El protocolo HTTP

El protocolo HTTP es un idioma básico formado por preguntas y respuestas que entienden todos los clientes web (es decir, los navegadores) y todos los servidores web (es decir, los programas que tienen almacenadas páginas web y que las ofrecen a los clientes que las solicitan). Aunque se utilicen diferentes clientes web (Firefox, Chrome, Safari, Spartan…) y en "el otro lado" esté funcionando distinto software de servidor web (Apache, Nginx, IIS…), el protocolo HTTP es un estándar que permite la comunicación entre ambos extremos de una forma universal. Concretamente, establece un conjunto muy específico de peticiones que un cliente web puede realizar, y un conjunto de respuestas predefinidas que un servidor web puede ofrecer a estas peticiones. Su versión más extendida -la 1.1- está desarrollada en el documento RFC 2616 (http://www.ietf.org/rfc/rfc2616.txt) y la más reciente -la 2.0- en el documento RFC 7540 (https://tools.ietf.org/html/rfc7540)

Formato de las peticiones, el método GET y manejo de URLs

Toda petición HTTP (realizada, por ejemplo, al escribir una dirección web en la barra de direcciones de un navegador y pulsar Enter) internamente no es más que un conjunto de líneas de texto. La primera línea siempre define el tipo de petición y tiene el formato siguiente: *<MÉTODO> <URI> <VERSIÓN>*. Las siguientes líneas (que podemos dividir por su significado y contenido entre "cabecera de la petición" y "cuerpo de la petición") aportan información adicional. Todas estas líneas se distinguen entre sí por su final, compuesto de los caracteres "\r\n" (retorno de carro más salto de línea).

El campo *<VERSIÓN>* simplemente indica la versión del protocolo utilizada para realizar la petición, y siempre tendrá como valor la cadena "HTTP/1.1". El campo *<MÉTODO>* representa la acción que el cliente pretende ejecutar sobre un determinado recurso indicado mediante el campo *<URI>* . Por ejemplo, lo más habitual -con diferencia- es que el cliente quiera obtener una página web; en ese caso, como *<MÉTODO>* se usará el verbo `GET` –en mayúsculas- y como *<URI>* se indicará la dirección de la página web a conseguir. Existen varios métodos posibles más que permiten al cliente interactuar con el servidor de otras formas, pero no los estudiaremos.

Tal como hemos dicho, el campo *<URI>* sirve, en el caso de usar el método GET, para indicar al servidor el recurso solicitado (el cual puede ser una página web, una fotografía…y en general, cualquier tipo de fichero). El formato de este campo (cuyo nombre viene de "Uniform Resource Identificator") está definido en el RFC 3986 pero, en la mayoría de ocasiones, utilizaremos un caso particular de URI llamado URL (de "Uniform Resource Locator") concretado en el RFC 1738 y que, en general, tiene este aspecto…:

protocolo://maquinaremota:puerto/ruta/recurso?c1=v1&c2=v2&c3=v3

…donde protocolo en nuestro caso siempre será la cadena "http" (aunque podría ser "https" si la conexión se estableciese sobre un canal cifrado), "maquinaremota" es el nombre DNS o dirección IP del servidor web, "puerto" es el número donde estará escuchando ese servidor (si dicho servidor usara el puerto nº80 no hará falta indicarlo), "/ruta/recurso" es la ruta dentro de la jerarquía de carpetas del servidor web donde se aloja el recurso en cuestión y por último, si ese recurso fuera de tipo dinámico (es decir, si fuera un programa ejecutable que cada vez pudiera retornar un resultado variable, como es por ejemplo una página PHP) y para solicitarlo/ejecutarlo se usara el método GET, se le podrían pasar parámetros de entrada mediante una cadena final con siguiente formato (fijarse en la sintaxis: la primera pareja clave/valor va precedida de "?" y después todas las demás están separadas entre sí por "&" ; a esta cadena de parejas clave/valor enviada al servidor para ser procesada se le llama "querystring").

