GLOBALRAIN

Practices for Secure Software Report

Table of Contents

Document Revision History

Version	Date	Author	Comments
1.0	04/15/2025	Ronny Valtonen	Initial Creation
1.1	4/16/2025		Final Edits

Client

ARTEMIS
FINANCIAL

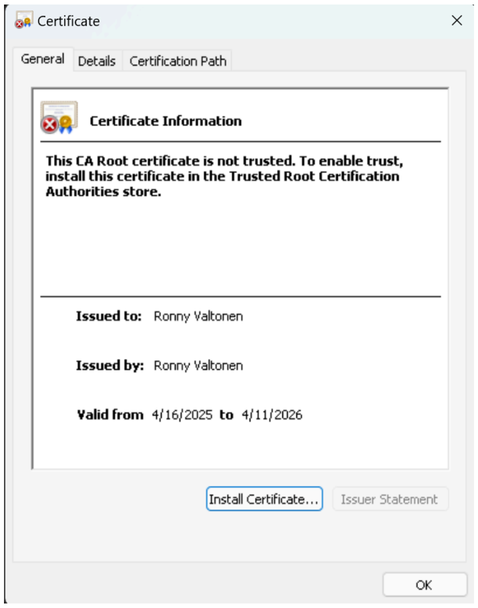
Developer
Ronny Z. Valtonen

1. Algorithm Cipher

To protect Artemis Financial, it is recommended to generate a SHA-256 hash because it is widely accepted and used. Unlike other ciphers that use keys, this sends a one way function. This way we generate a checksum and verify data integrity with data on the move. The bit level of SHA-256 is 256 bits with a block size of 512 bits. A cipher's **bit level** is what defines the security strength, referring to the length of a cryptographic key for encryption. A higher bit level provides more resistance to brute-force attempts. The use of **random numbers** in encryption ensures the keys are non-predictable which prevent attackers from pattern based attacks or simply guessing the encryption key. A **symmetric key** encryption such as AES uses the same key to encrypt and decrypt data. This makes it fast for bulk storage like database files, however it also means that the key needs to be shared to multiple people. This is why key management needs to be secure and the principle of least access is 'key.' An **asymmetric key** encryption uses a private key and public key to encrypt and decrypt data. This encryption type is great for digital signatures and authentication but is slower due to it being more hardware demanding than symmetric key encryption. Implementing asymmetric encryption to transmit symmetric keys is a typical use case.

In history, the most well known and oldest cipher is the Caesar Cipher which is a simple shift cipher and the most simple encryption techniques. A left shift of 4 means that the letter "D" would become "Z." This is now a popular coding homework for a lot of professors (Smith, James). Another encryption method that is widely known is the Enigma machine which was used in World War 2 in all branches of the German military (EnigmaHistory, n.d.). This was, during its time, the most secure encryption device to encipher top-secret messages. As hardware has improved, the need for stronger ciphers has become important. The emergence of DES in 1977 provided a great improvement for encryption. However, due to a short key length of only 56, it can be hacked in 22 hours and 15 minutes (Wikipedia contributors, 2025). Today we have RSA and AES which are the strongest ciphers to exist and with current supercomputers are impossible to brute-force. With hardware continuously improving, there are even stronger algorithms being produced like post-quantum encryption. The NIST has released 3 post-quantum encryption standards; the purpose of which quantum computers could break current encryption standards easily. Some experts predict that a device with the capability to break AES-256 could appear within a decade (NIST, 2025). Because this is a widely recognized algorithm with no currently known vulnerabilities, it is quick to implement. No two inputs will ever produce the same hash output. In this scenario, we can verify that the public key hasn't been tampered with. Using MessageDigest (https://docs.oracle.com/javase/8/docs/api/java/security/MessageDigest.html) it is easy to implement and reliable. Because it is industry standard, it is compatible with the JDK platform. Avoiding collisions is important because if an attacker is able to create a different file with the same hash, they could replace the public key with a malicious one without anyone knowing. SHA-256 entirely overrides this as even a remote possibility.

2. Certificate Generation



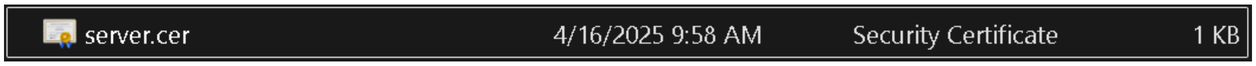
Certificate Information

This CA Root certificate is not trusted. To enable trust, install this certificate in the Trusted Root Certification Authorities store.

