

UNIDAD 3

Servicio HTTP

1. Servidores web

1.1. ¿Qué es HTTP?

El protocolo de transferencia de hipertexto o HTTP (*HyperText Transfer Protocol*) es un protocolo de la capa de aplicación que permite distribuir y compartir información entre sistemas mediante páginas web.

Este protocolo fue desarrollado por Sir Timothy Berners-Lee y su equipo. Crearon asimismo el lenguaje de etiquetas de hipertexto HTML (*HyperText Markup Language*) y el sistema de localización de objetos en la web o URL (*Uniform Resource Locator*).

A partir de 1990 HTTP se convirtió en el protocolo de la *World Wide Web* o WWW (Red Global Mundial), también creada por Berners-Lee, que representa la unión de hipertexto e Internet.

El IETF (Internet Engineering Task Force) o Grupo Especial sobre Ingeniería en Internet ha trabajado en la versión de HTTP 2.0 que intenta mejorar la velocidad y la conexión con dispositivos de telefonía móvil.

1.2. Localizador uniforme de recursos (URL)

Un localizador uniforme de recursos o URL se compone de una serie de elementos fijados por un estándar que se usan para nombrar recursos en Internet y permitir la localización o identificación de los mismos. Un URL identifica un único recurso.

Un navegador web localiza y muestra un recurso de forma adecuada mediante la introducción de un URL. Este último contiene diversa información que permite realizar de manera adecuada el proceso descrito.

El formato general de un URL es:

esquema://usuario:contraseña@máquina:puerto/directorio/archivo

Donde:

- **Esquema:** especifica qué protocolo debe ser usado para acceder a cada documento. Como, por ejemplo: HTTP, HTTPS, mailto, FTP... Utiliza los separadores “//” o “:/”.
- **Usuario:** indica el usuario. Le sigue como separador el símbolo de dos puntos (:).
- **Contraseña:** indica la contraseña. A continuación, se añade como separador una arroba (@).
- **Máquina:** indica el nombre del equipo donde está alojado el contenido. Puede ser una FQDN, una dirección IP o un nombre de dominio.
- **Puerto:** indica el puerto de escucha. Va precedido por el separador dos puntos (:).
- **Directorio:** consta de una secuencia de directorios separados por barras que definen la ruta a seguir para llegar al documento.
- **Archivo:** indica el nombre del archivo que contiene el recurso.

En un URL no tiene por qué aparecer todos los campos, como vemos en los siguientes ejemplos:

<http://ejemplo.com/recurso.html>
<http://empleado:123abc@192.168.100.1:80/alberto/a.html>

Donde tenemos:

Esquema	Usuario	Contraseña	Máquina	Puerto	Directorio	Archivo
http://			ejemplo.com			/recurso.html
http://	empleado	123abc@	192.168.100.1	:80	/alberto	/a.html

URN y URI

Cuando se habla de URL aparecen dos conceptos directamente relacionados con él: URN y URI.

- **URN** (*uniform resource name*) o nombre de recurso uniforme: identifica recursos en Internet, al igual que el URL, pero a diferencia de este último no indica cómo acceder a ellos. Un ejemplo de URN sería el ISBN de un libro, que identifica la obra, pero no nos indica en qué librería podemos conseguirla.
- **URI** (*uniform resource identifier*) o identificador uniforme de recurso: añade opcionalmente los campos *pregunta* y *fragmento* al final del URL:
 - *Pregunta*: son una o más variables separadas por punto y coma (;). Van precedidas por el separador "?". Con el separador "=" indicamos el valor de la variable.
 - *Fragmento*: es una zona del documento final. Va precedido del separador "#".

El URI puede ser clasificado como un localizador, un nombre o ambos.

En futuras especificaciones se recomendará el uso del término URI en lugar de URL y URN, que son más restrictivos.

A continuación, se muestran algunos ejemplos de URI:

```
foo://ejemplo.com:XXX/over/ther?name=exemple#nose
ftp://ftp.is.co.za/rfc/rfc1808.txt
http://www.ietf.org/rfc/rfc2396.txt
ldap://[2001:db8::7]/c=GB?objectClass?one
mailto:Jose.Dolz@ejemplo.com
news:comp.infosystems.www.servers.unix
telnet://192.0.2.16:80/
urn:oasis:names:specification:docbook:dtd:xml:4.1.2
```

Niveles del protocolo TCP/IP	
Aplicación	Transporte
HTTP	TCP (80) UDP (80)

Hipertexto

En informática, es el texto que aparece en pantalla y que permite acceder a otros recursos: textos, imágenes, gráficos, etc; relacionados mediante pulsaciones de ratón o teclado.

Actividad resuelta 3.1

Puertos utilizados por HTTP

Objetivo: se trata de conocer los puertos que utiliza el protocolo HTTP.

Descripción: acceder a los archivos que contienen información sobre los puertos utilizados por los servicios.

Linux

Ejecuta el siguiente comando en Linux:

```
$cat /etc/services | grep http
```

```
profesor2server@ubuntuserver:~$ cat /etc/services | grep http
# Updated from http://www.iana.org/assignments/port-numbers and other
# sources like http://www.freebsd.org/cgi/cvsweb.cgi/src/etc/services .
http      80/tcp      www          # WorldWideWeb HTTP
https     443/tcp     MCom         # http protocol over TLS/SSL
http-alt   8080/tcp    webcache    # WWW caching service
http-alt   8080/udp
```

Windows

Abre el archivo C:\WINDOWS\system32\drivers\etc\services en Windows para comprobar que son los mismos puertos estándar:

```
C:\Users\rafae>type C:\WINDOWS\system32\drivers\etc\services | find "http"
http      80/tcp      www www-http      #World Wide Web
https     443/tcp     MCom         #HTTP over TLS/SSL
https     443/udp     MCom         #HTTP over TLS/SSL
http-rpc-epmap 593/tcp     #HTTP RPC Ep Map
http-rpc-epmap 593/udp     #HTTP RPC Ep Map
```

1.3. Funcionamiento del protocolo HTTP

HTTP es un protocolo de pregunta/respuesta basado en el modelo cliente-servidor. Estos son los pasos que sigue en su funcionamiento:

1. El navegador web envía un mensaje de petición al servidor web.
2. El servidor que contiene los recursos o almacena los documentos envía un mensaje de respuesta.

Un navegador web o cliente web se denomina *User Agent* o UA (agente de usuario) y la información transmitida se llama recurso. Este último, como se ha visto, se identifica con un URL.

Mensajes HTTP

El mensaje HTTP contiene el estado de la solicitud y puede incluir cualquier información que pida el cliente:

- **Mensaje de petición:** el cliente indica la acción que quiere realizar, el recurso sobre el que se quiere realizar esta acción y cualquier otro dato necesario para que el servidor pueda atender la petición.
- **Mensaje de respuesta:** el servidor incluye en este paquete tanto información necesaria para el funcionamiento del protocolo, como, por ejemplo, el estado de la petición, así como el recurso solicitado por el cliente.

Sesión HTTP

Una sesión HTTP consiste en una serie de transacciones de red entre el cliente y el servidor. El cliente realiza una petición y lleva a cabo una conexión TCP estable con el puerto 80 del servidor, que

permanece a la escucha. El servidor procesará la información y transmitirá una respuesta compuesta por un mensaje de estado (en HTTP v1.1: 200 OK) y un mensaje con el recurso solicitado.

El mensaje de solicitud consiste en lo siguiente:

- Línea de solicitud (por ejemplo, *GET HTTP/1.1/images/logo.gif*, la cual solicita un recurso llamado *images/logo.gif* del servidor).
- Cabeceras (por ejemplo, *Accept-Language: en*).
- Una línea vacía.
- Un cuerpo de mensaje opcional.

La línea de solicitud y la cabecera deben acabar con <CR> <LF> (es decir, un retorno de carro seguido de un fin de línea). La línea vacía debe consistir en solo <CR> <LF>.

En HTTP v1.0, cuando un cliente se comunica con un servidor, se abre una conexión. Una vez el servidor le ha contestado, se cierra.

