Práctica 1. Protocolo HTTP: Servidores y Aplicaciones Web

1.1. La seguridad en la red

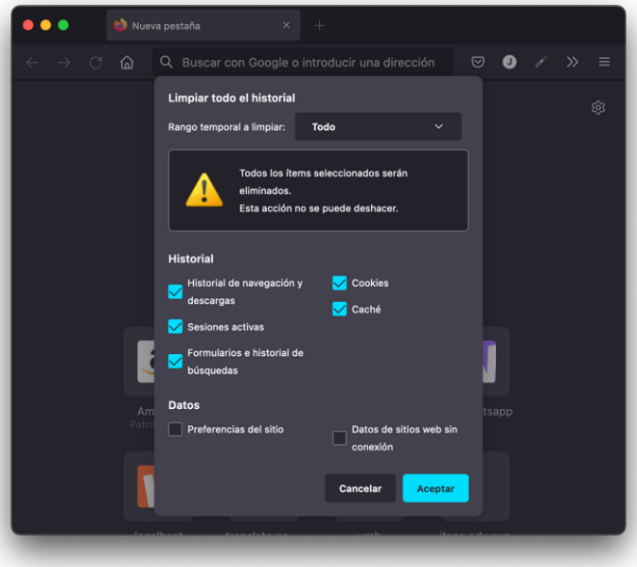
Cuando el investigador Tim Berners-Lee, trabajando para el CERN (European Organization for Nuclear Research), diseñó el protocolo HTTP, no tuvo en consideración la posibilidad de que las comunicaciones cliente/servidor fueran seguras, es decir, la información que intercambian los hosts en una comunicación HTTP convencional puede ser interceptada por terceros y, en consecuencia, dicha información puede ser utilizada con cualquier fin, malicioso o no.

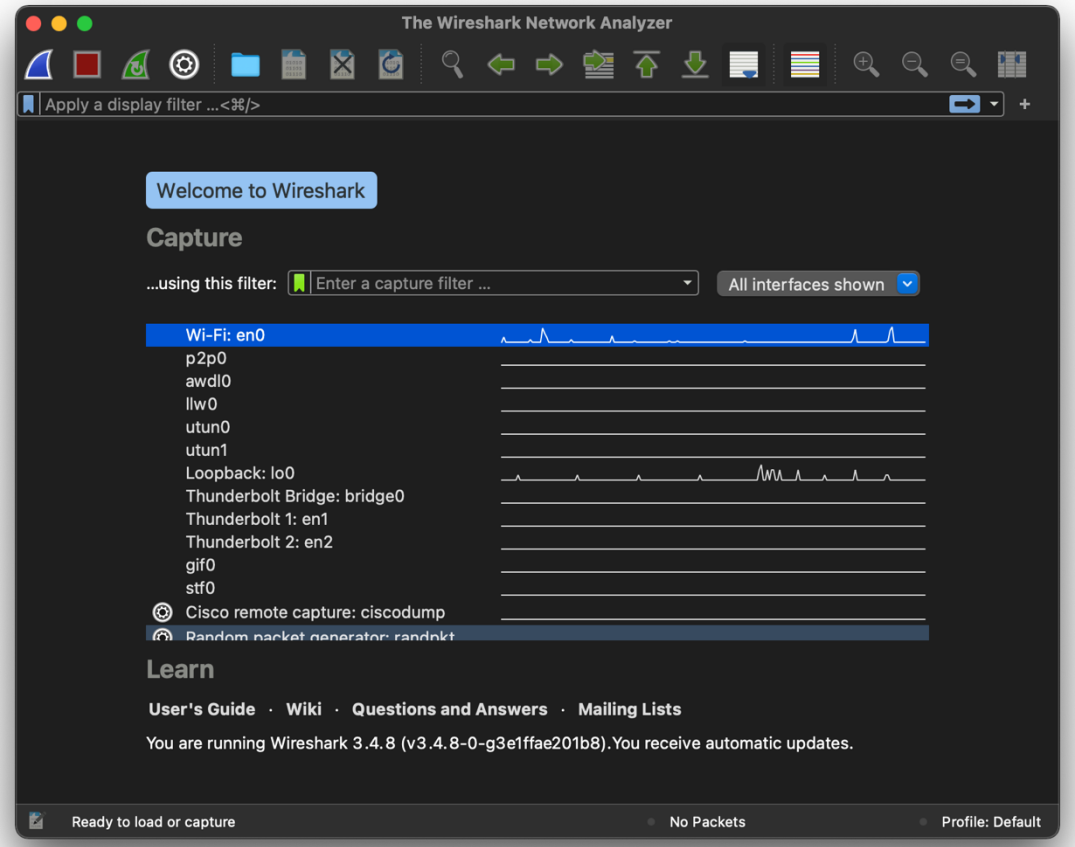
Vamos a utilizar la aplicación **Wireshark** para ilustrar lo sencillo que es capturar el tráfico de una conexión HTTP. Wireshark es un analizador de protocolos Open Source que actualmente está disponible para plataformas Windows, macOS y Unix. Su principal objetivo es el análisis de tráfico, pero además es una excelente aplicación didáctica para el estudio de las comunicaciones y para la resolución de problemas de red.