Issued to: Ronny Valtonen

Issued by: Ronny Valtonen

Valid from 4/16/2025 to 4/11/2026



server.cer 4/16/2025 9:58 AM Security Certificate 1 KB

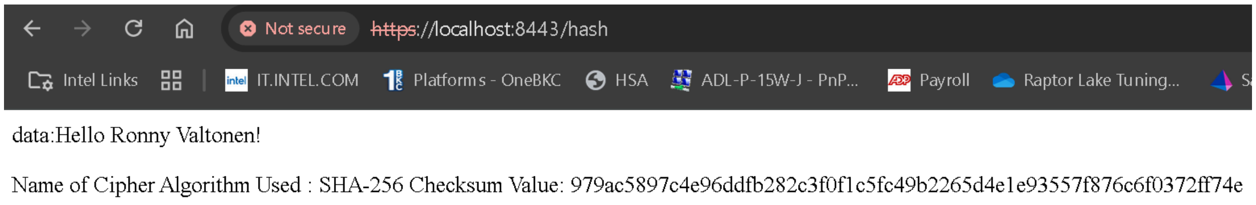
To successfully generate the server.cer file, I used the Java Keytool provided by the JVM and exported the CER file as a keystore 'keystore.jks' which allows my application to run the server using this self signed cert. Due to it being self signed and not using a CA signed cert, it will not be seen as secure to a client; this is why it is recommended to use a CA signed cert for real applications.

3. Deploy Cipher

```
data:Hello Ronny Valtonen!

Name of Cipher Algorithm Used : SHA-256 Checksum Value: 979ac5897c4e96ddfb282c3f0f1c5fc49b2265d4e1e93557f876c6f0372ff74e
```

4. Secure Communications



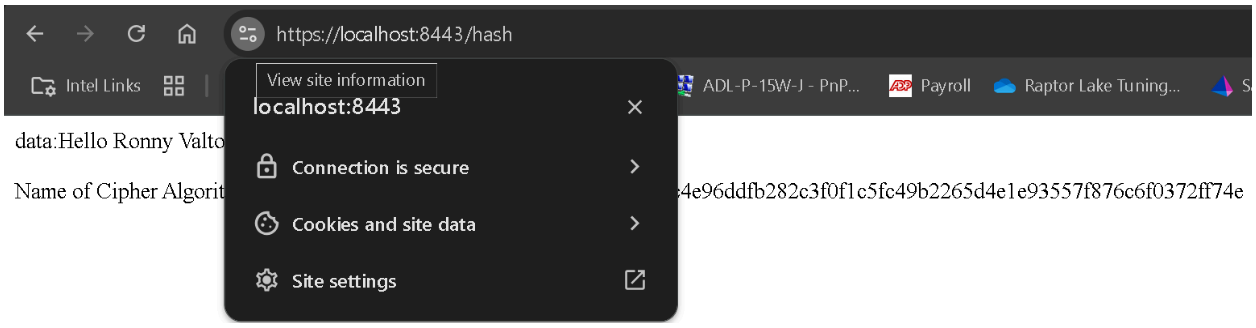
data:Hello Ronny Valtonen!

Name of Cipher Algorithm Used : SHA-256 Checksum Value: 979ac5897c4e96ddfb282c3f0f1c5fc49b2265d4e1e93557f876c6f0372ff74e

This is over HTTPS but still shows as not-secure because it is a self-signed cert. “Not Secure” is normal behavior on self-signed certs.

