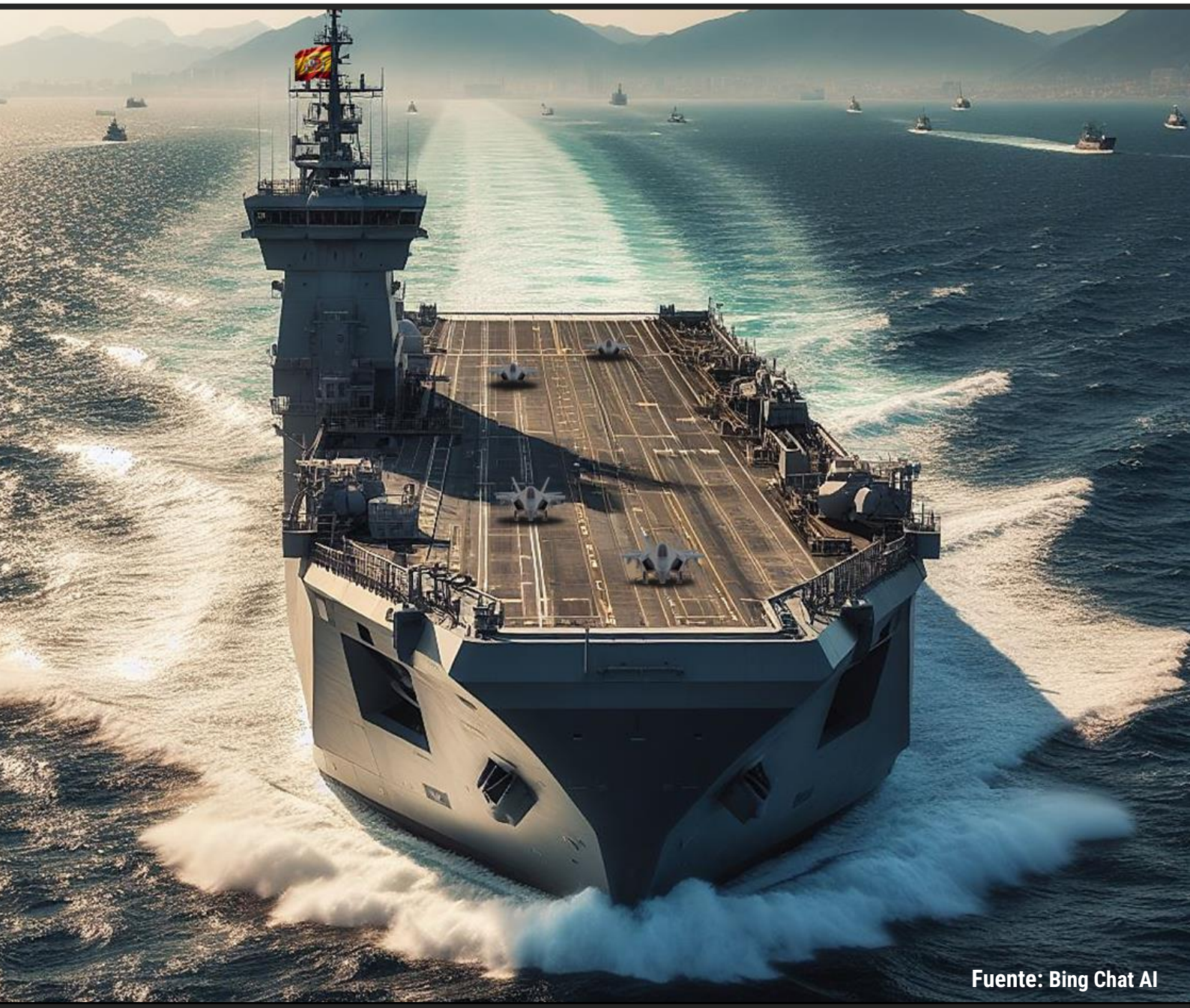


Fuente: Bing Chat AI

Administración de Apache 2

MÁSTER EN INGENIERÍA WEB 2023 – 2024

Administración de Sistemas Operativos v1.2 *("L-62 Princesa de Asturias")*

© José Manuel Redondo López. Universidad de Oviedo

Contenido

Bloque 1: Instalación y Administración de *Apache 2* 6

¿Qué es un servidor web?..... 6

¿Qué es *Apache 2*? 7

Instalación de *Apache 2* bajo sistemas operativos basados en *Debian* 9

Actualización del servidor..... 10

Estructura de una instalación de *Apache 2*..... 12

Organización y configuración básica de *Apache 2* 14

Contextos de aplicación de las directivas de *Apache 2*..... 15

Anidamiento de contenedores 19

¿Qué directiva usar? 21

Ficheros de configuración externos..... 22

Módulos de *Apache 2* 23

Extracción de información del servidor.....29

Anonimización de un servidor *Apache 2* 31

Comprobación de que no está permitido el listado de directorios 38

Optimización de los módulos de *Apache 2*.....42

Módulos de multiprocesamiento o MPMs..... 42

Selección de los módulos en la instalación de *Apache 2* 44

Deshabilitando módulos 49

Ejecución del servidor como un usuario concreto.....53

Configuración de permisos en directorios de una web.....59

Usuarios no root con permisos de edición sobre contenidos concretos 59

Bloque 2: Opciones de Configuración y Extensión 63

Estructura del procesamiento de peticiones de *Apache 2*.....63

Peticiones a *Apache 2* 64

El ciclo de procesamiento de peticiones de *Apache 2*..... 64

Modificación de la cadena de procesamiento de peticiones 75

Tipos MIME (*mod_mime*) 75

Ejemplo de uso de *content handlers*: Negociación de contenidos..... 77

Ejemplo de creación de un *content handler*: Instalación de *mod_python* 81

Ejemplo de filtro de salida: SSI (*Server Side Includes*)..... 86

Ejemplo de habilitación de un filtro de entrada..... 89

Ejemplo de creación de un filtro de salida manual..... 89

Ejemplo de instalación de un filtro predefinido: *mod_deflate*..... 90

Aspectos básicos de ciberseguridad en *Apache 2*.....95

Desactivación de los enlaces simbólicos, CGI *y Server Side Includes* 95
Problemas de seguridad con programas CGI y otro contenido dinámico 96
Seguridad con otras formas de crear páginas dinámicas 100
Problemas de seguridad con enlaces simbólicos 100
Limitación del tamaño de las peticiones y medidas anti-DoS básicas 104
Configuración y uso del *log* 107
Análisis de *log* con *awstats* 108
Otros programas para analizar *logs* 111