Wireshark implementa una amplia gama de filtros que facilitan la definición de criterios de búsqueda para los más de 1100 protocolos soportados actualmente y todo ello por medio de una interfaz sencilla e intuitiva que permite desglosar por capas cada uno de los paquetes capturados. Gracias a que Wireshark "entiende" la estructura de los protocolos, podemos visualizar los campos de cada una de las cabeceras y capas que componen los paquetes monitorizados, proporcionando un gran abanico de posibilidades al administrador de redes a la hora de abordar ciertas tareas en el análisis de tráfico.

https://www.wireshark.org/

Antes de empezar a usar la herramienta es muy importante borrar la caché del navegador que vayamos a usar; en el caso de la imagen siguiente es Mozilla Firefox. El objetivo es que las páginas web que usaremos de ejemplo no estén guardadas en la memoria caché del navegador a causa de una conexión anterior a ellas.

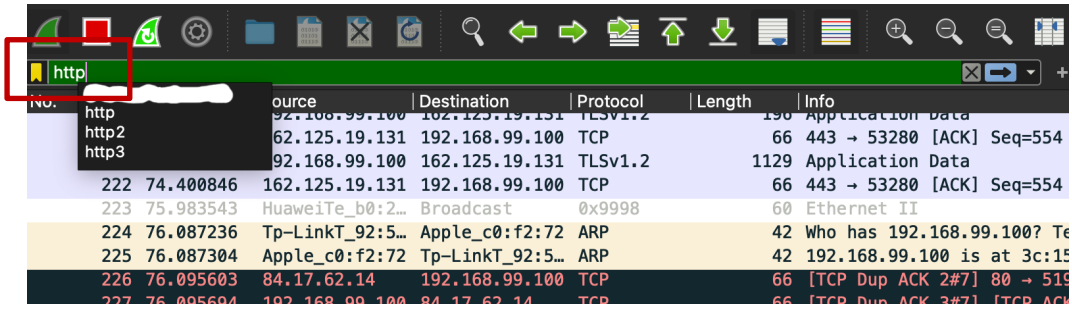




Tras ejecutar Wireshark lo primero que debemos hacer es identificar qué adaptador de red estamos usando para conectar a Internet. En el caso de la imagen anterior es `Wi-Fi:en0`.

En algunos sistemas es posible que surjan problemas de permisos para acceder al adaptador de red, en tal caso, basta con instalar el componente incluido `ChmodBPF` tal y como se muestra en la imagen siguiente.

Al hacer clic sobre este adaptador el programa comenzará a capturar todo el tráfico que se gestiona a través de él. Pero únicamente queremos analizar el tráfico generado por las conexiones HTTP que haremos acto seguido, por eso, aplicaremos un filtro de visualización de paquetes usando el texto `http`. Véase la imagen siguiente.

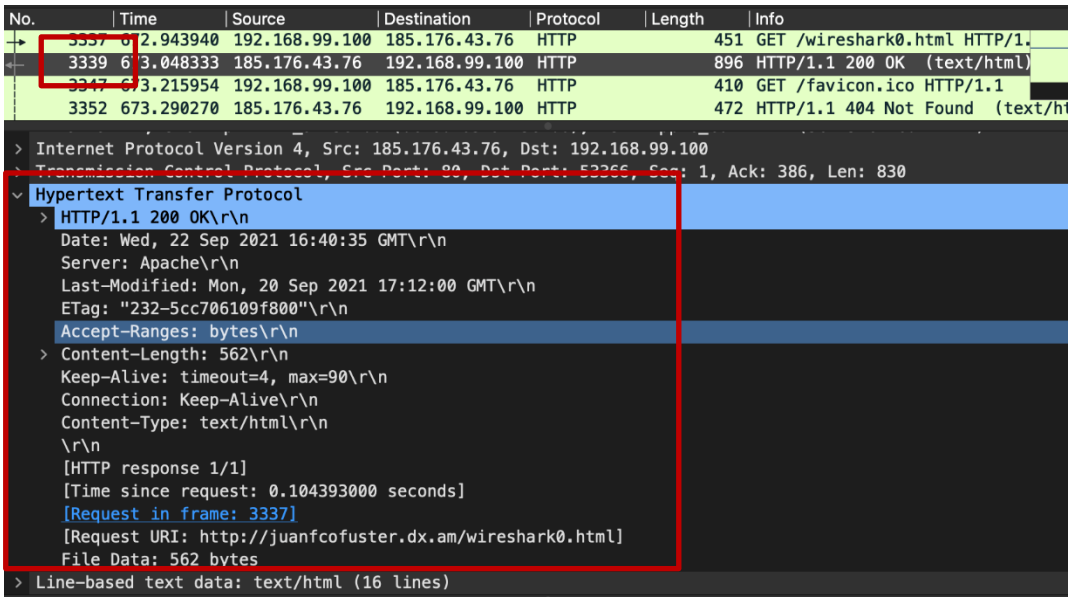


Usamos el navegador para conectar a nuestra primera página web de ejemplo: http://juanfcofuster.dx.am/wireshark0.html y observamos los paquetes capturados por la aplicación. En nuestro ejemplo:

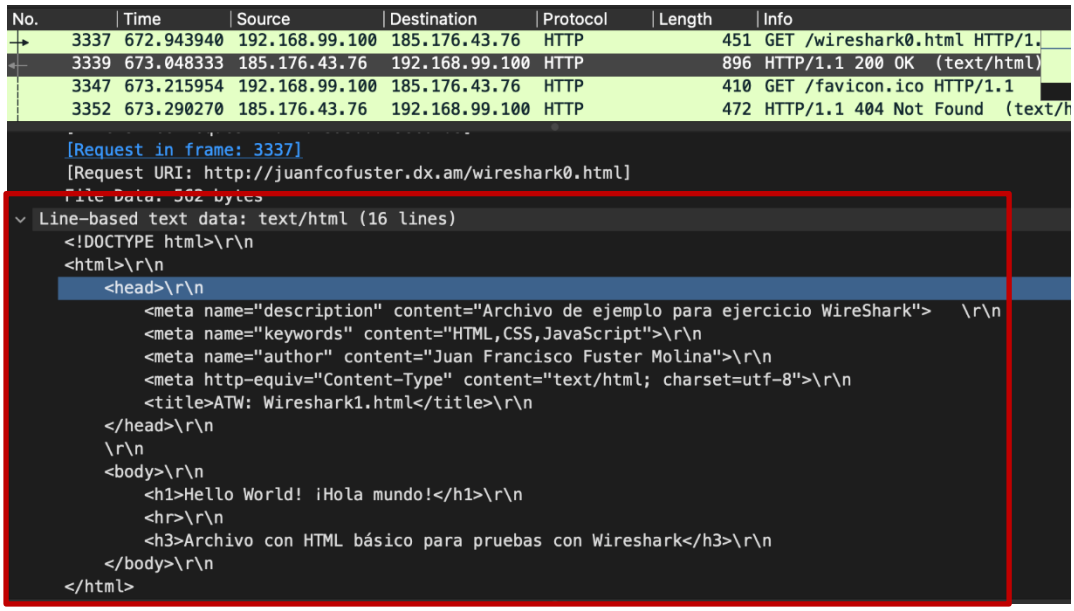
- el paquete 3337 contiene la petición de nuestro navegador (con una longitud de 451 bytes (`GET /wireshark0.html HTTP /1.1`)
- el paquete 3339 contiene la respuesta del servidor (con una longitud de 896 bytes, `HTTP/1.1 200 OK text/html`).

Podemos detener la captura de paquetes en este momento pulsando el botón **Stop capturing packets.** Nótese que estamos usando una conexión HTTP no cifrada.

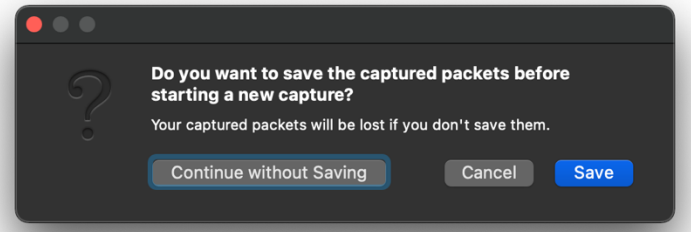
En el paquete 3339, dentro del apartado **Hypertext Transfer Protocol**, podemos observar la cabecera devuelta por el servidor.