Below for testing purposes of a CA-Signed cert

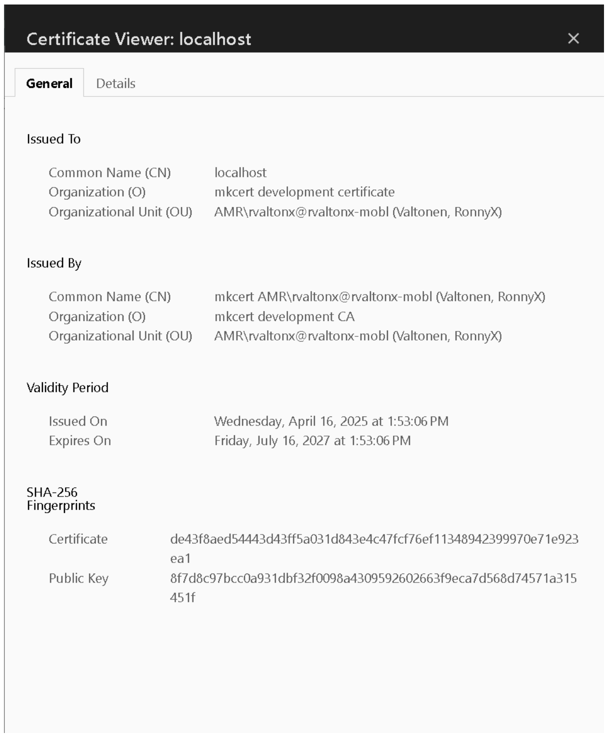
As explained above, I wanted to showcase the use of a public CA for Localhost - MKCert for development. First, I had to install MKCert though chocolatey, then using that, generate a PKCS12 for localhost. I also had to install the cert for my Windows Environment, after successfully getting everything setup, here we can now have secure HTTPS connections.



data:Hello Ronny Valto

Name of Cipher Algorit ... 4e96ddfb282c3f0f1c5fc49b2265d4e1e93557f876c6f0372ff74e

Here we can see that now the site shows as ‘Secure’ as it is no longer a self signed certificate. Using the CA Signed cert is more secure and strengthens trust with the client.



Issued To

Common Name (CN)	localhost
Organization (O)	mkcert development certificate
Organizational Unit (OU)	AMR\rvaltonx@rvaltonx-mobl (Valtonen, RonnyX)

Issued By

Common Name (CN)	mkcert AMR\rvaltonx@rvaltonx-mobl (Valtonen, RonnyX)
Organization (O)	mkcert development CA
Organizational Unit (OU)	AMR\rvaltonx@rvaltonx-mobl (Valtonen, RonnyX)

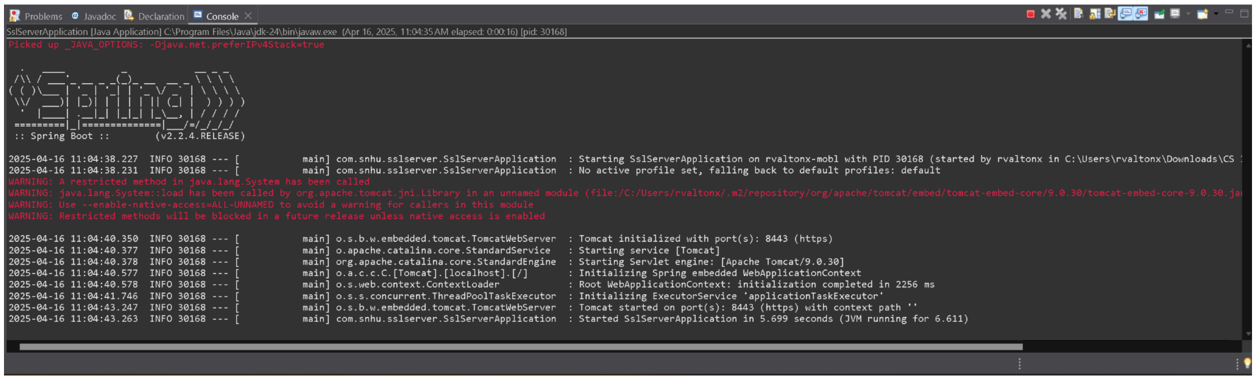
Validity Period

Issued On	Wednesday, April 16, 2025 at 1:53:06 PM
Expires On	Friday, July 16, 2027 at 1:53:06 PM

SHA-256 Fingerprints

Certificate	de43f8aed54443d43ff5a031d843e4c47fcf76ef11348942399970e71e923ea1
Public Key	8f7d8c97bcc0a931dbf32f0098a4309592602663f9eca7d568d74571a315451f

