INGENIERÍA DE SEGURIDAD US
Avda Reina Mercedes s/n. 41012 Sevilla

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

Araceli María Benítez Díaz

Paula Poley Ceballos

SCG, 3º ISA 2022

ÍNDICE

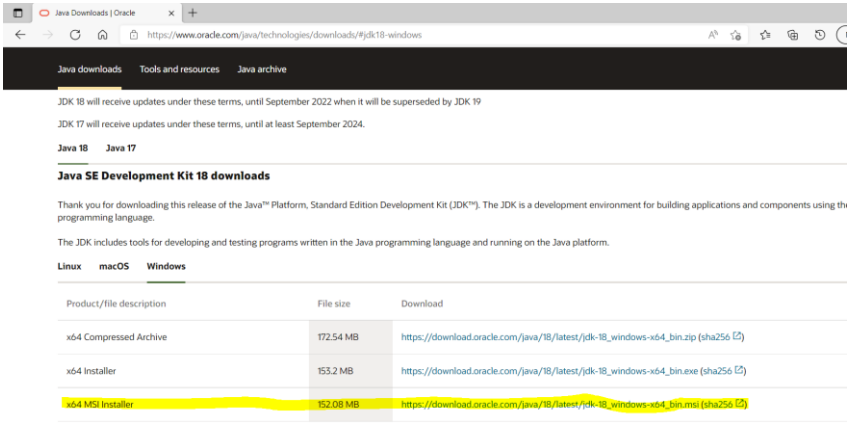
Vamos a comprobar login/Password de usuarios y un mensaje secreto con sockets seguros creando una infraestructura cliente-servidor. Hemos seguido una serie de pasos para crear la arquitectura segura.

Como Security Team elegimos usar el lenguaje Java para realizar el diseño.

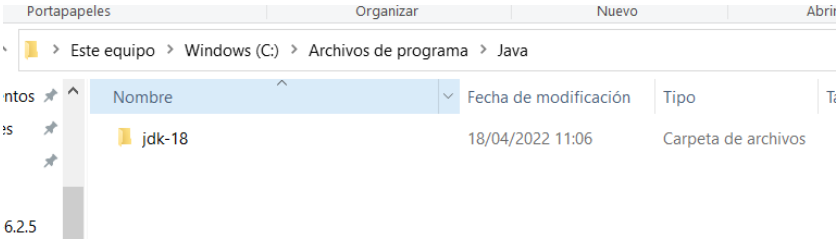
Para empezar:

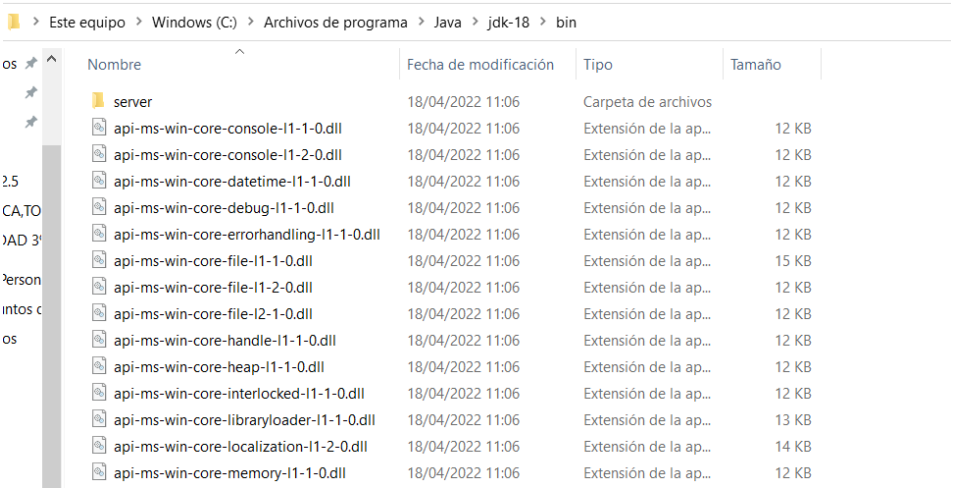
- ***Cambiamos el path de cmd***

1. Nos descargamos e instalamos jdk en la página de Oracle.

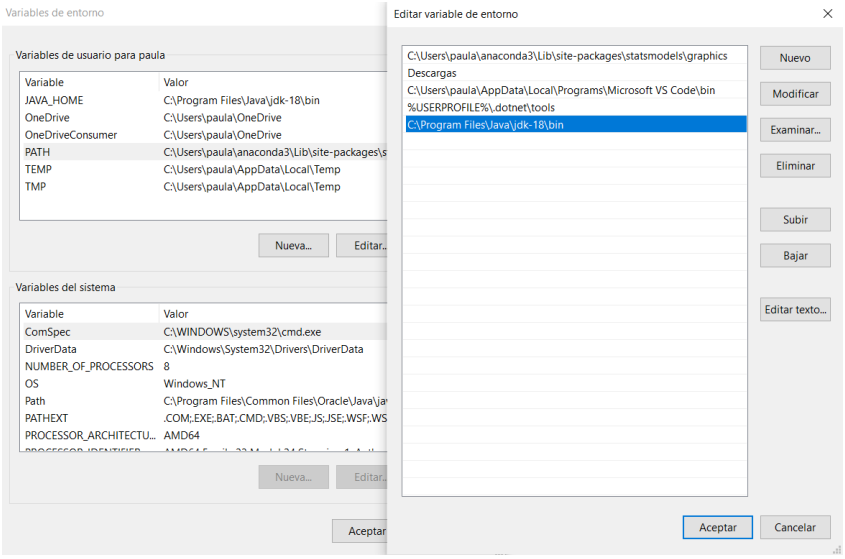


2. Buscamos la carpeta jdk-18 en nuestro equipo y dentro de ella la carpeta bin. Como podemos ver a continuación ya tenemos la ruta dicha carpeta.

