

# Securizar Servidor Apache

---

Servidores Web de Altas prestaciones

**Miguel Ángel Rodríguez**

**Grado en Ingeniería Informática**

En este documento se refleja, qué acciones tomar para poder securizar un servidor Apache. Qué medidas preventivas tomar, cómo identificar posibles anomalías, etc. Todo ello con ejemplos en los que se demuestra el efecto de las acciones tomadas.

## Índice

Directivas y Archivos de Configuración.....	3
Seguridad En La Instalación.....	4
Control de los Archivos Publicados .....	5
Ocultar Información.....	8
CGI y Módulos.....	13
¿CGI O MÓDULO? .....	13
MEDIDAS DE SEGURIDAD.....	14
Autenticación y Autorización .....	16
POR CONTRASEÑA .....	16
AUTORIZACIÓN A GRUPOS .....	18
OTROS FACTORES DE AUTENTICACIÓN .....	19
Comunicación HTTPS y Autenticación Con Certificado Cliente .....	20
VENTAJAS DE HTTPS SOBRE HTTP.....	20
CONFIGURACIÓN SEGURA DE HTTPS .....	20
COMPROBACIÓN DE LA SEGURIDAD DE HTTPS .....	22
AUTENTICACIÓN DEL USUARIO POR CERTIFICADO .....	22
AUTENTICACIÓN DEL USUARIO POR CERTIFICADO .....	23
Ataques de Denegación de Servicio .....	24
ATAQUES DOS .....	24
NÚMERO MÁXIMO DE PETICIONES CONCURRENTES.....	25
RESTRICCIONES SOBRE LAS PETICIONES.....	26
MONITORIZACIÓN DEL USO DE RECURSOS .....	26
Tabla de Ataques DOS.....	27
Monitorización .....	29
MONITORIZACIÓN DEL SERVICIO.....	29
CONFIGURACIÓN Y REVISIÓN DE LOGS.....	29
DETECCIÓN DE ATAQUES .....	30

---

## Directivas y Archivos de Configuración

---

Como cualquier aplicación, el servidor web apache contiene una serie de archivos que será necesario configurar correctamente durante su instalación para dotar al servidor de la funcionalidad deseada y asegurarlo de manera correcta. La configuración se realiza **a través de directivas o reglas en archivos de configuración concretos**.

Es importante resaltar que dependiendo de **la distribución de Linux/Unix utilizada con Apache, el número, las rutas y los nombres de los archivos de configuración de Apache pueden variar**. Con el comando `apache2ctl -V`, se pueden averiguar datos sobre una instalación concreta incluyendo cual es el archivo de configuración:

```
root@ServidorProduccion:/etc/apache2# apache2ctl -V
Server version: Apache/2.2.22 (Ubuntu)
Server built: Jul 22 2014 14:37:02
Server's Module Magic Number: 20051115:30
Server loaded: APR 1.4.6, APR-Util 1.3.12
Compiled using: APR 1.4.6, APR-Util 1.3.12
Architecture: 32-bit
Server MPM: Prefork
  threaded: no
    forked: yes (variable process count)
Server compiled with....
-D APACHE_MPM_DIR="server/mpm/prefork"
-D APR_HAS_SENDFILE
-D APR_HAS_MMAP
-D APR_HAVE_IPV6 (IPv4-mapped addresses enabled)
-D APR_USE_SYSVSEM_SERIALIZE
-D APR_USE_PTHREAD_SERIALIZE
-D SINGLE_LISTEN_UNSERIALIZED_ACCEPT
-D APR_HAS_OTHER_CHILD
-D AP_HAVE_RELIABLE_PIPED_LOGS
-D DYNAMIC_MODULE_LIMIT=128
-D HTTPD_ROOT="/etc/apache2"
-D SUEXEC_BIN="/usr/lib/apache2/suexec"
-D DEFAULT_PIDLOG="/var/run/apache2.pid"
-D DEFAULT_SCOREBOARD="logs/apache_runtime_status"
-D DEFAULT_LOCKFILE="/var/run/apache2/accept.lock"
-D DEFAULT_ERRORLOG="logs/error_log"
-D AP_TYPES_CONFIG_FILE="mime.types"
-D SERVER_CONFIG_FILE="apache2.conf"
root@ServidorProduccion:/etc/apache2#
```

Por ejemplo, en sistemas Debian/Ubuntu se utiliza la siguiente estructura de archivos de configuración diseñada para facilitar la administración:

- **/etc/apache2/apache2.conf**: el archivo raíz es el que incluye a los demás. No se debe modificar este archivo.
- **mods-enabled/\*.load y mods-enabled/\*.conf**: la finalidad de estos archivos es la carga y configuración de los módulos de Apache.
- **httpd.conf**: directivas aplicables a todos los servidores web.
- **ports.conf**: define en qué puertos «escuchará» Apache.
- **conf.d/**: directorio que contiene archivos de configuración para cada funcionalidad de apache (charset, php, security, etc.)
- **sites-enabled/**: directorio que contiene los archivos de configuración de cada virtual host.

---

## Seguridad En La Instalación

---

Afortunadamente, las instalaciones de Apache en sistemas Linux/Unix a través de los mecanismos de gestión de paquetes incluyen muchas de las opciones que se citan a continuación activadas o implantadas por defecto. Pero en alguna ocasión es posible que esas características no sean necesarias o no se vayan a utilizar por lo que en ocasiones es posible que se conviertan en un problema de seguridad si no se actúa correctamente, por lo que hay que analizar previamente las necesidades para desactivar todo aquello que no se vaya a utilizar.

### Deshabilitar los módulos que no se utilicen:

El deshabilitar los módulos que no se vayan a utilizar, va a ofrecer dos ventajas principales. Por un lado no sólo se evitarán ataques sobre estos módulos (a mayor número de módulos mayor número de posibilidades de ataque); y por otro, también Apache en ejecución consumirá menos recursos. Otra ventaja más de desactivar los módulos, es que se verá reducida la necesidad de emplear más recursos para su administración y control.

### Permisos sobre ficheros:

Durante la instalación se creará un usuario y grupo exclusivo para la ejecución de Apache. Este usuario y grupo se configura con las directivas User y Group. En cuanto a los permisos de ficheros, **el owner (propietario) del binario de Apache ha de ser root (super-administrador)** para poder abrir el puerto 80, aunque después cambia el usuario del proceso al propio de Apache. El resto de usuarios no ha de tener permisos de escritura sobre este fichero ya que entonces podría sustituirlo por otro malicioso que se ejecutaría con los máximos privilegios. Por ello se sugieren los siguientes permisos en los binarios de Apache:

```
root@ServidorProduccion:/etc/apache2# find /usr/lib/apache2 -type d | xargs chmod 755
root@ServidorProduccion:/etc/apache2# find /usr/lib/apache2 -type f | xargs chmod 644
root@ServidorProduccion:/etc/apache2#
```

Por otro lado, en muchas instalaciones por defecto otros usuarios tienen permisos de lectura sobre los archivos de configuración y de logs de Apache. Es muy recomendable eliminar estos permisos con los siguientes comandos:

```
root@ServidorProduccion:/etc/apache2# chmod -R go-r /etc/apache2
root@ServidorProduccion:/etc/apache2# chmod -R go-r /var/log/apache2
root@ServidorProduccion:/etc/apache2# _
```

---

## Control de los Archivos Publicados

---

Uno de los problemas más frecuentes en instalaciones de Apache, es que no se aplican las directivas adecuadas para controlar los archivos que se ubican en el directorio en el que se publican las páginas web. Este directorio está especificado mediante la directiva *DocumentRoot*. Si no se configura adecuadamente la directiva es posible que existan errores de configuración que puedan posibilitar el acceso a otros directorios o subdirectorios.

