INGENIERÍA DE SEGURIDAD US
E.T.S. Ingeniería Informática. Universidad de Sevilla
Avda Reina Mercedes s/n. 41012 Sevilla
Tlf/Fax 954 557 139 **E-mail** lsi@lsi.us.es **Web** www.lsi.us.es

PAI-2. BYODSEC-BRING YOUR OWN DEVICE SEGURO USANDO ROAD WARRIOR VPN SSL PARA UNA UNIVERSIDAD PÚBLICA

Introducción

En este **Proyecto de Aseguramiento de la Información** usaremos técnicas para poder asegurar la **integridad, confidencialidad y autenticidad en la transmisión de datos por redes públicas como Internet.**

En este proyecto, una determinada Universidad Pública 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 Universidad tales como correos electrónicos, bases de datos y archivos en servidores corporativos usando una VPN SSL. En el contexto empresarial esto se le conoce como Road Warriors, es decir, usuarios remotos que necesitan acceso Seguro a las infraestructuras de sus organizaciones.

Para la transmisión es fundamental la implementación de canales de comunicación seguros. En proyectos anteriores hemos visto cómo proceder al aseguramiento de la información mediante tecnologías de **aseguramiento de la integridad, por ejemplo, usando mecanismo de códigos de autenticación de mensajes (MAC).** Estos elementos son parte además de otros muchos para la implementación de canales de comunicación seguros.

Una de las maneras para crear canales de comunicación seguros es la utilización de ***VPN (Virtual Private Networks)*** que en su implementación puede usar Sockets o bien implementaciones concretas de terceros como ***OpenVPN o WireGuard***.

Ya se han varias formas para hacer la información integra, pero no confidencial y autenticada a través del canal de comunicaciones. Por ello en este proyecto una de las opciones posibles de implementación serían los sockets del tipo ***Secure Sockets Layers (SSL)*** como se observa en la Figura 1.

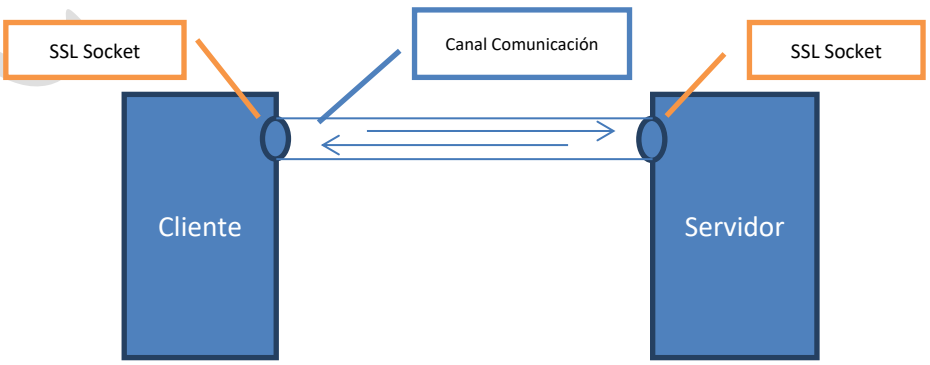

Figura 1: Canal de Comunicación Segura con SSL

SSL/TLS es un protocolo de comunicación seguro (implementa los requisitos de **autenticidad, confidencialidad e integridad**). Este protocolo utiliza una infraestructura basada en almacenes de claves y certificados.

Además, la Universidad Pública como Administración pública debe responder al cumplimiento legal exigido por la legislación española actual, al encontrarse la sede de dicha Universidad en España. En este caso, además del Reglamento de Protección de Datos Personales de la Unión Europea (**RGPD**) y de Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales (**LOPDGDD**), se exige el cumplimiento del Real Decreto 3/2010 del Gobierno de España, de 8 de enero, por el que se regula el Esquema Nacional de Seguridad (**ENS**) en el ámbito de la Administración Electrónica. Dicho ENS indica en su artículo 1 que está "… ***constituido por los principios básicos y requisitos mínimos requeridos para una protección adecuada de la información. Será aplicado por las Administraciones públicas para asegurar el acceso, integridad, disponibilidad, autenticidad, confidencialidad, trazabilidad y conservación de los datos, informaciones y servicios utilizados en medios electrónicos que gestionen en el ejercicio de sus competencias.***