5. Secondary Testing



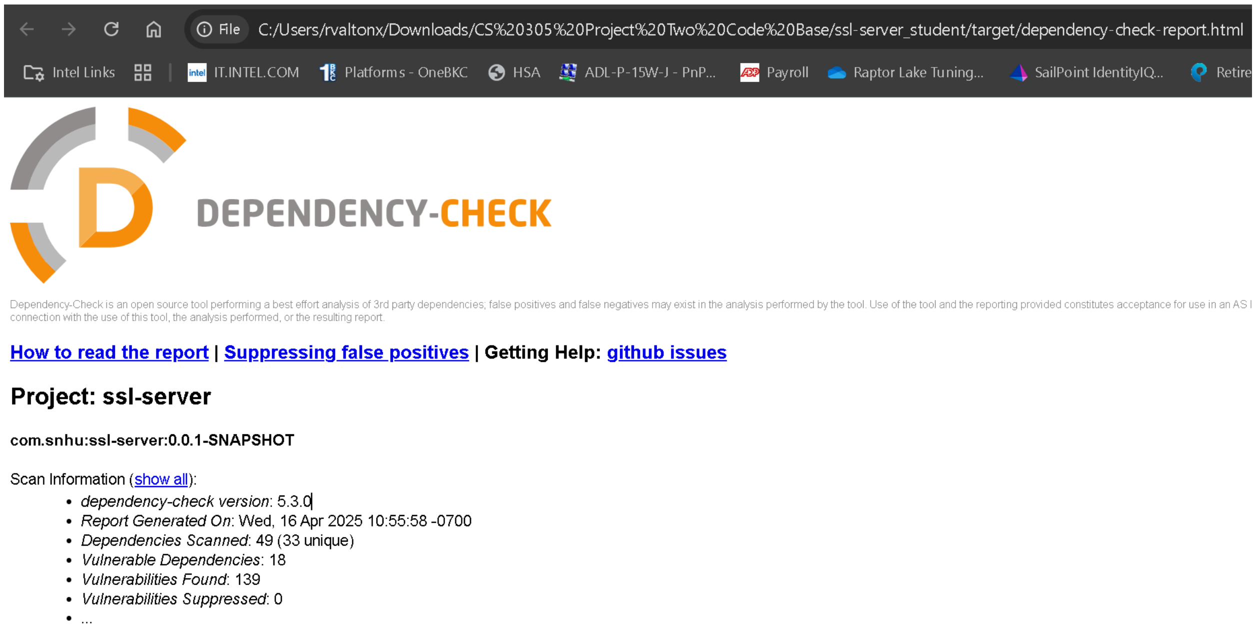
```
SslServerApplication [Java Application] C:\Program Files\Java\jdk-24\bin\javaw.exe  (Apr 16, 2025, 11:04:35 AM elapsed: 0:00:16) [pid: 30168]
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v2.2.4.RELEASE)

2025-04-16 11:04:38.227  INFO 30168 --- [           main] com.snhu.sslserver.SslServerApplication  : Starting SslServerApplication on rvaltonx-mobl with PID 30168 (started by rvaltonx in C:\Users\rvaltonx\Downloads\CS
2025-04-16 11:04:38.231  INFO 30168 --- [           main] com.snhu.sslserver.SslServerApplication  : No active profile set, falling back to default profiles: default
WARNING: A restricted method in java.lang.System has been called
WARNING: java.lang.System::load has been called by org.apache.tomcat.jni.Library in an unnamed module (file:/C:/Users/rvaltonx/.m2/repository/org/apache/tomcat/embed/tomcat-embed-core/9.0.30/tomcat-embed-core-9.0.30.ja
WARNING: Use --enable-native-access=ALL-UNNAMED to avoid a warning for callers in this module
WARNING: Restricted methods will be blocked in a future release unless native access is enabled

2025-04-16 11:04:40.350  INFO 30168 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8443 (https)
2025-04-16 11:04:40.377  INFO 30168 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-04-16 11:04:40.378  INFO 30168 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.30]
2025-04-16 11:04:40.577  INFO 30168 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-04-16 11:04:40.578  INFO 30168 --- [           main] o.s.web.context.ContextLoader            : Root WebApplicationContext: initialization completed in 2256 ms
2025-04-16 11:04:41.746  INFO 30168 --- [           main] o.s.s.concurrent.ThreadPoolTaskExecutor  : Initializing ExecutorService 'applicationTaskExecutor'
2025-04-16 11:04:43.247  INFO 30168 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8443 (https) with context path ''
2025-04-16 11:04:43.263  INFO 30168 --- [           main] com.snhu.sslserver.SslServerApplication  : Started SslServerApplication in 5.699 seconds (JVM running for 6.611)
```

Note that this shows warnings due to using a later version of Java which doesn't support the version of Tomcat, however in this application it does not cause issues.