### DENEGAR EL ACCESO POR DEFECTO

Por lo que se ha comentado anteriormente, es conveniente denegar el acceso a todos los directorios por defecto y permitir el acceso sólo al directorio especificado en el *DocumentRoot* explícitamente:

```
DocumentRoot /var/www
<Directory />
    Order deny,allow
    Deny from all
    Options None
    AllowOverride None
</Directory>
<Directory /var/www/>
    Options Indexes FollowSymLinks MultiViews
    AllowOverride None
    Order allow,deny
    allow from all
</Directory>
```

Las directivas *Allow* y *Deny* son utilizadas para permitir o denegar respectivamente el acceso a un directorio y *Order* especifica el orden en el que serán evaluadas.

La directiva `<Directory>` sirve para aplicar una serie de directivas sobre la carpeta especificada y a todas sus subcarpetas. Lo más común es que existan varias de estas directivas y que, como también se aplican a los subdirectorios, existan directivas contradictorias aplicadas a un directorio. En este caso, la directiva del `<Directory>` más específico es la que prevalece.

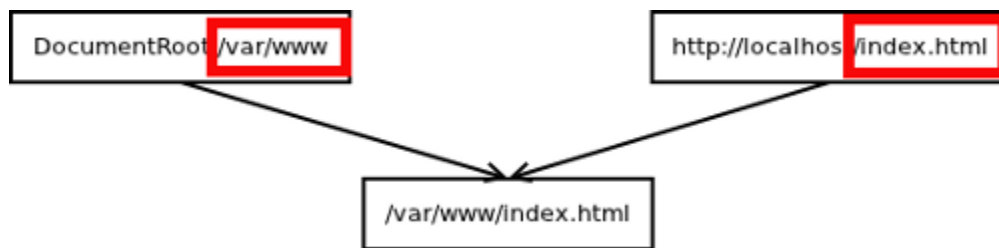
En el ejemplo anterior, existen directivas de acceso contradictorias en el directorio `/var/www/htdocs`, pero se imponen las que permiten el acceso por estar contenidas en un `<Directory>` más específico.

El cometido de las directivas *Options* y *AllowOverride* se explicarán en puntos posteriores.

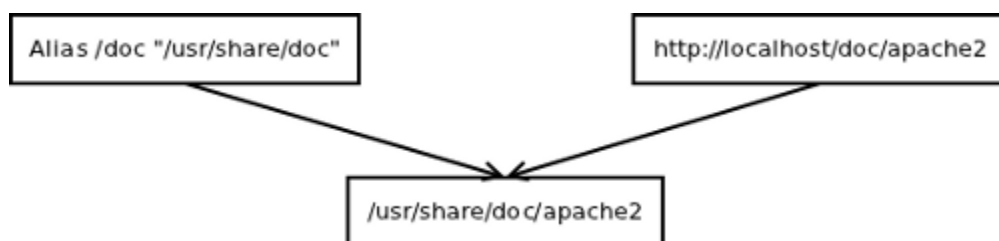
### REVISAR LAS DIRECTIVAS ALIAS

Apache permite mediante la directiva *alias* asignar cualquier tipo de ruta en el sistema de ficheros para que sea accesible desde la web. Se deben revisar estas reglas para verificar que no existan directorios asociados al servicio web que puedan ser mostrados.

Como ejemplo, para servir un archivo o página web desde el servidor Apache, se concatena el directorio especificado en *DocumentRoot* con la parte de la ruta de la URL:



Pero mediante la directiva *Alias* se puede modificar esta resolución y especificar otra ruta diferente. Si la ruta coincide con el primer parámetro de la URL, Apache servirá desde el directorio especificado en el segundo parámetro:



Para evitar filtrar información **se debe analizar la necesidad de esta directiva** y de las similares *AliasMatch*, *ScriptAlias* y *ScriptAliasMatch* y no implementarla si no se va a utilizar.

## EVITAR LA RESOLUCIÓN DE ENLACES SIMBÓLICOS

En sistemas Unix/Linux, Apache puede recibir una petición a un archivo que es, a nivel de sistema operativo del servidor, un enlace simbólico y devolver el archivo apuntado por él aunque se encuentre fuera del *DocumentRoot*.

Esta funcionalidad puede posibilitar que un atacante, que tenga permisos de escritura sobre el *DocumentRoot*, cree un enlace simbólico a archivos contenidos fuera del *DocumentRoot* para luego acceder a ellos a través del navegador.

Para deshabilitar esta posibilidad se debe aplicar la siguiente directiva:

```
<Directory /var/www/>
    Options -Indexes -FollowSymLinks
    AllowOverride None
    Order allow,deny
    allow from all
</Directory>
```

Como alternativa, se puede habilitar esta funcionalidad pero sólo si el archivo destino pertenece al mismo usuario que el enlace simbólico:

```
<Directory /var/www/>  
  Options -Indexes -FollowSymLinks +SymLinksIfOwnerMatch  
  AllowOverride None  
  Order allow,deny  
  allow from all  
</Directory>
```

---

## Ocultar Información

---

Una de las primeras fases de los ataques sobre los sistemas de información es la revelación de información (o *information disclosure*). Cuanta más información tenga el atacante, más fácil le resultara encontrar vulnerabilidades existentes en el sistema atacado.

A continuación, se detallarán varias medidas para tratar de exponer la mínima información posible.

### DESHABILITAR EL LISTADO DE FICHEROS

Si se le envía a Apache una URL que solicita un directorio y no un archivo concreto, Apache permite mostrar los contenidos de este directorio. Esta funcionalidad puede ser aprovechada por un atacante para descubrir archivos que el administrador del sitio no tenía intención de publicar.

Esta característica de Apache es controlada por la directiva *Options*, para desactivarla se debe aplicar la siguiente configuración:

```
<Directory /var/www/>
  Options -Indexes
  AllowOverride None
  Order allow,deny
  allow from all
</Directory>
```

Como su propio nombre indica, mediante esta directiva se controlan varias opciones de configuración. Puede aceptar varios valores, entre ellos *All* y *None* para activar o desactivar todas las opciones disponibles. De hecho, en la configuración del directorio raíz se recomienda desactivar todas por seguridad, lo que incluiría la opción *Indexes*:

```
<Directory />
  Options none
  AllowOverride None
  Order Deny, Allow
  Deny from all
  AllowOverride None
</Directory>
```



## LIMITAR EL ACCESO A ARCHIVOS POR EXTENSIÓN

En algunas ocasiones determinados ficheros deben existir dentro del *DocumentRoot* del servidor pero no se desea que estén disponibles, como los ficheros *.htaccess* y *.htpasswd*, cuya funcionalidad se explicará posteriormente. En estos casos, se pueden utilizar las directivas *Files* y *FilesMatch* para denegar su acceso.