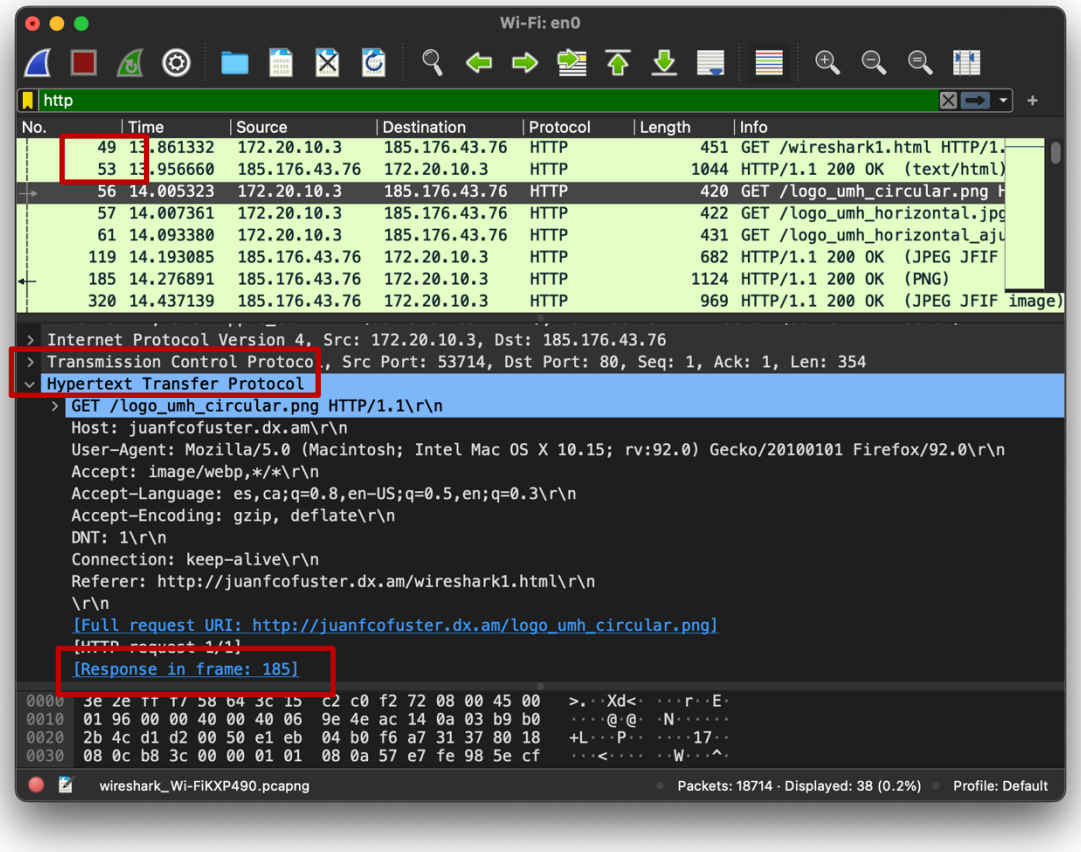


Más abajo, en el apartado **Line-base text data: text/html**, podemos encontrar el código completo del archivo HTML.



A continuación, conectaremos a otra página web de ejemplo, esta vez, una un poco más elaborada ya que contiene elementos multimedia incluidos como son imágenes: http://juanfcofuster.dx.am/wireshark1.html. Antes, recuerda poner Wireshark a capturar paquetes otra vez. No es necesario guardar los paquetes capturados en la conexión anterior: **Continue without Saving**.

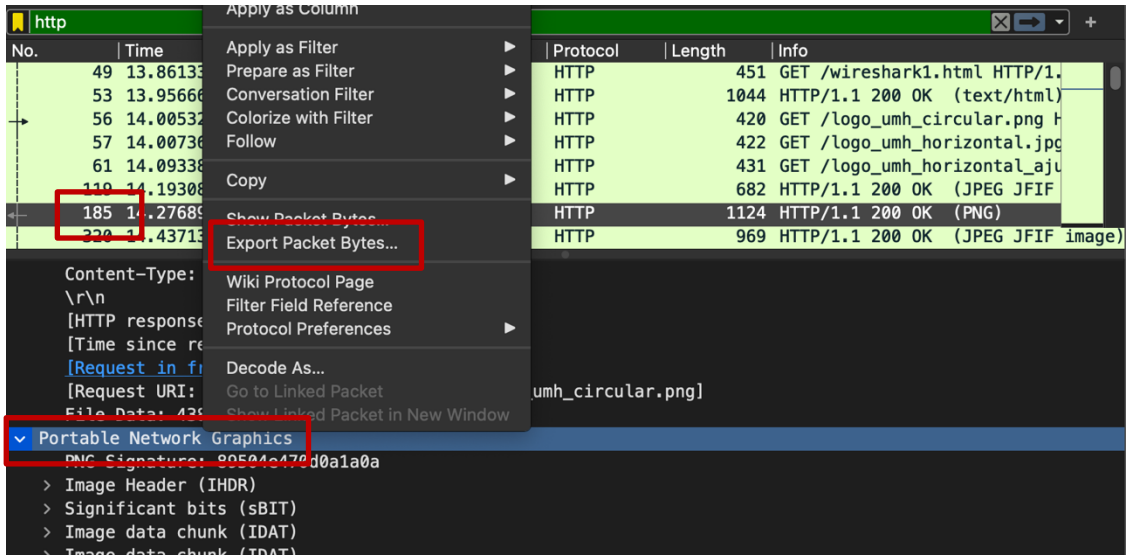




- El paquete 49 de la imagen anterior contiene la petición de la página web (`GET /wireshark1.html HTTP/1.1`)
- El paquete 53 contiene la respuesta del servidor (`HTTP/1.1 200 OK text/html`) e incluye el archivo HTML.
- Dado que esta página web incluye tres imágenes, los paquetes siguientes (56, 57 y 61) realizan peticiones (GET) para descargar cada una de las imágenes (`GET /logo_umh_circular.png`).