Política de Seguridad

La política de seguridad de la Universidad nos dice con respecto a la seguridad en las transmisiones de información cliente-servidor:

> *"… deberían ser **confidenciales e íntegras** y **además autenticadas**."*

Objetivos

1. Desarrollar/seleccionar **cómo llevar a la práctica de forma lo más eficiente posible los canales de comunicación segura para la transmisión *de credenciales (usuario, contraseñas) y mensajes usando*** el protocolo SSL/TLS asegurando autenticidad, confidencialidad e integridad.
2. Tener en cuenta que **el número de empleados concurrentes que usarán la aplicación son aproximadamente 300, realizar las pruebas de capacidad para demostrar que el sistema soporta este número de empleados.**
3. Utilizar alguna herramienta de análisis de tráfico que **permita comprobar la confidencialidad e integridad de los canales de comunicaciones seguros**.
4. Establecer **Cipher Suites robustos que usen en la versión TLS 1.3** evitando vulnerabilidades.
5. **Realizar pruebas que demuestren la confidencialidad de la VPN SSL y si hay pérdida de rendimiento al usar la VPN SSL frente a no usarla**.

Recomendaciones para el desarrollo

Arquitectura

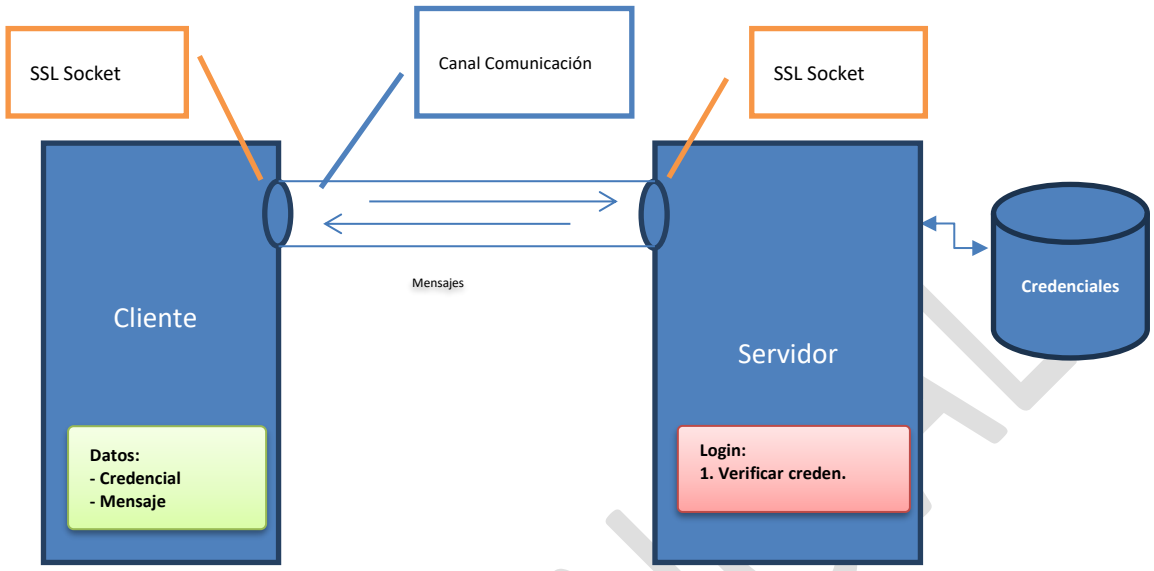

Figura 2: Arquitectura propuesta para la Universidad pública

Requisitos funcionales

1. **Registro de usuarios:**

   - Permitir que un nuevo usuario se registre proporcionando únicamente un nombre de usuario y una contraseña.
   - Informar al usuario si ya existe un registro con el mismo nombre de usuario.
   - No permitir modificaciones a los datos de usuarios después del registro.

2. **Inicio de sesión:**

   - Permitir a los usuarios registrados iniciar sesión introduciendo su nombre de usuario y contraseña.