En el siguiente ejemplo, se deniega el acceso a todos los ficheros que comienzan por los caracteres *".ht"* (entre ellos *.htaccess* y *.htpasswd*):

```
<Files ~ "^\.ht">
    Order allow,deny
    Deny from all
</Files>
```

La siguiente configuración evita que se publiquen los archivos de copia de seguridad:

```
<FilesMatch "(\.bak$|\.BAK$)">
    Order Allow,Deny
    Deny from all
</FilesMatch>
```

*Files* y *FilesMatch* sólo actúan sobre el nombre del archivo, para filtrar directorios, se debe utilizar *DirectoryMatch*. Mediante las directivas presentes a continuación se denegaría el acceso a todos los directorios CVS (utilizados en sistemas de control de versiones de código fuente):

```
<DirectoryMatch /CVS/>
    Order Allow,Deny
    Deny from all
</DirectoryMatch>
```

## INFORMACIÓN EN LA CABECERA "SERVER"

Las cabeceras http pueden contener información muy útil para un atacante. Cuando se realiza una petición hacia un servidor web, éste en las distintas respuestas http que ofrece, incluye la cabecera *Server* que generalmente contiene información sobre el software que ejecuta el servidor web:

The screenshot shows a network capture in Wireshark. The selected packet is an HTTP 200 OK response. The 'Request' tab is active, showing the following details:

- Request Version: HTTP/1.1
- Status Code: 200
- Response Phrase: OK
- Date: Thu, 23 Apr 2015 23:17:56 GMT\r\n
- Last-Modified: Mon, 23 Mar 2015 17:40:00 GMT\r\n
- ETag: "e19ab-c4-511f827c7740b"\r\n
- Accept-Ranges: bytes\r\n
- Vary: Accept-Encoding\r\n

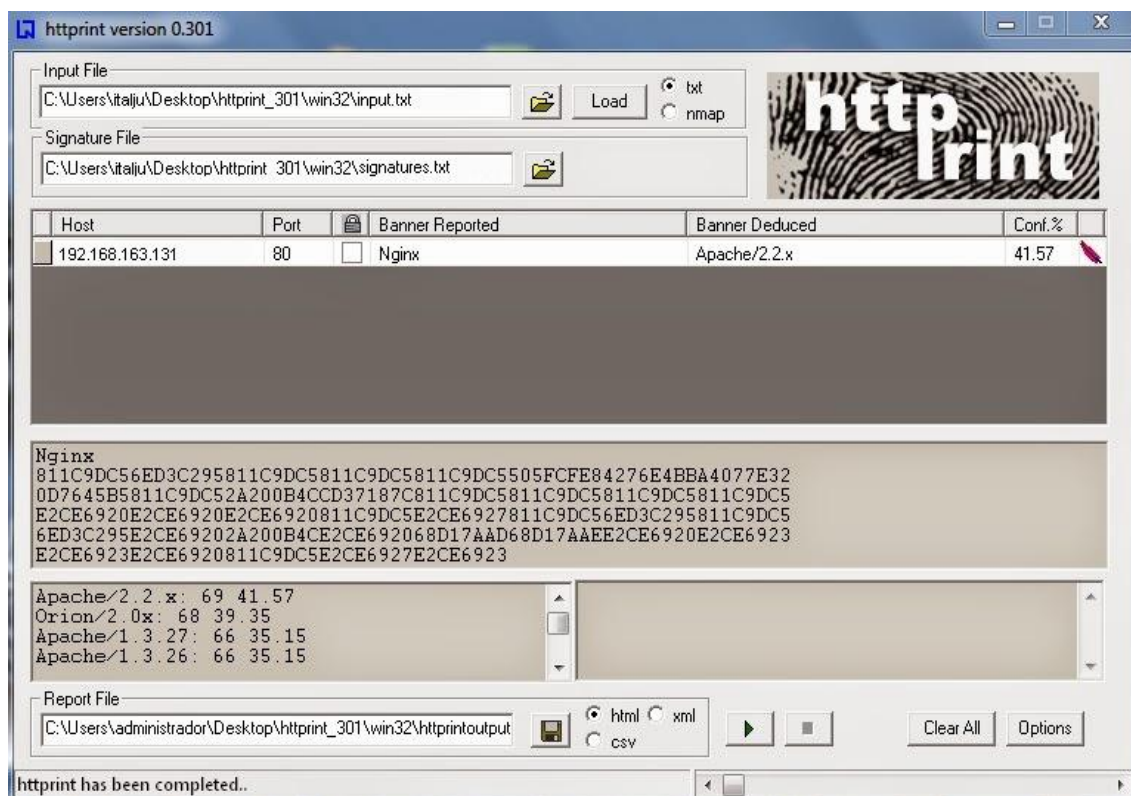
The 'Response' tab is also visible, showing the following details:

- Request Version: HTTP/1.1
- Status Code: 200
- Response Phrase: OK
- Date: Thu, 23 Apr 2015 23:17:56 GMT\r\n
- Server: Apache/2.2.22 (Ubuntu)\r\n
- Last-Modified: Mon, 23 Mar 2015 17:40:00 GMT\r\n
- ETag: "e19ab-c4-511f827c7740b"\r\n
- Accept-Ranges: bytes\r\n
- Vary: Accept-Encoding\r\n

Si se desea restringir esta información se puede utilizar la directiva *ServerTokens*, aunque no es posible limitar la información enviada completamente ya que la configuración más restrictiva es *Prod*, puesto que envía el nombre del servidor web.

Mediante el módulo *mod\_security* es posible modificar el valor de esta cabecera completamente, para ello deberíamos introducir: *SecServerSignature "(valor cabecera)"*

Aunque *mod\_security* es un módulo de gran utilidad, **se desaconseja su instalación únicamente para enmascarar el servidor** ya que un *hacker* dispone de múltiples herramientas de *fingerprinting* como HTTPPrint, que basándose en pequeñas diferencias de implantación del protocolo HTTP, permite obtener la versión del navegador web:

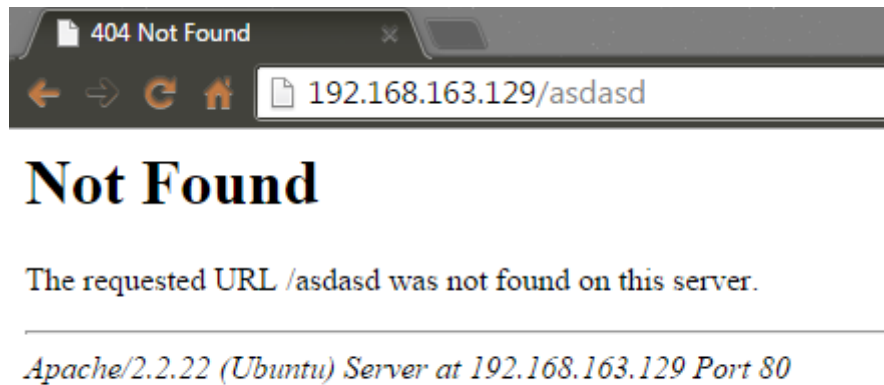


## INFORMACIÓN EN PÁGINAS GENERADAS POR EL SERVIDOR

Cuando un usuario se conecta a un recurso inexistente, listados, etc. de un servidor web Apache, este le puede ofrecer algún tipo de mensaje de error, advertencia, etc. La directiva *ServerSignature* establece si se desea mostrar la versión del servidor, el correo del administrador y el nombre del servidor virtual en las páginas generadas por Apache (por ejemplo, errores o listados de directorio FTP). Para no mostrar información se puede desactivar (establecer a *Off*).

La dirección de correo electrónico del administrador, que muestra en determinadas páginas de error para que el internauta pueda notificar el error, se establece mediante la directiva *ServerAdmin*. No es recomendable establecer una personal, sino una genérica para este fin específico.

Por otro lado, Apache muestra una página de error genérica cuando se produce un error que revela información sobre la versión del software ejecutada



Mediante la directiva *ErrorDocument* se puede modificar este texto, incluso invocar un script que lo construya dinámicamente. Aunque esta directiva es poco flexible ya que hay que crear una entrada por cada código de error. Por ejemplo:





---

## CGI y Módulos

---

Pocos portales en la actualidad ofrecen únicamente archivos HTML (contenido estático). La mayor parte crea las páginas web dinámicamente o implementa una lógica de negocio mediante un lenguaje de programación como PHP, Java o .NET.