C:/Users/rvaltonx/Downloads/CS%20305%20Project%20Two%20Code%20Base/ssl-server_student/target/dependency-check-report.html

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by the tool. Use of the tool and the reporting provided constitutes acceptance for use in an AS I
connection with the use of this tool, the analysis performed, or the resulting report.

How to read the report | Suppressing false positives | **Getting Help:** github issues

Project: ssl-server

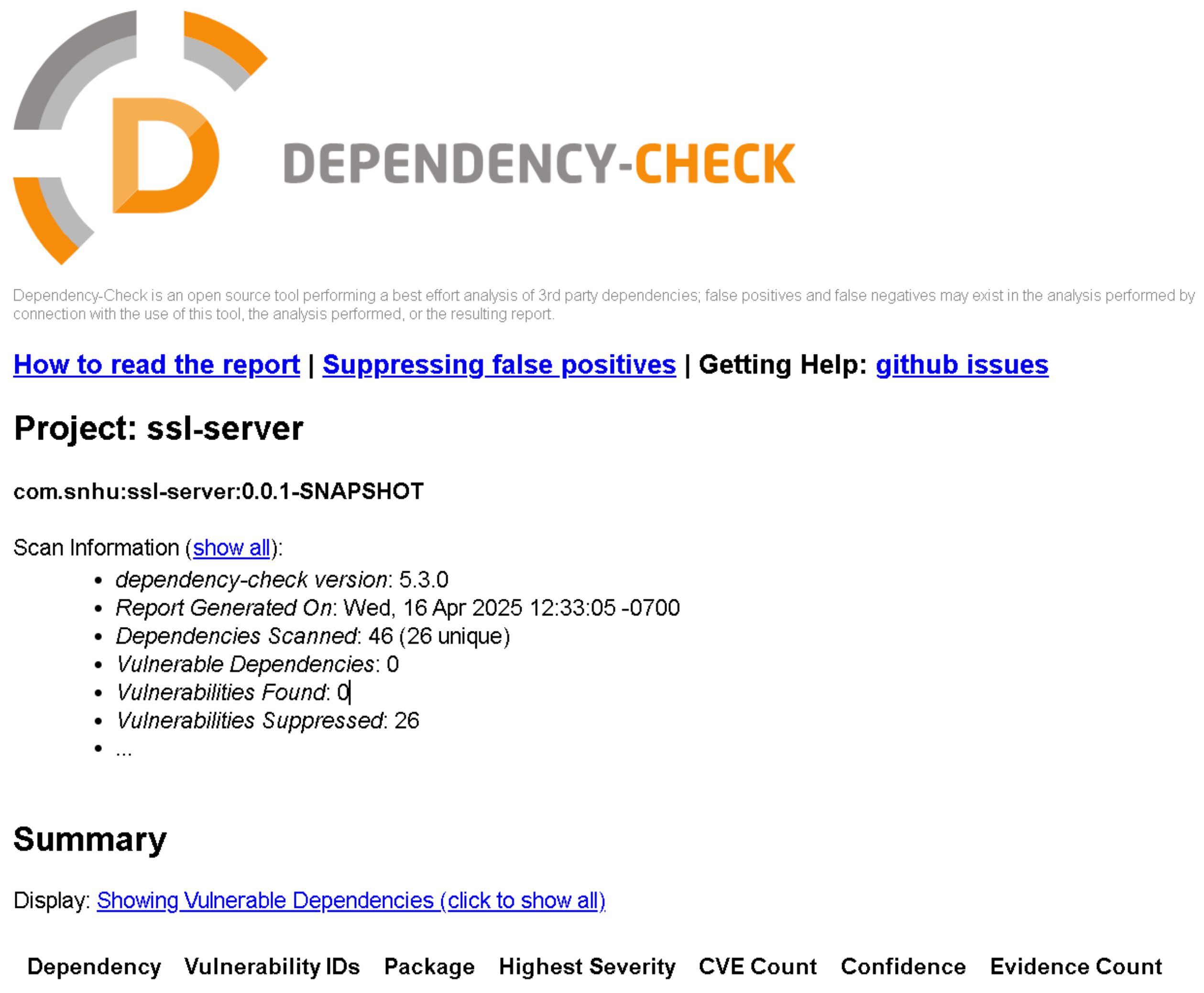
com.snhu:ssl-server:0.0.1-SNAPSHOT

Scan Information (show all):

- *dependency-check version*: 5.3.0
- *Report Generated On*: Wed, 16 Apr 2025 10:55:58 -0700
- *Dependencies Scanned*: 49 (33 unique)
- *Vulnerable Dependencies*: 18
- *Vulnerabilities Found*: 139
- *Vulnerabilities Suppressed*: 0
- ...