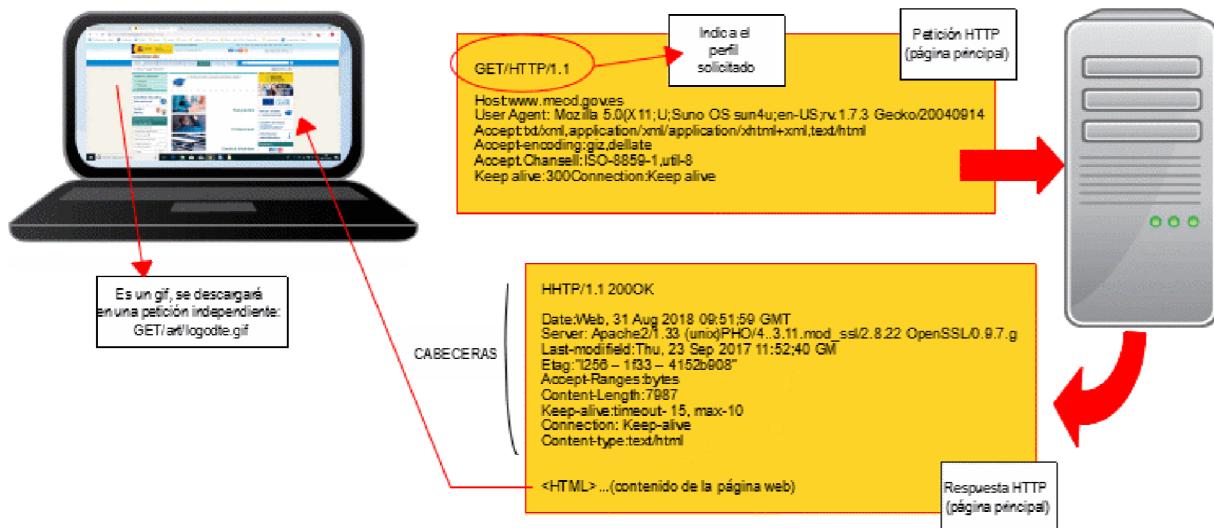
En HTTP v1.1 aumenta la velocidad al permitir que se realice por una misma conexión más de una transacción. Esto evita tener que volver a negociar la conexión TCP una vez enviada la primera petición.

Métodos de petición

El método indica la acción que se quiere realizar con el recurso. HTTP v1.1 define ocho métodos diferentes:

- **GET**: solicita una representación del recurso especificado. Mediante este método el cliente solicita, por ejemplo, una página HTML (index.html) o una imagen (foto.jpg).
- **POST**: envía datos para ser procesados (por ejemplo, un documento HTML) al recurso identificado. Los datos se incluyen en el cuerpo de la petición. Esto puede causar la creación de un nuevo recurso y/o las actualizaciones de recursos existentes. POST puede usarse para enviar a un servidor web los datos de un formulario.
- **HEAD**: pide una respuesta idéntica a la que correspondería a una petición GET, pero sin el cuerpo de respuesta (es decir, sin que nos devuelvan la página HTML o la imagen, por ejemplo). Es útil para recuperar la información escrita en cabeceras de respuesta, sin necesidad de llevar el contenido entero.
- **PUT**: carga una representación del recurso especificado.
- **DELETE**: suprime el recurso especificado.
- **TRACE**: provoca que el servidor devuelva en la respuesta la petición recibida. Se suele usar para tareas de diagnóstico.
- **OPTIONS**: devuelve los métodos HTTP que el servidor suministra para un URL especificado. Puede usarse para comprobar la funcionalidad de un servidor web solicitando "*" en vez de un recurso especificado.
- **CONNECT**: convierte la conexión de solicitud en un túnel transparente TCP/IP, por lo general para facilitar la comunicación SSL cifrada (el denominado protocolo HTTPS, que veremos más adelante) a través de un proxy HTTP no encriptado.

Veamos un ejemplo de funcionamiento de HTTP:



- Las peticiones y las respuestas llevarán unas cabeceras y, opcionalmente, un contenido (datos).
- Primero se descarga la página, y luego cada gif u objeto que haya en ella.
- Cada petición es independiente; si se usa HTTP/1.0 se hace en conexiones TCP distintas, si se usa HTTP/1.1 se pueden hacer varias en una misma conexión.

Códigos de estado

Desde la introducción de HTTP/1.0, la primera línea de la respuesta HTTP se denomina línea de estado e incluye un código de estado numérico (por ejemplo "404") y una frase explicativa textual (por ejemplo "No encontrado"). Existen una serie de frases explicativas estándar que pueden ser cambiadas, en todo caso, por el desarrollador web.

Veamos algunas de ellas:

Contenido	Descripción
100	Continuar.
101	Comutación de protocolos.
200	Ok.
201	Creado.
202	Aceptada.
300	Múltiples opciones.
301	Movido permanentemente.
302	Renombra un fichero remoto.
400	Solicitud incorrecta.
401	No autorizado.
402	Pago requerido.
500	Error interno del servidor.
503	Servicio no disponible.

Actividad resuelta 3.2

Esquema de una comunicación

Objetivo: conocer la estructura de un proceso de comunicación normal utilizando el protocolo HTTP.

Descripción: simular un proceso de comunicación con el protocolo HTTP.

Desarrollo

- El programa cliente establece la conexión con un servidor web.

Una forma de conocer más a fondo el esquema de comunicación es utilizar un cliente de telnet para conectarse a un servicio HTTP y enviar “a mano” los comandos necesarios. El servidor responderá en texto plano. Por ejemplo, desde una shell Linux puede teclearse *telnet localhost 80* para comenzar la comunicación. El problema es que muchos servidores web tienen el servicio telnet cerrado.

En el caso práctico se hace el telnet a **www.cordoba.es**, aunque puedes probar con cada una de las provincias andaluzas (alguna está protegida y no da información).

- Una vez dentro del servidor, se puede solicitar información sobre las cabeceras utilizando HEAD o algún documento utilizando GET. En nuestro caso pedimos información de cabeceras. Por ello escribimos:

```
alumno2smr@ubuntuserver:~$ telnet www.cordoba.es 80
Trying 45.60.34.3...
Connected to 49thliw.x.incapdns.net.
Escape character is '^].
HEAD /
HTTP/1.1 408 Request Timeout
Content-Type: text/html
Cache-Control: no-cache, no-store
Connection: close
Content-Length: 697
X-Iinfo: 5-24855270-0 0NNN RT(1635770997995 6077) q(-1 -1 -1 -1) r(60 -1) b2

<html style="height:100%"><head><META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW"><meta name="format-detection" content="telephone=no"><meta name="viewport" content="initial-scale=1.0"><meta http-equiv="X-UA-Compatible" content="IE=edge,chrome=1"></head><body style="margin:0px;height:100%"><iframe id="main-iframe" src="/_Incapsula_Resource?CWUDNSAI=0&xinfo=5-24855270-0%200NNN%20RT%281635770997995%206077%29%20q%28-1%20-1%20-1%29%20r%2860%20-1%29%20b2&incident_id=0-170988862891955973&edet=4&cinfo=ffffffff&rinfo=0&mth=HEAD" frameborder=0 width="100%" height="100%" marginheight="0px" marginwidth="0px">Request unsuccessful.
Incapsula incident ID: 0-170988862891955973</iframe></body></html>Connection closed by foreign host.
```

El servidor responde con una línea de estado, que incluye la versión de protocolo, un código de éxito o error y el texto explicativo a dicho código. Además, se enviarán varias cabeceras HTTP adicionales.

- El programa servidor cierra la conexión.

Connection closed by foreign host.

Gestión del estado de las conexiones. Cookies

HTTP no tiene memoria, por lo tanto, no guarda información de transacciones antiguas del cliente. Para ello deberemos utilizar un sistema externo: las llamadas cookies.

Una **cookie** es un conjunto de datos que recibe el cliente y almacena a petición del servidor web. Fueron desarrolladas por la empresa Netscape en el año 1994.