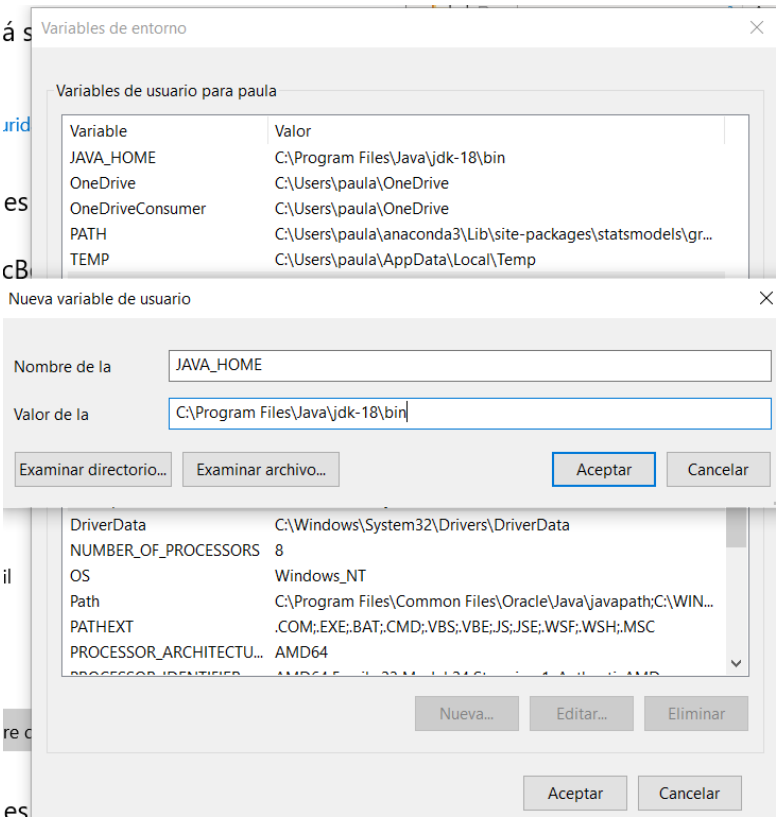




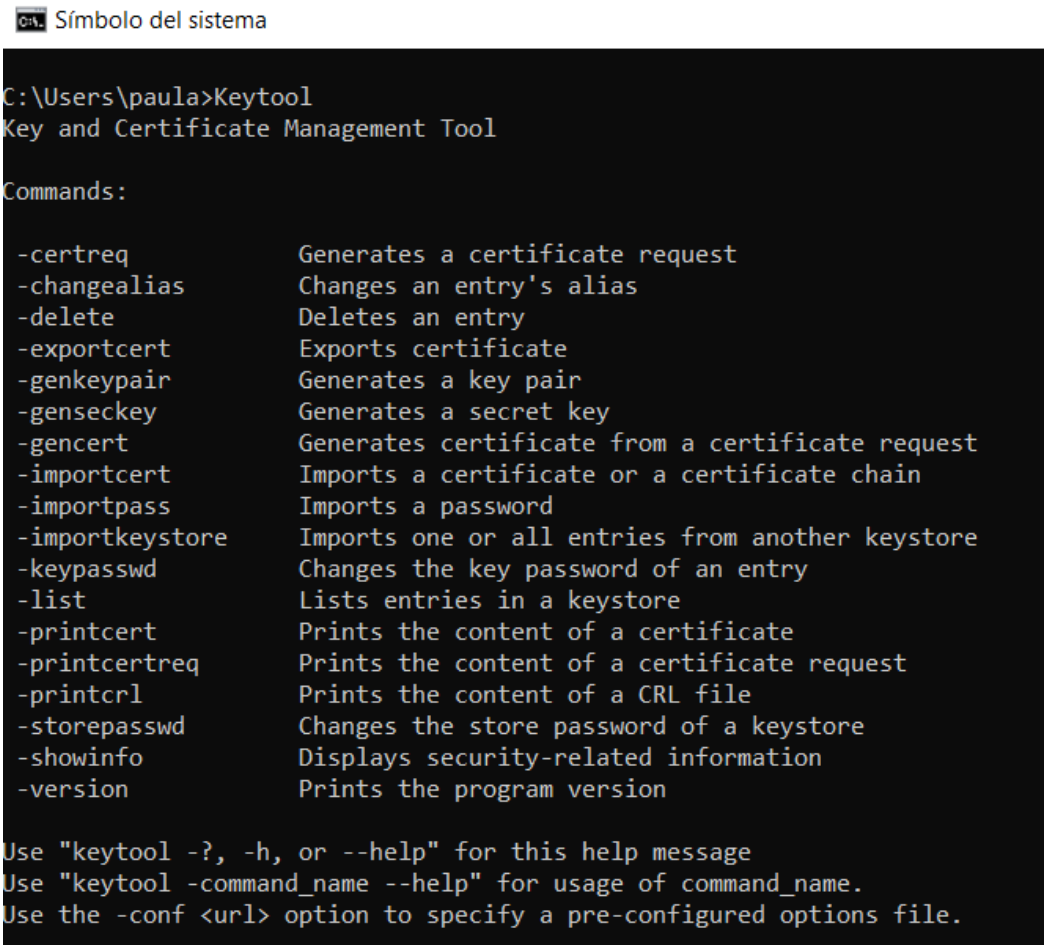
3. Una vez copiada la ruta de la carpeta bin cambiamos las propiedades de nuestro equipo. Buscamos en la configuración del equipo, editar las variables de entorno de esta cuenta . Seleccionamos PATH y lo editamos añadiendo la ruta de la carpeta y aceptamos.



Por último le damos a nueva y ponemos el nombre de la variable JAVA_HOME y el valor de ella que es la ruta de la carpeta bin.

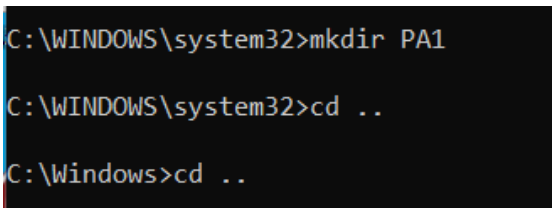


4. Una vez hecho esto abrimos cmd y ponemos Keytool para ver si ya funciona. En la figura siguiente comprobamos que funciona correctamente.