Here we can see a lot of vulnerable dependencies. Many of which are false positives as this application and use case is not affected by the noted vulnerability.

6. Functional Testing



DEPENDENCY-CHECK

Dependency-Check is an open source tool performing a best effort analysis of 3rd party dependencies; false positives and false negatives may exist in the analysis performed by connection with the use of this tool, the analysis performed, or the resulting report.

How to read the report | Suppressing false positives | Getting Help: github issues

Project: ssl-server

com.snhu:ssl-server:0.0.1-SNAPSHOT

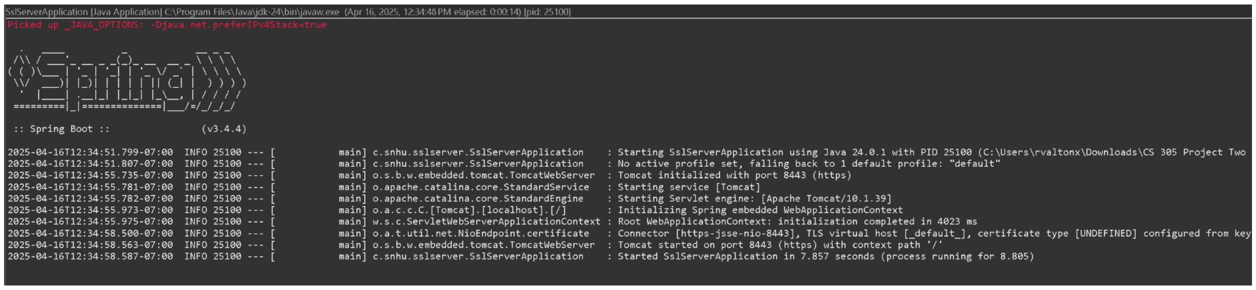
Scan Information (show all):

- *dependency-check version*: 5.3.0
- *Report Generated On*: Wed, 16 Apr 2025 12:33:05 -0700
- *Dependencies Scanned*: 46 (26 unique)
- *Vulnerable Dependencies*: 0
- *Vulnerabilities Found*: 0
- *Vulnerabilities Suppressed*: 26
- ...

Summary

Display: Showing Vulnerable Dependencies (click to show all)

Dependency	Vulnerability IDs	Package	Highest Severity	CVE Count	Confidence	Evidence Count



```
SslServerApplication [Java Application] C:\Program Files\Java\jdk-24\bin\javaw.exe  (Apr 16, 2025, 12:34:48 PM elapsed: 0:00:14) [pid: 25100]
Picked up _JAVA_OPTIONS: -Djava.net.preferIPv4Stack=true

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.4.4)

2025-04-16T12:34:51.799-07:00  INFO 25100 --- [           main] c.snhu.sslserver.SslServerApplication    : Starting SslServerApplication using Java 24.0.1 with PID 25100 (C:\Users\rvaltonx\Downloads\CS 305 Project Two
2025-04-16T12:34:51.807-07:00  INFO 25100 --- [           main] c.snhu.sslserver.SslServerApplication    : No active profile set, falling back to 1 default profile: "default"
2025-04-16T12:34:55.735-07:00  INFO 25100 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8443 (https)
2025-04-16T12:34:55.781-07:00  INFO 25100 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-04-16T12:34:55.782-07:00  INFO 25100 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.39]
2025-04-16T12:34:55.973-07:00  INFO 25100 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-04-16T12:34:55.975-07:00  INFO 25100 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 4023 ms
2025-04-16T12:34:58.500-07:00  INFO 25100 --- [           main] o.a.t.util.net.NioEndpoint.certificate   : Connector [https-jsse-nio-8443], TLS virtual host [_default_], certificate type [UNDEFINED] configured from key
2025-04-16T12:34:58.563-07:00  INFO 25100 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8443 (https) with context path '/'
2025-04-16T12:34:58.587-07:00  INFO 25100 --- [           main] c.snhu.sslserver.SslServerApplication    : Started SslServerApplication in 7.857 seconds (process running for 8.805)
```

After upgrading the Spring Boot to the latest Release version 3.4.4, this removed most vulnerabilities. And the 26 remaining vulnerabilities do not pertain to our use case, they can be safely suppressed. It was also a must to upgrade to Java 21. Running in Eclipse, the Project preferences Java JRE's had to be binded to JDK-24 instead of the JDK 1.8 for my application refactoring to work properly.