3. **Verificar credenciales:**

   - Validar las credenciales proporcionadas contra los datos almacenados en el servidor.
   - Denegar acceso si las credenciales no coinciden con un registro válido.

4. **Cerrar sesión:**

   - Permitir a los usuarios ya logados cerrar la sesión.

5. **Gestión de usuarios preexistentes:**

   - El sistema debe contar con un conjunto inicial de usuarios registrados para que puedan acceder sin necesidad de registrarse.

6. **Mensajes:**

   - Permitir a los usuarios autenticados realizar envió de mensajes de texto al servidor.

7. **Persistencia de datos:**

   - Registrar y almacenar de manera permanente:
     - Los datos de los usuarios.
   - Registrar mensajes enviados por usuario.
     - Nº de mensajes enviados

8. **Interfaz de comunicación:**

   - Proveer una interfaz (***usando sockets seguros***) que permita a los clientes interactuar con el sistema de registro, autenticación, y envío de mensajes.

Requisitos de Información

El sistema debe poder almacenar información relacionada con:

1. **Datos de usuarios:**
   - Cada usuario debe tener:
     - Nombre de usuario (único).
     - Contraseña.
2. **Registro inicial:**
   - El sistema debe contar con una base de datos inicial con:
     - Una lista de usuarios pre-registrados.
     - Sin mensajes realizadas previamente.
3. **Registro de mensajes:**
   - El sistema debe contar con un historial de mensajes:
     - Número de mensajes enviados por usuario y fecha.

El sistema debe gestionar la siguiente información:

4. **Mensajes:**
   - Cada envío de mensaje debe incluir:
     - Identificador de usuario.
     - Cadena de texto del mensaje a enviar (máximo 144 caracteres).
5. **Mensajes del sistema:**
   - Informar al usuario en las siguientes situaciones:
     - Usuario registrado exitosamente.
     - Usuario ya registrado.
     - Inicio de sesión exitoso o fallido.
     - Mensaje enviado y recibido correctamente.

Requisitos de seguridad

1. **Requisitos de Seguridad para las Credenciales de Usuario**
   - Almacenamiento Seguro de Credenciales
   - Verificación Segura de Credenciales
   - Protección Contra Ataques en Login (BruteForce)
2. **Requisitos de Seguridad para el envío de Mensajes**

- Integridad en el envío de mensajes.
- Confidencialidad en el envío de mensajes.
- Autenticidad en el envío de mensajes.

3. **Requisitos de Seguridad para la Base de Datos**
   - Integridad de los datos almacenados en base de datos.

Normas del entregable

- Cada grupo debe entregar a través de la Plataforma de Enseñanza Virtual y en la **actividad** preparada para ello un archivo zip, nombrado **PAI2-STX.zip**, que deberá contener al menos los ficheros siguientes:
  - ✓ **Documento en formato pdf que contenga un informe/resumen del proyecto** con los detalles más importantes de las decisiones, soluciones adoptadas y/o implementaciones desarrolladas, así como el resultado y análisis de las pruebas realizadas (máximo 10 páginas).
  - ✓ **Código fuente de las posibles implementaciones y/o scripts desarrollados y/o configuraciones, será necesario la entrega de los test, logs de las pruebas y trazas de los sniffers utilizados.**
- El plazo de entrega de dicho proyecto finaliza el **día *21 de octubre a las 23:59 horas***.
- Los proyectos entregados fuera del plazo establecidos serán considerados inadecuados por el cliente y por tanto entrarán en penalización por cada día de retraso entrega de 10% del total, hasta agotarse los puntos.
- **El cliente no aceptará envíos realizados por email, ni mensajes internos de la enseñanza virtual, ni correo interno de la enseñanza virtual. Toda entrega realizada por estos medios conllevará una penalización en la entrega del 10%.**

ANEXO – Ejemplo Java de infraestructura cliente-servidor SSL/TLS

Arquitectura Cliente-Servidor

En primer lugar, vamos a crear una infraestructura cliente-servidor mediante el uso de sockets seguros.

