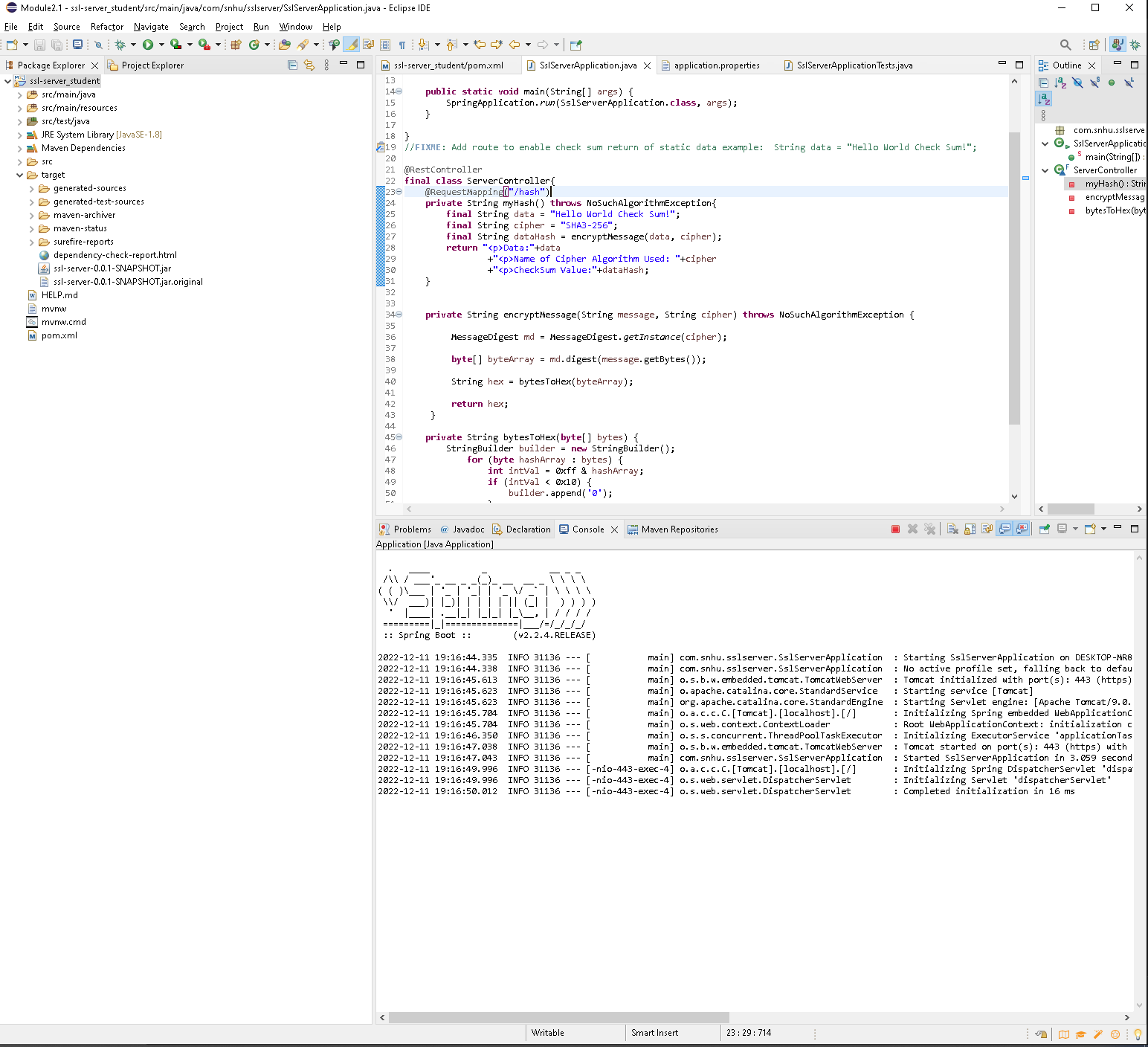


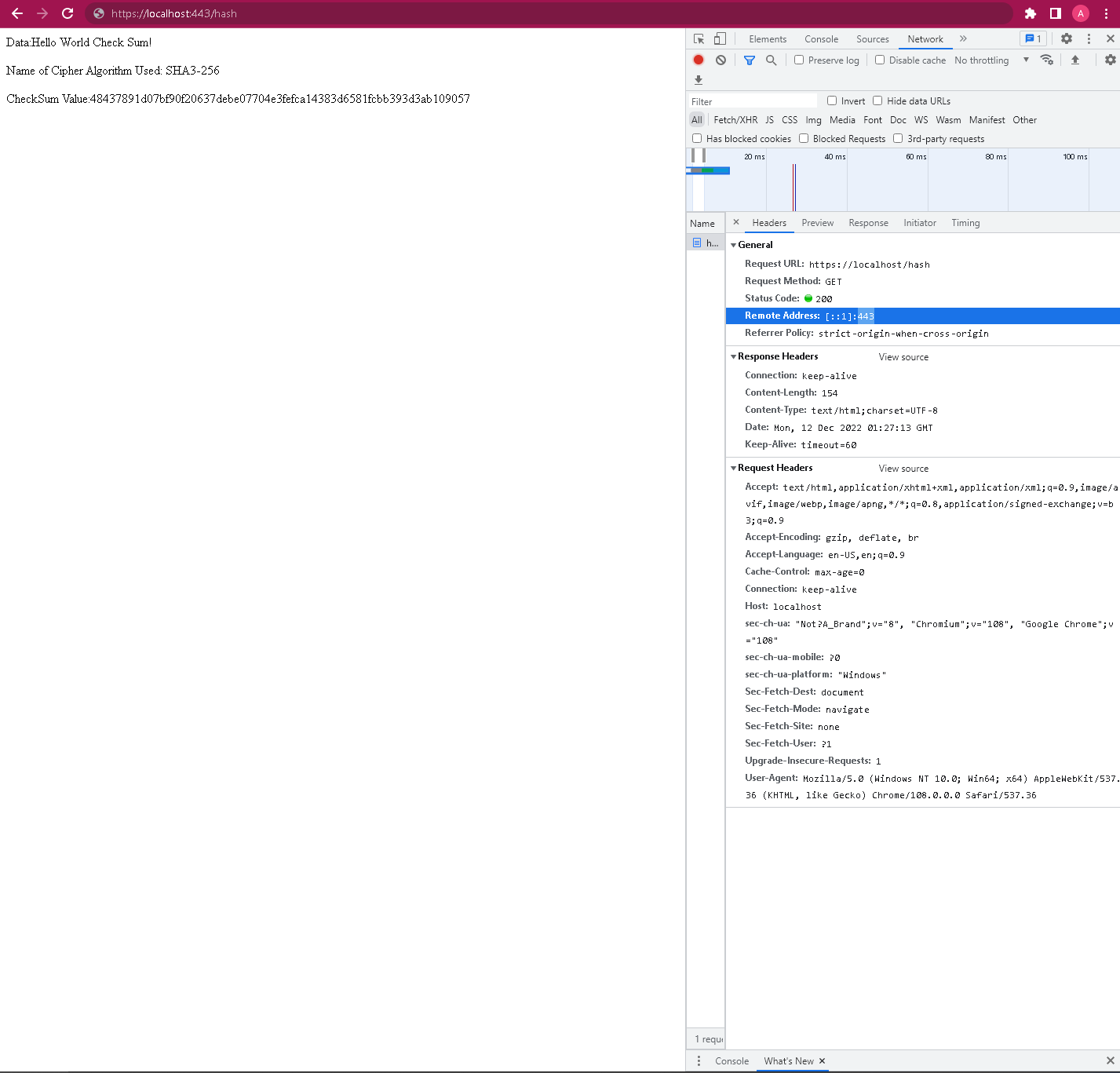
## Secondary Testing





## Functional Testing





## Summary

The software for Artemis Financial has been refactored to ensure that greater security is included in the application without adding any new dependency vulnerabilities (verified by above secondary dependency scanning). Through further review of the codebase for syntactical, logical, and security vulnerabilities, the following errors were found which existed prior to the above improvements to the codebase. Though more could be made, these establish an initial set of securities which do not add to the technical debt of the program, and will see positive results which layer onto the existing features. By applying a “security in depth” approach, we are able to ensure the security of each piece of the application. Everything from functional testing and hacking in the browser to code refinement and architectural adjustments will result in a more secure application.

**Cryptography**

* TLS1.3, HTTPS, and SHA3 implemented.

**Client/server**

* Strings and parent class of myHash function have been made immutable to prevent tampering once application runs.

**Encapsulation**

* Methods made private where possible to control access and transformations of data.

**Session**

* Adjusted port to 443 as previous port (8443) indicates to malicious actors that the Apache Tomcat server is being used and may facilitate further tampering.

## Industry Standard Best Practices

In our review and refactoring of the application for Artemis Financial, we have made a number of changes which allow the financial client to comply with the industry best practices in cybersecurity. Again, while addressing more of the dependency vulnerabilities identified through static testing