El servidor, gracias a las cookies, puede saber si un cliente se ha validado o no con antelación a la petición actual. De este modo, puede brindarle servicios propios de usuarios registrados, como acceso a zonas privadas o configuración de entornos. También se puede hacer un seguimiento del usuario a través del sitio web. De este modo es posible generar estadísticas de uso del sitio.

Existen dos tipos de cookies:

- **Origen:** habilitadas por el sitio que estamos visitando.
- **Third-Party cookies** o cookies a terceros: producidas por anuncios y otros elementos externos al sitio visitado. Estas permiten realizar un seguimiento de los usuarios a través de la web recopilando información acerca de sus preferencias. Estos datos permitirán realizar perfiles autorizados sobre los gustos de los usuarios. Existen leyes en diferentes países que las regulan.

Los usuarios pueden visualizar las cookies almacenadas por el UA y borrarlas. También puede configurar su recepción, permitiendo bloquear el acceso a determinadas cookies.

El nombre de "cookie" (galleta) proviene de una comparación con lo que los estadounidenses llaman "fortune cookie" (galletas de la suerte) debido a la información que hay dentro.

1.4. Los tipos MIME

Los tipos MIME (del inglés *Multipurpose Internet Mail Extensions*, extensiones multipropósito de correo de Internet) son una forma abierta y extensible de representar el contenido de los datos. Se trata de un estándar que especifica cómo debe transferir un programa los archivos multimedia (vídeo, sonido, o cualquier archivo que no esté codificado ASCII).

Como su nombre indica, en un primer momento fueron utilizados para extender las características del correo electrónico.

Hoy su uso se ha generalizado, por lo que también se les puede llamar IMT (*Internet Media Types*). MIME adjunta un archivo de cabecera a los documentos, en el cual indica el tipo de contenido del archivo. Esto permite al servidor web y al navegador manejar y mostrar correctamente los datos.

El tipo MIME indica el tipo de archivo que se está transfiriendo del servidor web al cliente o navegador web.

Los MIME se componen de tipos (indicados antes del carácter "/") y subtipos (indicados después del carácter "/"). Por ejemplo, son tipos MIME:

- **Text/html.** Define todos los archivos de texto que contienen código HTML.
- **Video/mpeg.** Detalla todos los archivos de imagen almacenadas en cualquier formato MPEG.
- **Image/*.** Define todos los archivos de imagen almacenados en cualquier formato: GIF, JPEG, BMP, etc.

El registro de los tipos MIME es de gran importancia, ya que asegura que dos tipos de contenido distintos nunca acabarán con el mismo nombre.

El prefijo especial x- queda reservado para tipos experimentales (desplegados sin que haya terminado el proceso de registro) o tipos de uso interno de organizaciones, por ejemplo: *image/x-fwf*.

El protocolo HTTP usa los tipos MIME en sus cabeceras con diferentes objetivos:

1. Informar al cliente del tipo de datos que está recibiendo del servidor. Esto se logra mediante el encabezado *Content-Type*. Así, un navegador típico puede manejar los datos de tres formas distintas según el tipo MIME indicado en *Content-Type*:
 - o Visualizar el documento, por ejemplo, con tipos *text/html*.
 - o Llamar a una aplicación externa, por ejemplo, con tipos *application/pdf*.
 - o Preguntar al usuario qué hacer ante un tipo que no se entiende, por ejemplo, a través de *image/x-fwf*.

Para ver un caso real, desde el navegador podemos hacer un GET de a una página.

2. Permitir la negociación de contenido. El cliente incluye en su petición los tipos MIME que acepta. De este modo, si un navegador puede soportar documentos de tipo *application/pdf*, lo indicará en la cabecera http:
Allow: application/pdf.
3. Encapsular una o más entidades dentro del cuerpo del mensaje mediante los tipos MIME multipart. Quizás el caso más conocido sea el tipo *multipart/form-data*, utilizado para encapsular los datos de un formulario en su envío hacia el servidor mediante el método POST.

Los tipos MIME se pueden referenciar desde tres lugares diferentes:

1. **Desde el servidor**, que indica al navegador el tipo de datos que envía. Por ejemplo, en el servidor Apache se puede indicar con la directiva *DefaultType* el tipo MIME por defecto que el servidor utilizará en los archivos cuyo tipo no pueda identificar automáticamente.
DefaultType text/plain
2. **Desde la página web**, en la cual se hace referencia a los tipos MIME a través de un enlace a un archivo externo, como una hoja de estilo:
`<link rel="stylesheet" href="hoja_estilo.css" type="text/css">`
 - o Puede especificarse como un atributo en otras etiquetas HTML, como *object* o *form* (atributo *enctype*).
 - o Mediante las etiquetas *<meta HTTP-EQUIV: ...>* podemos hacer que la página participe en el diálogo cliente-servidor especificando tipos MIME.
3. **Desde el navegador web** para que interprete el tipo MIME recibido desde el servidor, y también puede indicar qué tipos MIME acepta (cabecera *http-accept*). Si aparece *"*/*"*, significa que acepta cualquier tipo MIME.
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,/*;q=0.5*

Este es el contenido de la variable predefinida *HTTP_ACCEPT* de PHP que está disponible mediante la llamada a la función *phpinfo()*.

Es interesante saber que el servidor Apache permite especificar el tipo MIME por defecto para aquellos archivos que el servidor no pueda identificar de forma automática como pertenecientes a un tipo concreto:

DefaultType text/plain

Actividades propuestas

- Ejecuta la siguiente orden del protocolo HTTP desde una terminal y analiza la respuesta:
\$HEAD http://www.google.es.
- Ejecuta la orden *telnet* sobre el dominio **www.microsoft.com** y averigua el tipo de servidor que está utilizando.

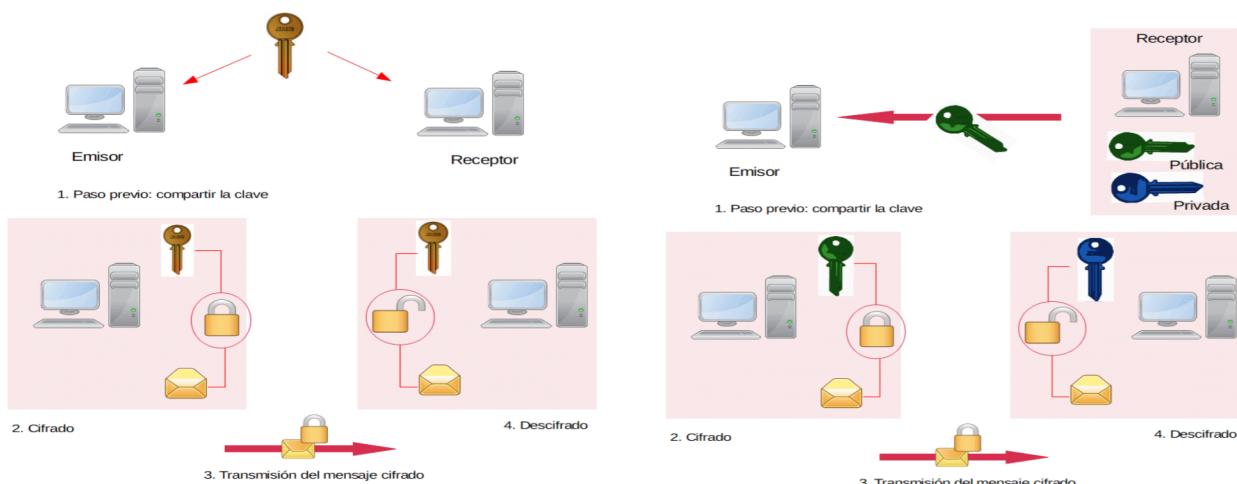
1.5. Sistema criptográfico

El sistema criptográfico es un conjunto de operaciones que permiten transmitir información de forma privada y segura entre emisor y receptor.

El mecanismo de cifrado consiste en aplicar un algoritmo sobre un texto en claro de forma que se obtenga otro compuesto por letras y símbolos que solo el receptor pueda leer.