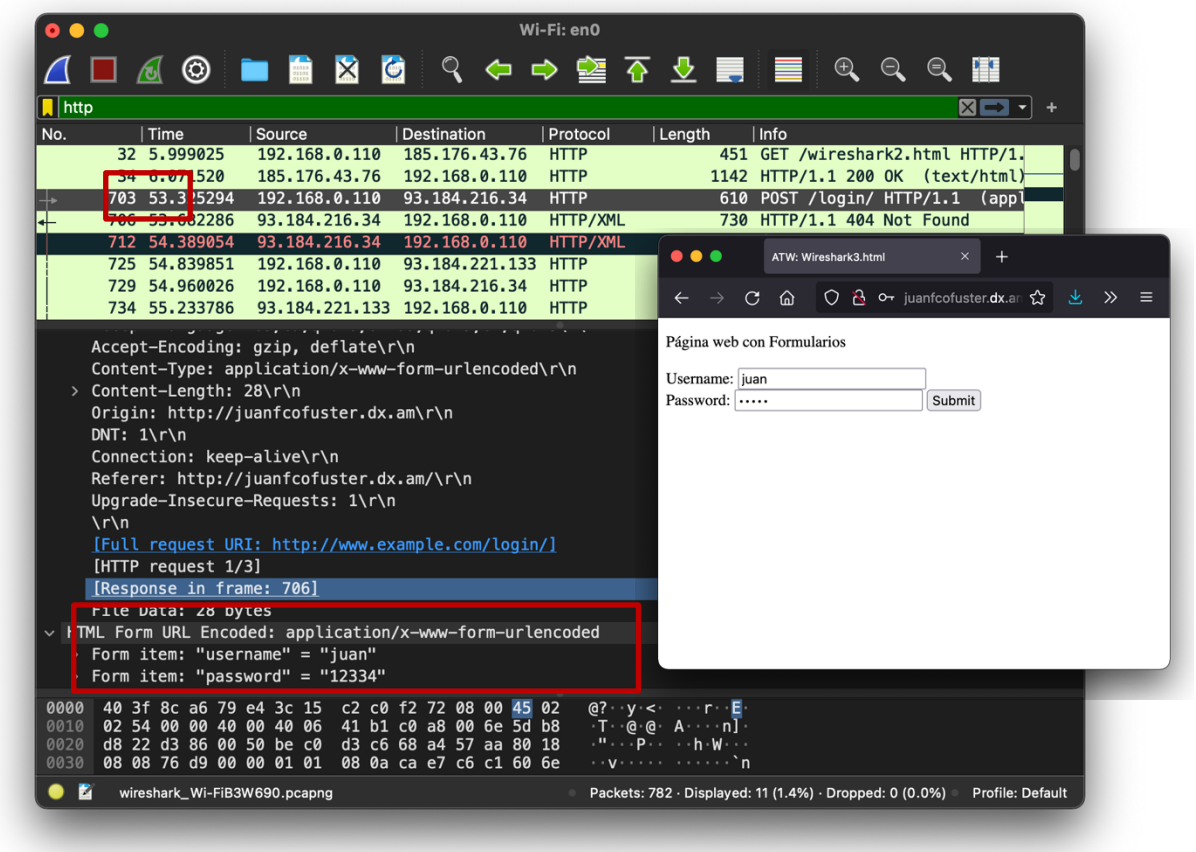
Wireshark es capaz de asociar peticiones y respuestas, así, en la imagen anterior podemos leer que la respuesta a la petición del paquete 56 se encuentra en el paquete 185 (`Response in frame: 185`).

Seleccionamos el paquete 185, buscamos la imagen descargada, imagen que está en formato PNG (`Portable Network Graphics`), que aparecerá tras la cabecera del servidor y, con el botón secundario, podemos exportar la imagen y guardar el archivo en nuestro ordenador. Repitiendo el mismo procedimiento podemos obtener las imágenes restantes.



Para terminar, realizaremos una nueva conexión a una página web que contiene un formulario de validación de usuario para acceder a una hipotética aplicación web: http://juanfcofuster.dx.am/wireshark2.html.

Al igual que antes, recuerda poner Wireshark a capturar paquetes nuevamente.



- En esta ocasión, el paquete 32 contiene la petición (`GET /wireshark2.html HTTP/1.1`)
- El paquete 34 contiene la respuesta del servidor con el archivo HTML (`HTTP/1.1 200 OK text/html`)
- El paquete 703 contiene los datos enviados por el usuario tras completar el formulario y pulsar el botón **Submit** (`POST /login/ HTTP/1.1 application/x-www-form-urlencoded`) en texto plano (`username = "juan" y password = "12334"`)

En este último ejemplo, dado que no se está utilizando una conexión cifrada entre cliente y servidor, los datos de validación del usuario se envían en texto plano a través de la red y son visibles para todos los usuarios que puedan capturar el paquete.

1. 2. El Protocolo HTTP

Este protocolo es el empleado para la comunicación entre el cliente y el servidor en la Web. Para poder desarrollar aplicaciones Web es necesario conocer su funcionamiento básico, la secuencia de conexiones que se establece entre cliente y servidor, y la información que intercambian entre ambos.

En esta segunda parte de la práctica se empleará el programa **Fiddler** y un pequeño programa, **ServidorHTTP**, implementado en Java que, simulando ser un cliente y un servidor web, permiten "espiar" la información que envía la otra parte.

El ciclo Petición / Respuesta

Para ver el funcionamiento de una petición/respuesta en HTTP se empleará el programa **Fiddler**, que permite efectuar peticiones manualmente, y observar la respuesta del servidor. Para ello descargar la última versión del programa de https://www.telerik.com/fiddler.

Ejecuta el programa Fiddler:

1. Utiliza la pestaña **Composer** para realizar una petición **GET** sobre la web `www.umh.es`.
2. Accede a la pestaña **Inspectors** para analizar la petición enviada por el cliente y **describir los resultados**.

Nótese que en esta petición **GET** sobre `www.umh.es` la "/" que aparece en la petición es equivalente a solicitar la página web principal del sitio, o fichero `index`.

3. Tras analizar la petición realizada mediante Fiddler, realiza la misma petición de la web `www.umh.es` mediante un navegador (por ejemplo, Google Chrome o Mozilla Firefox), **sin cerrar Fiddler**.

Se puede apreciar que aparece una nueva petición HTTP en Fiddler. Selecciona la nueva petición y, mediante la pestaña **Inspectors**, comprueba las diferencias entre la petición enviada anteriormente con Fiddler y esta última petición enviada por el navegador.