A. Creación de un keyStore – Compartición de claves

De acuerdo con la especificación del protocolo SSL/TLS se necesita para la autenticación de los servidores de los correspondientes certificados. En Java se puede realizar esto mediante la creación de un almacén o repositorio de certificados de seguridad para el protocolo SSL/TLS. Java viene provisto de una herramienta que permite crear de manera sencilla un almacén de certificados. Para crear el almacén de certificados ejecutamos en consola y con permisos de administración el siguiente comando:

```
keytool -genkey -keystore keystore.jks -alias ssl -keyalg RSA
```

Keytool hará una serie de preguntas para crear la creación del keyStore. Con esto ya tendremos en la ruta C:\ un archivo, por ejemplo, con nombre *SSLStore*. Este será el almacén de certificados que usaran nuestros Sockets cliente y servidor. De manera similar se podría realizar en un sistema basado en Linux.

Para usuarios de Windows, ***keytool*** se encuentra en la carpeta *bin* del JDK [1]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.

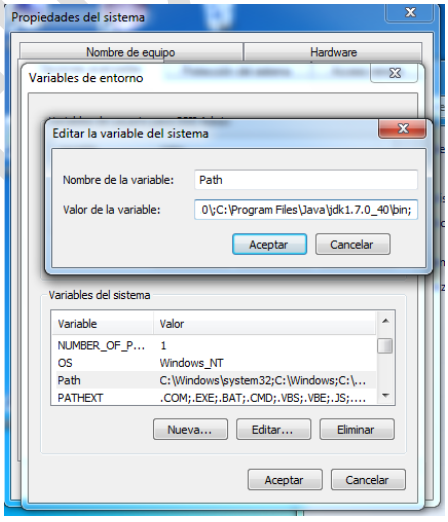

Figura 2: Configuración variable de entorno

B. Generación de un Socket TLS Servidor

Para utilizar comunicaciones seguras usando SSL/TLS entre cliente-servidor, utilizaremos Socket SSL/TLS. Crearemos una clase*, por ejemplo,* ***EjemploSocketSSLServer.java,*** donde instanciaremos el server socket, pero usando la implementación SSL/TLS (`SSLServerSocket`):

```
import javax.net.ssl.*; // Librerias SSL/TLS de JDK
```

[1] Cuidado porque los JRE de java no proporcionan dicha herramienta, sólo los JDK de Oracle.

```
...

SSLServerSocketFactory socketFactory = (SSLServerSocketFactory)
                                              SSLServerSocketFactory.getDefault();
serverSocket = (SSLServerSocket) socketFactory.createServerSocket(3343);

SSLSocket socket = (SSLSocket) serverSocket.accept();
```

El código se puede ejecutar en cualquier IDE de desarrollo (Eclipse o Visual Studio Code), pero debemos tener en cuenta que son dos procesos (servidor y cliente) diferentes con diferentes parámetros, por tanto, deberemos tener activada la correspondiente configuración de entorno. O también podemos no ejecutar directamente este código sobre el IDE, y hacerlo en el entorno de ejecución del sistema operativo especificando los procesos y los parámetros a la JVM de dónde se encuentra el almacén de claves y la clave de este como veremos en el apartado D.

C. Generación de un Socket TLS Cliente

Crearemos una clase ***EjemploSocketSSLClient.java*** donde crearemos una conexión a un socket server determinado pero usando la implementación SSL/TLS (**`SSLSocket`**), tal y como se muestra a continuación:

```
import javax.net.ssl.*; // Librerías SSL/TLS de JDK


...

SSLSocketFactory socketFactory = (SSLSocketFactory)  SSLSocketFactory.getDefault();
SSLSocket socket = (SSLSocket) socketFactory.createSocket("localhost", 8080);
```

Notar que la IP y el puerto de conexión deben coincidir con el puerto e IP del servidor y el puerto definido en la clase del servidor.

D. Compilar y ejecutar código cliente y servidor desde consola

1. Desde una consola con permisos de administración y en el directorio donde se encuentre la clase Java del cliente o servidor, ejecutaremos el siguiente comando:

   ```
   javac[2] EjemploSocketSSLServer.java
   javac EjemploSocketSSLCliente.java
   ```

   Esto va a generar un archivo .class con el código interpretable por la JVM de Java, y que utilizaremos desde la misma línea de comandos para su ejecución.

