

# PAI 1

PAI-1. BYODSEC-BRING YOUR OWN DEVICE SEGURO  
PARA UNA ENTIDAD HOSPITALARIA USANDO ROAD  
WARRIOR VPN TLS

**Grupo 11**

Alejandro Inglés Martínez

Israel Brea Piñero

---

# INDICE

---

## **1. Introducción**

## **2. Pruebas de comunicación sin seguridad**

### **2.1 Lado del cliente**

### **2.2 Lado del servidor**

### **2.3 Lado del sniffer**

## **3. Creación de almacenes de certificados**

### **3.1 Creación de un keystore**

#### **3.1.1 Configuración del Path y HOME\_PATH**

#### **3.1.2 Información adicional importante**

## **4. Pruebas de comunicación con seguridad**

### **4.1 Lado del servidor**

### **4.2 Lado del cliente**

### **4.3 Lado del sniffer**

## **5. Prueba de capacidad**

## **6. Código fuente aplicaciones java cliente-servidor**

### **6.1 BYODClienteSinSeguridad**

### **6.2 BYODServerSinSeguridad**

### **6.3 BYODCliente300Users**

### **6.4 BYODServer300Users**

---

# 1. INTRODUCCIÓN

---

Una determinada entidad hospitalaria nos solicita la implementación de la Política de Seguridad Bring your Own Device (BYOD) seguro para sus empleados, que consiste en que éstos utilicen sus propios dispositivos para realizar sus trabajos, pudiendo tener acceso a recursos de la entidad tales como correos electrónicos, bases de datos y archivos en servidores corporativos usando una Road Warrior VPN TLS. Para la transmisión de todos estos elementos es fundamental la implementación de canales de comunicación seguros.

En este trabajo se han implementado sockets del tipo Secure Sockets Layers (SSL), utilizando el lenguaje de programación Java. Estos sockets tendrán incorporados el protocolo TLS. TLS es un protocolo de comunicación seguro que utiliza una infraestructura basada en almacenes de claves y certificados, lo que va a asegurar que la información transmitida sea confidencial, íntegra y autenticadas.

Para mostrar la importancia en la transmisión vamos a crear una infraestructura cliente-servidor para la comprobación de login/password de usuarios y un mensaje secreto mediante el uso de sockets seguros.

---

## 2. PRUEBAS DE COMUNICACIÓN SIN SEGURIDAD

---

En esta sección veremos la importancia de crear un socket de transmisión segura usando el protocolo TLS, cómo va a funcionar el servidor y el cliente, y, lo que podrá ver un man-in-the-middle si consigue conectarse al puerto del servidor.

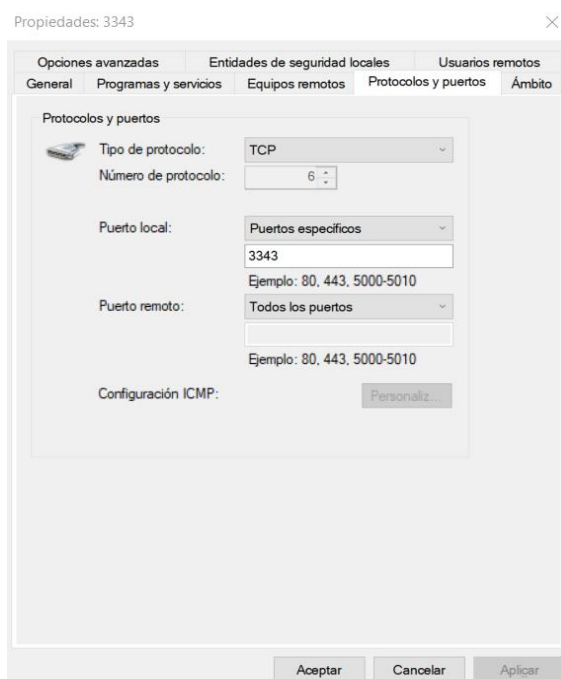
### 2.1 Lado del servidor

El código del servidor implementa un socket en Java que escucha continuamente a los clientes que se conecten en el puerto escogido. Cuando un cliente se conecta, el servidor espera a que el cliente envíe un nombre de usuario y una contraseña. Si el servidor recibe esta información, simplemente envía una respuesta de confirmación al cliente. Todo este proceso se realiza con un socket **sin seguridad**.

```
C:\Users\farne\OneDrive - UNIVERSIDAD DE SEVILLA\Escritorio\ws_Seguridad\PAI1\src\PAI1>java BYODServerSinSeguridad.java
Waiting for connection...
```

Img 1. Server esperando la conexión de clientes.

Para que la conexión entre cliente y servidor funcione correctamente, es importante que el puerto de conexión implementado en el código java esté permitiendo su acceso en el servidor. Por lo tanto, habría que permitir la conexión tanto en el firewall, como en el router del servidor.



Img 2. Configuración avanzada del firewall para permitir el acceso por el puerto 3343.

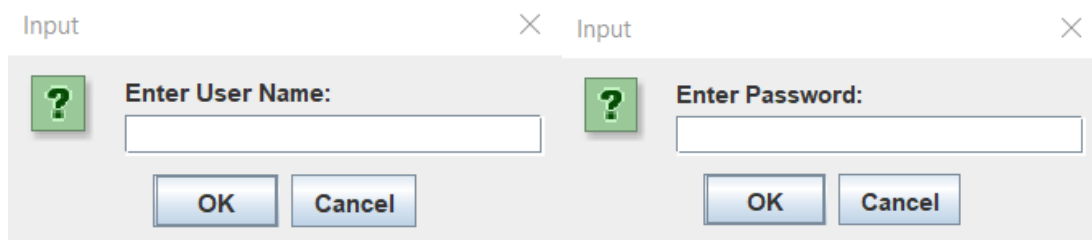
## 2.2 Lado del cliente

En el caso del cliente el código implementa un socket en Java **sin seguridad** que se conecta a un servidor remoto con una dirección IP y un puerto. El objetivo del cliente es enviar al servidor un nombre de usuario y una contraseña, y recibir una respuesta del servidor.

```
Microsoft Windows [Versión 10.0.19044.2846]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>cd C:\Users\farme\OneDrive - UNIVERSIDAD DE SEVILLA\Escritorio\ws_Seguridad\PAI1\src\PAI1
C:\Users\farme\OneDrive - UNIVERSIDAD DE SEVILLA\Escritorio\ws_Seguridad\PAI1\src\PAI1>java BYODClienteSinSeg1.java
```

Img 3. Ejecución del programa java del cliente. Conexión con el servidor.



The image shows two separate input dialog boxes. The first is titled 'Enter User Name:' and the second is titled 'Enter Password:'. Both boxes have a green question mark icon on the left, a text input field in the center, and 'OK' and 'Cancel' buttons at the bottom right.