will be critical in the future, we have implemented a number of controls to make the company and application more secure. As companies in the financial sector have some of the greatest chance of cybersecurity breach, it is important to resolve and to anticipate the potential issues which malicious actors may exploit (Secure Hash Algorithm 256 - Glossary | CSRC, n.d.).

We have implemented the most secure cryptographic ciphers to ensure that transmissions with clients are trusted and immune to most attacks. We have refactored the code at a low level to ensure that the handling of data between functions is secure, and employed hashing to ensure data integrity remains for transmissions (Transport Layer Security (TLS) - Glossary | CSRC, n.d.). Through this and several other reviews of the codebase, Global rain developers have made great strides in bringing the industry best practices to the client. All code is submitted with this document and we are available to answer any questions for our client.

References

*block cipher - Glossary | CSRC*. (n.d.). <https://csrc.nist.gov/glossary/term/block_cipher>

*Hash Functions | CSRC*. (n.d.-b). <https://csrc.nist.gov/Projects/hash-functions>

*NATIONAL VULNERABILITY DATABASE*. (n.d.). Retrieved November 6, 2022, from <https://nvd.nist.gov/vuln/detail/CVE-2019-7701>

*Post-Quantum Cryptography*. (2022, January 24). NIST. <https://www.nist.gov/programs-projects/post-quantum-cryptography>

*Secure Hash Algorithm 256 - Glossary | CSRC*. (n.d.). <https://csrc.nist.gov/glossary/term/secure_hash_algorithm_256>

*Secure Coding Guidelines for Java SE*. (2022, October). Oracle.com. <https://www.oracle.com/java/technologies/javase/seccodeguide.html>

*Transport Layer Security (TLS) - Glossary | CSRC*. (n.d.). <https://csrc.nist.gov/glossary/term/transport_layer_security>