5. Creamos un directorio llamado PA1 con el comando → mkdir PA1

6. Salimos con el comando cd .. que lo escribiremos dos veces.



Atención: todo esto lo estamos creando en system32 y se creará como un archivo fuera pero queremos que se cree en una carpeta. Esto lo podemos hacer de dos maneras, de forma manual creando la carpeta o poniendo mkdir c:\PA1 . Llamando a la carpeta PA1.

- ***<u>Creamos un KeyStore (Almacén de claves)</u>***

Creamos un KeyStore (repositorio de certificados de seguridad) para autentificar los servidores de los correspondientes certificados según el protocolo TLS. Los pasos que seguiremos serán:

1. Escribimos Keytool –genkey –keystore c:\SSLStore -keyalg RSA –alias SSLCertificate
   Atención.
   Sustituimos SSLStore por nuestro directorio creado, es decir, PA1 y al lado ponemos -keyalg RSA porque trabajamos con ello.
   A continuación nos pregunta una serie de preguntas que contestaremos o podremos obviarlas.

```
C:\>Keytool -genkey -keystore c:\PA1 -keyalg RSA -alias SSLCertificate
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [no]:  paula
What is the name of your organizational unit?
  [us]:
What is the name of your organization?
  [universidad]:
What is the name of your City or Locality?
  [sevilla]:
What is the name of your State or Province?
  [es]:
What is the two-letter country code for this unit?
  [es]:
Is CN=paula, OU=us, O=universidad, L=sevilla, ST=es, C=es correct?
  [no]:  yes

Generating 2.048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 90 days
        for: CN=paula, OU=us, O=universidad, L=sevilla, ST=es, C=es
```

   Podemos apreciar que la última linea nos dice que se ha generado correctamente y tendrá 90 días de validación. Toda esta información es para crear el certificado. Nos pide todo esa información para identificar la entidad que va a utilizarlo.

2. Abrimos Virtual Studio y le damos a open folder y abrimos nuestro store .
   Anteriormente hemos metido los dos socket proporcionados por el profesor. (ClientSocket.java y SSocket.java)

3. Ahora tenemos dos formas de inicializarlo.
    a. Desde visual studio
    En el servidor le damos a botón derecho y a run java.
    En la pestañan inferior nos sale Waiting for connection...

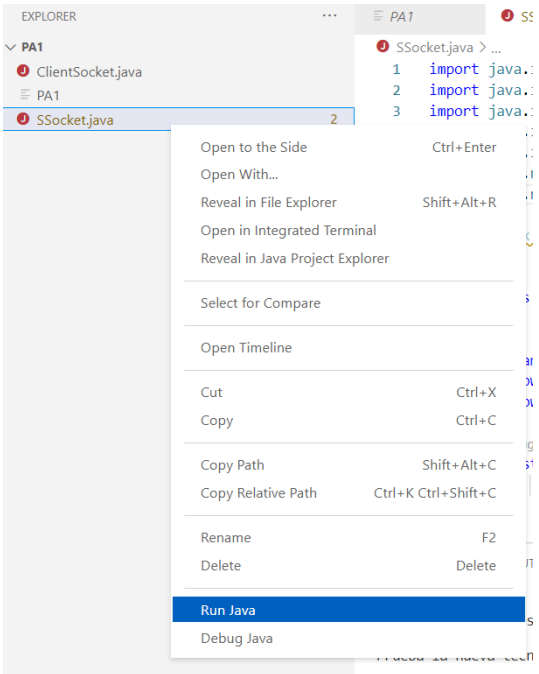
¿Qué es lo que está sucediendo?.

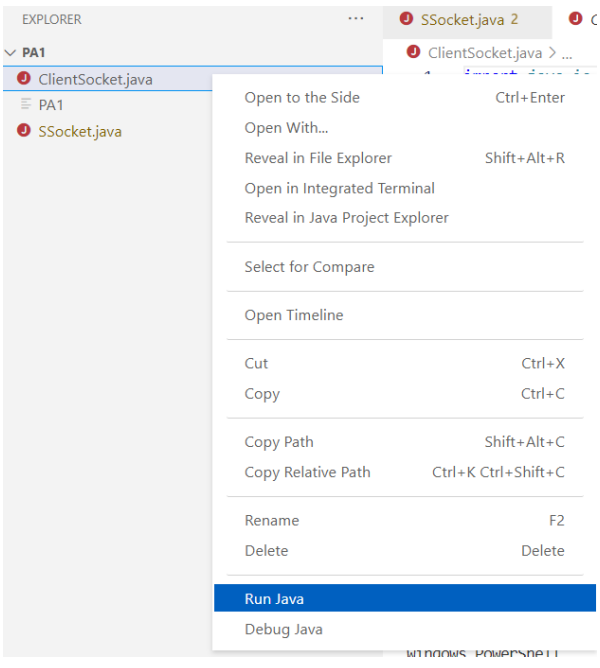
-De manera gráfica el servidor está detrás de una puerta esperando a que alguien le llame, está en espera.

Hacemos lo mismo con el cliente.

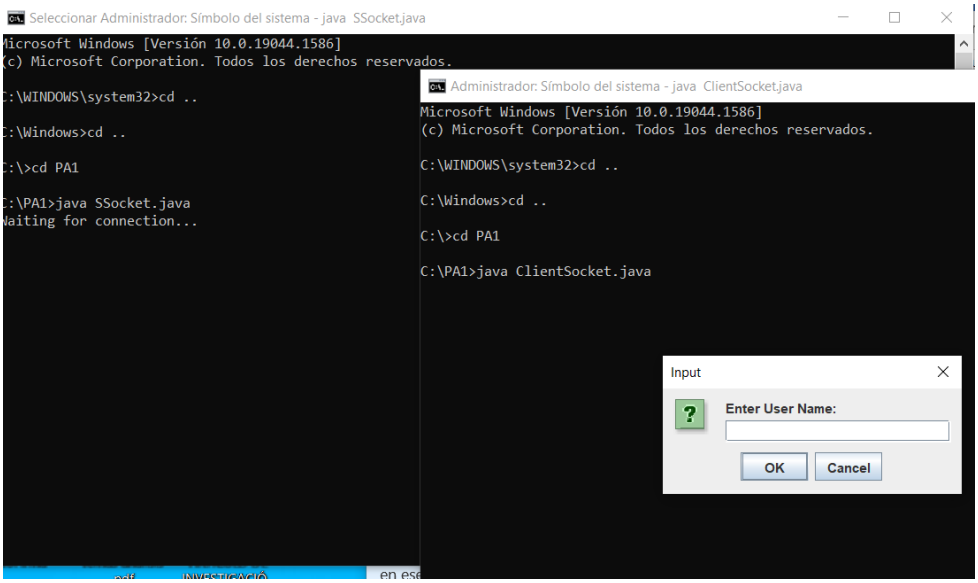
¿Y ahora que ocurre?.