Img 4. Campos para rellenar usuario y contraseña que serán enviados al servidor.

```
at java.base/java.net.Socket.<init>(Socket.java:519)
at java.base/java.net.Socket.<init>(Socket.java:293)
at Pai1.BYODClienteSinSeg1.main(BYODClienteSinSeg1.java:22)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:78)
at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.base/java.lang.reflect.Method.invoke(Method.java:567)
at jdk.compiler/com.sun.tools.javac.launcher.Main.execute(Main.java:415)
at jdk.compiler/com.sun.tools.javac.launcher.Main.run(Main.java:192)
at jdk.compiler/com.sun.tools.javac.launcher.Main.main(Main.java:132)
C:\Users\User\Desktop\ws_scgi\Pai_1\src\Pai1> java BYODClienteSinSeg1.java
C:\Users\User\Desktop\ws_scgi\Pai_1\src\Pai1> java BYODClienteSinSeg1.java
va.net.ConnectException: Connection timed out: connect
at java.base/sun.nio.ch.Net.connect0(Native Method)
at java.base/sun.nio.ch.Net.connect(Net.java:519)
at java.base/sun.nio.ch.Net.connect(Net.java:519)
at java.base/sun.nio.ch.NioSocketImpl.connect(NioSocketImpl.java:150)
at java.base/java.net.SocksSocketImpl.connect(SocksSocketImpl.java:191)
at java.base/java.net.Socket.connect(Socket.java:645)
at java.base/java.net.Socket.connect(Socket.java:595)
at java.base/java.net.Socket.<init>(Socket.java:519)
at java.base/java.net.Socket.<init>(Socket.java:293)
at Pai1.BYODClienteSinSeg1.main(BYODClienteSinSeg1.java:22)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:78)
at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.base/java.lang.reflect.Method.invoke(Method.java:567)
at jdk.compiler/com.sun.tools.javac.launcher.Main.execute(Main.java:415)
at jdk.compiler/com.sun.tools.javac.launcher.Main.run(Main.java:192)
at jdk.compiler/com.sun.tools.javac.launcher.Main.main(Main.java:132)
C:\Users\User\Desktop\ws_scgi\Pai_1\src\Pai1> java BYODClienteSinSeg1.java
```

A small 'Message' dialog box is overlaid on the terminal output, displaying an information icon and the text 'User, ale' with an 'OK' button.

Img 5. Mensaje de respuesta del servidor.

## 2.3 Lado del sniffer

En nuestro trabajo vamos a usar RawCap, un sniffer para Windows que se utiliza desde línea de comandos. Para inspeccionar el archivo de salida de RawCap hemos utilizado Wireshark.

```
C:\Users\farme>RawCap.exe localhost
NETRESEC RawCap version 0.2.1.0

Usage: RawCap.exe [OPTIONS] <interface> <pcap_target>
<interface> can be an interface number or IP address
<pcap_target> can be filename, stdout (-) or named pipe (starting with \\.\pipe\)

OPTIONS:
-f          Flush data to file after each packet (no buffer)
-c <count>  Stop sniffing after receiving <count> packets
-s <sec>    Stop sniffing after <sec> seconds
-m          Disable automatic creation of RawCap firewall entry
-q          Quiet, don't print packet count to standard out

INTERFACES:
0.  IP      : 169.254.163.139
    NIC Name : Ethernet
    NIC Type  : Ethernet

1.  IP      : 25.3.29.221
    NIC Name : Hamachi
    NIC Type  : Ethernet

2.  IP      : 172.23.16.1
    NIC Name : vEthernet (WSL)
    NIC Type  : Ethernet

3.  IP      : 192.168.56.1
    NIC Name : Ethernet 2
    NIC Type  : Ethernet
```

Img 6. Ejecución del programa de sniffer en el Man-in-the-middle. En nuestro caso, es el propio host del servidor.

Administrador: Símbolo del sistema

```
10.  IP      : 192.168.1.134
    NIC Name : Wi-Fi
    NIC Type  : Wireless80211

11.  IP      : 169.254.187.172
    NIC Name : Conexión de red Bluetooth
    NIC Type  : Ethernet

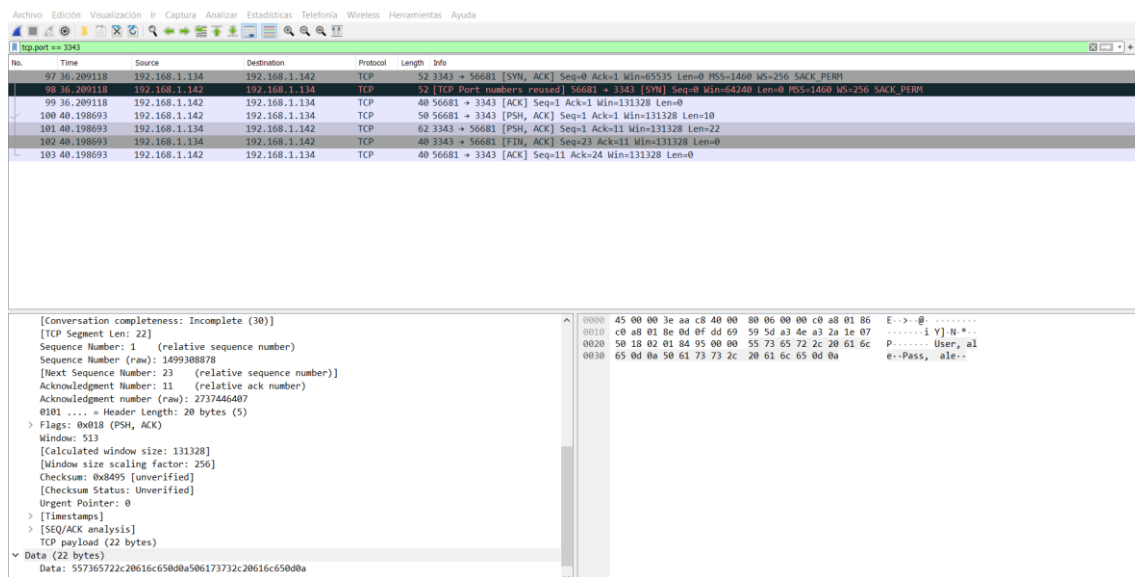
12.  IP      : 127.0.0.1
    NIC Name : Loopback Pseudo-Interface 1
    NIC Type  : Loopback

Example 1: RawCap.exe 0 dumpfile.pcap
Example 2: RawCap.exe -s 60 127.0.0.1 localhost.pcap
Example 3: RawCap.exe 127.0.0.1 \\.\pipe\RawCap
Example 4: RawCap.exe -q 127.0.0.1 - | Wireshark.exe -i - -k

C:\Users\farme>RawCap.exe 10 dumpfile.pcap
Sniffing IP : 192.168.1.134
Output File : C:\Users\farme\dumpfile.pcap
--- Press [Ctrl]+C to stop ---
Packets      : 115^C
C:\Users\farme>
```

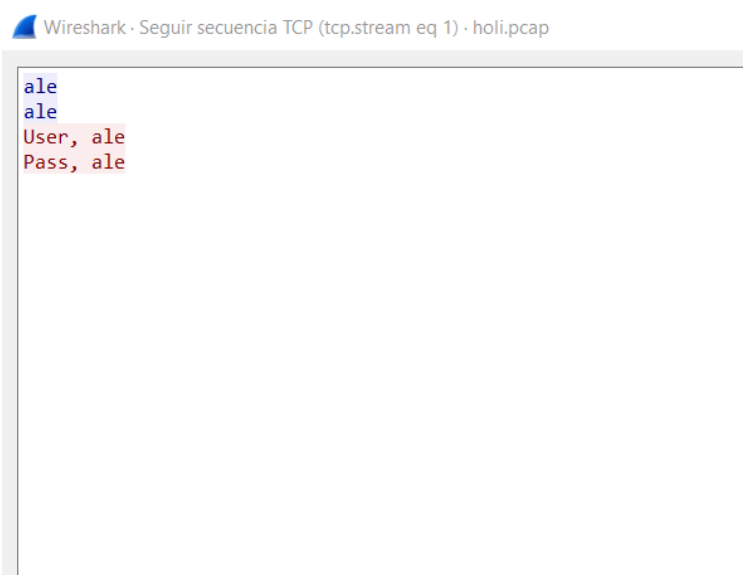
Img 7. RawCap procede a recoger el tráfico de la red indicada.