En el siguiente capítulo se explicarán las distintas maneras posibles en Apache de «ejecutar código» y las medidas de seguridad que se deberían tener en cuenta.

Aunque no se citan en este informe por su extensión, **la mayor fuente de riesgo se encuentra en los posibles errores de programación de los propios scripts**, como puede ser no validar la entrada, errores en la gestión de sesión o condiciones de carrera.

### ¿CGI O MÓDULO?

Los dos mecanismos que permiten a Apache servir páginas web dinámicas se diferencian principalmente en el modo de invocación del intérprete o programa que crea el contenido dinámico. Mediante el mecanismo de módulos, el intérprete o máquina virtual se encuentra integrada en el proceso de Apache y mediante CGI es un proceso externo e independiente.

Por ello, el mecanismo del módulo es más eficiente y puede utilizar características de Apache, como los archivos .htaccess que se describirán más adelante. Pero, por otro lado, todos los scripts se ejecutan bajo el mismo usuario, el de Apache, lo que en servidores compartidos permitiría acceder a los scripts de servicios web de otros usuarios. En cambio, mediante CGI se pueden utilizar la herramienta suEXEC para definir unos permisos sobre los scripts que no permitan al usuario de Apache tener acceso a ellos. Esta herramienta permite ejecutar los scripts mediante el usuario y grupo definido en la directiva SuexecUserGroup.

Por otra parte, **si existe algún problema en la ejecución** que hace que el programa se «cuelgue», como módulo afectará a todo el servidor, en cambio, como **CGI, al crear un nuevo proceso**, que ejecuta la petición **sólo afectará a la petición determinada**.

	CGI	MÓDULO
<b>Eficiente</b>	No	Sí
<b>.htaccess</b>	No	Sí
<b>Control de Acceso</b>	Sí	No
<b>Resistencia a errores</b>	Sí	No

**Una alternativa a estas dos tecnologías, es utilizar FastCGI, que incorpora las ventajas de CGI junto con mecanismos más eficientes de ejecución.**

## MEDIDAS DE SEGURIDAD

Mediante la directiva *ScriptAlias*, similar a *Alias* descrita en puntos anteriores, se especifica el patrón de las URLs que corresponden a scripts y el directorio en el que se encuentran, de manera que los scripts que estén contenidos en ese directorio serán ejecutados:

```
ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
```

El directorio al que hace referencia *ScriptAlias* ha de encontrarse fuera del *DocumentRoot*, sino se podrían mostrar su código fuente accediendo al directorio final sin usar el alias.

Para evitar la ejecución de scripts en el *DocumentRoot* se puede utilizar la directiva *Options* vista anteriormente:

```
DocumentRoot /var/www
<Directory />
    Options FollowSymLinks -ExecCGI
    AllowOverride None
</Directory>
```

Actualmente, las páginas web integran contenido dinámico mediante etiquetas HTML especiales, como `<?` en PHP, utilizando directivas de Apache como las siguientes:

```
Action application/x-httpd-php modules/libphp.so
<FilesMatch "\.php$">
    SetHandler application/x-httpd-php
</FilesMatch>
```

### Corrígelolo!

Al introducir las anteriores directivas, es posible que se muestre el siguiente mensaje al reiniciar el servicio de apache:

*Invalid command 'Action', perhaps misspelled or defined by a module not included in the server configuration*

Para solucionarlo, tan solo hay que habilitar el módulo actions como sigue:

```
root@ServidorProduccion:/etc/apache2/sites-enabled# a2enmod actions
Enabling module actions.
To activate the new configuration, you need to run:
  service apache2 restart
root@ServidorProduccion:/etc/apache2/sites-enabled# /etc/init.d/apache2 restart
* Restarting web server apache2
... waiting
root@ServidorProduccion:/etc/apache2/sites-enabled# [ OK ]
```

Estas directivas, en resumen, dictan que cualquier petición de archivo con extensión .php sea servida por libphp.so, la librería dinámica que ejecuta PHP. Esta configuración evita que se pueda acceder al código fuente, ya que no devolverá el archivo directamente sino que lo hará a través de libphp.so. Aunque, por seguridad, es conveniente tratar de aislar la información confidencial que utilicen estos scripts en un directorio fuera de *DocumentRoot* para que si se produjera un error en el *handler* (*manejador*) no se tuviera acceso al código fuente.

Por último, se debe **mantener actualizado el *framework* de ejecución** para evitar que se exploten vulnerabilidades, como la publicada en mayo del 2012 en PHP, que permitía la ejecución de código arbitrario.

---

## Autenticación y Autorización

---

El control de acceso de los usuarios a los distintos servicios, páginas o directorios **puede realizarse a nivel de aplicación web**, validación de credenciales contra un sistema de base de datos por ejemplo, **o a nivel del servidor web**. A continuación se describirá este último. Se informará de las diferentes maneras de autenticar a un usuario (por contraseña, certificado o dirección IP) y de cómo autorizar el acceso.

### POR CONTRASEÑA

Como se explicará a continuación, en un primer lugar hay que tomar las siguientes decisiones:

- Método de autenticación entre navegador y servidor web: *Basic* o *Digest*.
- Mecanismo de almacenamiento de contraseñas: archivo sin formato, archivo DBM o LDAP.

En los ejemplos siguientes se supondrá que se utilizará como mecanismo de almacenamiento un archivo sin formato. Para crear el archivo que contendrá las contraseñas se debe ejecutar el siguiente comando:

```
htpasswd -c /usr/local/apache/passwords migue
```

El comando solicitará la contraseña del usuario que se está creando, en este caso, "migue".

Es recomendable que este archivo, como todos los que contienen información confidencial, se encuentre **fuera del DocumentRoot y que sea sólo accesible por el usuario de Apache**, aunque Apache almacene el *hash* como una combinación de la contraseña y un valor aleatorio de 32 bits (*salt*).

Hay que tener en cuenta que estos métodos de autenticación son únicamente **recomendables para pocos usuarios** (por ejemplo, administradores). La razón de que no se puedan utilizar para los usuarios comunes del portal es que acceder al archivo de contraseñas y buscarla en él por el nombre de usuario, para verificar la contraseña en cada petición, puede saturar el servidor. Para este tipo de usuarios, lo mejor es una autenticación a nivel de aplicación mediante cookies.

A continuación, se debe especificar el directorio protegido por contraseña. Se puede hacer de dos formas distintas, mediante la directiva *Directory* o mediante archivos *.htaccess*.

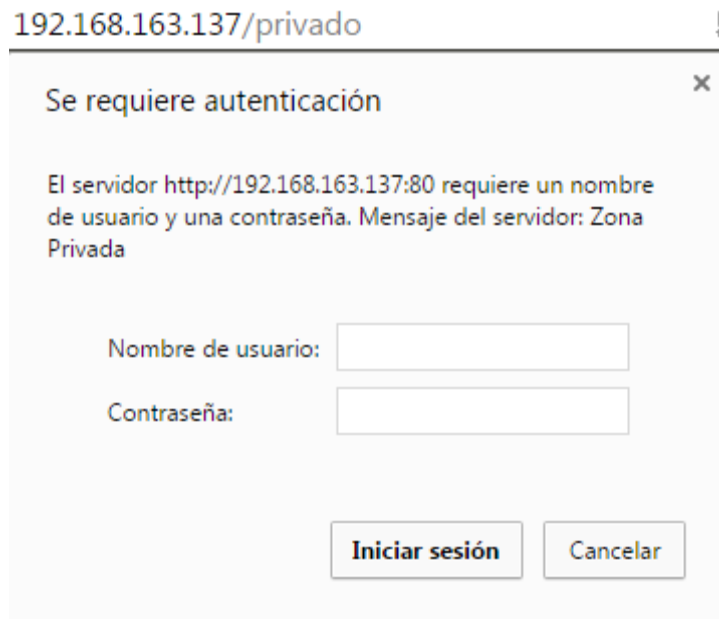
Los **archivos .htaccess** contienen directivas de configuración que se aplican al directorio en el que están contenidos dentro del sistema de ficheros. Su razón de ser es permitir, en servidores compartidos, que los distintos administradores puedan configurar su servidor sin interferir con el resto y sin necesidad de reiniciar el servicio. Como contrapartida, esta búsqueda y procesamiento de los archivos