?clave=valor&otraclave=otrovalor&otramás=suvalor&yotra=suvalor…

Por ejemplo, si escribimos en la barra de direcciones de un navegador una URI real tal como http://www.google.com/search?q=guapo y pulsamos Enter, éste generará automáticamente una petición HTTP de tipo GET (este método es el utilizado por defecto por los navegadores si no se indica lo contrario) para acceder a un recurso llamado "search" ubicado en el servidor "www.google.com" (en otras palabras, al buscador de Google), al cual le estaremos indicando una pareja clave-valor (concretamente, la pareja "q=guapo"); con esta querystring, el buscador de Google estará recibiendo la información necesaria para realizar una búsqueda (clave "q") usando la palabra "guapo". De hecho, podemos comprobar fácilmente cómo, si accedemos a la página del buscador de Google a través de un navegador normal, dependiendo de lo que escribimos en el cuadro de búsqueda, así se modifica la cola "search?q=" de la dirección visible en la barra de direcciones.

Es importante tener en cuenta que un computador que actúe como servidor web normalmente tiene sus recursos compartidos (páginas web, imágenes, documentos, etc) alojados dentro de una carpeta "raíz" determinada (establecida en la configuración del programa Apache/Nginx/IIS/etc.), cuya ruta real dentro de su sistema operativo no es conocida por el cliente porque este solamente "ve" a partir de esa carpeta en adelante. Por ejemplo, si esa carpeta tuviera de ruta real "/var/www/html/misitioweb" y ahí dentro hubiera una subcarpeta "imagenes" con todas las imágenes de nuestro sitio web, para solicitar una imagen llamada "mifoto.png", en el navegador deberíamos escribir http://www.miservidor.com/imagenes/mifoto.png y no http://www.miservidor.com/var/www/html/misitioweb/imagenesmifoto.png.

Hay que hacer notar también la diferencia de formato existente entre las URL escritas en la barra de un navegador (las cuales tienen el aspecto ya conocido de protocolo://maquinaremota…) y las URL que aparecen formando parte de la primera línea interna de una petición HTTP, en las que se omite el protocolo, el nombre/dirección del servidor y su puerto de escucha (quedando, por tanto, con un aspecto similar a "/ruta/recurso?c1=v1&c2=v2&c3=v3"). Esta diferencia es debida a que los valores omitidos en la URL de la petición HTTP ya son especificados al establecer la conexión entre cliente y servidor (proceso que es previo al envío de la petición propiamente dicha) y, por tanto, volverlos a indicar sería redundante. Por ejemplo, la primera línea de una petición HTTP de tipo GET que solicitara (a un determinado servidor ya contactado) una página llamada "index.html" ubicada directamente dentro de la carpeta "raíz", debería tener un aspecto como: `GET /index.html HTTP/1.1`. Así pues, y solo para acabar de aclarar este aspecto: ¿cuál debería ser la petición GET necesaria para obtener, por ejemplo, la página http://arduino.cc/en/Reference/HomePage? Respuesta: `GET /en/Reference/HomePage HTTP/1.1`

Finalmente, debemos saber que no todos los caracteres están permitidos en una URL: solamente podemos utilizar las letras del alfabeto inglés minúsculas y mayúsculas, los dígitos del 0 al 9, los caracteres -_.!*'() y ciertos caracteres con significado especial dentro de las URLs, los cuales son: espacio en blanco,",#,$,%,&,+,´,/,:,;,<,=,>,?,@,[,\,],{,|,} y ~. Si deseáramos escribir alguno de estos últimos sin mantener su significado especial (es decir, si quisiéramos, por ejemplo, que "?" dejara de marcar el inicio de la "querystring" para pasar a ser un simple interrogante) deberíamos "codificarlos" según la tabla mostrada a continuación:

Carácter	Codificación
	%20
"	%22
#	%23
$	%24

%	%25
&	%26
+	%2B
,	%2C
/	%2F
:	%3A
;	%3B
<	%3C
=	%3D
>	%3E
?	%3F
@	%40
[	%5B
\	%5C
]	%5D
{	%7B
|	%7C
}	%7D
~	%7E

Así pues, si una página web tuviera por ejemplo un espacio en blanco en su nombre (pongamos que se llama "datos clientes.html"), una URL válida incluiría dicho nombre transformado así: "datos%20clientes.html".

Las peticiones POST

Además del método GET, otro método ampliamente utilizado es el método `POST`. Este método está específicamente diseñado para enviar información desde el cliente (contenida en el cuerpo de la petición) hacia el recurso identificado por *<URI>* (ubicado en el servidor), para que éste la procese según lo tenga programado. Un caso típico es el envío de datos desde un formulario web hacia una página PHP.