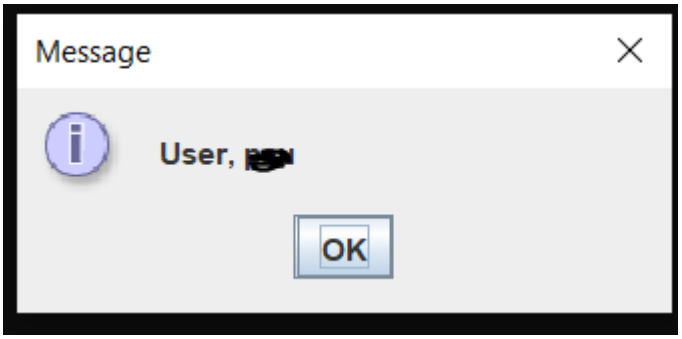
-El cliente llama a la puerta, el servidor la abre y pueden mantener una comunicación entre ellos.





b. Desde cmd .
Entramos en nuestra carpeta PA1, ejecutamos primero el servidor y después el cliente.
Nos sale satisfactorio ya que nos pregunta el usuario y la contraseña.





- ***Generación de un Socket TLS Servidor y un Socket TLS Cliente***

Con Socket SSL/TLS conseguiremos usar comunicaciones seguras entre cliente-servidor.

1. Entramos en el servidor y cambiamos lo subrayado por:
   SSLServerSocketFactory socketFactory = (SSLServerSocketFactory)
   SSLServerSocketFactory.getDefault();
   serverSocket = (SSLServerSocket) socketFactory.createServerSocket(7070);

SSocket.java 2 | ClientSocket.java

SSocket.java > SSocket > main(String[])

```
15      * @param args
16      * @throws IOException
17      * @throws InterruptedException
18      */
        Run | Debug
19      public static void main(String[] args) throws IOException,
20                                                     InterruptedException {
21
22
23          // perpetually listen for clients
24          ServerSocket sSocket = new ServerSocket( port: 3343);
25          while (true) {
26
27              // wait for client connection and check login information
28              try {
29              System.err.println( x: "Waiting for connection...");
30
31              Socket socket = sSocket.accept();
32
33              // open BufferedReader for reading data from client
34              BufferedReader input = new BufferedReader(new
35                                                        InputStreamReader(socket.getInputStream()));
36
37              // open PrintWriter for writing data to client
38              PrintWriter output = new PrintWriter(new
39                                                   OutputStreamWriter(socket.getOutputStream()));
40              String userName = input.readLine();
```

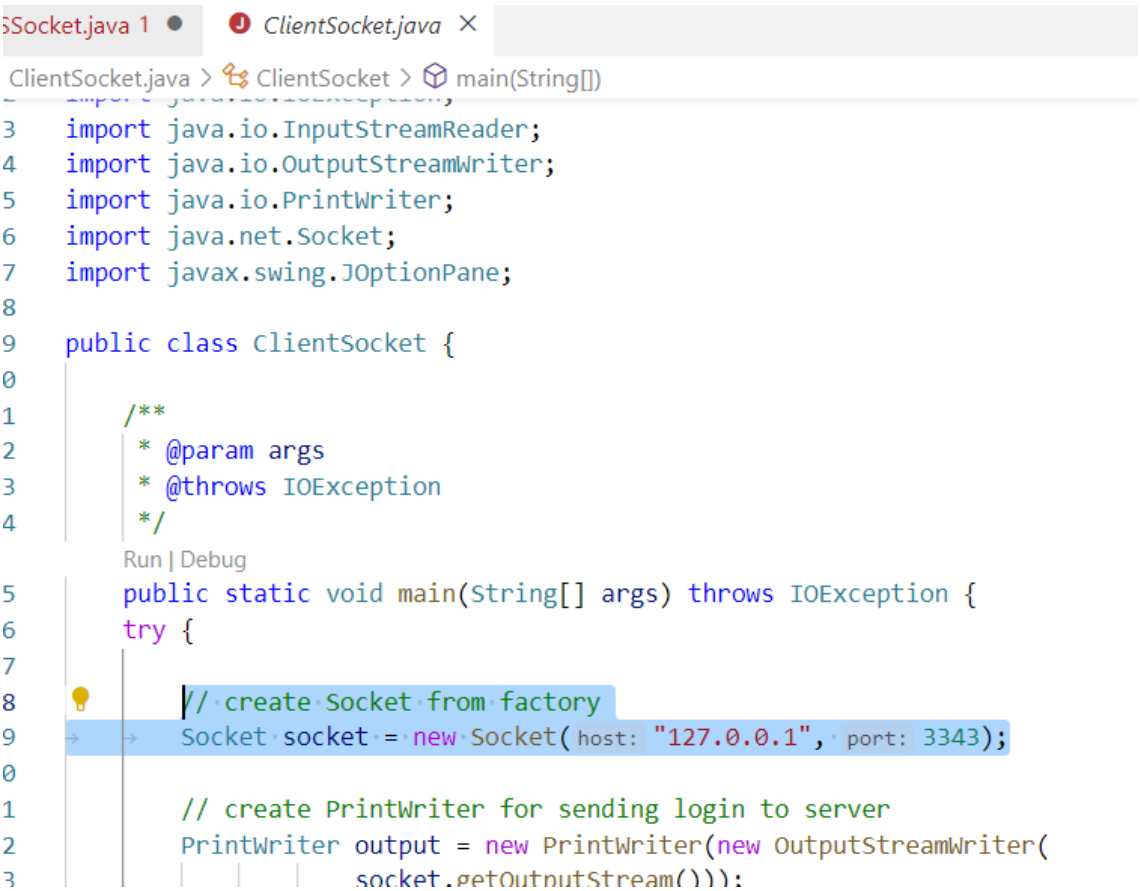
```
// perpetually listen for clients
SSLServerSocketFactory socketFactory = (SSLServerSocketFactory)
SSLServerSocketFactory.getDefault();
serverSocket = (SSLServerSocket) socketFactory.createServerSocket(7070);
```

¿Qué ocurre? .Hay algunos fallos que habrá que cambiar.