Los dos métodos de cifrado que se estudian en esta unidad utilizan algoritmos de clave simétrica y algoritmos de clave asimétrica:

- Clave simétrica:** es el sistema que utiliza la misma clave tanto para cifrar como para descifrar el mensaje. Previamente, esta debe ser compartida entre el emisor y el receptor. Ofrece como ventaja su velocidad. Por ello se utiliza para cifrar grandes cantidades de datos. Su inconveniente es que, al viajar en claro, la clave puede ser interceptada por un tercero.
- Clave asimétrica:** es el procedimiento que corrige el problema de la clave simétrica mediante el uso de dos claves: una pública, que se puede enviar a todos los usuarios, y otra privada, que su propietario debe mantener en secreto. Estas claves son complementarias, es decir, lo que se cifra con una solamente lo puede descifrar la otra y viceversa. Su principal ventaja es que, como solo se transmite la clave pública, aunque sea interceptada, las transmisiones seguras no se verán comprometidas. Sus principales inconvenientes son que resulta un proceso muy lento y que debe existir el modo de asegurar que la clave pública realmente pertenezca a su dueño. Los dos elementos básicos utilizados por este sistema de cifrado son: el certificado digital y la firma electrónica.



Firma electrónica

La firma electrónica amplía el concepto de firma manuscrita. Al firmar electrónicamente un documento, además de asegurar que el firmante original es el origen de dicho documento y que él no pueda negar que envió el mensaje, se verifica que su contenido no ha sido modificado. Estas características se denominan **autenticidad, no repudio e integridad** de datos.

El proceso de firma electrónica consiste en cifrar el resumen de un documento con la clave privada del emisor. Así, todos los destinatarios que tengan la clave pública del emisor podrán verificar su procedencia.

Certificado digital

Es el documento electrónico que acredita la correspondencia de una clave pública con su propietario. Mediante un certificado digital se resuelve el segundo problema en el cifrado de clave asimétrica, pero se plantea uno nuevo: saber si el certificado es auténtico o si ha sido falsificado.

Dado que las claves públicas son accesibles por todos los usuarios y que estos no suelen tener relación entre sí, es necesario encontrar el modo que permita que un usuario confíe en el certificado de otro.

Para ello se acude a las **autoridades de certificación** o CA (*Certification Authority*), entidades en las que todos los usuarios confían y que se encargan de distribuir y publicar los certificados digitales. Un ejemplo de CA es la Fábrica Nacional de Moneda y Timbre.

También existen los certificados que son firmados por la misma entidad cuya identidad se certifica, los llamados **autofirmados**. La desventaja de estos certificados es que los navegadores no los reconocen automáticamente.

Autoridades de certificación (CA)

Los navegadores web admiten los certificados firmados electrónicamente por las CA que tienen en su lista sin informar al usuario.

1.6. Funcionamiento del protocolo HTTPS

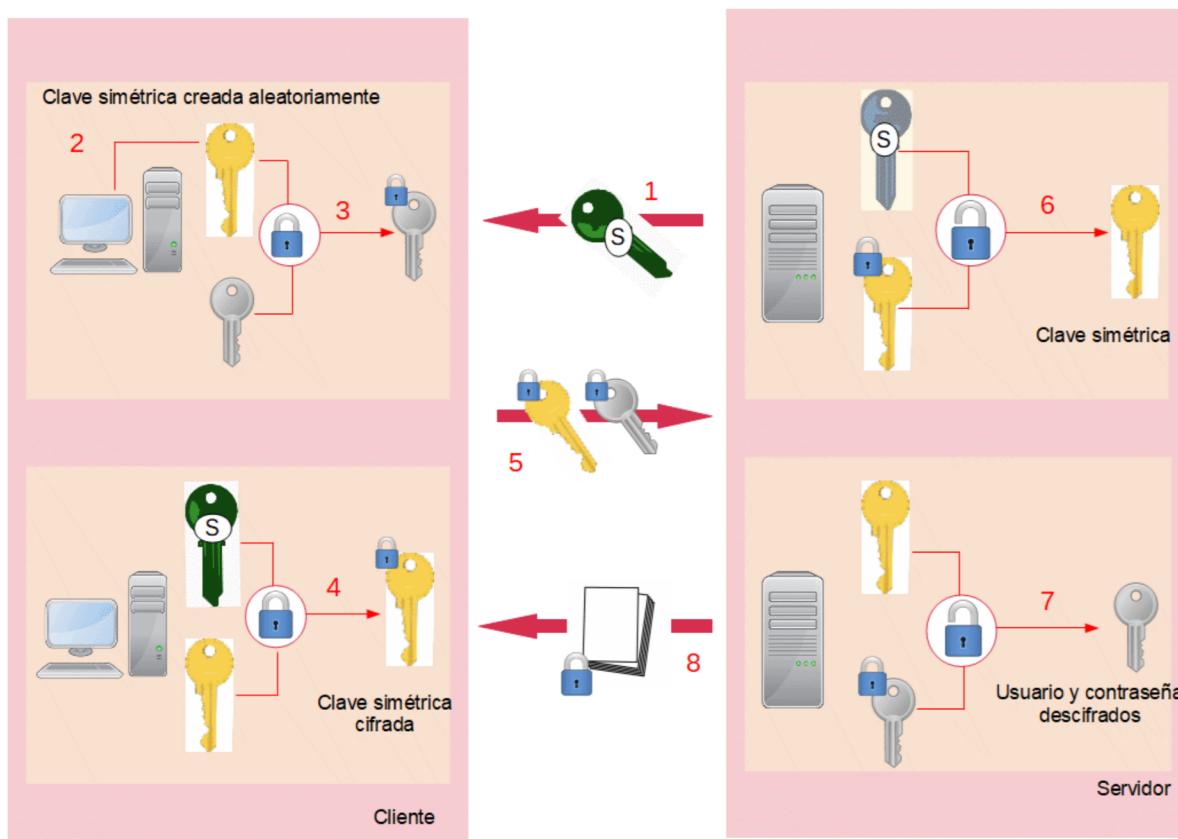
HTTPS se basa en el empleo de los protocolos criptográficos SSL/TLS (*Secure Socket Layer/Transport Layer Security*) para establecer conexiones seguras entre el cliente y el servidor.

El proceso de cifrado mediante claves asimétricas es muy lento, por lo que no se aplica para cifrar páginas web. En su lugar, HTTPS emplea una combinación de algoritmos de clave asimétrica (SSL) y simétrica (TLS).

Con el protocolo HTTPS, el servidor se autentica frente al cliente mediante el certificado digital, mientras que el cliente lo hace a través de un usuario y una contraseña. El procedimiento es el siguiente:

1. La primera vez que el cliente contacta con el servidor este último le envía, mediante su certificado, la clave pública.
2. El cliente acepta dicha clave y genera otra clave simétrica de forma aleatoria.
3. El cliente cifra un nombre de usuario y una contraseña para acceder al servicio web mediante la clave simétrica generada.
4. El cliente cifra la clave simétrica con la clave pública del servidor. Para ello utiliza el sistema de criptografía asimétrica.
5. Se transmiten al servidor tanto el usuario y la contraseña cifrados como la clave simétrica cifrada.

6. El servidor descifra la clave simétrica a través de su clave privada.
7. La clave simétrica permite descifrar el nombre de usuario y la contraseña que han viajado cifrados a través de la red.
8. El servidor comprueba si el usuario tiene acceso al servicio web. En caso afirmativo, la conexión se habrá completado. A partir de este momento cifrará las páginas web con la clave simétrica antes de transmitirlas y el cliente, una vez las reciba, podrá descifrarlas usando esa misma clave.



El protocolo HTTPS es rápido y seguro porque aporta la rapidez del cifrado simétrico y aumenta su seguridad, ya que, gracias al cifrado asimétrico, la clave simétrica es transmitida entre el cliente y el servidor sin comprometer su confidencialidad.

Este protocolo también permite que el cliente se autentique usando sus propias claves asimétricas en vez del usuario y la contraseña.