4. Completa la siguiente tabla con las diferencias, si existen, en relación con las siguientes partes de la petición: **línea de petición**, **cabeceras de cliente** y **cabeceras genéricas**.

PETICIÓN FIDDLER		PETICIÓN NAVEGADOR	
LINEA DE PETICIÓN:			
CABECERAS DE CLIENTE:			
CABECERA	**VALOR / UTILIDAD**	**CABECERA**	**VALOR / UTILIDAD**
Accept		**Accept**	
Accept-Languaje		**Accept-Languaje**	
Accept-Encoding		**Accept-Encoding**	
User-Agent		**User-Agent**	
Host		**Host**	
CABECERAS GENÉRICAS:			
CABECERA	**VALOR / UTILIDAD**	**CABECERA**	**VALOR / UTILIDAD**
Connection		**Connection**	

Para seguir **analizando la respuesta del servidor** tendremos que emplear la pestaña **Inspectors**. En la parte inferior veremos las pestañas **Headers** y **Body**, con las que podremos analizar la respuesta del servidor en cuanto a cabeceras, códigos de estado, etc. Y también el código HTML que el servidor devuelve al cliente.

5. Comentar los resultados obtenidos completando la siguiente tabla acerca de los siguientes aspectos de la respuesta: **código de estado**, **cabecera genérica**, **cabecera de servidor** y **cabecera de entidad**.

RESPUESTA DEL SERVIDOR	
CÓDIGO DE ESTADO (VALOR / UTILIDAD):	
CABECERAS GENÉRICAS	
CABECERA	**VALOR / UTILIDAD**
Date	
Connection	
CABECERAS DE SERVIDOR	
CABECERA	**VALOR / UTILIDAD**
Accpet-Ranges	
Server	
CABECERAS DE ENTIDAD	
CABECERA	**VALOR / UTILIDAD**
Content-Lenght	
Content-Type	
Content-Location	
Content-Languaje	

Ahora, copiar y pegar el **cuerpo de la entidad**, con el código HTML (desde la etiqueta **<html>** hasta el final de la respuesta) en un documento de texto plano y abrirlo con el navegador, para comprobar que se trata de una página web (aunque no estarán las imágenes, ni los formatos especificados por las hojas de estilo CSS, ni otro contenido externo al documento HTML).

Observar la petición del cliente

En el ejemplo anterior se ha realizado una petición **GET** de manera directa, sin enviar información adicional, pero la realidad es que los navegadores suelen incluir en su petición cabeceras con información variada. Para observar la petición de un navegador real, se empleará el programa **ServidorHTTP** (incluido en los archivos de la práctica).

Tras ejecutar nuestro **ServidorHTTP** (su ejecución requiere de la máquina virtual de Java disponible en el equipo):

1. Se puede elegir el puerto en el que debe "escuchar" nuestro servidor ficticio (por defecto es el 80, aunque habría que seleccionar otro si tuviéramos otro servicio escuchando ese puerto, como el IIS, Tomcat, Apache…).
2. Una vez elegido el puerto, pulsar el botón de **Esperar Petición**. El servidor quedará a la espera de la conexión de un cliente.
3. Para hacer dicha petición, se puede emplear cualquier navegador (por ejemplo: Google Chrome o Mozilla Firefox), en cuya barra de direcciones habrá que teclear: `http://localhost`.

Ya que nuestro "servidor ficticio" está situado en la máquina local, y **localhost** es el nombre que se le da a ésta en Internet.

En caso de haber puesto al servidor escuchando en un puerto distinto del 80, habría que ponerlo al final de la URL precedido de dos puntos, por ejemplo `http://localhost:8080`.

En el cuadro de texto denominado **petición del cliente**, debe aparecer la información enviada por el navegador. Observar que la primera línea es muy similar a la petición manual hecha en el apartado anterior. No obstante, el navegador incluye información adicional: formatos admitidos, idioma preferido para las páginas, nombre del programa, versión y sistema operativo en el que se está ejecutando...

En el recuadro de **respuesta del servidor** se puede escribir la información que queremos devolver al navegador. Para probarlo, teclear cualquier mensaje, y pulsar el botón **Enviar**. El texto aparecerá en la pantalla del navegador (esto no es del todo correcto según el protocolo HTTP, ya que la respuesta debería venir precedida de un código de estado, aunque casi todos los navegadores la tomarán como válida).

Cabeceras del servidor

El servidor puede, enviando las cabeceras apropiadas, proporcionar información sobre los datos que sirve y, en algunos casos, forzar al cliente a realizar determinadas acciones. Por ejemplo, la cabecera **Refresh** se utiliza cuando la información se ha cambiado de sitio y se quiere hacer que el navegador se redirija automáticamente a una nueva URL. Para comprobarlo, arrancamos nuestro **ServidorHTTP**, en la barra de URL del navegador tecleamos `http://localhost` y como respuesta del servidor escribimos:

```
HTTP/1.1 300
Refresh: 5;URL=https://www.umh.es
(línea en blanco)
En 5 segundos será redirigido a la U.M.H.
```

Como se habrá comprobado, la cabecera **Refresh** redirige al navegador automáticamente a una nueva URL pasado un número de segundos especificado. **¿Qué significado tiene el código de estado 300 que estamos enviando en el inicio de la respuesta del servidor?**

Cookies

Las **cookies** son datos que el servidor puede almacenar en el cliente. Se crean mediante la cabecera del servidor set-cookie (que no es parte del estándar HTTP pero que implementan prácticamente todos los navegadores y servidores). Esta cabecera admite una lista de cookies en el formato **nombre1=valor1;nombre2=valor2;...** Por ejemplo, teclear lo siguiente como respuesta a una petición HTTP:

```
HTTP/1.1 200 OK
Set-cookie:nombre=Juan
(línea en blanco)
La cookie nombre ha sido almacenada en su ordenador
```

Cuando una cookie se almacena en el navegador del cliente, éste la envía automáticamente de vuelta al servidor en las siguientes peticiones que se realizan al mismo sitio.