Una vez capturado el tráfico con RawCap, y obtenido el archivo .pcap se puede abrir con la herramienta Wireshark, donde se podrá visualizar todo el tráfico de la red. Utilizando el filtro tcp.port=3343 podemos ver todos los paquetes transmitidos por ese puerto.



Img 8. Visualizado de paquetes del puerto 3343 desde Wireshark.

Si utilizamos la opción de “Follow TCP Stream” podemos observar todos los datos que se transmiten en la conexión del Socket. Como se puede observar en la *img 9*, el usuario y la contraseña del cliente se transmiten en claro, por lo que el man-in-the-middle podría obtener las credenciales fácilmente. Por lo tanto, se ve la importancia de crear un cliente y un servidor que funcionen con un protocolo seguro.



Img 9. Información dentro de “Follow TCP Stream”.

---

## 3. CREACIÓN DE ALMACENES DE CERTIFICADOS

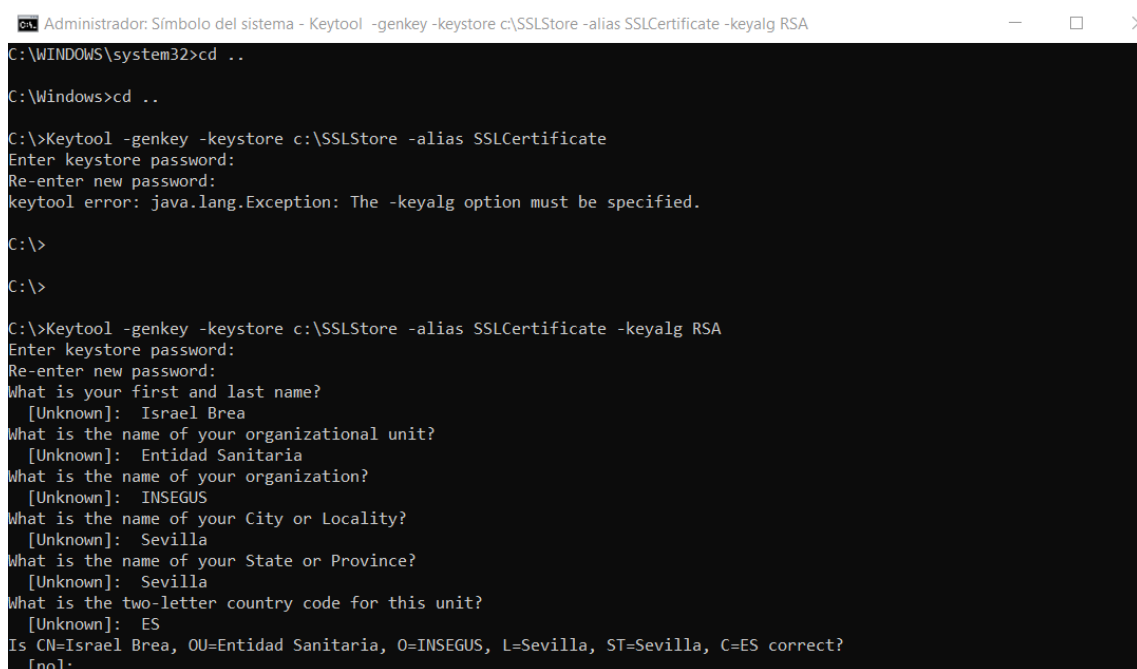
---

Como hemos visto en el apartado anterior, la transmisión mediante sockets seguros es muy importante. Para ello, se han implementado sockets del tipo Secure Sockets Layers (SSL), que utiliza una infraestructura basada en almacenes de claves y certificados, lo que va a asegurar que la información transmitida sea confidencial, íntegra y autenticadas.

### 3.1 Creación de un keystore

En primer lugar, se debemos crear un almacén de certificados(keystore). Que utilizaremos para la autenticación de los servidores de los correspondientes certificados. Crearemos el almacén de certificados desde el terminal y con permisos de administrador.

Keytool hará una serie de preguntas para crear el keyStore. La información solicitada al crear un keystore es crítica para garantizar la autenticidad y validez del certificado. La recolección precisa y completa de esta información es esencial para garantizar que el certificado cumpla con los requisitos necesarios para su uso previsto.



```
Administrador: Símbolo del sistema - Keytool -genkey -keystore c:\SSLStore -alias SSLCertificate -keyalg RSA
C:\WINDOWS\system32>cd ..
C:\Windows>cd ..
C:\>Keytool -genkey -keystore c:\SSLStore -alias SSLCertificate
Enter keystore password:
Re-enter new password:
keytool error: java.lang.Exception: The -keyalg option must be specified.
C:\>
C:\>
C:\>Keytool -genkey -keystore c:\SSLStore -alias SSLCertificate -keyalg RSA
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: Israel Brea
What is the name of your organizational unit?
[Unknown]: Entidad Sanitaria
What is the name of your organization?
[Unknown]: INSEGUS
What is the name of your City or Locality?
[Unknown]: Sevilla
What is the name of your State or Province?
[Unknown]: Sevilla
What is the two-letter country code for this unit?
[Unknown]: ES
Is CN=Israel Brea, OU=Entidad Sanitaria, O=INSEGUS, L=Sevilla, ST=Sevilla, C=ES correct?
[no]:
```

Img 10. Creación de un KeyStore (Almacen de certificados de seguridad para el protocolo SSL).