Pero hemos visto que el método GET también permite que un cliente pueda enviar datos al servidor mediante la creación de una querystring al final de la URL del recurso solicitado. Entonces, ¿cuál es la diferencia entre ambos métodos a la hora de enviar datos al servidor? ¿Cuándo convendrá utilizar un método y cuándo otro? Pues la diferencia principal está en el lugar dentro de la petición donde se encuentran los datos a enviar al servidor: con el método GET se envían, tal como hemos dicho, dentro de la propia URL solicitada (en forma de querystring, y por tanto, visibles para todo el mundo en la barra de direcciones del navegador) y con el método POST se envían dentro del cuerpo de la petición (y por tanto, permanecen algo más internos).

Los datos enviados mediante peticiones GET están escritos en un formato llamado "URL-encode"; esto significa que cumplen dos requisitos: siguen la estructura –ya vista- de una querystring y usan las reglas de codificación de caracteres mostradas en la tabla anterior. Los datos enviados mediante POST pueden estar escritos igualmente en el formato "URL-encode" (es decir, seguir la estructura de querystring y estar codificados convenientemente -eso sí, dentro del cuerpo de la petición-) pero también pueden tener otro formato más específico llamado "multipart", diseñado especialmente para la transferencia de datos binarios (útil, pues, para la transmisión de ficheros) u otros más recientes, como JSON. El formato utilizado por defecto en todas las peticiones POST, de todas formas, es siempre el "URL-encode".

Cabeceras de las peticiones

Las peticiones HTTP no se componen solamente de una sola línea de tipo *<MÉTODO> <URI> <VERSIÓN>* sino que están formadas por más líneas, las cuales pueden formar parte o bien de la llamada "cabecera" de la petición o bien del "cuerpo" de la petición. La separación entre cabecera y cuerpo se realiza gracias a una línea en blanco (es decir, una línea que contiene solamente los caracteres "\r\n" y ninguno más) entre ambas secciones.

Las líneas de cuerpo solamente aparecen en peticiones de tipo POST porque son las que contienen los datos que el cliente envía al servidor (tal como ya se ha comentado).

Las líneas de la cabecera sirven para informar al servidor sobre detalles técnicos de la petición, permitiendo que éste pueda componer una respuesta más adecuada. Todas estas líneas tienen la forma *Nombre:Valor* (no se distinguen mayúsculas de minúsculas) y están definidas en el documento RFC 2616. Todas ellas son opcionales excepto una: la línea "Host:xxxx" (donde "xxxx" representa en este caso el nombre DNS -o dirección IP- del servidor web al que el cliente quiere conectar). A continuación se lista una (muy) breve selección de las líneas de cabeceras más comunes:

Host : Sirve para indicar, como hemos dicho, el nombre DNS (seguido de ":" y un nº de puerto si éste fuera diferente de 80) del servidor HTTP al que se le envía la petición. Por ejemplo, *Host: elpuig.xeill.net:4567* Esta cabecera puede parecer redundante, pero en casos donde un servidor HTTP con una única dirección IP tiene asociados varios nombres DNS (algo bastante habitual) esta cabecera permite localizar correctamente el recurso indicado en la URI. Es la única línea de cabecera obligatoria.

Accept : Sirve para indicar al servidor, separados por comas, qué tipos MIME (de los recursos solicitados) el cliente es capaz de aceptar. Los recursos cuyo tipo MIME no esté incluido en la lista especificada en esta línea de cabecera, serán rechazados por el cliente. Un ejemplo (que solamente acepta páginas HTML) podría ser este: *Accept: text/html* .
NOTA: Un tipo MIME es una cadena (cuyo formato está definido en el documento RFC 1341 (https://www.ietf.org/rfc/rfc1341.txt) que sirve para clasificar un recurso según su naturaleza: texto, imagen, audio... Cada tipo MIME consta de un tipo principal genérico ("text", "image", "audio"…) y un subtipo más concreto (adecuado al tipo principal) que indica el formato específico de dicho recurso. La lista oficial de tipos MIME está en la web de la organización internacional IANA (aquí: http://www.iana.org/assignments/media-types/media-types.xhtml) pero algunos de los tipos MIME más habituales son: *text/plain* (texto plano genérico), *text/html* (código HTML), *text/css* (código CSS), *application/javascript* (código Javascript), *application/json* (datos de tipo JSON), *application/x-www-urlencoded* (datos en formato "URL-encode"), *text/csv* (datos en formato CSV), *image/gif* (imagen GIF), *image/jpeg* (imagen JPG), *image/png* (imagen PNG), *application/pdf* (documento PDF), *application/gzip* (datos comprimidos de tipo Gzip) o *application/octet-stream* (datos sin estructura reconocida), entre otros. Para indicar en conjunto todos los subtipos de un tipo dado, se puede usar el símbolo especial "*", así, por ejemplo: *image/**. Igualmente, para indicar todos los tipos MIME posibles, se puede escribir */*.