1.7. Arquitectura de las aplicaciones web

Las aplicaciones web son aquellas cuyo código es descargado total o parcialmente desde la web cada vez que son ejecutadas. También se conocen como aplicaciones basadas en navegador, dado que se ejecutan dentro de estos programas. Algunos ejemplos de aplicaciones web bien conocidas son Wikipedia, Google, eBay o Facebook.

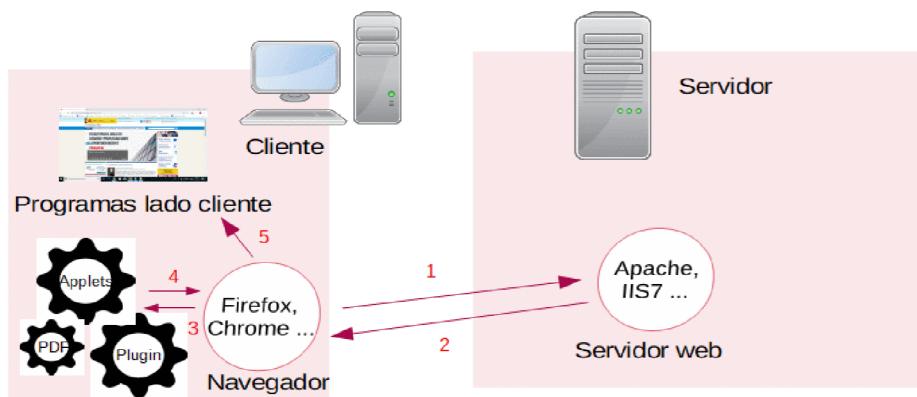
Este tipo de aplicaciones hacen uso de la tecnología cliente-servidor tanto a través de una intranet como de Internet.

Ejecución de código en el cliente

El cliente representa en pantalla un documento generado a partir del código HTML. Además, se hace cargo de la ejecución de las sentencias que componen los *scripts* (programas), que se encuentran incorporados en el código de las mismas. Por ejemplo, *pluggins* como PDF o *applets*.

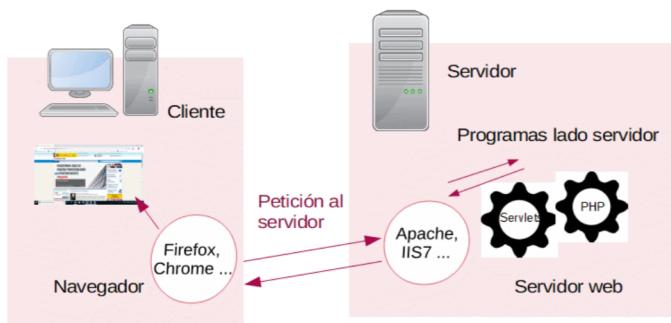
En la siguiente figura se muestra el funcionamiento de una aplicación web, cuyos pasos se describen a continuación:

1. El navegador realiza una petición al servidor web.
2. El servidor envía el documento completo en formato HTML, junto con las sentencias de código incorporado o un mensaje de error.
3. Los programas del lado del cliente son llamados por el navegador para que traduzcan los *scripts* en tiempo de ejecución.
4. El código interpretado se devuelve al navegador. Normalmente, se traduce a formato HTML.
5. El navegador genera el documento y lo muestra en pantalla.



Ejecución de código en el servidor

La diferencia respecto a la ejecución de código en el lado del cliente consiste en que son los programas del lado del servidor quienes interpretan y ejecutan los *scripts*, para luego generar como resultado código HTML, que posteriormente enviarán al navegador web.



Vocabulario

Plugin o complemento: es un pequeño programa que se instala en el navegador para ofrecerle nuevas funcionalidades.

Applet: conjunto de sentencias insertadas en una página web para dotarla de interactividad. Los navegadores web necesitan de un *plugin* Java para poder ejecutarlas.

Servlet: es la tecnología Java que extiende y mejora las funcionalidades de los servidores web. Se encarga de responder peticiones del navegador, por ello se ejecutan en el servidor.

Ejecución de código mixta

Es una mezcla de los dos modelos vistos con anterioridad, donde existe código de programación para ejecutar en los dos lados, tanto en el cliente como en el servidor.

1.8. Servidor virtual

Se denomina **alojamiento virtual** a la técnica que permite hospedar diversos sitios web sobre un mismo servidor de páginas web. A cada uno de estos sitios se le llama **servidor virtual**.

Los servidores virtuales pueden crearse utilizando varios métodos:

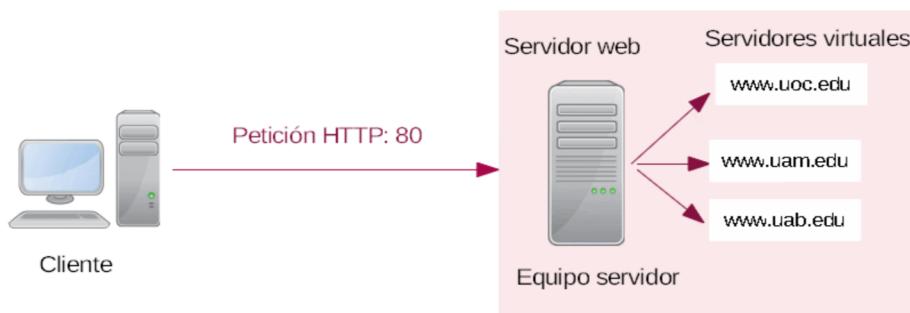
- **Alojamiento basado en dirección IP:** cada servidor virtual debe tener asignada una dirección IP diferente. Este método tiene varias desventajas, entre las que se encuentran: la gestión que comporta la asignación de las direcciones IP o la creación de interfaces virtuales en el servidor web, la administración de las direcciones ante los organismos oficiales y el agotamiento de las direcciones IP disponibles.
- **Alojamiento basado en número de puerto TCP no estándar:** asigna un puerto diferente a cada servidor virtual. No se utiliza debido a que, para ello, el usuario debería conocer el puerto en el que escucha el servidor web.
- **Alojamiento basado en nombre de dominio:** cada servidor virtual tiene asignado su propio nombre de dominio. Por ejemplo, un servidor web podría alojar varios sitios web con nombres de dominio diferentes, como *www.uoc.edu*, *www.uam.es* o *www.uab.es*.

Debido a su sencillez, el último método descrito es el más empleado. Además, puede utilizarse con varios servicios de red y hacer uso de una única dirección IP.

Para implementar el servidor virtual basado en nombre de dominio, es necesario realizar dos acciones:

- Un servidor web se deberá configurar para que pueda identificar los nombres de los servidores virtuales. Además, el navegador web rellena con el nombre de un sitio web el campo llamado *host*, que pertenece a la cabecera del mensaje HTTP, y realiza la petición. Esto permite al servidor web reconocer el servidor virtual al que va dirigida la solicitud. En la versión 1.1 del protocolo HTTP aparece por primera vez esta nueva característica.
- Los nombres de *host* deberán registrarse en el servidor DNS para que pueda resolver la dirección IP del servidor web.

A veces se realizan peticiones a servidores virtuales que ya no existen. Si se configura un servidor virtual por defecto, se asegura que la petición será atendida.



2. Servicios web en sistemas GNU/Linux

Un **servidor web** es un programa que, haciendo uso del protocolo HTTP, atiende las peticiones de los navegadores o clientes web y les proporciona los recursos solicitados.

La arquitectura utilizada es cliente-servidor. El equipo cliente hace una petición de página web al servidor, y este atiende dicha solicitud. Por lo tanto, el objetivo de un servidor web es servir páginas web a los clientes (navegadores) que lo solicitan.

El equipo cliente hace la petición a través del navegador o cliente web. Esta aplicación:

1. **Es la interfaz de usuario:** atiende sus peticiones de páginas web, muestra los resultados y proporciona al usuario herramientas que facilitan su comunicación con el servidor.
2. **Se comunica con el servidor web,** es decir, transmite las peticiones de los usuarios.