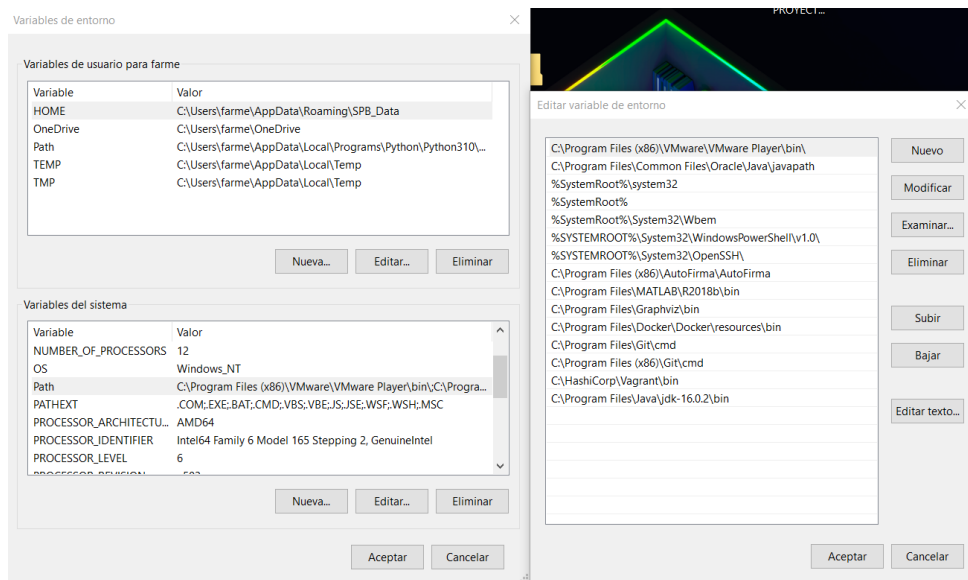
Con esto ya tendremos en la ruta C:\ un archivo SSLStore el almacén de certificados que usaran nuestros Sockets cliente y servidor.

```
Generating 2.048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 90 days
for: CN=Alejandro, OU=Entidad Sanitaria2, O=INSEGUS2, L=Sevilla, ST=Sevilla, C=ES
```

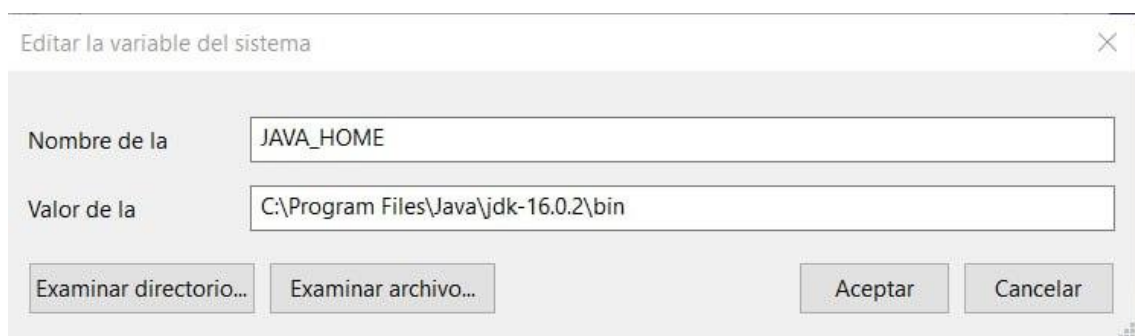
Img 11. Almacén de certificados creado.

### 3.1.1. Configuración del Path y HOME\_PATH

Para usuarios de Windows, keytool se encuentra en la carpeta bin del JDK instalado en el sistema. Por lo tanto, para usar keytool desde línea de comandos sin tener que entrar en la ruta debemos configurar las variables de entorno JAVA\_HOME y PATH con la ruta hacia la carpeta bin de la JDK de Java.



Img 12. Configura la variable de entorno PATH con la ruta hacia la carpeta bin de la JDK de Java.



Img 13. Configura la variable de entorno JAVA\_HOME con la ruta hacia la carpeta bin de la JDK de Java.

### **3.1.1. Información adicional importante**

En nuestro caso, se ha utilizado el mismo keystore tanto para el cliente como para el servidor, entonces ambos tendrán acceso a las mismas claves privadas y certificados. Esto puede ser útil en algunas situaciones, como en pruebas locales o en redes de confianza limitada.

Sin embargo, en situaciones donde se requiere mayor seguridad y confidencialidad, es recomendable que cada máquina tenga su propio keystore. De esta manera, se evita que un atacante pueda comprometer ambas máquinas a través de una sola vulnerabilidad en el keystore compartido.

Si utilizamos dos máquinas distintas, el keystore deberá estar presente en cada una de ellas. Además, es importante tener en cuenta la ubicación y los permisos de archivo del keystore, ya que pueden influir en la seguridad y la accesibilidad de las claves y los certificados.

---

## 4. PRUEBAS DE COMUNICACIÓN CON SEGURIDAD

---

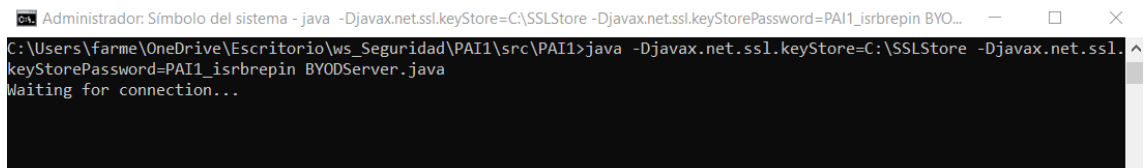
### 4.1 Lado del servidor

El código correspondiente al servidor espera conexiones de clientes y verifica su información de inicio de sesión antes de enviar o recibir cualquier información. Además, utiliza SSL (Secure Socket Layer) para proporcionar integridad y confidencialidad en las comunicaciones, y autenticación mediante certificados digitales.

Tiene implementado un método que toma un nombre de usuario y una contraseña como entrada y verifica si la combinación es correcta, utilizando un diccionario predefinido de nombres de usuario y contraseñas. Devuelve un mensaje de éxito o error según el resultado de la verificación.

Se crea un socket de servidor SSL en el puerto 3343 y se espera a que los clientes se conecten. Cuando un cliente se conecta, se le solicita su información de inicio de sesión (nombre de usuario y contraseña) y se verifica su autenticidad. Si la verificación es correcta, se envía un mensaje de éxito al cliente y se cierra la conexión. Si la verificación falla, se envía un mensaje de error al cliente y se cierra la conexión. El servidor sigue esperando conexiones de clientes en un bucle infinito.

Para que funcione la ejecución del servidor en la línea de comandos es necesario iniciar como administrador.



```
Administrador: Símbolo del sistema - java -Djavax.net.ssl.keyStore=C:\SSLStore -Djavax.net.ssl.keyStorePassword=PAI1_isrbrepin BYO...
C:\Users\farne\OneDrive\Escritorio\ws_Seguridad\PAI1\src\PAI1>java -Djavax.net.ssl.keyStore=C:\SSLStore -Djavax.net.ssl.
keyStorePassword=PAI1_isrbrepin BYODServer.java
Waiting for connection...
```

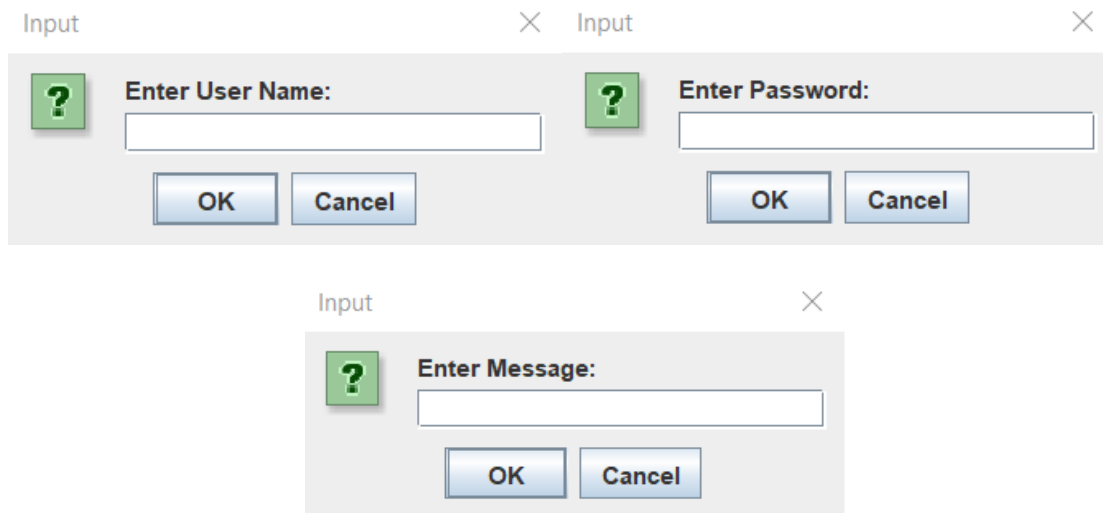
Img 14. Servidor con puerto SSL activado(Protocolo TLS).