Content-Type : Esta línea de cabecera solamente aparece en peticiones de tipo POST y sirve para informar al servidor sobre el tipo MIME de los datos enviados en el cuerpo de la petición (el cual suele ser "application/x-www-urlencoded" o "application/json").Si están en formato "URL-encode" (que es lo predeterminado), el valor de esta línea de cabecera será, por tanto, *Content-Type: application/x-www-form-urlencoded* . Siempre viene acompañada de otra línea de cabecera llamada **Content-Length**, la cual sirve para informar al servidor del tamaño (en bytes) del cuerpo de la petición (es decir, de los datos transferidos); un ejemplo: *Content-Length: 348*

Date : Sirve para informar al servidor de la fecha y hora en la que la petición fue enviada. Su valor se ha de expresar (en las pocas ocasiones que pueda ser necesario indicar esta línea de cabecera) en un formato (definido en el documento RFC 1123 (http://www.ietf.org/rfc/rfc1123.txt) que tiene el siguiente aspecto general: "díadelasemana, díadelmes mes año hora:minuto:segundo GMT". Por ejemplo, Date:Sat, 6 Jun 2016 10:10:10 GMT

Referer : Sirve para informar al servidor de la URI del recurso que fue solicitado justo antes del recurso actual. En la práctica, su valor suele ser la dirección de la página web que contenía el enlace que ha llevado a la petición vigente hacia la página actual. Por ejemplo, *Referer: http://www.rebelion.org*

User-Agent : Sirve para informar al servidor sobre qué programa concreto (y su versión) es el cliente que realiza la petición. Por ejemplo, *User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:32.0) Firefox/32.0* . Esta línea de cabecera nos puede venir bien para simular ser un determinado navegador y comprobar el comportamiento del servidor según este dato.

Códigos de las respuestas

La respuesta HTTP enviada por un servidor a un cliente (correspondiente a la petición hecha anteriormente por éste) también se compone internamente de varias líneas de texto terminadas con los caracteres "\r\n". La primera de estas líneas siempre define el tipo de respuesta ofrecida y tiene el formato siguiente: <*VERSIÓN*> <*CÓDIGO*> <*FRASE*>. Las siguientes líneas (que podemos dividir en cabecera de la respuesta y cuerpo de la respuesta) aportan, respectivamente, información sobre detalles más técnicos de la respuesta y el contenido propiamente dicho de ésta.

El campo <*VERSIÓN*> simplemente indica la versión del protocolo utilizada para realizar la respuesta, y siempre tendrá como valor la cadena "HTTP/1.1". El campo <*CÓDIGO*> es un número de tres dígitos que define qué tipo de respuesta se está proporcionando. El campo <*FRASE*> contiene una breve explicación textual del valor de <*CÓDIGO*>; el documento RFC 2616 da recomendaciones sobre estas frases, pero son opcionales y pueden cambiarse como se estime oportuno.

Los valores posibles del campo <*CÓDIGO*> se pueden clasificar en cinco categorías, dependiendo del resultado obtenido en el servidor tras el procesamiento de la petición. Estas categorías vienen identificadas por el dígito de más a la izquierda, sirviendo los otros dos para especificar más al detalle el tipo de respuesta concreta. A continuación presentamos algunos de los códigos de respuesta comunes:

1xx (Información) : El servidor indica que la petición ha sido recibida y que procederá a procesarla. Se usa en casos muy específicos que no veremos.