2. Para ejecutar las clases una vez compiladas simplemente debemos de ejecutar el siguiente comando donde se establecen el keyStore en el servidor y el trustStore para el cliente:

   - **`EjemploSocketSSLServer.java`**:
     ```
     java -Djavax.net.ssl.keyStore=C:\SSLStore -
     Djavax.net.ssl.keyStorePassword=PASSWORD EjemploSocketSSLServer
     ```
   - **`EjemploSocketSSLCliente.java`**:
     ```
     java -Djavax.net.ssl.trustStore=C:\SSLStore -Djavax.net.ssl.
     trustStorePassword=PASSWORD EjemploSocketSSLCliente
     ```

[2] El compilador javac sólo está disponible en las JDK.

Análisis de tráfico de red en comunicaciones (sniffers)

Una manera de inspeccionar la información que fluye entre los Sockets es usar herramientas para analizar tráfico de red o sniffers. Los sniffers se interponen en los canales de comunicación y capturan toda la información que fluye por ellos. Existen sniffers muy sofisticados como *Wireshark* que permiten analizar múltiples protocolos, y otros sniffers más simples, por ejemplo, tcpdump que sólo capturan la información de red y hacen un volcado de la información de los paquetes en un archivo dump.

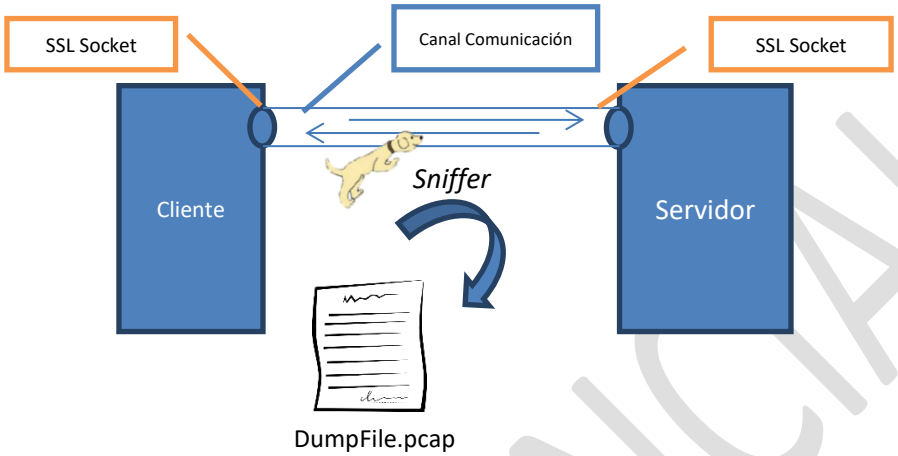

Figura 3: Monitorización del tráfico de red

En este proyecto si estamos en Windows, podríamos usar *RawCap* un sniffer para Windows que se utiliza desde línea de comandos, y/o *Wireshark*. Para inspeccionar el archivo de salida de RawCap podremos usar Wireshark. En Linux no es necesario usar *RawCap*, podría usarse directamente Wireshark o un sniffer como *tcpdump* junto con *Wireshark*.

RawCap

RawCap nos hará dos preguntas: (1) la interfaz queremos escuchar; y (2) ruta y nombre del archivo donde queremos volcar la información (con extensión pcap). En nuestro caso vamos a capturar el tráfico en *loopback* o *localhost*, ya que el canal de comunicación se estable en nuestra propia máquina. *RawCap* interceptará todos los paquetes de información y los escribirá en el archivo especificado. En la siguiente imagen vemos a *RawCap* en marcha escuchando en la interfaz loopback, y la ruta (c:\ dumpfile.pcap) del fichero donde escribirá la información.