## 4.2 Lado del cliente

Este código implementa un cliente de red que establece una conexión SSL con un servidor en la dirección IP "X" y el puerto "3343". El cliente envía al servidor un nombre de usuario, una contraseña y un mensaje, y luego envía esta información al servidor para su autenticación. Luego, espera una respuesta del servidor y muestra la respuesta al usuario en una ventana emergente. Finalmente, cierra las conexiones y sale de la aplicación.

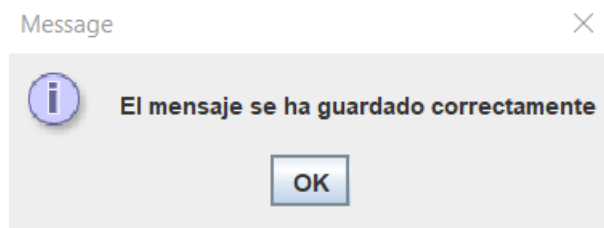
```
C:\Users\farne\OneDrive\Escritorio\ws_Seguridad\PAI1\src\PAI1>java -Djavax.net.ssl.trustStore=C:\SSLStore -Djavax.net.ssl.trustStorePassword=PAI1_isrbrepin BYODCliente.java
```

Img 15. Llamada a cliente con protocolo TLS activado.



The image shows three separate input dialog boxes. The first two are titled 'Input' and contain fields for 'Enter User Name:' and 'Enter Password:' respectively, each with 'OK' and 'Cancel' buttons. The third is also titled 'Input' and contains a field for 'Enter Message:' with 'OK' and 'Cancel' buttons. Each dialog box has a green question mark icon in the top left corner.

Img 16. Campos para rellenar usuario, contraseña y mensaje que serán enviados al servidor.



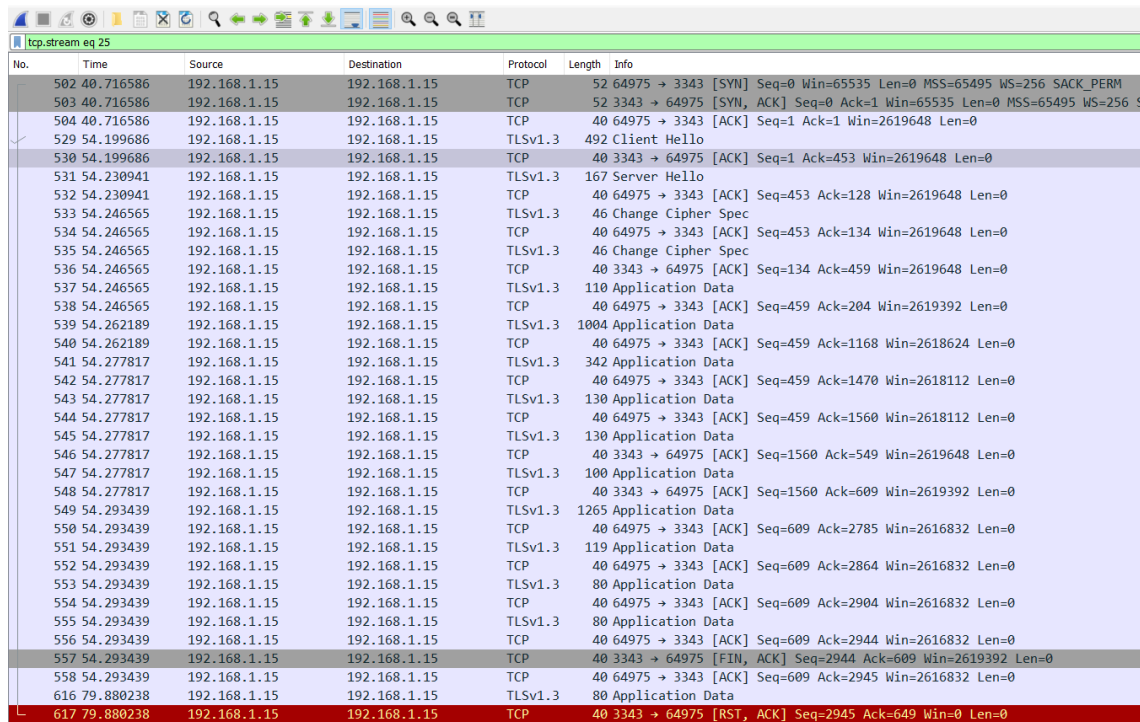
The image shows a message dialog box titled 'Message'. It contains an information icon (i) and the text 'El mensaje se ha guardado correctamente'. There is an 'OK' button at the bottom.

Img 17. Mensaje de respuesta del servidor tras autenticar las credenciales.

## 4.3 Lado del sniffer

Volvemos a realizar el mismo procedimiento para capturar el tráfico en la red. Una vez capturado el tráfico con RawCap, y obtenido el archivo .pcap lo abrimos con Wireshark, donde se podrá visualizar todo el tráfico de la red. Utilizamos de nuevo el filtro tcp.port=3343.

En la *img 18* se puede observar que se está utilizando el **protocolo TLSv1.3** en la transmisión de información en el puerto 3343.