2xx (Éxito) : El servidor indica que la petición ha sido recibida y procesada con éxito. El código más típico de esta categoría es 200, obtenido cuando todo ha sido satisfactorio y el recurso solicitado es devuelto correctamente (si el método de la petición fue GET, se devuelve el recurso en sí; si fue POST, el resultado de la acción, etc).

3xx (Redirección) : El servidor indica que la petición no se ha completado y se deben realizar más acciones para conseguir finalizarla. Generalmente se usa como respuesta a peticiones GET para señalar un cambio de localización del recurso solicitado (es decir, una nueva URI de una página ya existente). Esta nueva localización se indica en una cabecera enviada por el servidor justo para ese propósito: *Location.* Si el cambio es permanente, se

emplea el código 301 y para peticiones siguientes el cliente debe usar la ubicación proporcionada en *Location*; si el cambio es temporal existen otros códigos (302, 303, 307…) y las siguientes peticiones deben seguir haciéndose a la URI original. Normalmente estos códigos son interpretados automáticamente por cualquier navegador actual, realizando la acción pertinente sin necesidad de notificarlo al usuario.

4xx (Error de cliente) : El servidor indica que la petición proveniente del cliente está mal formada o contiene errores. El código devuelto más común en estos casos es el 404, que indica que el servidor no ha encontrado el recurso solicitado en la URI de la petición. Pero hay muchos más: el código 400 indica que la sintaxis de la petición HTTP es errónea; el código 401 indica que el usuario no ha presentado las credenciales adecuadas para poder estar autorizado a acceder a un recurso protegido (el proceso de autenticación y autorización de recursos está definido en un documento RFC específico, el 2617 (https://www.ietf.org/rfc/rfc2617.txt); el código 403 indica que el recurso está protegido sin posibilidad de presentar credenciales, etc.

5xx (Error de servidor) : El servidor indica que, aunque la petición es correcta, ha fallado al procesarla. Los distintos motivos se señalan con un código concreto. Por ejemplo, 500 avisa de un error indeterminado o 501 indica que el servidor no soporta el método de la petición, entre otros.

Cabeceras de las respuestas

Las respuestas HTTP no se componen solamente de una sola línea de tipo *<VERSIÓN> <CÓDIGO> <FRASE>* sino que, tal como ya se ha dicho, están formadas por más líneas, las cuales pueden formar parte o bien de la llamada "cabecera" de la respuesta o bien del "cuerpo" de la respuesta. La separación entre cabecera y cuerpo se realiza gracias a la existencia de una línea en blanco (es decir, una línea que contiene solamente los caracteres "\r\n" y ninguno más) entre ambas secciones.

Las líneas de cuerpo solamente aparecen en respuestas a peticiones de tipo GET y son el contenido del recurso propiamente solicitado (es decir, lo que habitualmente suele ser el código de una página web). Este contenido es el que los navegadores obtienen, interpretan y, como resultado, muestran en pantalla. Así pues, aunque el cuerpo de las respuestas a peticiones GET no está ceñido a ningun formato en particular; en cada respuesta el servidor deberá indicar al cliente el tipo MIME concreto del contenido enviado (mediante la directiva *Content-Type*, que veremos enseguida) para que dicho cliente lo pueda interpretar convenientemente al recibirlo.

Las líneas de la cabecera sirven para informar al cliente sobre detalles más técnicos de la respuesta, permitiendo que éste pueda obtener una información más completa sobre el recurso en cuestión. Todas estas líneas tienen la forma *Nombre:Valor* (no se distingue entre mayúsculas y minúsculas) y también están definidas en el documento RFC 2616. A continuación se lista una breve selección de las líneas de cabeceras más habituales en las respuestas de los servidores web:

Connection : Sirve para informar al cliente sobre si la conexión TCP creada por el servidor para enviar la respuesta HTTP continuará abierta o no. Si es que sí (valor k*eep-alive)*, esa conexión TCP podrá ser reutilizada para recibir más peticiones HTTP posteriores del mismo cliente; si no (valor *close*), será cerrada y para enviar una nueva respuesta HTTP el servidor tendrá que crear una nueva conexión.