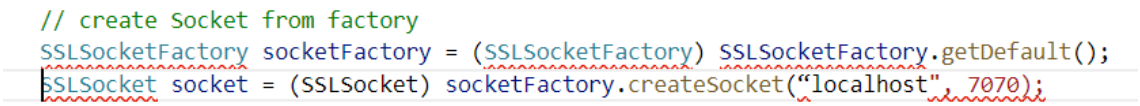
Primero importamos SSLServerSocketFactory y SSLServerSocket . Luego creamos una variable local y cambiamos serverSocket por sSocket que es como inicialmente lo tenía escrito el profesor. En la captura siguiente vemos que se ha cambiado todo correctamente y no hay ningún error.

```
// perpetually listen for clients
SSLServerSocketFactory socketFactory = (SSLServerSocketFactory)
SSLServerSocketFactory.getDefault();
SSLServerSocket sSocket = (SSLServerSocket) socketFactory.createServerSocket(7070);
```

2. Cambiamos los subrayado también en el cliente por:
SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
SSLSocket socket = (SSLSocket) socketFactory.createSocket("localhost", 7070);



Nos sale también errores, lo resolvemos importando SSLSocket y SSLSocketFactory



```
// create Socket from factory
SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
SSLSocket socket = (SSLSocket) socketFactory.createSocket("localhost", 7070);
```

3. Una vez que ya hemos cambiado el servidor y el cliente lo inicializamos. Abrimos una ventana cmd para el servidor y pondremos:

```
C:\PA1>
C:\PA1>java SSocket.java
Waiting for connection...
```

Y otra ventana para el cliente:

```
Administrador: Símbolo del sistema
Microsoft Windows [Versión 10.0.19044.1586]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\WINDOWS\system32>cd ..

C:\Windows>cd ..

C:\>cd PA1

C:\PA1>java ClientSocket.java
java.net.SocketException: Socket is closed
        at java.base/sun.security.ssl.SSLSocketImpl.getInputStream(SSLSocketImpl.java:883)
        at ClientSocket.main(ClientSocket.java:46)
        at java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)
        at java.base/java.lang.reflect.Method.invoke(Method.java:577)
        at jdk.compiler/com.sun.tools.javac.launcher.Main.execute(Main.java:421)
        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)
```

Al escribir java CLientSocket.java, que podremos escribirlo o simplemente dándole al tabulador se escribe solo, veremos cómo nos sale error en el cliente (captura anterior) y error en el servidor (captura siguiente)

```
C:\PA1>
C:\PA1>java SSocket.java
Waiting for connection...
javax.net.ssl.SSLHandshakeException: No available authentication scheme
        at java.base/sun.security.ssl.Alert.createSSLException(Alert.java:131)
        at java.base/sun.security.ssl.Alert.createSSLException(Alert.java:117)
        at java.base/sun.security.ssl.TransportContext.fatal(TransportContext.java:358)
        at java.base/sun.security.ssl.TransportContext.fatal(TransportContext.java:314)
        at java.base/sun.security.ssl.TransportContext.fatal(TransportContext.java:305)
        at java.base/sun.security.ssl.CertificateMessage$T13CertificateProducer.onProduceCertificate(CertificateMessage.java:972)
        at java.base/sun.security.ssl.CertificateMessage$T13CertificateProducer.produce(CertificateMessage.java:961)
        at java.base/sun.security.ssl.SSLHandshake.produce(SSLHandshake.java:440)
        at java.base/sun.security.ssl.ClientHello$T13ClientHelloConsumer.goServerHello(ClientHello.java:1246)
        at java.base/sun.security.ssl.ClientHello$T13ClientHelloConsumer.consume(ClientHello.java:1182)
        at java.base/sun.security.ssl.ClientHello$ClientHelloConsumer.onClientHello(ClientHello.java:840)
        at java.base/sun.security.ssl.ClientHello$ClientHelloConsumer.consume(ClientHello.java:801)
        at java.base/sun.security.ssl.SSLHandshake.consume(SSLHandshake.java:396)
        at java.base/sun.security.ssl.HandshakeContext.dispatch(HandshakeContext.java:480)
        at java.base/sun.security.ssl.HandshakeContext.dispatch(HandshakeContext.java:458)
        at java.base/sun.security.ssl.TransportContext.dispatch(TransportContext.java:201)
        at java.base/sun.security.ssl.SSLTransport.decode(SSLTransport.java:172)
        at java.base/sun.security.ssl.SSLSocketImpl.decode(SSLSocketImpl.java:1500)
        at java.base/sun.security.ssl.SSLSocketImpl.readHandshakeRecord(SSLSocketImpl.java:1415)
        at java.base/sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:450)
        at java.base/sun.security.ssl.SSLSocketImpl.ensureNegotiated(SSLSocketImpl.java:915)
        at java.base/sun.security.ssl.SSLSocketImpl$AppInputStream.read(SSLSocketImpl.java:1006)
        at java.base/sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:270)
        at java.base/sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:313)
        at java.base/sun.nio.cs.StreamDecoder.read(StreamDecoder.java:188)
        at java.base/java.io.InputStreamReader.read(InputStreamReader.java:176)
        at java.base/java.io.BufferedReader.fill(BufferedReader.java:162)
        at java.base/java.io.BufferedReader.readLine(BufferedReader.java:329)
        at java.base/java.io.BufferedReader.readLine(BufferedReader.java:396)
        at SSocket.main(SSocket.java:43)
        at java.base/jdk.internal.reflect.DirectMethodHandleAccessor.invoke(DirectMethodHandleAccessor.java:104)
        at java.base/java.lang.reflect.Method.invoke(Method.java:577)
        at jdk.compiler/com.sun.tools.javac.launcher.Main.execute(Main.java:421)
        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)
```