No.	Time	Source	Destination	Protocol	Length	Info
502	40.716586	192.168.1.15	192.168.1.15	TCP	52	64975 → 3343 [SYN] Seq=0 Win=65535 Len=0 MSS=65495 WS=256 SACK_PERM
503	40.716586	192.168.1.15	192.168.1.15	TCP	52	3343 → 64975 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=65495 WS=256 S
504	40.716586	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=1 Ack=1 Win=2619648 Len=0
529	54.199686	192.168.1.15	192.168.1.15	TLSv1.3	492	Client Hello
530	54.199686	192.168.1.15	192.168.1.15	TCP	40	3343 → 64975 [ACK] Seq=1 Ack=453 Win=2619648 Len=0
531	54.230941	192.168.1.15	192.168.1.15	TLSv1.3	167	Server Hello
532	54.230941	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=453 Ack=128 Win=2619648 Len=0
533	54.246565	192.168.1.15	192.168.1.15	TLSv1.3	46	Change Cipher Spec
534	54.246565	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=453 Ack=134 Win=2619648 Len=0
535	54.246565	192.168.1.15	192.168.1.15	TLSv1.3	46	Change Cipher Spec
536	54.246565	192.168.1.15	192.168.1.15	TCP	40	3343 → 64975 [ACK] Seq=134 Ack=459 Win=2619648 Len=0
537	54.246565	192.168.1.15	192.168.1.15	TLSv1.3	110	Application Data
538	54.246565	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=459 Ack=204 Win=2619392 Len=0
539	54.262189	192.168.1.15	192.168.1.15	TLSv1.3	1004	Application Data
540	54.262189	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=459 Ack=1168 Win=2618624 Len=0
541	54.277817	192.168.1.15	192.168.1.15	TLSv1.3	342	Application Data
542	54.277817	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=459 Ack=1470 Win=2618112 Len=0
543	54.277817	192.168.1.15	192.168.1.15	TLSv1.3	130	Application Data
544	54.277817	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=459 Ack=1560 Win=2618112 Len=0
545	54.277817	192.168.1.15	192.168.1.15	TLSv1.3	130	Application Data
546	54.277817	192.168.1.15	192.168.1.15	TCP	40	3343 → 64975 [ACK] Seq=1560 Ack=549 Win=2619648 Len=0
547	54.277817	192.168.1.15	192.168.1.15	TLSv1.3	100	Application Data
548	54.277817	192.168.1.15	192.168.1.15	TCP	40	3343 → 64975 [ACK] Seq=1560 Ack=609 Win=2619392 Len=0
549	54.293439	192.168.1.15	192.168.1.15	TLSv1.3	1265	Application Data
550	54.293439	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=609 Ack=2785 Win=2616832 Len=0
551	54.293439	192.168.1.15	192.168.1.15	TLSv1.3	119	Application Data
552	54.293439	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=609 Ack=2864 Win=2616832 Len=0
553	54.293439	192.168.1.15	192.168.1.15	TLSv1.3	80	Application Data
554	54.293439	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=609 Ack=2904 Win=2616832 Len=0
555	54.293439	192.168.1.15	192.168.1.15	TLSv1.3	80	Application Data
556	54.293439	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=609 Ack=2944 Win=2616832 Len=0
557	54.293439	192.168.1.15	192.168.1.15	TCP	40	3343 → 64975 [FIN, ACK] Seq=2944 Ack=609 Win=2619392 Len=0
558	54.293439	192.168.1.15	192.168.1.15	TCP	40	64975 → 3343 [ACK] Seq=609 Ack=2945 Win=2616832 Len=0
616	79.880238	192.168.1.15	192.168.1.15	TLSv1.3	80	Application Data
617	79.880238	192.168.1.15	192.168.1.15	TCP	40	3343 → 64975 [RST, ACK] Seq=2945 Ack=649 Win=0 Len=0

Img 18. Tráfico del puerto 3343 capturado por Wireshark.

Dentro de Client Hello podemos observar mucha información relevante, entre ellas podemos ver aquellos Cipher Suites que va a permitir el cliente. Dentro de Server Hello se observa cuál va a utilizar el servidor para la transmisión.

Si quisiéramos cambiar el Cipher Suite utilizado en la comunicación. Bastaría con configurarlo en la aplicación servidora o cliente o incluso en ambas. En general, en la realidad como no disponemos del código fuente ni tenemos los permisos para configurar los servidores donde corren las aplicaciones servidoras a las cuales nos conectamos. Lo configuramos en una aplicación cliente (que corre en nuestra máquina) como por ejemplo es el navegador. Aunque en este caso, como ambas aplicaciones corren en nuestra máquina y podemos configurarlas, podríamos modificarla en una cualquiera o en ambas.



**Cipher Suites (in order of preference)**

TLS_GREASE_DA (0xdada)	-
TLS_AES_128_GCM_SHA256 (0x1301) Forward Secrecy	128
TLS_AES_256_GCM_SHA384 (0x1302) Forward Secrecy	256
TLS_CHACHA20_POLY1305_SHA256 (0x1303) Forward Secrecy	256
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b) Forward Secrecy	128
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f) Forward Secrecy	128
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c) Forward Secrecy	256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030) Forward Secrecy	256
TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9) Forward Secrecy	256
TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xc0a8) Forward Secrecy	256
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013) <b>WEAK</b>	128
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014) <b>WEAK</b>	256
TLS_RSA_WITH_AES_128_GCM_SHA256 (0x9c) <b>WEAK</b>	128
TLS_RSA_WITH_AES_256_GCM_SHA384 (0x9d) <b>WEAK</b>	256
TLS_RSA_WITH_AES_128_CBC_SHA (0x2f) <b>WEAK</b>	128
TLS_RSA_WITH_AES_256_CBC_SHA (0x35) <b>WEAK</b>	256

(1) When a browser supports SSL 2, its SSL 2-only suites are shown only on the very first connection to this site. To see the suites, close all browser windows, then open this exact page directly. Don't refresh.

Img 19. Cipher suites aceptados por nuestro navegador y seguridad de estos. Qualys SSL Labs

```

> Frame 49: 492 bytes on wire (3936 bits), 492 bytes captured (3936 bits)
Raw packet data
> Internet Protocol Version 4, Src: 192.168.1.142, Dst: 192.168.1.134
> Transmission Control Protocol, Src Port: 52445, Dst Port: 3343, Seq: 1, Ack: 1, Len: 452
√ Transport Layer Security
  √ TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 447
  √ Handshake Protocol: Client Hello
    Handshake Type: Client Hello (1)
    Length: 443
    Version: TLS 1.2 (0x0303)
    Random: f1133dd03fbae44850b9364aa494215ee184967c64e0141f80b3999f121cd27f
    Session ID Length: 32
    Session ID: 67de3ca0d694e6ad444decdec9ed2dd1b55a2bbcdfdd128e905f9396c72e1f6b
    Cipher Suites Length: 98
  √ Cipher Suites (49 suites)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_DHE_RSA_WITH_AES_256_GCM_SHA384 (0x009f)

```

Img 20. Lista de Cipher Suites admitidas por el cliente.

```

> Frame 50: 167 bytes on wire (1336 bits), 167 bytes captured (1336 bits)
Raw packet data
> Internet Protocol Version 4, Src: 192.168.1.134, Dst: 192.168.1.142
> Transmission Control Protocol, Src Port: 3343, Dst Port: 52445, Seq: 1, Ack: 453, Len: 127
√ Transport Layer Security
  √ TLSv1.3 Record Layer: Handshake Protocol: Server Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 122
  √ Handshake Protocol: Server Hello
    Handshake Type: Server Hello (2)
    Length: 118
    Version: TLS 1.2 (0x0303)
    Random: d95df4e559df3f979667dc9679ac51fdb856e4e835a019e0f1b3950a835d7098
    Session ID Length: 32
    Session ID: 67de3ca0d694e6ad444decdec9ed2dd1b55a2bbcdfdd128e905f9396c72e1f6b
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Compression Method: null (0)
    Extensions Length: 46
  > Extension: supported_versions (len=2)
  > Extension: key_share (len=36)
    [JA3S Fullstring: 771,4866,43-51]
    [JA3S: 15af977ce25de452b96affa2addb1036]

```

Img 21. Cipher Suite utilizado por el servidor.