Content-Type : Sirve, en respuestas a peticiones GET, para informar al cliente sobre cuál es el tipo MIME del contenido del cuerpo del recurso devuelto. También informa (pero sólo si el tipo MIME es textual, como *text/plain* o *text/html*) de su sistema de codificación (ASCII,

UTF-8,etc), para que el cliente pueda mostrar dicho contenido correctamente. Esto último se indica mediante la palabra especial "charset" tras un punto y coma, así: *Content-Type: text/html; charset=UTF-8* (si el sistema de codificación no se indicara, el cliente empleará uno por defecto que dependerá de su configuración, el cual puede no coincidir con el del recurso). Salvo raras excepciones, UTF-8 es el sistema de codificación recomendado por ser el más versátil, óptimo y compatible entre clientes

Content-Length : Sirve para informar al cliente del tamaño (en bytes) del contenido del cuerpo del recurso devuelto (una vez comprimido, si es el caso). Si el valor de la línea *Connection* es *close*, el valor de esta línea es fácilmente calculable y se puede indicar directamente, así: *Content-Length: 348* . No obstante, si *Connection* es *keep-alive* (por ejemplo cuando se ofrece un fichero en "streaming"), el servidor no sabe de antemano este dato, por lo que esta línea de cabecera suele ser sustituida por la línea *Transfer-Encoding: chunked*, la cual indica al cliente que el recurso es enviado "a pedazos" progresivamente hasta enviar una marca final.

Date : Sirve para informar al cliente de la fecha y hora en la que la respuesta fue generada. Su valor se ha de expresar en un formato definido en el documento RFC 1123 (igual que ocurre con la línea de cabecera de cliente homónima).

Expires : Sirve para informar al cliente de la fecha y hora en la que la respuesta dada se considerará caducada. Su valor se ha de expresar en un formato definido en el documento RFC 1123.

Location : Sirve para indicar al cliente la nueva URI de un recurso cuando la URI solicitada (correspondiente a ese recurso) ya no es válida. En general, gracias a esta información un navegador actual será capaz de redireccionar automáticamente su petición a la URI recién indicada. Por ejemplo, *Location: http://www.fbi.gov*

Server : Sirve para informar al cliente del nombre y sistema del servidor. Por ejemplo, *Server: Apache/2.4 (Unix)*

Pruebas en un computador con Ncat y Curl

La mejor manera de comprender el funcionamiento interno del protocolo HTTP es utilizando un cliente TCP simple como `netcat` (ya instalado por defecto en Ubuntu) porque con él deberemos generar las peticiones HTTP "a mano" y recibiremos las respuestas HTTP "tal cual". Así, pues, si abrimos un terminal y, por ejemplo, ejecutamos el comando `netcat www.rclibros.es 80` estaremos conectando con el servidor web oficial de RCLibros, pero tendremos que indicarle qué queremos de él: el cursor nos estará indicando que debemos escribir la petición deseada (hay que ir rápido: si nos entretenemos el servidor cortará la conexión). Un ejemplo de petición podría ser éste:

```
GET / HTTP/1.1
Host:www.rclibros.es
```

donde la URI "/" indica que queremos obtener la página inicial por defecto. Tras pulsar dos veces Enter (para dejar una línea en blanco indicando que la petición no tiene más cabecera ni cuerpo), netcat enviará dicha petición y al instante deberíamos ver por pantalla cómo obtenemos dos cosas: las líneas de cabecera devueltas por el servidor y tras una línea en blanco, el código de la página web solicitada (esto es, el de la página inicial del sitio "www.rclibros.es"). Más concretamente, las líneas de cabecera más relevantes obtenidas con la petición de ejemplo anterior son:

HTTP/1.1 301 Moved Permanently

Date: Sun, 23 Aug 2015 18:27:08 GMT
Server: Apache
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Location: http://rclibros.es/
Content-Type: text/html; charset=UTF-8
Content-Length: 0

De las líneas anteriores se puede deducir que el servidor web Apache en realidad no nos está ofreciendo ningún contenido -es decir, no obtenemos ningún código de ninguna página en el cuerpo de la respuesta-; tan solo nos indica que la página inicial de RCLibros ya no se encuentra disponible en "www.rclibros.es" sino que deberemos dirigirnos a "rclibros.es" Si probamos entonces de escribir `netcat rclibros.es 80` y en la consola que se nos ofrece...:

```
GET / HTTP/1.1
Host: rclibros.es
```

...obtendremos, ahora sí, el código fuente de la página principal de RCLibros.