Para comprobarlo, hacer una nueva petición a localhost (cuidando de no cerrar la ventana del navegador desde que se almacenó la cookie, porque si no ésta se perdería) y observar que a las cabeceras adicionales que envía el navegador se añade una nueva cookie con los datos almacenados.

Se pueden crear cookies que persistan hasta una fecha fijada, aunque se cierre la ventana del navegador.

Explicar qué significado tiene el código de estado 200 que estamos enviando en el inicio de la respuesta del servidor.

Paso de parámetros

En muchas ocasiones, la URL que se teclea en el navegador no representa una página web estática, sino un programa que se debe ejecutar en el servidor y devolver su salida en forma de página Web. En ese caso, al programa llamado se le pueden pasar parámetros en la petición HTTP, con el formato: **GET nombre-programa?parametro1=valor1&parametro2=valor2&...**

Por ejemplo, cuando se rellena un formulario Web, y se pulsa el botón de envío, se suele llamar a un programa en el servidor que recibe como parámetros los datos tecleados en el formulario. Para comprobarlo, abrir en el navegador el fichero **formulario.html** que se proporciona como ejemplo, rellenar los campos con datos y **pulsar la tecla Intro** (empleando el programa **ServidorHTTP** para "espiar" la petición que hace el navegador). A continuación, revisar el código del fichero **formulario.html** e **indicar en qué parte del código se envían las variables mediante el método GET**. Observar también cómo pone el navegador en la petición URL lo que se ha escrito en los campos del formulario HTML.

1.3. Servidores y Aplicaciones Web

Un servidor Web puede devolver al cliente páginas estáticas, almacenadas previamente en el servidor y que no sufren ningún tipo de modificación, o páginas dinámicas generadas mediante alguno de los lenguajes utilizados para programar en la web en función de los parámetros recibidos en la petición. Estos lenguajes proporcionan una capa de alto nivel que evita tener que leer directamente el texto de la petición y de la respuesta HTTP.

Cada tecnología (JSP, PHP, ASP, JavaScript, ...) dispone de un API propio, normalmente orientado a objetos, que permite acceder de manera más sencilla a la información que

intercambian cliente y servidor. Además, en muchos casos, se implementan funcionalidades no disponibles directamente en el protocolo, por ejemplo, la posibilidad de almacenar objetos que se mantienen durante varios ciclos petición/respuesta (útil, por ejemplo, para implementar un carrito de la compra), lo que comúnmente se llama manejo de sesiones.

Las aplicaciones Web pueden ejecutar código en el cliente o en el servidor. En esta parte de la práctica veremos un ejemplo de los dos tipos de programas.

Aplicaciones Web en el servidor

No todos los servidores Web son compatibles con todas las tecnologías de programación Web. El estándar en este sentido es CGI, ya que la práctica totalidad de servidores existentes pueden ejecutar programas de este tipo, pero es una tecnología que se está abandonando en favor de otras más modernas como ASP, PHP o JSP. Aquí se verá un ejemplo de programa en JSP (que utiliza el lenguaje Java), por lo que se necesita un servidor compatible con esta tecnología. El más difundido de los servidores JSP es Tomcat.

El servidor *Tomcat*

Tomcat es un servidor Web compatible con las tecnologías de JSP y Servlets, ambas basadas en programas Java que se ejecutan en el servidor. El propio Tomcat está escrito en Java, por lo que es multiplataforma (aunque se pueden obtener versiones autoinstalables para sistemas específicos, por ejemplo, Windows). Tomcat es Open Source y la última versión puede obtenerse de https://tomcat.apache.org/ . Puede funcionar de forma independiente (**stand alone**) o como parte del servidor Apache, que será descrito en un tema posterior.

Para poner en marcha el servidor, utilizar el icono **Start Tomcat** que debe aparecer en el grupo de programas **Apache Tomcat X.0** (donde X es la versión). Aparecerá una ventana de MS-DOS en la que Tomcat indica cuándo está listo para recibir peticiones.

Por defecto, el servidor funciona en el puerto **8080**, por lo que, para acceder a su página principal, tendremos que ir al navegador y escribir en la barra de direcciones:

```
http://localhost:8080
```

Comprobar que la página principal se corresponde con el archivo index.html que, en la versión Windows, está en el directorio **C:\Program Files\Apache Software Foundation\Tomcat X.0\webapps\ROOT** (si se ha instalado Tomcat en el directorio por defecto, en cualquier caso, estará en webapps\ROOT).