Este error es bueno, es decir, estamos utilizando una comunicación segura pero no se entienden el servidor y el cliente, no está configurado correctamente.
Un ejemplo sería que el servidor ya le ha abierto la puerta al cliente pero el servidor se comunica en italiano y el cliente se comunica en español. Entre ellos no se entienden, entonces tendrá que hablar en un idioma común como es el inglés. Pues eso mismo es lo que vamos a hacer a continuación.

- ***Compilar y ejecutar código cliente y servidor desde consola***

1. Creamos un archivo.class (con el código interpretable por la JVM de Java) en la consola con permiso de administración siguiendo el siguiente comando:
   javac BYODServer.java
   que en este caso cambiaremos BYODServer por nuestra clase servidor SSocket. Todo lo haremos desde nuestro directorio.

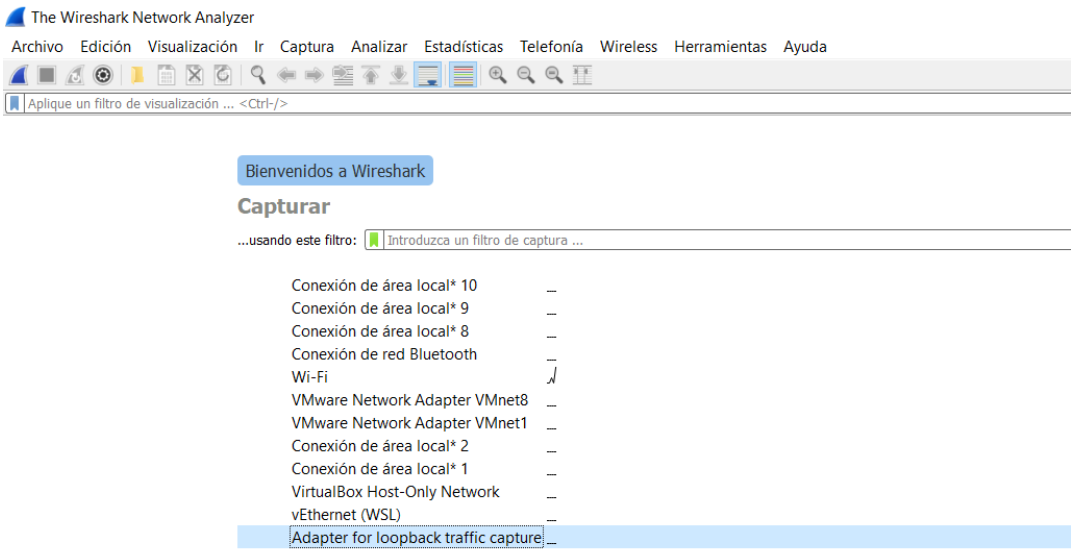
2. Abrimos un blog de notas y escribimos :

   java –Djavax.net.ssl.keyStore=C:\SSLStore –
   Djavax.net.ssl.keyStorePassword=PASSWORD BYODServer

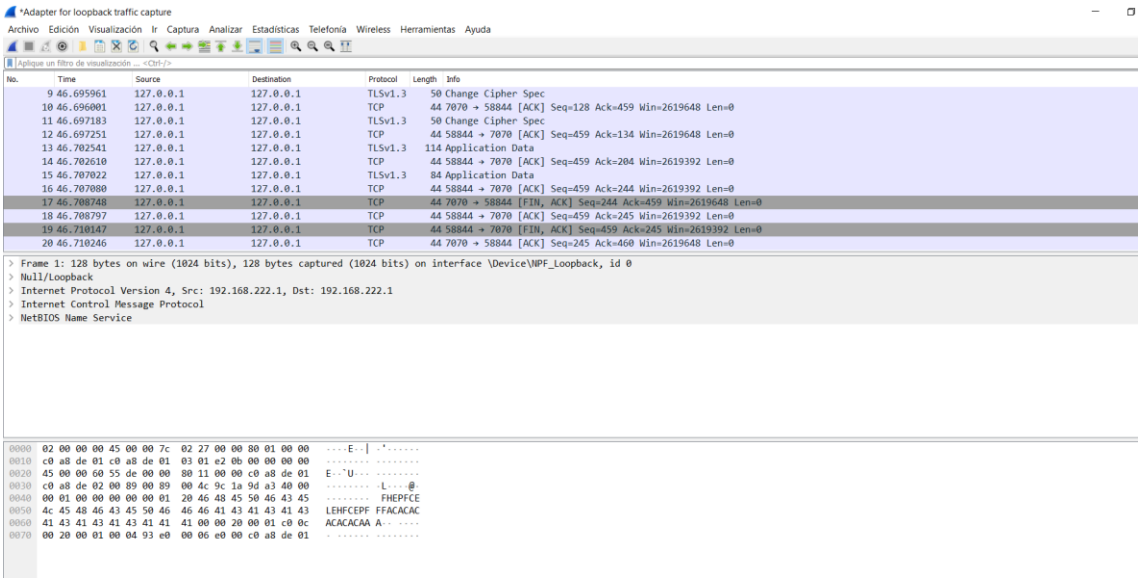
   java –Djavax.net.ssl.trustStore=C:\SSLStore –Djavax.net.ssl.
   trustStorePassword=PASSWORD BYODCliente

   Cambiamos el nombre de nuestra clase. BYODServer por SSocket , BYODCliente por ClientSocket. Nuestra contraseña y SSLStore por PA1.

