****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 5](#_Toc33111308)

[4. Secure Communications 5](#_Toc33111309)

[5. Secondary Testing 5](#_Toc33111310)

[6. Functional Testing 8](#_Toc33111311)

[7. Summary 9](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **10/11/21** | **Larry Arndt** |  |

## Client



## Developer

Larry Arndt

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

We recommend EdDSA which uses twisted Edwards curves. The algorithm is resistant to many forms of attack and while it does not require the use of random numbers it can incorporate them and that is recommended. The specific curve 25519 has been proven to be fault tolerant to hash collisions. It uses a 32-byte private key and a 32-byte public key which makes it asymmetric. The digital verification signature is calculated from a formula that hashes the private key and the message with modular division in one direction and with the public key in the other direction. The hashing is typically SHA-512. The digital signatures are 64 bytes. It was invented in 2011, by D. Bernstein, N. Duif, T. Lange, P. Schwabe, and B. Yang. Today it supersedes ECDSA because it provides collision resilience on hash functions, so they don’t break the system. Also, a unique random number for each signature is not required. With curve 25519 used for authentication you get several benefits. It is considered as secure as a ~3000 bit RSA key. Creating a new signature does not require random input. It is resilient to hash function collisions. It is immune to many side-channel attacks. It is very fast.

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

Text

Description automatically generated

## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.

Graphical user interface, text, application, Word

Description automatically generated

## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

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

Graphical user interface, text, application

Description automatically generated

## 5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

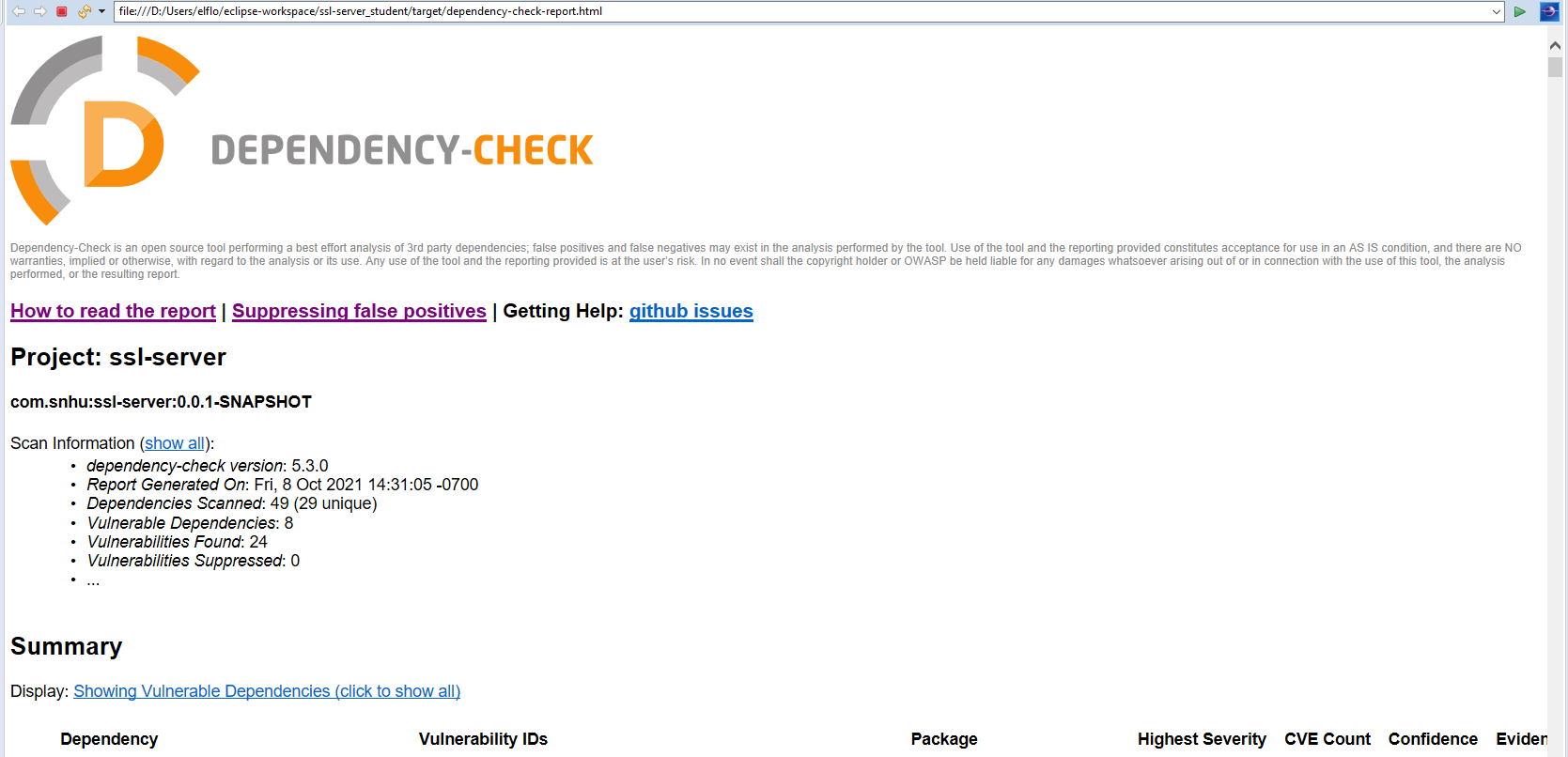
* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

Here is the code I added and it is running on the 8443 port

Graphical user interface, text, application

Description automatically generated

This is the before dependency check before I added any code to the project



Here is the after dependency check that shows no new dependencies added.

Graphical user interface, text, application, email

Description automatically generated

## 

## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.

The application properties are configured and the keystore files correctly located the build completes successfully.

Graphical user interface, application

Description automatically generated

Anticipates a possible exception being thrown for algorithm selection.

Graphical user interface, text, application

Description automatically generated

There is no log in to the program and the session does not time out or expire. The port has to be manually shut down the process from the command prompt to end the session. These are security issues.

## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.
* APIs are important because their interactions safeguard the application where it must communicate between interfaces.
* Client/Server security; secure communications with server-side authentications for web sessions, ensuring proper right entitlements.
* Cryptography is key to prevent sensitive data exposure over the web environment with proper encryption methods.
* Code error for secure error handling; catching all thrown exceptions with proper remedies.

The process I use for adding security is to manually inspect the code and run a vulnerability checker on the application as well, I have found some areas of need on input validation and error handling and software with vulnerabilities because it is out of date and needs upgraded, also the system does not run a internet session that would time out and there is no login credentials being checked. These are some of the things I have noticed. It would be best to generate a log in system for the employees and the client and get the sessions to be more dynamic where they do not stay open indefinitely.