Programas en JSP

JavaServer Pages (JSP) es una tecnología que permite crear páginas web dinámicas mediante la inclusión de código Java en páginas HTML. El código Java se ejecuta en el servidor al solicitar el cliente la página JSP, y genera en ese momento el HTML. El cliente recibe únicamente el HTML generado, con lo que el proceso del servidor es transparente para él.

Veamos un ejemplo sencillo de página JSP:

Ejemplo de página JSP. (Fichero saludo.jsp)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ page import = "java.util.Calendar" %>
<html>
    <head>
        <title>Saludo</title>
    </head>
    <body>
        <%
            Calendar ahora = Calendar.getInstance();
            int hora = ahora.get(Calendar.HOUR_OF_DAY);
        %>
        Hola mundo,
        <% if ((hora>20)||(hora<6)) { %>
             buenas noches
        <% }
          else if ((hora>=6)&&(hora<=12)) { %>
            buenos días
        <% }
          else { %>
            buenas tardes
        <% } %>
    </body>
</html>
```

Para probar el ejemplo, copiar el fichero **saludo.jsp** dentro de la carpeta ROOT de Tomcat, y solicitar el fichero al servidor mediante la dirección

```
http://localhost:8080/saludo.jsp
```

Para comprobar que el proceso del JSP es transparente al navegador, ver en el mismo el código fuente de la página. Se observará únicamente código HTML, resultado de la evaluación del código Java que contenía la página.

El ejemplo anterior genera un texto cuyo contenido varía en función de la hora actual. Obsérvese que la página contiene código Java incrustado entre símbolos **<%** y **%>**. Cuando el servidor web encuentra código HTML lo envía tal cual, al cliente, pero cuando encuentra el código Java, lo ejecuta (la primera vez que se solicita la página JSP, el código es compilado, por lo que la primera ejecución sufrirá un ligero retraso). Este código se puede utilizar para generar HTML como en el siguiente fragmento:

```
<% if ((hora>20)||(hora<6)) %>
    buenas noches
```

En este caso, se enviaría al cliente el texto "buenas noches" si se cumple la condición Java expresada en la primera línea. Una forma alternativa de hacerlo sería generando el HTML mediante un **println** de Java:

```
<% if ((hora>20)||(hora<6))
    out.println("buenas noches"); %>
```

Aquí se utiliza el objeto predefinido **out**, que representa la salida enviada al cliente. De este modo, la generación de HTML se introduce dentro del propio código Java (de hecho, JSP utiliza internamente este método, convirtiendo todo el HTML de la página en sentencias Java del tipo **`out.println`**). Probar ambas formas de enviar la salida al cliente.

Aplicaciones Web en el cliente

También se puede ejecutar código en el navegador web. En este caso, a diferencia del anterior, el navegador recibe el código del programa junto con el código HTML. Este código puede aparecer compilado o directamente como código fuente, dependiendo del lenguaje de programación. En el caso de JavaScript se recibe el código fuente, mientras que en el caso de los applets Java, se recibe ya compilado.

Para el siguiente ejemplo se deberá modificar el programa JSP anterior, con código JavaScript que pregunta el nombre al usuario para "personalizar" el saludo. Para ello, modificar el fichero anterior y llamarlo **saludo2.jsp**.

Página JSP con código JavaScript (Fichero saludo2.jsp)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%@ page import = "java.util.Calendar" %>
<html>
    <head>
        <title>Saludo</title>
    </head>
    <body>
        <%
            Calendar ahora = Calendar.getInstance();
            int hora = ahora.get(Calendar.HOUR_OF_DAY);
        %>
        Hola
        <script language="JavaScript">
        nombre = prompt("¿Cómo te llamas?");
        document.write(nombre); </script>
        <% if ((hora>20)||(hora<6)) { %>
            buenas noches
        <% }
        else if ((hora>=6)&&(hora<=12)) { %>
            buenos días
        <% }
        else { %>
            buenas tardes
        <% } %>
    </body>
</html>
```

El código adicional aparece sombreado. Si se copia el fichero al directorio ROOT de Tomcat y se solicita al servidor como se ha hecho antes, se puede observar que en el código fuente de la página, visto desde el navegador, ha "desaparecido" el código Java, pero **sigue visible el código JavaScript**.

Páginas estáticas

Aunque Tomcat es un servidor de Aplicaciones, también puede emplearse para servir **páginas web estáticas**, si bien, en este caso, es preferible emplear otros servidores que se concibieron en su origen para servir este tipo de páginas, como Internet Information Services (IIS) o Apache, que serán estudiados en profundidad más adelante. Como ejemplo de funcionamiento, copiaremos el fichero **saludoestatico.html** al directorio **ROOT** de Tomcat, y haremos la siguiente petición desde el navegador:

```
http://localhost:8080/saludoestatico.html
```

Si echamos un vistazo al código fuente desde el navegador, y lo comparamos con el fichero HTML original, podremos comprobar que no existe ninguna diferencia, pues al tratarse de una página estática, el servidor la ha enviado tal cual, sin realizar ninguna modificación.

Normas de entrega

Se deberá redactar memoria descriptiva con cada uno de los pasos empleados en la realización de la práctica. La fecha de entrega recomendada, así como el peso de la práctica se explicará en clase y se notificará con un mensaje en el campus virtual.

El documento deberá comprimirse en formato **zip** y se deberá subir a la tarea correspondiente del campus virtual.