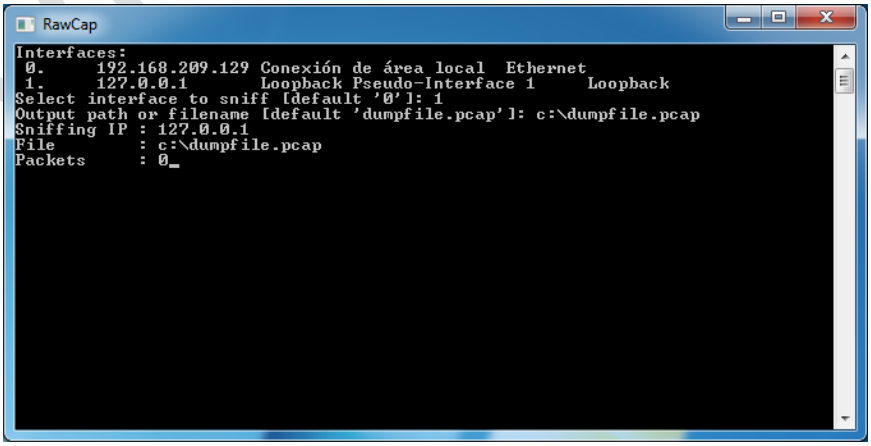

Figura 4: RawCap capturando información en la interfaz localhost

Wireshark

Una vez capturado el tráfico con ***RawCap*,** y obtenido el archivo. pcap se puede abrir con la herramienta ***Wireshark*** de tal manera que podemos observar el intercambio de paquetes que se ha producido en la interfaz seleccionada. Igualmente podemos usar directamente ***Wireshark***, para realizar la captura de paquetes y realizar el análisis del tráfico.