.htaccess puede penalizar el rendimiento del servidor. Su uso se configura mediante AllowOverride.

```
<Directory /var/www/privado>
    AuthType Basic
    AuthName "Zona Privada"
    AuthBasicProvider file
    AuthUserFile /claves/passwords
    Require valid-user
</Directory>
```

Tras indicar esta directiva y reiniciar el servicio, al intentar acceder al directorio *privado*, se nos mostrará un mensaje como el que aparece a continuación:



192.168.163.137/privado

Se requiere autenticación

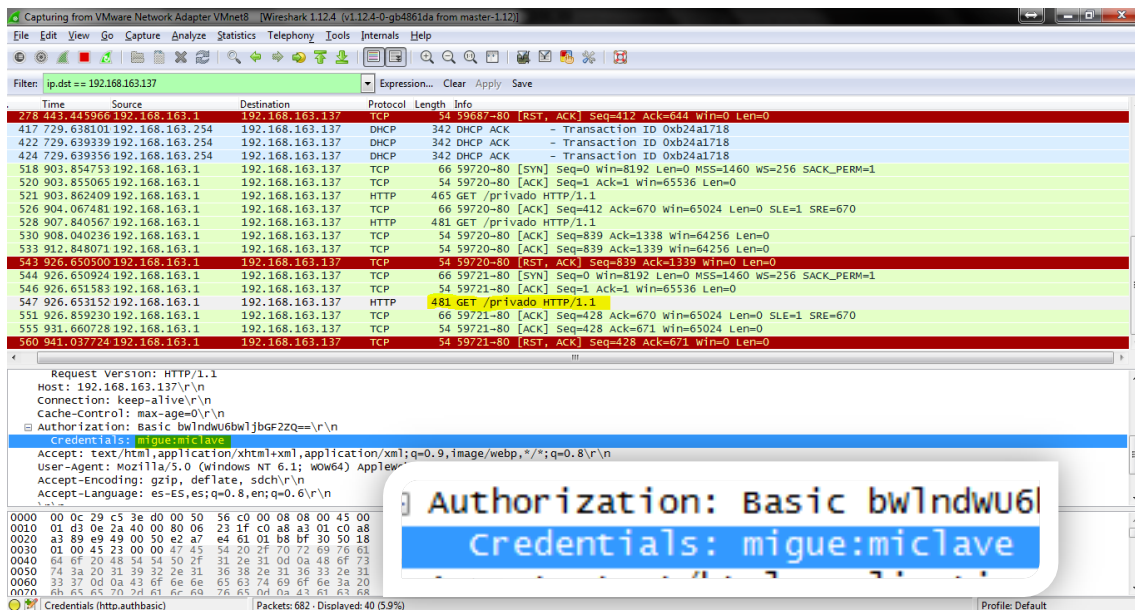
El servidor <http://192.168.163.137:80> requiere un nombre de usuario y una contraseña. Mensaje del servidor: Zona Privada

Nombre de usuario:

Contraseña:

Iniciar sesión Cancelar

Las directivas se explican por sí mismas, sólo destacar AuthType. Esta directiva establece el protocolo de autenticación entre el navegador y el servidor. Puede ser **Basic, que envía la contraseña en claro, o Digest, que envía la hash de la contraseña.** Para evitar ataques de repetición, que permitirían a un atacante autenticarse reenviando un *hash* capturado, en el proceso se incluye un *nonce*, una cadena de texto aleatoria que envía el servidor en la solicitud de autenticación, que se incluye en la generación del *hash*. Como el *hash* enviado cambia a cada petición, sólo es válido una vez.



Aunque no es posible capturar la contraseña, todavía son factibles los ataques de *man-in-the-middle*. Por ello **es recomendable únicamente realizar este tipo de autenticación, ya sea Basic o Digest, sobre HTTPS**; como la comunicación está cifrada de extremo a extremo, no es posible capturar y modificar la comunicación.

Otra directiva interesante es *AuthBasicProvider* y, en el caso de autenticación *Digest*, *AuthDigestProvider*. Estas directivas establecen el método de almacenamiento de las contraseñas en el servidor. En los ejemplos anterior tiene el valor *file*, lo que significa que se almacenan en un archivo. Aunque puede tener otros valores como *ldap*, si se desea autenticar al usuario contra un servicio de LDAP, o *dbm* para hacerlo contra un archivo DBM. Para más información, se puede consultar la documentación del módulo `mod_authnz_ldap`.

## AUTORIZACIÓN A GRUPOS

En el ejemplo de configuración se aprecia cómo se permite el acceso a cualquier usuario válido mediante la configuración "*Require valid-user*". Del mismo modo, se puede permitir el acceso a usuarios que, además de válidos, pertenezcan a un determinado grupo. Ello lo haremos con la directiva *Require group (nombre grupo)*

También existe la posibilidad de hacer tal autorización a un grupo de usuarios, indicándolos individualmente, en lugar de en grupo como en el ejemplo anterior. Para ello tan solo se ha de teclear: *Require user (usuario1) (usuario2) (...)*

## OTROS FACTORES DE AUTENTICACIÓN

A partir de la versión 2.4 de Apache la funcionalidad de la directiva *Require* se ha ampliado, entre otras cosas, permite especificar rangos de IPs autorizadas que antes se implementaban mediante la directiva *Allow*. Esto permitirá añadir host autorizados para conectarse con ese servidor. Muy útil si se dispone de una IP fija.

Ello podrá hacerse con: *Require ip (ip)*

Por otro lado, se ha definido las directivas, como *RequireAll* y *RequireAny*, que permiten la **composición de requisitos de acceso**. En el siguiente ejemplo, extraído de la web de Apache, se permite el acceso a todo el mundo excepto si pertenecen a la red 192.168.163.255, su host es `"*.virus.mipaginaestainfectada.com"`, `"*.te.infectare"` o `"*.pc"`:

```
<RequireAll>
  Require all granted
  <RequireNone>
    Require ip 192.168.163
    Require hosts virus.mipaginaestainfectada.com, te.infectare
    Require hosts pc
  </RequireNone>
</RequireAll>
```

En el siguiente ejemplo, se considera la situación en la que, en una empresa, se quiera otorgar acceso a cierta documentación a los empleados de esa empresa pero no a los que estén con contrato temporal:

```
<Directory /var/www/docs>
  <RequireAll>
    Require group empleados
    Require not group emp_temporales
  </RequireAll>
</Directory>
```

---

## Comunicación HTTPS y Autenticación Con Certificado Cliente

---

Hasta hace unos años, la comunicación entre servidor y cliente se realizaba “en claro” para la mayor parte de los servicios sobre los que viaja “información personal o confidencial”, webmail, acceso a redes sociales, etc. Esto suponía un grave problema de seguridad ya que un usuario malintencionado con los conocimientos suficientes, podría espiar los datos que viajaban a través de la red. Para dar solución a este problema, se comenzaron a implementar sistemas de cifrado entre servidor y cliente que utilizaban algoritmos de clave pública y privada pasando del protocolo “http” al “https”.

Se recomienda que la clave privada asociada al certificado del sitio web tenga una longitud de **2048 bits**.