Si utilizamos la opción de “Follow TCP Stream” como hicimos anteriormente, podemos observar todos los datos que se transmiten en la conexión del Socket. Como se puede observar en la *img 20*, el usuario, la contraseña y el mensaje se transmiten encriptados, por lo que el man-in-the-middle cuando acceda, nada más encontrará basura encriptada. Por lo tanto, se ve la importancia de crear un cliente y un servidor que funcionen con un protocolo seguro.

```
.&$.
.....2.&$.
.....#.....3.k.i...1j.2.....X.h^...
.H...A$.A.t>...c.Y...2>...S...<t.9...+l5...<j..w.5..4rS...v.T...Y...z...v..R.....k
j...G?6.R...!|...n$.d...Z...M..b..I.Z..V.]...^.....+.....3.$...
uG.N...v...i...I%...d1...A...a.p...e.8.[5.b"A.*X...W.....Xz=2...[:R...l...s...5.l.....S...`.)Z
6.0..&Kv...jt...l.n.W.6x...I...n.v.f0.P\D...[Q.n5k...[...p%08)...y.FGQj...K.3..X.n/...@.7...S...
7~...Z...k...k.E3.2gk~...N.j...$.zRu...L2.S.k.';T(.d...#...v...x...[!s/.m..._{.x.0...
V...e..u...>...a.K"\xuEj}...@)?...T...s)2...+Sh>...%K.....Kb.s...).i.9...s{.Q.l~>...a...8B.k$ea...jk...?k.,4
3=..Jc:v..9MF.7.7gz...A[...x...x..h[XT_v..tp.....p....]u.#...A.....;m.f...-g1.x.PJ.mj..f=...2-..rSqn.o...
...b{02.J}...BH.5.....%.3.dSW
..\...H...l.oA...u..W.....].r.0..].0..f...5.J...r..k.Us....._J0../.W.D..?&vc...;.>*.1.U.z...&=
.^...e0.....z...}&{.....L...<
}.....1...9...m..I...w...LE.s.....to'.C2~...).y...a...r...}.N...K.....:SD.k.....
2#.m3...D@pB...R..S.W...J...p.x'.J...K.*.X...;t.a...B...'.]...`b.bb.../If9..p.to..LX....?<...JI...&.....7...=B.$..8.
%$#...3g>..u.j..S...}.Eaq.f........0(i.....)C5.....!UN...{..9.T.o.w.....!.....
SZ~jc?/.5.....z.....t.....
.D.R2.*.....@a!..o%.G;z...j..".w.....pg..Lv...s..0.
.h.....L..$...k..0.=."N.a...m.W...#.)06..ua.ml...D.y@..b..I...[.0...7@..\j...],...m.Nt..../.0...9..."td.7..?2.
....R..3...2#.5.5i..G...d.[...h.....x*.#...U)wp...A...X...r..W.h...z/...<
.c..R..phmh`i>...u...^_@~...v.S..X...D...+K..v.....U)...T..6.....)S0...b`..;^[...e1r...$.&.o]{&...<...h<...l.D5qu...k...
6..o...E):...7.j~.wW@...9].....".TO#w.....]...GL...U.p.(..V..W.....?.+n.
(. ....b.....I]H~...#9..l5...bokg*...M
{..(.9...9.T#=L...~GS.E.H+...l.V...+RuW...E...3.@...Js...17...&..q.Jb.....a...j...tS.
..TH.....e./\2.]~.#hgkK..0..._Z4V...x...7.....}.Uf...i..C.pl...ac2.P...Xx>Y.k.....b%...k...j...f....;T+...}.d,
4.Rw..._g..'.Z..p..j...
..<W.S...*.x..!..3K...j...V.6G..
..d8...I.....>..U...}.Ab...V1r...q,...:..@..6.CR...9U.*J
B0%.....Y.....iI1.t
lfs..xS...$.o.6f.G@8.....C.4.)I....{>..8...U...~.pn...E...s.....R.I...
...21^#. \...y.....;awt...01p..Drz.F..Q...):...v.o.....L.....U.....s.}IZ...[E]...d...k.....6.h.K.t.)..7..
(=..8C...0..g..).`.....Q.R.A...I...a...QIF...&%t..B...l].....]d...}m..
...6...@.F.*;...v"...Az~...c...:'.[.kU..w8.t..0..o.....).....IN7...3.....n.@..A...r...w78.I.4...~...KsQ...V.
{.Y]^....
.0.g.A6...j&6...%Yw0...-...X..|m...uUI..}1.....2..V...`q..I...B...Br4.G...v/j.K.....^l...F.l... ..z...J]....1...$.6
..WS...8f...9.wt.dg..d4...h...UQ...OUL.V.....{...0..kz0X.1.*...;R..\4@...l...bs.
..o.0...l3.9x...FH...Fu.....E...
.65..7.....]a{.0.S..H\...9.U...D.....I~.....-p...r;...t...y
c.a...y...].J.U.g..K...7...0>=.\{.....c.A.+~^D..T..$.W.t...Soz..&../...$`...-Vc...#..au1...#..V..cM..l..
3.....^N9l...#C.
.p...[.lDWB...vm12.S...s4..P.....#..Z
```

Img 22. Información dentro de “Follow TCP Stream” con protocolo TLS.



---

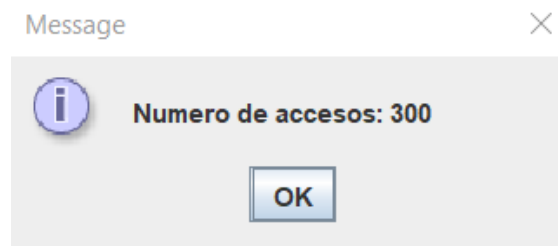
# PRUEBA DE CAPACIDAD

---

Finalmente, queremos comprobar que la conexión es capaz de soportar al menos los 300 clientes de forma concurrente. La manera ideal y más realista de hacer esto hubiera sido mediante la creación de múltiples hilos para manejar varias conexiones de clientes simultáneamente (en paralelo). Sin embargo, en nuestro caso, entra un cliente cada segundo (en serie).

En el código del cliente se crean 300 conexiones de cliente SSL con el servidor consecutivas, una cada segundo. El servidor tiene implementado un contador que dirá el número de clientes que han accedido.

Recaltar, que implementamos en el código fuente de la app cliente la espera entre un cliente y otro. Porque sin ella, no daba a tiempo a que la app servidora cerrase y abriera el socket en cada iteración, lo que hacía que en la siguiente iteración de la nueva petición del cliente notificara que el socket estuviera cerrado.



*Img 23. Mensaje que el servido le manda al cliente con el número de accesos que le han llegado.*

Como podemos observar en la *img 23* el servidor es capaz de soportar 300 clientes de manera consecutiva.