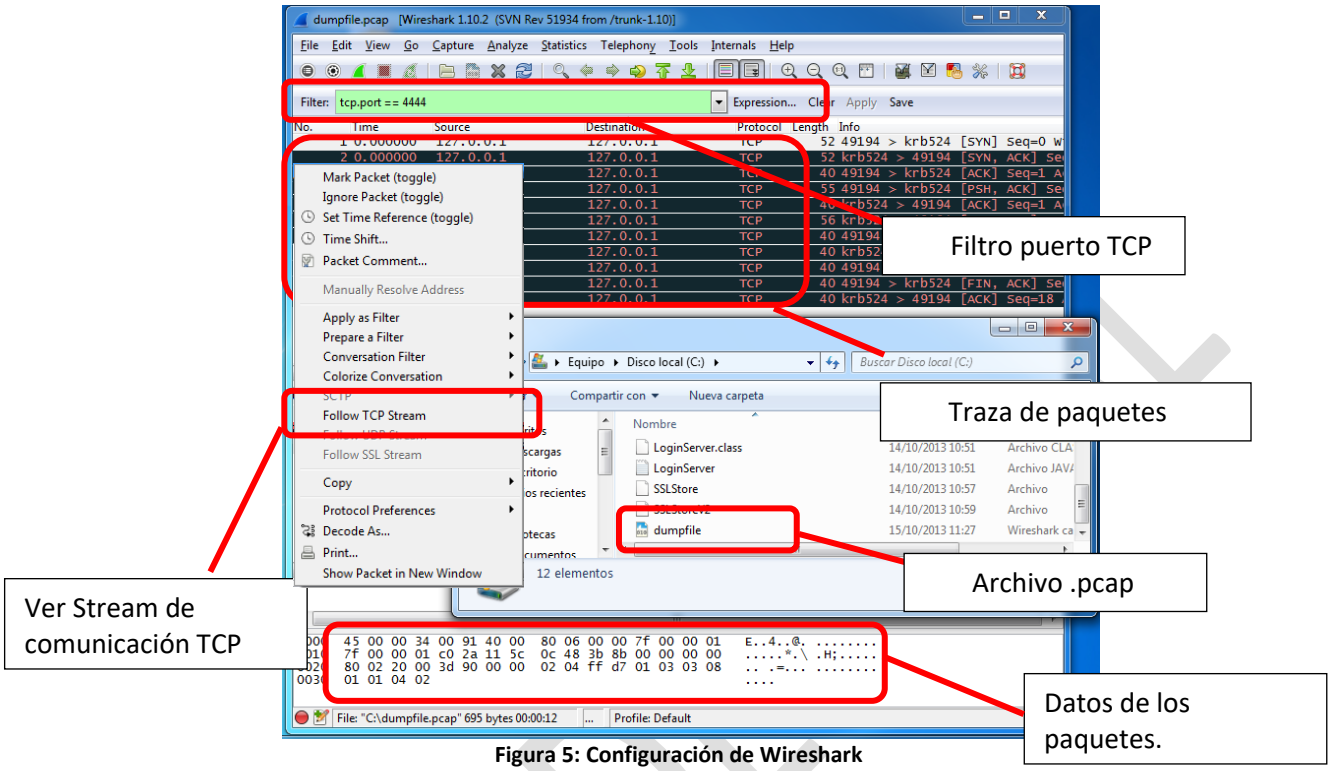

Figura 5: Configuración de Wireshark

En el caso de la Figura 5 al filtrar los paquetes capturados para observar sólo aquellos que utilicen el puerto 4444. Si utilizamos la opción de *"Follow TCP Stream*" podemos observar todos los datos que se transmiten en la conexión de los Sockets que hemos llevado a cabo. Del mismo modo podemos tener una vista más detallada de los datos de los paquetes si marcamos alguno de los paquetes y observamos su información a nivel de red, IP o TCP, en la ventana correspondiente como en la Figura 6.

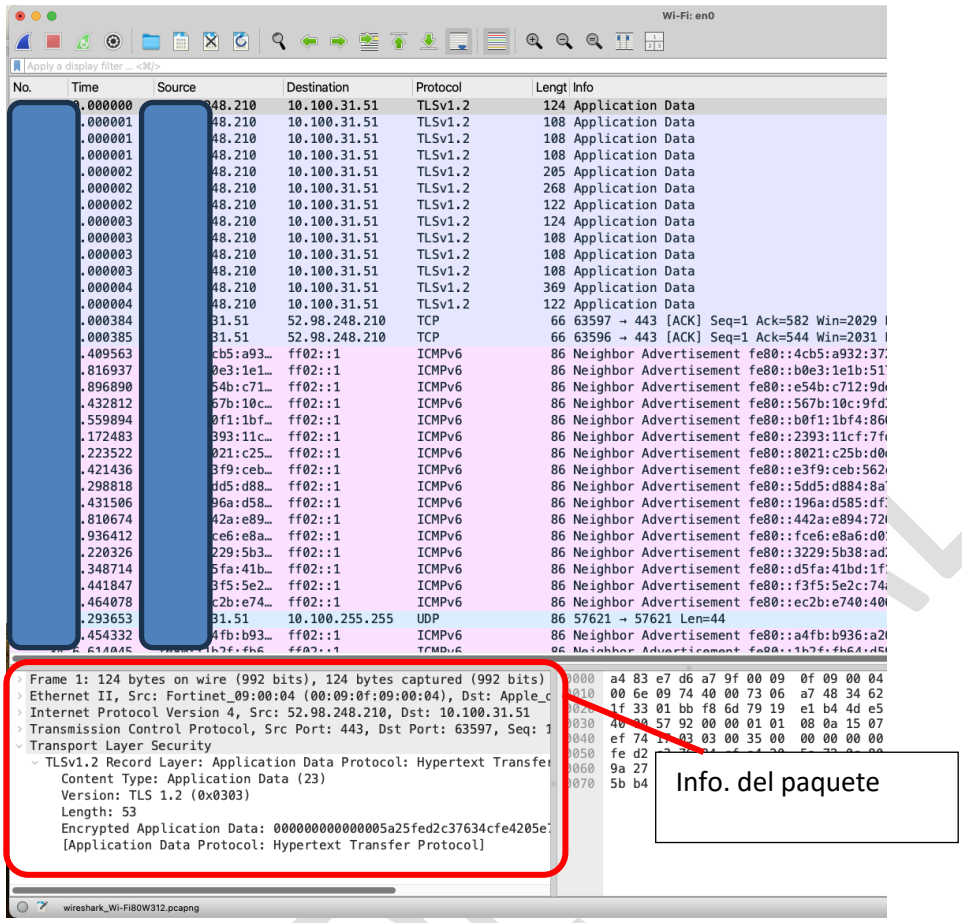

Figura 6: Análisis de datos de paquetes de Wireshark