### VENTAJAS DE HTTPS SOBRE HTTP

HTTPS se diferencia de HTTP en que utiliza el protocolo SSL/TLS lo que le proporciona las siguientes ventajas:

- Comunicación cifrada que por tanto imposibilita que se capture la información (*eavesdropping*).
- Autenticación del servidor, si su certificado ha sido emitido por una CA de confianza, se evitarán los ataques de tipo *man-in-the-middle*.

Por contra, el proceso de cifrado y descifrado lo realiza el servidor, lo que puede llegar a suponer un problema en páginas web con mucho volumen de visitas.

### CONFIGURACIÓN SEGURA DE HTTPS

#### Protección de los archivos de clave privada

Al crearse el archivo de clave privada puede especificarse una contraseña con la que se cifrará el archivo. Al iniciar Apache esta contraseña será solicitada al administrador para que Apache pueda descifrar y utilizar la clave privada.

Si este archivo no se encuentra cifrado debe tener permisos de acceso sólo por el usuario root. El servicio de Apache se inicia con el usuario root para después crear varios procesos que proporcionan el servicio web pero que son ejecutados con el usuario de Apache.

#### Suites de cifrado seguras

La directiva *SSLCipherSuite* establece los protocolos de autenticación y de cifrado que admite el servidor web. Se recomienda establecerla a los siguientes valores:

```

<IfModule mod_ssl.c>
<VirtualHost _default_:443>
    ServerAdmin webmaster@localhost
    SSLHonorCipherOrder On
    SSLCipherSuite RC4-SHA:HIGH:!ADH
    DocumentRoot /var/www
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
    <Directory /var/www/>
        Options Indexes FollowSymLinks MultiViews
        AllowOverride None
        Order allow,deny
        allow from all
    </Directory>

    ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
    <Directory "/usr/lib/cgi-bin">
        AllowOverride None
        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
        Order allow,deny
        Allow from all
    </Directory>

    ErrorLog ${APACHE_LOG_DIR}/error.log

    # Possible values include: debug, info, notice, warn, error, crit,
    # alert, emerg.
    LogLevel warn

    CustomLog ${APACHE_LOG_DIR}/ssl_access.log combined
    [ línea 1/174 (0%), col 1/21 (4%), car 0/7526 (0%) ]
root@ServidorProduccion:/etc/apache2/sites-available# service apache2 reload
* Reloading web server config apache2
root@ServidorProduccion:/etc/apache2/sites-available# /etc/init.d/apache2 restart
* Restarting web server apache2
... waiting
root@ServidorProduccion:/etc/apache2/sites-available#

```

*SSLHonorCipherOrder* es necesaria para que las preferencias del servidor tenga prioridad sobre las del cliente en la negociación de qué algoritmos criptográficos utilizar. La configuración anterior establece lo siguiente:

En primer lugar, si el cliente lo admite, utilizar RC4-SHA. Esta medida tiene el objetivo de evitar, si el cliente lo permite, los algoritmos de cifrado CBC que son vulnerables al ataque Beast que permiten descifrar la información.

El parámetro "HIGH" establece que se utilicen claves de por lo menos 128 bits. Este valor se refiere a la clave de sesión que cifra las comunicaciones, a diferencia de la clave privada del certificado de 2048 bits que se utiliza en la autenticación y en el establecimiento de los parámetros de la comunicación.

El valor "!ADH" elimina de los algoritmos anteriores los que utilizan el protocolo *Anonymous Diffie-Hellman*, porque no proporcionan autenticación del servidor.

## No permitir la renegociación SSL/TLS

Existe una vulnerabilidad en el protocolo SSL/TLS en la renegociación de los parámetros de comunicación que permite a un atacante enviar peticiones al servidor como procedentes del cliente (*spoofing*).

Motivado por esta vulnerabilidad, Apache ha desactivado esta funcionalidad por defecto.

## Permitir únicamente acceso por HTTPS

Lo más común es que los portales sean en su mayoría accesibles por HTTP y sólo las páginas en las que se transmite información confidencial, como formularios o que permiten realizar acciones, sean sobre HTTPS.

Esta doble configuración podría permitir que partes accesibles por HTTPS lo fueran también por HTTP. Este problema se puede evitar con la siguiente configuración que obliga a que el directorio al que se refiere sólo pueda ser accedido por HTTPS:

```
<Directory /var/www/privado>  
    SSLRequireSSL  
</Directory>
```

Como medida de seguridad adicional se puede configurar Apache para que utilice *HTTP Strict Transport Security* o HSTS, para informar al navegador del usuario de que el acceso debe realizarse por HTTPS. Esta medida es útil en el caso de que se produzca un error en el servidor o en la configuración que permita el acceso por HTTP a contenidos restringidos a HTTPS.

## COMPROBACIÓN DE LA SEGURIDAD DE HTTPS

Se pueden comprobar el nivel de seguridad de la configuración HTTPS de un portal a través de la web [www.trustworthyinternet.org/ssl-pulse/](http://www.trustworthyinternet.org/ssl-pulse/), que muestra un informe con las debilidades en la configuración y cómo solventarlas, o utilizar una herramienta como sslyze.

## AUTENTICACIÓN DEL USUARIO POR CERTIFICADO

Además de por contraseña, Apache permite la autenticación de un usuario mediante certificado. El servidor únicamente ha de verificar la firma del certificado de los usuarios. Por ello, sólo necesita el certificado de la CA que emite los certificados clientes, que no tiene porqué ser el mismo que el utilizado en el protocolo HTTPS.

En los ejemplos siguientes se autorizará el acceso a aquellos clientes que presenten el certificado de autenticación del DNIE. La configuración de Apache debe ser la siguiente:

```
# Establece el archivo que contiene el certificado  
# de las CA raíz:  
SSLCACertificateFile "/etc/httpd/ssl.crt/dnie-acraiz-sha256.pem"  
# Establece que la cadena de validación se compone de 2 niveles:  
# una CA raíz y 3 subordinadas:  
SSLVerifyDepth 2  
# Obliga a que el cliente presente un certificado válido:  
SSLVerifyClient require  
# Para exportar la información del certificado  
# y ser usada en CGI:  
SSLOptions +StdEnvVars +ExportCertData
```

La funcionalidad de comprobación de la revocación del certificado mediante OCSP no la proporciona Apache, ha de realizarse mediante CGI.

## AUTENTICACIÓN DEL USUARIO POR CERTIFICADO

Mediante la directiva *Require* y la nueva forma de definir expresiones lógicas en Apache se puede hacer uso de toda la información del certificado cliente para definir unos requisitos de acceso complejos que utilicen entre sus requisitos determinada información del certificado.

En el siguiente ejemplo, extraído de la web de Apache, sólo se permite el acceso a aquellos usuarios que se conecten en horario laboral, con un certificado de la organización "Snake Oil, Ltd.", de la unidad organizativa "Staff", "CA" o "Dev" que no hayan utilizado protocolos inseguros, o bien que se conecten desde la red 192.76.162.255:

```
Require (%{SSL_CIPHER} !~ m/^(EXP|NULL)-/ \
and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd." \
and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"} \
and %{TIME_WDAY} >= 1 and %{TIME_WDAY} <= 5 \
and %{TIME_HOUR} >= 8 and %{TIME_HOUR} <= 20 ) \
or %{REMOTE_ADDR} =~ m/^192\.76\.162\.([0-9])+/
```

Como alternativa se podría utilizar la directiva *SSLRequire*, pero esta directiva tiene previsto eliminarse (está en estado *deprecated*).