2.1. El servidor web

El servidor web tiene la tarea de atender peticiones recibidas desde los navegadores o clientes web, y debe hacerlo de forma eficiente y segura. La ubicación lógica de un servidor web es Internet, pero existen servidores web que dan servicio a intranets e incluso a redes locales.

Si desde el navegador un usuario hace clic sobre un enlace o una página web, está enviando una solicitud al servidor web que aloja dicha página para que se la muestre (servir una página). Si la encuentra, el servidor web la envía, y si no la encuentra, devuelve al cliente un mensaje de error. Cuando este recibe la página web, interpreta el código HTML mostrando de forma correcta las fuentes, colores, imágenes que la componen, etc.

Pero el servidor web, además de servir páginas web HTML que no cambian (estáticas), también permite la ejecución de pequeños programas, en diferentes lenguajes de programación, que proporcionan dinamismo a las páginas web. Estos lenguajes son PHP, CGI, applets/servlets de Java, etc.

Para que el navegador pueda llegar al servidor web (ya sea en una red local o en Internet) el servidor debe tener asignada una dirección IP única que identifica el equipo en la red.

Cuando el navegador hace una petición al servidor web, utiliza un nombre que identifica al servidor, y esta solicitud se transmite a un servidor DNS. Este resuelve el nombre y devuelve la dirección IP que corresponde al nombre. Entonces, la solicitud del navegador se encamina al servidor web correcto. La información que el usuario proporciona al navegador para conectar con un servidor web se llama URL (*Uniform Resource Locators*), que indica la forma de acceder a un recurso utilizando un protocolo de comunicación.

Actividades propuestas

3. ¿Cómo comprobarías que el servidor web Apache está ejecutándose?
4. ¿Cómo comprobarías en qué puerto está escuchando Apache2?
5. ¿Cómo comprobarías el PID del proceso correspondiente al servicio Apache2?

Vamos a realizar un servidor web con las siguientes funcionalidades:

- Todos los datos van a estar organizados y centralizados evitando duplicidades y garantizando la integridad.
- Este servicio permite alojar distintos tipos de datos.

- El acceso a estos datos se puede restringir, de forma que cada usuario vea solo la información para la que está autorizado. Además, la información podrá ser cifrada.
- Los usuarios ya conocen la herramienta que les permitirá acceder al servidor: el navegador web.
- Actualmente, este servicio solo podrá ser usado desde la red de área local, aunque, en cualquier momento, se puede permitir el acceso desde el exterior.

2.2. Instalación del servidor

Sigue las indicaciones que se realizan a continuación para proceder a la instalación del servidor HTTP, utilizando el modo gráfico:

1. Abre una sesión gráfica en el servidor.
2. Abre el gestor de paquetes Synaptic.
3. Haz clic en el botón *Recargar* para actualizar la lista de paquetes disponibles en los repositorios de Internet que están configurados. Espera unos segundos mientras termina este proceso.
4. Haz clic sobre el botón *Buscar* para acceder a la herramienta de búsqueda.
5. Escribe *apache2* en el cuadro de texto y haz clic en el botón *Buscar*.
6. Selecciona *apache2* haciendo clic sobre el nombre del paquete y lee la información adicional mostrada debajo de la lista.
7. Haz clic sobre la casilla de verificación que se encuentra delante del nombre del paquete que has seleccionado. De esta manera lo marcas para instalar.
8. Se abrirá un diálogo que te advierte que para poder instalar apache2 es necesario marcar otros paquetes. Haz clic en el botón *Marcar* para permitir estos cambios adicionales.
9. Asegúrate de que la casilla de verificación correspondiente al paquete apache2 se encuentra marcada y haz clic sobre el botón *Aplicar* para iniciar la instalación.
10. Se abrirá la ventana *Resumen* que muestra información sobre la instalación que vas a realizar. Analízala y haz clic en el botón *Aplicar* para comenzar la descarga e instalación de los paquetes. Durante este proceso se abre la ventana de diálogo *Aplicando cambios*, que se cerrará automáticamente al finalizar la instalación para dar paso a la ventana *Cambios aplicados*.
11. Lee atentamente el contenido del diálogo *Cambios aplicados* y, a continuación, haz clic sobre el botón *Cerrar*.
12. Haz clic en el botón *Cerrar* de la ventana de Synaptic para salir de la aplicación.

Si por el contrario optas por el modo texto, tienes que introducir la orden:

```
$sudo apt-get install apache2
```

Borrado de paquetes

A veces queremos borrar un paquete totalmente, porque la configuración que hemos hecho no está bien, o porque ya no lo utilizamos. La orden para hacerlo rápidamente es:

```
$sudo apt-get purge nombre del paquete
```

En este caso sería:

```
$sudo apt-get purge apache2
```

Tanto en un caso como en otro, para comprobar que el servidor está funcionando abrimos un navegador y ponemos:

<http://localhost>

Podemos cambiar la página por defecto del servidor Apache2. Para ello accedemos al directorio `/var/www/html`. Renombra el fichero `index.html` como `index.bak` y crea en él un nuevo archivo `index.html`. Con un editor de texto escribimos en el fichero el contenido siguiente:

```
<center>
<h1>Bienvenidos al Servidor Apache2</h1>
<h2>Servicios en Red</h2>
</center>
```

Vuelve al navegador y refresca el servidor web. Debe aparecer la página de inicio conforme a la que hemos creado.

2.3. Configuración del servidor

Los archivos de configuración de Apache2 están disponibles en el directorio `/etc/apache2`. El archivo principal de configuración es `/etc/apache2/apache2.conf`.

Es importante antes de realizar cualquier cambio en estos archivos hacer una copia de seguridad, ya que si Apache detecta errores en el archivo de configuración no arrancará.

Entramos en Webmin para realizar la configuración de manera más sencilla. Si al entrar y elegir *Servidores* no aparece el servidor apache2 le damos a *Reajusta módulos*. Ahora aparecerá el servidor web *apache Webserver*.

Dentro del archivo de configuración `apache2.conf`, al que podemos acceder desde terminal o desde Webmin en la pestaña *Configuración global*, en el botón *Editar Archivos de Configuración*, nos podemos encontrar directivas de configuración de los siguientes tipos:

- **Directivas globales**, que controlan las características del servidor.
- **Directivas que controlan el funcionamiento del servidor principal**, y valores por defecto para los servidores virtuales.
- **Directivas de configuración** para los servidores virtuales, que permiten que las peticiones web sean enviadas a diferentes servidores.

En principio no hacemos cambios en la configuración.

Caso práctico 1

Objetivo: mostrar la activación y desactivación de los módulos de Apache y su configuración básica.
Descripción: utilización del módulo *userdir* como ejemplo de activación y desactivación de módulos Apache.

Peso de la nota: 15% de la nota del tema.

Se evaluará:

- Funcionamiento de la práctica (5 puntos).
- Conocimiento de los cambios realizados (2 puntos).
- Dominio de los archivos de configuración (2 puntos).
- Integración con el servidor DNS (1 punto) (cuando montes el servidor DNS puedes volver a esta práctica y hacer esta parte. Tendrás un punto más en ella).

Desarrollo:

1. Creación del usuario *alumno01*.

Para el correcto desarrollo de la práctica deberemos tener creado en nuestro servidor un usuario de nombre *alumno01*. Para ello tenemos, como siempre, dos opciones: a través del terminal, o bien a través del entorno gráfico, en este caso del programa *Usuarios y grupos*. Puede que no lo tengas instalado, pero no te preocupes, te da la opción de instalarlo.

La cuenta creada no necesita grandes permisos, por lo que puede ser una cuenta estándar. Le ponemos como contraseña *albaytar*.

2. Instación del módulo *userdir* (*mod_userdir*).

En Webmin, acceder a *Servidor Web Apache*>*Configuración global*>*Configurar módulos de Apache*. Se mostrará una pequeña parte de la lista de módulos disponibles desde Apache2.

El módulo *userdir* permite a cualquier usuario crear su propio espacio web en un directorio dentro de su *home* (carpeta que se crea automáticamente al usuario y donde entra por defecto al abrir una sesión).