Generación de páginas de error personalizadas 112
Bloque 3: Mejora de la Seguridad del Servidor 116
Configuración de restricciones de acceso 116
Módulo *mod_access_compat* 116
Módulo *mod_authz_host* 121
Autenticación basada en el servidor 134
Autenticación básica 134
Autenticación *Digest* 139
Autenticación con los usuarios del sistema operativo 141
Autenticación con un LDAP 144
Habilitación del tráfico cifrado 145
Comprobación de revocación de certificados 149
Uso de suites de cifrado lo más seguras posible 150
Paso a producción: *Let's Encrypt* y *Certbot* 153
Más información 153
Instalación y puesta en marcha de WAFs 155
Instalación de *mod_security 2* 155
Configurar *mod_security 2* 156
Mod_security 3 y contenedores 172
Medidas anti-DoS/DDoS: *mod_evasive* 173
Puesta en marcha y configuración 174
Detección y bloqueo de intrusiones 178
Instalación y configuración 178
Jails adicionales 181
Bloque 4: Explotación de Servidores *Apache 2* bajo *Linux* 183
Configuración de múltiples host virtuales 183
Más ejemplos de configuración de *VirtualHost* 187
Instalación de módulos típicos de lenguajes de servidor 192
Lenguaje *Java* 193

Plataforma *.Net* 197
Lenguaje *PHP* 200
Configuración personalizada de webs particulares 202
Reescritura de URIs: *Mod_rewrite* 206
Uso básico del módulo 206
Cómo se ejecuta *mod_rewrite* 211
Ejemplos comunes de uso 215
Bloqueo de *User Agent Strings* 232
Impedir el robo de ancho de banda: Otra alternativa 234
Optimización automática del servicio de páginas web: Pagespeed 236
¿Qué es *Pagespeed*? 236
Instalación y puesta en marcha 236
Configurar los filtros de *Pagespeed* 237
Potenciales problemas con los contenidos/recursos 244
Uso de *Pagespeed* en diferentes localizaciones 246
Páginas de administración 248
Optimización de recursos de otros dominios 250
Pagespeed y HTTPs 251
Probando *Pagespeed* 252
***WebDAV* 261**
¿Qué es *WebDAV*? 261
Clientes *WebDAV* 262
Instalación de soporte *WebDAV* en *Apache 2* 262
Configuración de *Apache 2* para el uso de *WebDAV* 264
Publicar contenido mediante *WebDAV* 265
Protección de contenidos con TLS 268
Mejores prácticas en webs de subida de archivos 271
Técnicas de ataque 271
Técnicas de defensa 273
Geolocalización de clientes con *mod_geoip* 274
Variables de entorno 275
Ejemplos de uso 277
Bloque 5: Servidores *Apache 2* en Arquitecturas Multiservidor 279
¿Qué vamos a ver? 279
***Proxies* web con *mod_proxy* 281**
Proxy forward 281
Reverse Proxy 282
Arquitectura de un *proxy* 283