Notar que netcat es un simple cliente TCP que no entiende el contenido del recurso obtenido al realizar una petición GET (lo que normalmente es el código de una página web); así que lo que hace es simplemente "vomitarlo" por pantalla. Si la obtención de esta página la hubiera realizado un navegador, éste sí que habría procedido seguidamente a interpretar dicho contenido y el resultado de dicha interpretación lo habría "renderizado" gráficamente, mostrando finalmente la página web tal como la visualiza un usuario estándar.

Usar netcat para estudiar tráfico HTTP puede resultar un poco tedioso. Para una interacción con servidores web más cómoda y rápida, es más adecuado emplear un cliente HTTP propiamente dicho, tal como el comando **Curl** (http://curl.haxx.se), aplicación de terminal libre, gratuita y multiplataforma. Este programa es capaz de realizar peticiones HTTP de cualquier tipo a un servidor dado, mostrando la respuesta obtenida por pantalla. A continuación se muestran algunos casos de uso comunes de esta herramienta (las URL son fictícias, pero los ejemplos de peticiones GET se pueden probar contra cualquier dirección web y los de peticiones POST se pueden probar con un servidor de prueba diseñado precisamente para estos menesteres, disponible públicamente en http://www.posttestserver.com/post.php) :

- Para realizar una petición de tipo GET (con querystring) y obtener el contenido del recurso solicitado: `curl -X GET http://www.a.com/blog/?fecha=1979`

- Para realizar una petición de tipo GET (y obtener el contenido del recurso solicitado) enviando además líneas de cabeceras de cliente personalizadas: `curl -X GET http://www.a.com -H "Nombre=Valor" H "otra..."`

- Para realizar una petición GET (y obtener el contenido del recurso solicitado) mostrando, además, las líneas de cabecera enviadas por Curl de forma predeterminada (las cuales aparecerán precedidas del símbolo ">"): `curl -X GET http://www.a.com -v`

- Para realizar una petición de tipo GET y visualizar por la salida estándar, además del contenido del recurso solicitado, las cabeceras de respuesta del servidor (si tan solo nos interesaran estas últimas y no quisiéramos por tanto obtener el contenido del recurso solicitado, deberíamos sustituir el parámetro –i por –I): `curl -X GET http://www.a.com -i`

- Para realizar una petición de tipo GET y seguir las indicaciones de una posible línea *Location* presente en la cabecera de la respuesta del servidor (por defecto, Curl no realiza

ninguna redirección): curl -X GET http://www.a.com -L

• Para realizar una petición de tipo GET y obtener el contenido del recurso solicitado, guardándolo en un fichero (en otras palabras, para descargar una página web): curl -X GET http://www.a.com/pag.html -o fichero.html

•Para realizar una petición de tipo POST enviando datos en formato "URL-encode" dentro de su cuerpo (Curl establece automáticamente el valor de la cabecera de cliente *Content-Type* a *application/x-www-form-urlencoded*): curl -X POST http://www.a.com/login -d "user=ana&pass=1234" (si los datos están dentro de un fichero de texto, el parámetro -d se ha de escribir así: -d @fichero.txt)

Curl tiene muchísimas más posibilidades de las mostradas en la lista anterior (como el poder pausar/reanudar una descarga, o el limitar el ancho de banda usado en una descarga, o el utilizar combinaciones de nombres en URIs, o el realizar múltiples descargas a la vez, etc, etc). Para un conocimiento más exhaustivo de éstas, remito a su página oficial, donde hay disponible una documentación excelente.

Si Curl aún nos siguiera pareciendo demasiado farragoso, para estudiar más cómodamente el comportamiento del protocolo HTTP podríamos utilizar las funcionalidades diseñadas para ello que están integradas en los navegadores más importantes. Por ejemplo, Firefox ofrece un panel llamado "Red" (disponible con la combinación de teclas CTRL+SHIFT+Q) que, entre otras cosas, permite conocer en tiempo real las cabeceras de las peticiones que son realizadas (¡incluso editarlas y volverlas a enviar!) y las cabeceras de las respuestas correspondientes. Chrome ofrece una funcionalidad parecida mediante el menú "Red" de su consola de desarrollador.