---

## Ataques de Denegación de Servicio

---

Un ataque de denegación de servicio, básicamente consiste en saturar los recursos de algún tipo de servicio, En el caso del servidor Apache, o cualquier servidor web, se podría decir que un ataque se podría llevar a cabo mediante peticiones ilegítimas a través de aplicaciones diseñadas para tal fin o aprovechando algún fallo de seguridad, de modo que le falten recursos para procesar las legítimas.

**Por muchos recursos de los que disponga un servidor Apache, un ataque de denegación de servicio puede saturar el servidor.** Hay que tener en cuenta que un ataque DoS puede ser distribuido, DDoS o *Distributed Denial of Service*, y contar con un gran número de máquinas que siempre superarán en recursos al servidor o servidores web. Estos ataques pueden ser realizados sin conocimiento del usuario, porque su equipo ha sido infectado y forma parte de una botnet, o conscientemente por razones de «hacktivismo».

Existen varias medidas para mitigar esta amenaza aunque si el ataque es suficientemente grande, y puede serlo ya que una botnet puede componerse de cientos de miles de equipos, únicamente se puede evitar con la colaboración del ISP (*Internet Service Provider*).

### ATAQUES DOS

A continuación, se citarán algunos tipos de ataques DoS y cómo mitigarlos.

#### Por fuerza bruta

Se basa simplemente en generar un gran número de peticiones HTTP de manera que el servidor no pueda responder a todas. Es posible realizarlo mediante una aplicación especialmente diseñada para tal fin.

#### Ataques SYN Flood

Son más refinados que el anterior ya que tratan de colapsar el servidor a nivel de TCP/IP y no de aplicación. Esto se consigue colapsando una estructura de datos en la que se almacena información sobre las conexiones TCP llamada *Transmission Control Block* o TCB. Para ello, en el *handshake* de establecimiento de sesión TCP, el atacante envía el SYN inicial, el servidor responde con SYN-ACK pero el atacante no envía el ACK final.

En Linux se puede mitigar con el mecanismo de SYN cookies que sólo reserva espacio para la conexión cuando se recibe el mensaje de confirmación final. Se puede activar de la siguiente manera:

```
root@ServidorProduccion:/etc/apache2/sites-available# echo 1 > /proc/sys/net/ipv4/tcp_syncookies
root@ServidorProduccion:/etc/apache2/sites-available# _
```

Otra forma de logarlo es mediante Syn caches que utiliza una estructura de datos aparte y, en el caso de que se complete el *handshake*, copia los datos a la estructura de datos TCB. Este mecanismo sólo está implementado por defecto en FreeBSD.

No es útil tratar de evitar ataques de *sync-flooding* aumentando el tamaño de la cola de espera mediante la directiva ListenBacklog.



## Bandwidth Attacks

Este ataque, también conocido como *hotlinking*, consiste en incrustar enlaces en varios portales web a imágenes o archivos del sitio web atacado de manera que al visitar uno de estos portales automáticamente se realiza una costosa petición al sitio web atacado.

Mediante este ataque se puede utilizar contenido de otros sitios web sin permiso ni «pagar» por el ancho de banda y también se puede llegar a colapsar el servidor atacado.

Este ataque se puede evitar mediante una regla de `mod_rewrite` que prohíba todas las peticiones de acceso a archivos cuya cabecera `HTTP_REFERER`, que contiene el sitio web desde el que se ha realizado la petición, no sea el del propio sitio web:

```
RewriteEngine on
RewriteCond %{HTTP_REFERER} !^http://servidorproduccion.com/*$ [NC]
RewriteCond %{HTTP_REFERER} !^http://sitio.com*$ [NC]
RewriteRule .*\. (jpg|jpeg|gif|png|bmp|zip)$ - [F,NC]
```

## NÚMERO MÁXIMO DE PETICIONES CONCURRENTES

Las consecuencias de no configurar Apache según el número de peticiones previstas podrían llegar a ser las mismas que las de un ataque DoS: el servidor deja de responder a peticiones legítimas. Por esta razón es conveniente configurar la cantidad de recursos que se ponen a disposición de Apache en función del hardware disponible y del número y coste de las peticiones estimadas.

Según la carga y capacidad del servidor se puede configurar el número de peticiones máximo que el servidor será capaz de servir en un determinado momento. Esta configuración depende en gran medida del modelo de gestión de peticiones que está utilizando Apache o *multiprocessing modules* (MPMs). Existen varios tipos:

- `prefork`: utiliza múltiples procesos. Cada uno de ellos se ocupa de una petición. Es el utilizado por defecto en sistemas Unix/Linux.
- `winnt`: Apache se ejecuta en un sólo proceso que contiene varios hilos, *threads*, de ejecución. Se usa en sistemas Windows.
- `worker`: ejecuta varios procesos que a su vez lanzan varios hilos.

La directiva `MaxRequestWorkers`, o `MaxClients` en versiones anteriores a la versión 2.3.13, establece el número máximo de peticiones que se gestionarán concurrentemente. El máximo valor que puede configurarse en esta directiva se controla mediante `ServerLimit`. Las peticiones concurrentes que superen ese número se colocarán en una cola, cuyo tamaño máximo gobierna la directiva `ListenBacklog`, a la espera de que un proceso, en el MPM `prefork`, o hilo, en `winnt` o `worker`, complete su petición o que venza un *timeout* de espera.

Por ejemplo, en la siguiente imagen, correspondiente a un terminal de comandos, se aprecian los procesos de una instancia de Apache que utiliza el MPM `prefork`. Cinco son los que ejecuta en un inicio para gestionar peticiones.

Otro proceso se inició para gestionar una petición y no se ha finalizado porque todavía no se ha alcanzado el valor especificado en *MaxSpareServers*. Por último, el proceso que se ejecuta como root corresponde al hilo principal de Apache.

```
root@ServidorProduccion:/var/www# ps aux | grep apache2
root      4564  0.0  0.7 34984 7696 ?        Ss   04:25   0:00 /usr/sbin/apache2 -k start
www-data  4569  0.0  0.3 35008 3996 ?        S    04:25   0:00 /usr/sbin/apache2 -k start
www-data  4570  0.0  0.3 35008 3996 ?        S    04:25   0:00 /usr/sbin/apache2 -k start
www-data  4571  0.0  0.3 35008 3996 ?        S    04:25   0:00 /usr/sbin/apache2 -k start
www-data  4572  0.0  0.3 35008 3996 ?        S    04:25   0:00 /usr/sbin/apache2 -k start
www-data  4573  0.0  0.3 35008 3996 ?        S    04:25   0:00 /usr/sbin/apache2 -k start
root      4578  0.0  0.0  4412   828 tty1      S+   04:28   0:00 grep --color=auto apache2
root@ServidorProduccion:/var/www#
```

También hay que tener en cuenta si se desea que varias peticiones se gestionen sobre la misma conexión TCP (conexiones persistentes). En este caso el proceso o hilo, después de servir una petición, quedará a la espera de nuevas peticiones. Mejora el rendimiento en páginas web que incluyen varios archivos, como imágenes, aunque mientras el proceso/hilo está a la espera no puede aceptar nuevas peticiones.

## RESTRICCIONES SOBRE LAS PETICIONES

Quizás la directiva más crítica en este sentido es *LimitRequestBody*, útil para limitar el tamaño máximo de las peticiones y archivos que se pueden subir al servidor y que podrían llegar a saturarlo.

## MONITORIZACIÓN DEL USO DE RECURSOS

Teniendo en cuenta que es realmente complejo evitar un ataque DoS en el servidor, la solución más realista es monitorizar el servidor y tener preparado un plan de actuación (que generalmente conlleva acciones por parte del ISP) en caso de que se detecte un ataque.