7. Summary

After going through an architecture review, it was found that ensuring APIs, Cryptography, Client/Server communication, Code Error, Code Quality and Encapsulation were the main areas of security that had to be managed. As there is no input validation, this did not have to be reviewed. Using the SHA-256 cryptographic algorithm was the best choice as it will handle any future data transfer for the company and is protected from brute force vulnerabilities. Changing the server application.properties to support HTTPS composed secure client/server communication. I encapsulated the hash generation into a try-catch statement to ensure there is secure error handling, this prevents an attacker from understanding any possible vulnerabilities. The ServerController is securely mapped where only required data is accessible outside of the class. I also ensured that any vulnerabilities found from the dependency check were either suppressed from false positives or by updating the dependencies where possible. After all of this was completed, a final code review was committed which found no new vulnerabilities from the re-factored code and any previously vulnerabilities were fixed/suppressed. These steps added layers of security to the software application. The remaining vulnerabilities did not pertain to the application, thus can be suppressed.

8. Industry Standard Best Practices

To further discuss, I used industry standard best practices to maintain the software applications existing security by updating out-of-date and/or deprecated dependencies as well as upgrading to a latest supported Java version for the application. Using older dependencies and older JVM's can introduce vulnerabilities to the rest of the application. I also used an industry standard hashing algorithm (SHA-256) to ensure a widely used algorithm is approved by NIST - this ensures it is resistant to collisions, keeping data reliably secure. Furthermore, using try/catch blocks prevents unhandled exceptions which could reveal possible vulnerabilities to an attacker and leaking technical details to the client.

I also used the OWASP dependency check Maven plugin to check for any possible vulnerable dependencies. I ran this prior to any code changes so I could ensure I did not add any new vulnerabilities. Once I refactored the code, I ran it again to compare to the first check and began fixing vulnerabilities that could be fixed. Some dependencies could not be upgraded, however upgrading the Spring Boot dependency to the latest release removed most vulnerabilities in which I could then go through and check for false positives and false negatives.

The value of applying industry standard best practices for secure coding ensures good trust between the company and the client. Not only this but it also ensures that data on the move will meet local and global standards like NIST and FIPS compliant. This ensures a secure reputation and further builds trust in clients that their data is secure in your application(s).

References

“Java Security Standard Algorithm Names,” n.d., https://docs.oracle.com/javase/9/docs/specs/security/standard-names.html#cipher-algorithm-names.

Jim Manico and August Detlefsen, *Iron-Clad Java: Building Secure Web Applications (Oracle Press)*, 2014, http://dl.acm.org/citation.cfm?id=2826076.

Auth, “Signing Algorithms,” Auth0 Docs, n.d., https://auth0.com/docs/get-started/applications/signing-algorithms#:~:text=HS256%20(HMAC%20with%20SHA-256,shared%20between%20the%20two%20parties.

James Smith Project Structure, “Writing Secret Messages With a Caesar Cipher | Golang Project Structure,” *Golang Project Structure* (blog), June 27, 2022, https://golangprojectstructure.com/caesar-cipher-secret-messages/.

“Enigma History,” n.d., https://www.cryptomuseum.com/crypto/enigma/hist.htm.

Wikipedia contributors, “Data Encryption Standard,” Wikipedia, March 28, 2025, https://en.wikipedia.org/wiki/Data_Encryption_Standard#:~:text=The%20EFF%27s%20DES%20cracker,DES%20key%20in%2056%20hours.&text=Together%2C%20Deep%20Crack%20and%20distributed,22%20hours%20and%2015%20minutes.

Chad Boutin, “NIST Releases First 3 Finalized Post-Quantum Encryption Standards,” NIST, February 4, 2025, https://www.nist.gov/news-events/news/2024/08/nist-releases-first-3-finalized-post-quantum-encryption-standards.