Selecciona en módulo *userdir*. Pulsa en *Habilitar módulos seleccionados* para activarlo.

3. Configuración del módulo *userdir*

Para el módulo *userdir*, el archivo *userdir.load* contiene la directiva de carga del módulo correspondiente.

LoadModule userdir_module /usr/lib/apache2/modules/mod_userdir.so

El archivo *userdir.conf* contiene la configuración propiamente dicha. (recuerda que podemos llegar a través de *Editar Archivos de Configuración*)

Esta configuración indica que el directorio donde estarán los archivos HTML de los usuarios es *public_html* (se le puede cambiar el nombre). Por seguridad, el usuario *root* está desactivado. Podemos indicar usuarios concretos activados, pero en ambos casos deben ser usuarios del sistema. La sección *<Directory>* está indicando que de los *home* de todos los usuarios podrán ser listados sus contenidos (*Options indexes*). Es decir, cualquier usuario cuyo *home* contenga el directorio *public_html* podrá publicar su contenido en el servidor Apache2 correspondiente.

Para la configuración del módulo *userdir* en modo gráfico, accede a *Servidor Web Apache*>*Hosts virtuales existentes*>*Servidor Automático* (el superior)>*Opciones de documento*. Ahí podrás ver lo que se ha comentado y que has podido observar en el fichero *userdir.conf*

4. Utilización del módulo *userdir*

El usuario *alumno01* crea su propia página web y la cuelga en nuestro servidor web. Para ello, *alumno01* debe estar permitido en *userdir*. Y lo está, ya que hemos dejado activos a todos los usuarios del sistema, excepto a *root* (*userdir disabled root*).

Vamos a crear para *alumno01* el directorio */home/alumno01/public_html* y en él ubicamos su página web. Para este caso práctico simplemente se crea un archivo *index.html* dando la bienvenida a la página personal de *alumno01*.

Al acceder a la URL <http://localhost/~alumno01> (para sacar el símbolo ~ en Linux hay que pulsar la tecla *AltGr* y la ñ) vemos el texto del archivo *index.html* creado. Si tenemos creado y operativo el servidor de DNS podemos acceder mediante la URL: <http://servidor.cieloazul.com/~alumno01>. Sino será a través de la IP.

De esta forma se accede al *home* de usuario, y no al directorio */var/www/html*, al que accederíamos si escribiésemos solo **localhost** (o **servidor.cieloazul.com** con el servidor DNS operativo).

Prueba a acceder a la página web del servidor principal y a la de alumno01 desde uno de los clientes. ¿qué debes de poner en lugar de localhost?

Si se quiere acceder a la página web sin necesidad de utilizar el carácter “~” habrá que definir un alias en el archivo */etc/apache2/mods-available/alias.conf*. Desde el entorno gráfico de Webmin accedemos a la opción *de Alias y Redireccionamientos* del primer servidor (el que define las opciones por defecto). Ahora creamos el alias de */alumno01/* para */home/alumno01/public_html/* en *Aliases de directorio de documento*, en las casillas *De* y *Para* (no olvidar aplicar los cambios en el servidor). Ahora comprobamos que podemos acceder de una forma más sencilla sin necesidad de poner el símbolo “~”.

Recomiendo que miréis si ha hecho el cambio en el fichero para estar más seguros.

Caso práctico 2

Objetivo: mostrar la creación, configuración y utilización de hosts virtuales en Apache2 desde Ubuntu GNU/Linux utilizando Webmin.

Peso de la nota: 15% de la nota del tema.

Se evaluará:

- Funcionamiento de la práctica (5 puntos).
- Conocimiento de los cambios realizados (2 puntos).
- Dominio de los archivos de configuración (2 puntos).
- Integración con el servidor DNS (1 punto) (cuando montes el servidor DNS puedes volver a esta práctica y hacer esta parte. Tendrás un punto más en ella).

Desarrollo:

1. Creación de un host virtual.

Comienza por crear un host virtual. Para ello accede a *Servidores>Servidor Web Apache>Crear host virtual*

Indica, en las opciones disponibles, que dirección tiene (la del servidor, hasta que lo hagamos con DNS), que este servidor atenderá peticiones de cualquier dirección, utilizará el puerto por defecto (el 80), que la raíz para documentos será `/var/www/virtualA` y que su nombre será **virtualA.cieloazul.com**. Pulsa en *Crear Ahora y Aplicar cambios*.

Como se comprueba, aparece el servidor por defecto genérico, el servidor virtual que identifica el dominio y el servidor virtual que se acaba de crear.

2. Configuración del host virtual

Para acceder a la configuración del nuevo host virtual, haz clic sobre la bola del mundo de la entrada correspondiente: se mostrarán las opciones disponibles.

Accede a *Editar directivas* para conocer las incluidas en el archivo de configuración del host virtual. Aparecerán las siguientes:

```
DocumentRoot /var/www/virtualA
ServerName virtualA.cieloazul.com
<Directory "/var/www/virtualA">
allow from all
Options None
Require all granted
</Directory>
```

El primer host virtual de la lista es el que se usa por defecto. Cuando una petición entra en el servidor Apache2 desde un navegador web a través de una IP dada, Apache2 comprueba el nombre de dominio que se está solicitando y muestra el contenido a dicho nombre de dominio.

3. Resolución de nombres para nuestro "servidor"

Una vez creado el host virtual, para que se resuelva el nombre hay que acceder desde Webmin a *Servidor DNS Server* y añadir el registro correspondiente a **virtualA.cieloazul.com** en la zona DNS **cieloazul.com**. (lo tendremos que hacer cuando creamos el servidor DNS)

4. Acceso al host virtual **virtualA.cieloazul.com**

Crea en `/var/www/virtualA` un archivo `index.html` de bienvenida a nuestro host virtual. Podemos poner el mensaje "Bienvenidos al Servidor Virtual Apache2: **virtualA.cieloazul.com**".

Si introducimos en el navegador la dirección: *virtualA.cieloazul.com*, deberá aparecer la página web creada (cuando tengamos el servidor DNS activo).

Como no tenemos todavía el servidor DNS activo vamos a hacer un pequeño truco. Nos vamos al fichero */etc/hosts*. En este fichero está la información de las traducciones que hace antes de llamar al servidor DNS. Añadimos la siguiente línea:

192.168.10.2 virtualA.cieloazul.com

Ahora ya funcionará correctamente. ¿Servirá también en los clientes? Recordar quitar esa línea cuando montemos el servidor DNs, sino os dará conflictos.

Importante: si una petición URL referencia a un servidor virtual que no existe, responde el primer servidor virtual de la lista. Es conveniente situar en el primer lugar de la lista un servidor virtual que solo muestre una página indicando que ese servidor no existe. ¿Te atreves a hacerlo?

Caso práctico 3

Objetivo: se trata de mostrar la configuración y utilización de los módulos correspondientes a la autenticación básica en Ubuntu GNU/Linux.

Peso de la nota: 15% de la nota del tema.

Se evaluará:

- Funcionamiento de la práctica (5 puntos).
- Conocimiento de los cambios realizados (2 puntos).
- Dominio de los archivos de configuración (2 puntos).
- Integración con el servidor DNS (1 punto) (cuando montes el servidor DNS puedes volver a esta práctica y hacer esta parte. Tendrás un punto más en ella).

1. Activación de módulos

El módulo que controla este método de autenticación es *auth_basic* y tiene la ventaja de que está soportado por todos los navegadores web. Sin embargo, un inconveniente que tiene es que el login y la contraseña no van cifrados del navegador web al servidor.

Comprueba si está instalado utilizando Webmin. Para ello tienes que ir a *Configurar módulos de Apache* (como vimos para el módulo *userdir*).

3. Utilización de archivos *.htaccess*

Los archivos *.htaccess* permiten a los usuarios que no tienen permisos modificar la configuración, y así poder ejercer algún control sobre el comportamiento del servidor Apache2.