Aunque más adelante se expondrán otras herramientas de monitorización, para una supervisión sencilla del consumo de recursos se puede utilizar el módulo *mod\_status*. *mod\_status* ofrece información a través de una página web sobre el consumo de recursos del servidor Apache:

- El tiempo de *uptime*.
- El total de accesos y de cantidad de información enviada.
- Algunos valores medios como el número de peticiones por segundo, número de bytes servidos por segundo o el número de bytes por petición.
- El estado de cada proceso de Apache y su porcentaje de uso de la CPU.

Si se está utilizando el HTTPS muestra información sobre el estado de la caché SSL/TLS. También, sobre Apache utilizado como proxy, activando la directiva *ProxyStatus*.

Para habilitarlo se debe añadir la siguiente configuración:

```
ExtendedStatus On
<Location /server-status>
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow from 192.0.2.0/24 Seguridad en Apache 30
</Location>
```

No se debe olvidar imponer restricciones de acceso, mediante IP por ejemplo, sobre la web de mod\_status.

La URL admite un par de parámetros opcionales:

- "refresh=N": para que el resultado se refresque cada N segundos.
- auto: sirve los resultados en modo texto para ser procesados fácilmente por otras herramientas.

## Tabla de Ataques DOS

A continuación se muestra una tabla con una relación entre ataques DOS, a qué afecta y en qué consiste:

Nombre del ataque	Nivel OSI	Tipo de ataque	Explicación del ataque
<b>ICMP echo request flood</b>	L3	Recursos	También denominado Ping Flood. Envío masivo de paquetes (ping), que implican una respuesta por parte de la víctima (pong) con el mismo contenido que el paquete de origen.
<b>IP Packet Fragment Attack</b>	L3	Recursos	Envío de paquetes IP que remiten voluntariamente a otros paquetes que nunca se envían, saturando así la memoria de la víctima.
<b>SMURF</b>	L3	Ancho de banda	Ataque por saturación ICMP que usurpa la dirección de origen para redirigir las múltiples respuestas hacia la víctima.
<b>IGMP Flood</b>	L3	Recursos	Envío masivo de paquetes IGMP (protocolo de gestión de grupos de internet)
<b>Ping of Death</b>	L3	Explotación	Envío de paquetes ICMP que explotan fallos del sistema operativo
<b>TCP SYN</b>	L4	Recursos	Envío masivo de solicitudes de conexión

<b>Flood</b>		TCP	
<b>TCP Spoofed SYN Flood</b>	L4	Recursos	Envío masivo de solicitudes de conexión TCP usurpando la dirección de origen
<b>TCP SYN ACK Reflection Flood</b>	L4	Ancho de banda	Envío masivo de solicitudes de conexión TCP a un gran número de máquinas, usurpando la dirección de origen por la dirección de la víctima. En ancho de banda de la víctima queda saturada por las respuestas a dichas peticiones.
<b>TCP ACK Flood</b>	L4	Recursos	Envío masivo de acuses de recibo de segmentos TCP
<b>TCP Fragmented Attack</b>	L4	Recursos	Envío de segmentos TCP que remiten voluntariamente a otros que nunca se envían, saturando la memoria de la víctima
<b>UDP Flood</b>	L4	Ancho de banda	Envío masivo de paquetes UDP (sin necesidad de establecer conexión previa)
<b>UDP Fragment Flood</b>	L4	Recursos	Envío de datagramas que remiten voluntariamente a otros datagramas que nunca se envían, saturando así la memoria de la víctima
<b>Distributed DNS Amplification Attack</b>	L7	Ancho de banda	Envío masivo de peticiones DNS usurpando al dirección de origen de la víctima hacia un gran número de servidores DNS legítimos. Como la respuesta tiene un mayor volumen que la pregunta, el ataque se amplifica
<b>DNS Flood</b>	L7	Recursos	Ataque de un servidor DNS mediante el envío masivo de peticiones
<b>HTTP(S) GET/POST Flood</b>	L7	Recursos	Ataque de un servidor web mediante el envío masivo de peticiones
<b>DDoS DNS</b>	L7	Recursos	Ataque de un servidor DNS mediante el envío masivo de peticiones desde un gran número de máquinas controladas por el atacante

---

## Monitorización

---

Todo sistema de información, sobre todo aquellos expuestos como los servidores web, necesitan ser monitorizados para detectar errores, ataques o cargas de trabajo excesivas.

Se pueden supervisar varios aspectos de los servidores web. Una manera de clasificarlos es la siguiente:

- Caídas y errores del servidor web.
- Consumo de recursos.
- Ataques informáticos.

### MONITORIZACIÓN DEL SERVICIO

Anteriormente se describió `mod_status`, un módulo utilizado para supervisar la carga de los procesos e hilos de Apache. Además del consumo de recursos, quizás el evento más importante que se debe detectar son las caídas del servicio o demonio, de Apache.

Existen varias herramientas que se pueden utilizar para detectar estos eventos y ejecutar de nuevo el servicio en caso de caídas, caben destacar Nagios y Monit.

### CONFIGURACIÓN Y REVISIÓN DE LOGS

Apache posee dos tipos de logs. Uno almacena información sobre las URLs a las que se han accedido y otro recoge los errores de Apache.

Es conveniente revisar los archivos de log de Apache para detectar errores de configuración, de archivos inexistentes o errores en CGI.

#### Log de error

En este archivo se almacena información sobre eventos importantes, como inicio de la ejecución de Apache, y sobre los errores que se han producido. La localización de este archivo se establece mediante *ErrorLog* y su detalle mediante *LogLevel*:

```
LogLevel warn
ErrorLog ${APACHE_LOG_DIR}/error.log
```

En sistemas Unix, se puede filtrar la información de interés de este fichero mediante comandos como el siguiente que genera una lista de errores únicos:

```
cat error.log | cut -d']' -f 4-99 | sed -e "s/, referer.*//g" | sort | uniq
```

## Log de acceso

El archivo de log y su formato se configura mediante la directiva *CustomLog*. Se puede utilizar *LogFormat* para especificar un formato aparte. En el ejemplo siguiente se ha creado un formato llamado *combined*:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\"" combined
```

```
CustomLog log/access_log combined
```

Las especificaciones del formato *combined* tienen el siguiente significado:

- %h: dirección IP de la petición.
- %l: nombre de usuario de identd o protocolos similares.
- %u: usuario que ha realizado la petición si se ha solicitado autenticación.
- %t: la fecha de la petición.
- \"%r\": la URL de la petición entre comillas dobles.
- %>s: código de respuesta del servidor.
- %b: tamaño de la respuesta, sin incluir las cabeceras.
- \"%{Referer}i\": la cabecera *Referer*.
- \"%{User-agent}i\": la cabecera *User-Agent*.

Se pueden utilizar herramientas como AWStats o W3Perl que procesan la información de este fichero para extraer estadísticas y facilitar su acceso mediante formularios, gráficos y rankings.

En vez de guardar la información directamente en un archivo se puede enviar a otra aplicación mediante *pipes*. Este mecanismo se utiliza para poder rotar los archivos de log automáticamente mediante la aplicación *rotatelog*s que incluye Apache.

## DETECCIÓN DE ATAQUES

La última fase en el proceso de monitorización es la detección de ataques. Sería extenso definir en profundidad las distintas herramientas que se pueden utilizar con este objetivo. Pero, por citar alguna, se podría utilizar un *Web Application Firewall* (WAF) como *mod\_security*, entre los ataques que detecta por defecto se pueden resaltar los siguientes:

- Violaciones del protocolo HTTP.
- IPs pertenecientes a listas negras.
- Detección de malware utilizando el API Google Safe Browsing.
- Detección y prevención de ataques de denegación de servicio.
- Ataques web comunes.