Protocolos seguros de comunicaciones (SSL/TLS)

DANIEL RUIZ MAGRO DA2D1A

Contenido

Explicación del Programa

Este programa implementa un servidor y un cliente que **se comunican de forma segura usando SSL/TLS**.

El ServidorSSL:

- Escucha en un puerto seguro (8443) y espera conexiones de clientes.
- Carga un **certificado** desde keystore.jks para establecer una conexión cifrada.
- Recibe un mensaje del cliente y responde confirmando la recepción.

El ClienteSSL:

- Se conecta al servidor en el puerto 8443 utilizando **SSL/TLS**.
- Envia un mensaje al servidor y espera una respuesta.

La comunicación está cifrada y autenticada mediante un **certificado digital** almacenado en keystore.jks.

Explicación del Servidor (ServidorSSL)

El servidor SSL tiene la función de aceptar conexiones seguras de clientes y comunicarse con ellos de manera cifrada mediante **SSL/TLS**.

1. **Carga del certificado (keystore.jks)**
   - El servidor necesita autenticarse, por lo que carga su **almacén de claves (keystore)**, donde está almacenado su certificado digital y clave privada.
   - Para acceder a este archivo, se usa una **contraseña** que lo protege.
2. **Configuración del contexto SSL**
   - Se configura el **gestor de claves** (KeyManagerFactory), que se encarga de manejar el certificado del servidor.
   - Luego, se crea un **contexto SSL (SSLContext)** con la información de seguridad necesaria.
3. **Creación del socket SSL seguro**
   - Con la configuración SSL lista, se genera un **socket seguro (SSLServerSocket)** en el puerto **8443**.
   - El servidor queda **esperando conexiones** de clientes en este puerto.
4. **Aceptación de clientes**

- Cuando un cliente intenta conectarse, el servidor **acepta la conexión** y establece un canal seguro.

5. **Recepción de mensajes**
    - Una vez que la conexión está establecida, el servidor **lee el mensaje** enviado por el cliente.
6. **Envío de respuesta al cliente**
    - Después de recibir el mensaje, el servidor **responde con una confirmación** de que el mensaje fue recibido de manera segura.
7. **Cierre de conexión**
    - Finalmente, se cierran las conexiones con el cliente, aunque el servidor **permanece activo**, esperando nuevas conexiones.

Explicación del Cliente (ClienteSSL)

El cliente se encarga de conectarse de forma segura al servidor, enviar un mensaje y recibir una respuesta.

1. **Configuración del contexto SSL**
    - Se configura un **contexto SSL (SSLContext)** para poder crear un **socket seguro (SSLSocket)**.
2. **Conexión al servidor**
    - Se usa la dirección **"localhost"** y el puerto **8443** para conectarse al servidor.
    - La conexión se establece mediante un **socket seguro (SSLSocket)**, lo que garantiza la seguridad de la comunicación.
3. **Envío de un mensaje al servidor**
    - Una vez conectado, el cliente **envía un mensaje** al servidor a través del socket seguro.
4. **Recepción de la respuesta del servidor**
    - El cliente **espera la respuesta** y la muestra en pantalla.
5. **Cierre de la conexión**
    - Finalmente, el cliente **cierra la conexión** y finaliza su ejecución.

Antes de ejecutar el programa, es necesario para que el servidor pueda usar **TLS**, tener un certificado. Se genera uno **autofirmado** con keytool, con ayuda del siguiente comando en el cmd:

```
PS D:\2°GS\PSP,PMDM\Protocolos seguros comunicaciones (SSL-TLS)\protocolosSegurosComunicacionesSSLyTLS> & "C:\Program Files\Java\jdk-21\bin\keytool.exe" -genkeypair -alias mi_certificado -keyalg RSA -keysize 2048 -validity 365 -keystore keystore.jks -storepass 123456
```

(Asegúrate de tener keystore.jks en la misma carpeta que los archivos .java)

Explicación del comando:

- -genkeypair → Genera un par de claves (pública y privada).
- -alias mi_certificado → Nombre del certificado (puedes cambiarlo).
- -keyalg RSA → Algoritmo de clave (RSA, recomendado).
- -keysize 2048 → Tamaño de la clave (mínimo 2048 bits por seguridad).
- -validity 365 → Duración del certificado (en días).
- -keystore keystore.jks → Nombre del archivo donde se guardará el certificado.
- -storepass 123456 → Contraseña del keystore

Se pide información sobre el certificado:

```
Enter the distinguished name. Provide a single dot (.) to leave a sub-component empty or press ENTER to use the default value in braces.
What is your first and last name?
  [Unknown]:  Daniel Ruiz
What is the name of your organizational unit?
  [Unknown]:  IT
What is the name of your organization?
  [Unknown]:  MiEmpresa
What is the name of your City or Locality?
  [Unknown]:  Madrid
What is the name of your State or Province?
  [Unknown]:  Madrid
What is the two-letter country code for this unit?
  [Unknown]:  ES
Is CN=Daniel Ruiz, OU=IT, O=MiEmpresa, L=Madrid, ST=Madrid, C=ES correct?
  [no]:  yes

Generating 2.048 bit RSA key pair and self-signed certificate (SHA384withRSA) with a validity of 365 days
        for: CN=Daniel Ruiz, OU=IT, O=MiEmpresa, L=Madrid, ST=Madrid, C=ES
PS D:\2°GS\PSP,PMDM\Protocolos seguros comunicaciones (SSL-TLS)\protocolosSegurosComunicacionesSSLyTLS>
```

Para ejecutarlo:

1. Se compilan los archivos:
   javac ServidorSSL.java ClienteSSL.java
2. Se inicia el **servidor** en una terminal: java ServidorSSL
3. Luego, en otra terminal, se ejecuta el **cliente**: java ClienteSSL

Código Fuente

Javadoc adjuntado en la entrega junto con los archivos .java comentados en líneas de código.

Repositorio en GitHub

https://github.com/daniielrm05/protocolosSegurosComunicacionesSSLyTLS_Daniel_Ruiz.git

Bibliotecas Usadas

javax.net.ssl.*

- **SSLServerSocket / SSLServerSocketFactory** → Para crear el servidor SSL.
- **SSLSocket / SSLSocketFactory** → Para crear la conexión segura desde el cliente.
- **SSLContext** → Configura el contexto SSL y permite el uso de TLS.

java.security.*

- **KeyStore** → Para cargar el almacén de claves (keystore.jks).
- **KeyManagerFactory** → Para gestionar las claves del servidor y establecer la autenticación.

java.io.*

- **FileInputStream** → Para leer el archivo de claves (keystore.jks).
- **InputStreamReader / BufferedReader** → Para leer datos del socket (mensajes).
- **PrintWriter** → Para enviar mensajes entre cliente y servidor.

Dependencias

El código usado no necesita **dependencias externas**, ya que todas las clases utilizadas pertenecen a la **biblioteca estándar de Java (JDK)**.