---

# CÓDIGO FUENTE APLICACIONES JAVA CLIENTE - SERVIDOR

---

## 6.1 BYODClienteSinSeguridad

```
package PAI1;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;
import javax.swing.JOptionPane;

public class ClientSocket {

    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {
        try {

            // create Socket from factory --> crea el socket
            Socket socket = new Socket("192.168.1.15", 3343);

            // create PrintWriter for sending login to server -->imprime (es
            lo que se va a mandar) en la red el nombre de user y contrase a
            PrintWriter output = new PrintWriter(new OutputStreamWriter(
                socket.getOutputStream()));
            // prompt user for user name
            String userName = JOptionPane.showInputDialog(null,
                "Enter User Name:");

            // send user name to server
            output.println(userName);

            // prompt user for password
            String password = JOptionPane.showInputDialog(null,
                "Enter Password:");

            // send password to server
            output.println(password);

            output.flush(); //lanzarlo

            // create BufferedReader for reading server response-->asegurar
            la integridad para que se forme una cola de mensajes
            //los mensajes se van encolando en ella
```

```

        BufferedReader input = new BufferedReader(new InputStreamReader(
            socket.getInputStream()));

        // read response from server-->cuando viene la respuesta del
server, lee lo que entra y la mete en una cadena(response)

        String response = input.readLine();

        // display response to user

        JOptionPane.showMessageDialog(null, response);

        // clean up streams and Socket
        output.close();
        input.close();
        socket.close();

    } // end try

    // handle exception communicating with server
    catch (IOException ioException) {
        ioException.printStackTrace();
    }

    // exit application
    finally {
        System.exit(0);
    }
}
}

```

## 6.2 BYODServerSinSeguridad

```

package PAI1;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

import javax.net.SocketFactory;

public class BYODServerSinSeguridad {

    /**
     * @param args
     * @throws IOException
     * @throws InterruptedException
     */
}

```

```

    */
    public static void main(String[] args) throws IOException,
        InterruptedException {

        // perpetually listen for clients
        ServerSocket sSocket = new ServerSocket(3343);
        while (true) {

            // wait for client connection and check login information
            try {
                System.err.println("Waiting for connection...");

                Socket socket = sSocket.accept();

                // open BufferedReader for reading data from client
                BufferedReader input = new BufferedReader(new
                    InputStreamReader(socket.getInputStream()));

                // open PrintWriter for writing data to client
                PrintWriter output = new PrintWriter(new
                    OutputStreamWriter(socket.getOutputStream()));
                String userName = input.readLine();
                String password = input.readLine();
                output.println("User, " + userName);
                output.println("Pass, " + password);

                output.close();
                input.close();
                socket.close();

            } // end try

            // handle exception communicating with client
            catch (IOException ioException) {
                ioException.printStackTrace();
            }

        } // end while
    }
}

```

## 6.3 BYODCliente300Users

```

package PAI1;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.Socket;

```

```

import javax.net.ssl.SSLSocket;
import javax.net.ssl.SSLSocketFactory;
import javax.swing.JOptionPane;

public class BYODCliente300Users {

    /**
     * @param args
     * @throws IOException
     */
    public static void main(String[] args) throws IOException {
        try {
            // create Socket from factory
            SSLSocketFactory socketFactory = (SSLSocketFactory)
SSLSocketFactory.getDefault();
            String serverName = "192.168.1.15";
            int serverPort = 3343;
            String userName, password, message;
            PrintWriter output;
            BufferedReader input;

            userName = JOptionPane.showInputDialog(null, "Enter User Name:");

            // prompt user for password
            password = JOptionPane.showInputDialog(null, "Enter Password:");

            // prompt user for message
            message = JOptionPane.showInputDialog(null, "Enter Message:");
            // repeat the connection process 300 times
            for (int i = 1; i <= 300; i++) {
                // create socket
                SSLSocket socket = (SSLSocket)
socketFactory.createSocket(serverName, serverPort);

                // create PrintWriter for sending login to server
                output = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));

                // send user name, password, and message to server
                output.println(userName);
                output.println(password);
                output.println(message);
                output.flush(); // send the data to the server

                // read response from server
                input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                String response = input.readLine();

                // display response to user
                JOptionPane.showMessageDialog(null, response);

                // close streams and socket
                output.close();
                input.close();
                socket.close();

                // pause for a second between connections

```

```

        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
} catch (IOException ioException) {
    ioException.printStackTrace();
} finally {
    System.exit(0);
}
}
}

```

## 6.4 BYODServer300Users

```

package PAI1;

import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.KeyManagementException;
import java.security.KeyStore;
import java.security.KeyStoreException;
import java.security.NoSuchAlgorithmException;
import java.security.UnrecoverableKeyException;
import java.security.cert.CertificateException;
import java.security.cert.CertificateFactory;
import java.security.cert.X509Certificate;
import java.util.HashMap;

import javax.net.SocketFactory;
import javax.net.ssl.KeyManager;
import javax.net.ssl.KeyManagerFactory;
import javax.net.ssl.SSLContext;
import javax.net.ssl.SSLServerSocket;
import javax.net.ssl.SSLServerSocketFactory;

/*
Cuando abramos el programa estamos abriendo una puerta (y si estamos
conectados a la red)

PARA DARLE SEGURIDAD--> serversocket-->sslserversocket (una nueva clase)(se
consigue la integridad y la confidencialidad )

```

Falta la autenticidad--> el servidor necesita un certificado digital--  
>keytools  
El cliente debe de confiar en el keystore del server

Para cerrar esto cerra el proceso del S0perativo

\*/

```
public class BYODServer300Users {
    public static String auxiliar(String clave, String password) {
        String res="Si sale esto WTF";
        // Crear un nuevo HashMap
        HashMap<String, String> diccionario = new HashMap<>();

        // Agregar los pares clave-valor
        diccionario.put("user1", "clave1");
        diccionario.put("user2", "clave2");
        diccionario.put("user3", "clave3");
        diccionario.put("user4", "clave4");
        diccionario.put("user5", "clave5");

        // Comprobar si la clave existe y la contraseña es correcta
        if (diccionario.containsKey(clave)) {
            if (diccionario.get(clave).equals(password)) {
                res="El mensaje se ha guardado correctamente";
            } else {
                res="Contraseña incorrecta";
            }
        } else {
            res="No existe ese user";
        }
        return res;
    }

    /**
     * @param args
     * @throws IOException
     * @throws InterruptedException
     * @throws KeyStoreException
     * @throws CertificateException
     * @throws NoSuchAlgorithmException
     * @throws UnrecoverableKeyException
     * @throws KeyManagementException
     */
    public static void main(String[] args) throws IOException,
        InterruptedException, KeyStoreException, CertificateException,
        NoSuchAlgorithmException, UnrecoverableKeyException, KeyManagementException {
        SSLServerSocketFactory socketFactory = (SSLServerSocketFactory)
        SSLServerSocketFactory.getDefault();
        SSLServerSocket sSocket = (SSLServerSocket)
        socketFactory.createServerSocket(3343);
        int counter = 0;
        while (true) {
            try {
                System.err.println("Waiting for connection...");
                Socket socket = sSocket.accept();
            }
        }
    }
}
```

```

        counter++;
        BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
        PrintWriter output = new PrintWriter(new
OutputStreamWriter(socket.getOutputStream()));
        String userName = input.readLine();
        String password = input.readLine();
        String mensaje = input.readLine();
        output.println("Numero de accesos: " + counter);
        output.close();
        input.close();
        socket.close();
    } catch (IOException ioException) {
        ioException.printStackTrace();
    }
}
}
}
}

```