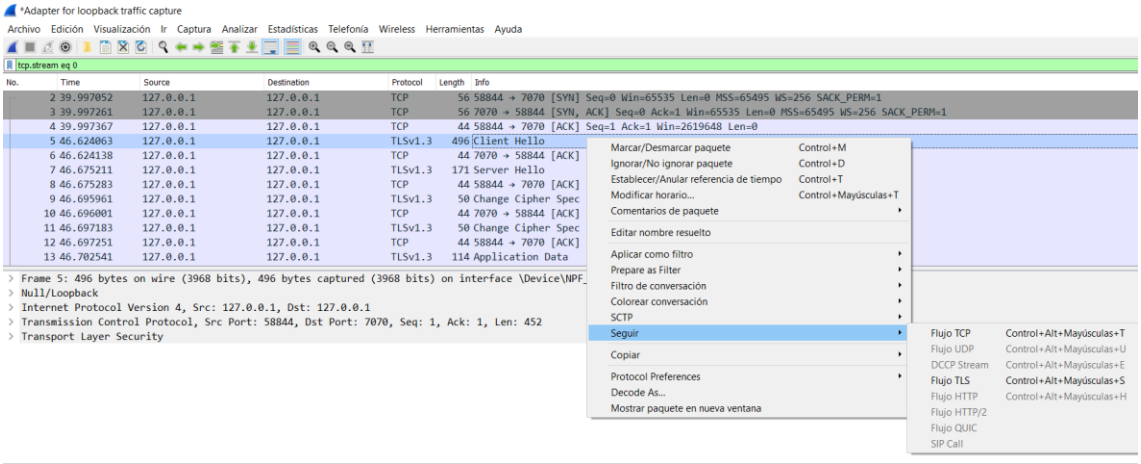
3. En el siguiente paso nos descargaremos wireshark y capturamos tráfico entre cliente y servidor. Wireshark es un sniffers , es decir se interponen en los canales de comunicación entre el servidor y cliente y capturan toda la información que navega por ellos. Seleccionamos la búsqueda en “Adapter for loopback traffic capture” (es una red para realizar conexiones entre los Hosts)



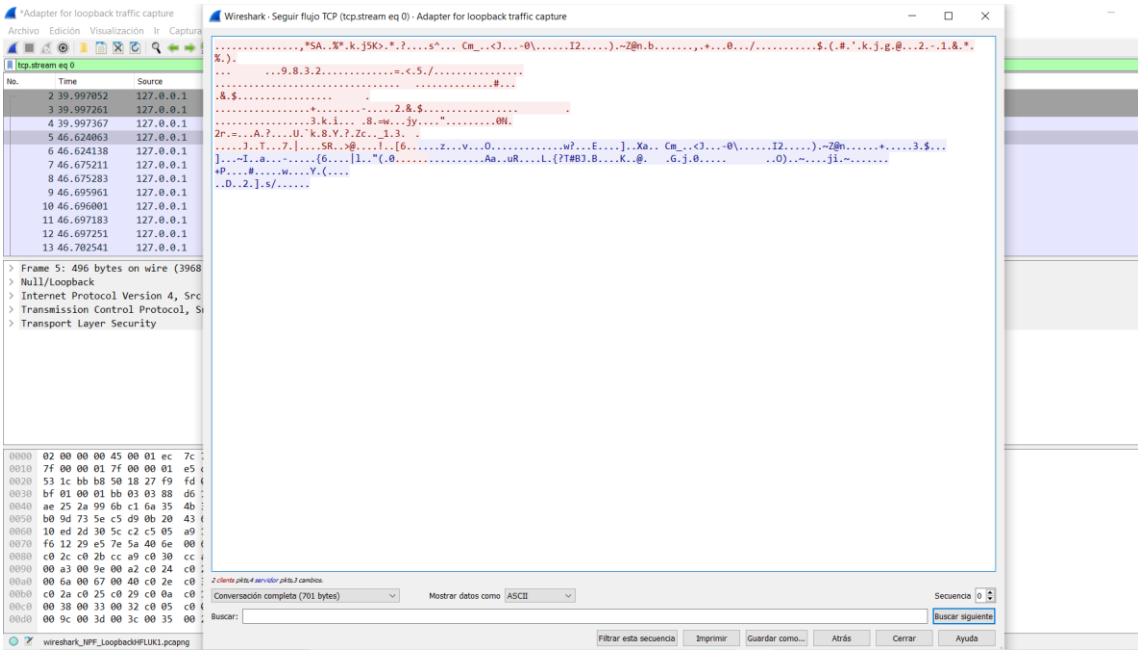
4. Le damos al símbolo de aleta de tiburón para capturar tráfico. Inicializamos primero el servidor y después el cliente mediante cmd como ya hemos hecho anteriormente. Una vez que ya hemos puesto el usuario y la contraseña, paramos el tráfico dándole al cuadrado rojo de la barra superior y veremos lo siguiente.



Como vemos ha capturado muchos paquetes que pasan por la interfaz que hemos seleccionado. Para que nos salga solo en la pantalla aquellos que utilizan el protocolo TCP añadimos un filtro en la barra superior y ponemos tcp.stream eq 0 .



Como podemos comprobar en la captura anterior el TLS utilizado es TLSv1.3. Hacemos click derecho a la conexión seleccionada (Client Hello) y le damos a seguir flujo TCP.

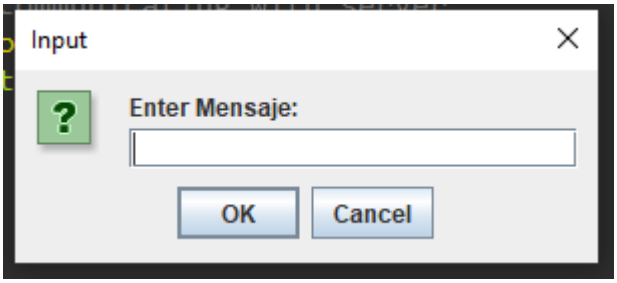


¿Qué quiere decir la captura anterior?
Lo que demuestra es que el canal es seguro. No podemos ver ni obtener el nombre de usuario ni la contraseña … no podemos ver nada porque la información está cifrada. Como se pueden comunicar entre ellos utilizan el mismo keyStore en Cliente y Servidor, si se utilizara dos máquinas distintas estarían también en el Servidor y en el Cliente.