El archivo *.htaccess* se sitúa en el directorio al que tiene que afectar, junto a todos sus directorios, y las modificaciones introducidas no requieren reiniciar el servidor web. Para que el servidor haga caso de los archivos *.htaccess* hay que incluir la directiva *AllowOverride* (permite sobreescritura).

Hay que asegurarse de que el nuevo directorio está contemplado dentro del host virtual por defecto "default" (el que escucha en el puerto 80 y tiene la raíz para documentos en /var/www/html) y que se permite la utilización de archivos *.htaccess*.

Para eso añadimos las siguientes líneas en *Editar Directivas*, de ese host virtual.

```
<Directory "/var/www/html/directorio_protegido">
    AllowOverride AuthConfig
</Directory>
```

Salva los cambios.

3. Directorios a proteger mediante autenticación básica

Antes que nada, hay que crear el directorio que queremos proteger. Para ello nos vamos al terminal y creamos el directorio: /var/www/html/directorio_protegido.

Para configurar el módulo *auth_basic*, hay que acceder a *Webmin>Herramientas>Directorios web protegidos>Aregar protección para un nuevo directorio*.

En la interfaz que aparece debemos introducir:

- Ruta al directorio: `/var/www/html/directorio_protegido`.
- Encriptación de contraseña: *Encriptación Unix*.
- Dominio de autenticación: *directorio_protegido*.

A continuación, hay que pulsar el botón *Crear*. Así, se mostrará la pantalla de *Directorios Webs Protegidos*, en la que se requerirá especificar qué usuarios podrán acceder al dominio de autenticación *Directorio protegido*.

Debemos dar de alta al usuario *alumno01*, dándole al botón *agregar un nuevo usuario* para que se pueda validar en el dominio de autenticación definido. Como usuario pondremos lógicamente *alumno01* y como contraseña *albaytar* (tener en cuenta que la contraseña de usuario y la de acceso a su espacio web no tienen por qué ser la misma).

De este modo se habrá creado un archivo `.htpasswd` dentro del directorio protegido, cuyo contenido es el usuario y la contraseña encriptada (*¿sabes mostrar ficheros ocultos?*)

Es conveniente comprobar, además, que se ha creado un archivo `.htaccess` en el directorio `/var/www/html/directorio_protegido` que contiene una entrada para los usuarios permitidos de este tipo:

Require user alumno01

O la siguiente entrada:

Require valid-user

Esta última es más genérica y solicita las credenciales a todos los usuarios para los que se ha establecido esa condición.

Para comprobar que estos ficheros existen, y que puedo preguntar cuando corrija la práctica, una manera sería a través del terminal, yendo al directorio y listando con `ls -a`.

4. Acceso al directorio protegido

Abre el navegador en la página del servidor por defecto `ubuntuserver.cieloazul.com` y, en la URL, escribe `http://ubuntuserver.cieloazul.com/directorio_protegido`.

Al introducir usuario y contraseña nos mostrará el contenido del directorio.

(como todavía no tenemos montado el servidor DNS, es necesario trabajar con IPs.)

(para comprobar que funciona, podéis hacerlo en primer lugar en el propio servidor, y si va bien, hacerlo con un cliente).

Caso práctico 4

Objetivo: vamos a activar la seguridad en Apache2, en Ubuntu GNU/Linux.

Peso de la nota: 15% de la nota del tema.

Se evaluará:

- Funcionamiento de la práctica (5 puntos).
- Conocimiento de los cambios realizados (2 puntos).
- Dominio de los archivos de configuración (2 puntos).
- Integración con el servidor DNS (1 punto) (cuando montes el servidor DNS puedes volver a esta práctica y hacer esta parte. Tendrás un punto más en ella).

En esta práctica vamos a configurar Apache como un servidor seguro utilizando el protocolo HTTPS. Se utilizará Webmin en aquellos pasos en los que se pueda realizar una configuración gráfica, y el resto de acciones se harán utilizando la terminal en modo comando.

1. Activación desde Webmin del módulo SSL

En Webmin entra en *Servidores>Servidor web Apache>Configuración Global>Configuración módulos de Apache*.

Activa la casilla correspondiente al módulo SSL y, a continuación, pulsa en *Habilitar módulos seleccionados*.

2. Creación del host virtual

Desde la pestaña de *Creación de host virtual*, crea uno específico para conexiones seguras, basado en el host virtual por defecto y con directorio de inicio para documentos */var/www/ssl/htdocs* que debes crear como usuario sudo.

En este directorio, deja un archivo *index.html* preparado para la prueba de conexión.

3. Obtención del certificado

Una vez en el nuevo host virtual, accede a las *Opciones SSL*; debes comprobar que se suministran unos certificados para el servidor y que se han de obtener.

En un entorno de servidor de producción real, estos certificados se deberían solicitar a una autoridad de certificación. Pero en el caso del servidor del aula se va a proceder transformando nuestro servidor en una CA.

La orden para crear el certificado SSL es *apache2-ssl-certificate*, pero no va incluida en Ubuntu. Se ha de descargar e instalar el archivo *apache2-ssl.tar.gz*. para ello, ejecuta la siguiente orden desde un terminal:

```
$sudo wget http://librarian.launchpad.net/7477840/apache2-ssl.tar.gz
```

Al descomprimir el archivo deja en el directorio dos archivos: *ssl.cnf* y *apache2-ssl-certificate*

ssleay.cnf copiar en */usr/share/apache2*

apache2-ssl-certificate copiar a */usr/sbin*

A continuación, crea el directorio en donde se guardará el certificado SSL, y que es */etc/apache2/ssl*

Añade permiso de ejecución a la orden *apache2-ssl-certificate*. En este momento se

generará el certificado ejecutando la siguiente orden:

```
$sudo apache2-ssl-certificate
```

De esta forma se ha creado el archivo `/etc/apache2/ssl/apache.pem`, que contiene las claves (RSA PRIVATE KEY) y el certificado (CERTIFICATE) pertenecientes al servidor `ubuntuserver.cieloazul.com`, firmado por nosotros mismos.

4. Instalar el certificado

A continuación, crea los directorios y archivos siguientes:

```
$cd /etc/apache2/ssl
```

```
$sudo mkdir miCA
```

```
$sudo mkdir miCA/private
```

Las órdenes siguientes deberán ejecutarse desde `/etc/apache2/ssl`:

- Copia las claves de la CA al directorio correspondiente con el nombre de `cakey.pem`:

```
$sudo cp apache.pem miCA/private/cakey.pem
```

- Copia el certificado de la CA al lugar necesario:

```
$sudo cp apache.pem miCA/cacert.pem
```

- Crea el fichero `miCA/serial` (o editalo si ya estaba creado) y guárdalo en modo texto. Su contenido debe ser una línea con el valor "01", como se muestra a continuación:

```
$more miCA/serial
```

```
01
```

Ahora lo único que queda pendiente es incluir en el sitio las directivas que indican que el acceso al mismo se hace mediante el protocolo HTTPS.

5. Certificados para Webmin

Una vez que estén disponibles los certificador desde Webmin, accede a las *Opciones SSL* del servidor virtual seguro e indica los caminos hasta ellos.

Introduce en *Archivo de Certificado/Clave privada* en valor `/etc/apache2/ssl/apache.pem`

De cualquier forma, Webmin no gestiona bien los certificados. Para tener la completa seguridad de que se han incluido en el archivo de configuración del host virtual seguro, edita dicho archivo. Deberás incluir en él, al final, las directivas siguientes:

```
SSLEngine on
```

```
SSLCertificateFile /etc/apache2/ssl/mica/cacert.pem
```

```
SSLCertificateKeyFile /etc/apache2/ssl/miCA/private/cakey.pem
```

Sal de *Opciones SSL* y, desde Webmin, aplica los cambios en Apache.

6. Comprobación de funcionamiento

Al abrir el navegador e ir a `https://servidor.cieloazul.com` se mostrará un mensaje de aviso diciendo que el certificado que tiene disponible no es fiable (ya lo sabíamos) y añadimos una excepción.

Una vez confirmada la excepción, ya se tiene disponible el mensaje incluido en el archivo `index.html` preparado.