- ***<u>Comprobación del servidor VPN para soportar a los 300 empleados del hospital de forma concurrente.</u>***

Creación de una pantalla de mensaje:

```
String mensaje = JOptionPane.showInputDialog(null,
                 "Enter Menssage:");
output.println(mensaje);
```



Algunas de las soluciones que hemos pensado han sido:

Bucle for:

Queríamos realizar un bucle que enviara 10 usuarios al servidor a la vez.

```
public static void main(String[] args) throws IOException {
    int i=1;
    for(i=1; i<=10;i++) {
try {

    // create Socket from factory
    SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
    SSLSocket socket = (SSLSocket) socketFactory.createSocket("localhost", 7070);



    //Socket socket = new Socket("127.0.0.1", 7070);

    // create PrintWriter for sending login to server

    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);
```

Bucle while:

Al igual, que con el bucle for, mientras que se cumpliese la condición, se enviaran 10 usuarios a la vez al servidor.

```
public static void main(String[] args) throws IOException {
    int i=1;
    while(i<=10) {
try {

    // create Socket from factory
    SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
    SSLSocket socket = (SSLSocket) socketFactory.createSocket("localhost", 7070);



    //Socket socket = new Socket("127.0.0.1", 7070);

    // create PrintWriter for sending login to server

    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);
```

Creación de una segunda clase con un bucle for para ejecutar la clase ClientSocket:

Con nuestra clase ClientSocket creada, con una segunda clase ClientesSocket creamos un bucle for para ejecutar la clase al menos 100 veces, para que así se enviasen 100 clientes al servidor.

```
  public static void main(String[] args) throws IOException {
  try {

      // create Socket from factory
      SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
      SSLSocket socket = (SSLSocket) socketFactory.createSocket("localhost", 7070);
      //Socket socket = new Socket("127.0.0.1", 7070);

      // create PrintWriter for sending login to server

      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:");
      output.println(password);
      // prompt user for password
      String mensaje = JOptionPane.showInputDialog(null,
                      "Enter Mensaje:");
      output.println(mensaje);
///         send password to server
```

```
public class ClientesSocket {

    public static void main(String[] args) throws IOException {
        try {

            // create Socket from factory
            SSLSocketFactory socketFactory = (SSLSocketFactory) SSLSocketFactory.getDefault();
            SSLSocket socket = (SSLSocket) socketFactory.createSocket("localhost", 7070);


            for (int i = 0; i < 100; i++)
            {
                ClientSocket.writeUTF("Este es el cliente número " + (i+1) + "\n");
            }

            socket.close();
```

```
Waiting for connection...
javax.net.ssl.SSLHandshakeException: Remote host terminated the handshake
        at java.base/sun.security.ssl.SSLSocketImpl.handleEOF(SSLSocketImpl.java:1661)
        at java.base/sun.security.ssl.SSLSocketImpl.decode(SSLSocketImpl.java:1470)
        at java.base/sun.security.ssl.SSLSocketImpl.readHandshakeRecord(SSLSocketImpl.java:1370)
        at java.base/sun.security.ssl.SSLSocketImpl.startHandshake(SSLSocketImpl.java:437)
        at java.base/sun.security.ssl.SSLSocketImpl.ensureNegotiated(SSLSocketImpl.java:878)
        at java.base/sun.security.ssl.SSLSocketImpl$AppInputStream.read(SSLSocketImpl.java:969)
        at java.base/sun.nio.cs.StreamDecoder.readBytes(StreamDecoder.java:297)
        at java.base/sun.nio.cs.StreamDecoder.implRead(StreamDecoder.java:339)
        at java.base/sun.nio.cs.StreamDecoder.read(StreamDecoder.java:188)
        at java.base/java.io.InputStreamReader.read(InputStreamReader.java:181)
        at java.base/java.io.BufferedReader.fill(BufferedReader.java:161)
        at java.base/java.io.BufferedReader.readLine(BufferedReader.java:326)
        at java.base/java.io.BufferedReader.readLine(BufferedReader.java:392)
        at SSocket.main(SSocket.java:40)
Caused by: java.io.EOFException: SSL peer shut down incorrectly
```

Con hilos

Otra idea sería crear un hilo, en otra clase “Con hilos”

```
    String atributo;
    public ConHilos(int i)
    {
        atributo = "clientes" + i;
    }

    public static void main(String[] args)
    {
        for (int i = 0; i < 4; i++)
        {
            Thread socket = new Thread(new ConHilos(i));
            socket.start();
        }
    }
```

Poner el bucle for en el servidor:

La última opción era crear un bucle for en el servidor, con eso conseguimos que en una misma conexión el servidor soportase i clientes, pero de uno en uno, no todos a la vez. Así que tuvimos que desechar esa idea.

```
public class SSocket {

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

SSLServerSocketFactory socketFactory = (SSLServerSocketFactory)
SSLServerSocketFactory.getDefault();
SSLServerSocket sSocket = (SSLServerSocket) socketFactory.createServerSocket(7070);
// perpetually listen for clients
//ServerSocket sSocket = new  ServerSocket(3343);
while (true) {
int i;
// wait for client connection and check login information
try {
    for (i=0;i<=10;i++) {
System.err.println("Waiting for connectionn...");
Socket socket = sSocket.accept();
System.err.println("Sirvo al cliente: " + i );
socket.close();
    }
```

```
C:\PA1>java -Djavax.net.ssl.keyStore=.\SSLStore -Djavax.net.ssl.keyStorePassword=SeguridaD SSocket
Waiting for connectionn...
Sirvo al cliente 0
Sirvo al cliente 1
Sirvo al cliente 2
Sirvo al cliente 3
Sirvo al cliente 4
Sirvo al cliente 5
Sirvo al cliente 6
Sirvo al cliente 7
Sirvo al cliente 8
Sirvo al cliente 9
Waiting for connectionn...
Sirvo al cliente 0
Sirvo al cliente 1
```