Instalación de un *proxy forward* 284
Instalación de un *reverse proxy* 290
Montar una infraestructura *Round-Robin DNS* 295
Pruebas de funcionamiento 299
Clúster *Load-Balanced* con *Apache 2* bajo *Linux* 301
Algoritmos de balanceo de carga 301
Creación de la infraestructura 302
Uso de *rsync* para sincronizar contenidos web 305
El programa *rsync* 306
Instalación y replicación manual 306
Replicación automática 307
Pruebas de funcionamiento 311
Gestión de sesiones en arquitecturas multi-servidor 314
¿De qué trata este bloque? 314
Sesiones *Sticky* 314
Clúster PHP con replicación de sesión usando *memcached* 315
Clúster *Tomcat* horizontal con replicación de sesiones bajo *Linux* 323
Otras arquitecturas compuestas 334

Bloque 1: Instalación y Administración de *Apache 2*

¿Qué es un servidor web?

En pocas palabras, un servidor web no es más que **un programa encargado de mostrar a sus clientes determinados recursos que se le solicitan una vez procesados**, si estos tienen permiso para acceder a los mismos. Esta es una visión muy simplificada de lo que es un servidor web, y a lo largo de este libro se verán muchos matices y detalles que se esconden detrás de esta funcionalidad. No obstante, esta idea debe permanecer en nuestra cabeza a medida que estudiamos las opciones de un servidor web, puesto que nunca debemos perder de vista cuál es el propósito principal del mismo: **servir ficheros**. Y hablamos de ficheros porque un servidor **no sólo sirve páginas HTML**, sino que también puede servir otro gran nº de ficheros diferentes, según la configuración que posea.

En el origen de la web, un servidor web era un programa relativamente simple, cuyo propósito **era servir páginas HTML de texto plano** a los posibles clientes (en su mayor parte navegadores) que éstos procesaban y mostraban. Hoy en día esta funcionalidad **es mucho más amplia**: las páginas requieren un procesamiento en el servidor con algún lenguaje conocido (páginas dinámicas) antes de ser servidas al cliente, y el conjunto de servicios que se suministran puede ir mucho más allá del mero servicio de páginas web. Sin embargo, de todos los posibles tipos de programas que pueden existir, un servidor web es un programa "especial" por varios motivos. De todos ellos, podemos destacar los siguientes:

- **Es un programa con una elevada complejidad y extremadamente configurable**: El programa está pensado para ser útil en un enorme conjunto de situaciones, por lo que cuenta con gran nº de opciones para poder ser usado en cualquier escenario posible.
- **Es un programa sometido a un gran nº potenciales ataques y atacantes**: Un servidor web está expuesto a sus clientes. De ellos, muchos serán clientes estándar sin un fin malicioso, pero otros **sacarán partido de sus posibilidades para hacer alguna clase de daño** al mismo o al contenido que sirve. La seguridad de la web depende de dos pilares fundamentales: cómo se han programado las páginas que se sirven en el servidor y también cómo está configurado dicho servidor. Este libro se ocupará de este segundo aspecto, aunque debe tenerse en cuenta que si ambos no están funcionando de manera adecuada y coordinada nunca tendremos seguridad real.

Desde el punto de vista del administrador, un servidor web debe proporcionar un servicio eficiente y seguro, y esas son las pautas en las que centraremos los contenidos impartidos en este libro. Otros aspectos igualmente importantes, pero que ya no son puramente responsabilidad del administrador, como la adecuada programación de las páginas web, caen fuera del ámbito de lo que vamos a describir aquí.

Por otro lado, hemos dicho que un servidor **es un programa altamente configurable**. Tanto es así que para una sola persona es prácticamente imposible conocer cada detalle de cada una de sus opciones, pudiendo darse el caso de que un administrador se encuentre perdido ante una abrumadora cantidad de opciones que se pudiese encontrar. El objeto de este libro **NO** es conocer todas y cada una de las posibles opciones que tiene *Apache 2*, sino dar a conocer un conjunto amplio de las mismas que se han juzgado como las

más interesantes desde el punto de vista de la administración y la configuración del servidor ante casos típicos y con una secuenciación de "menos a más" que haga el conjunto del libro más fácil de seguir y, llegado el momento de ir más allá de los aspectos básicos, dar a elegir al lector aquellas opciones que más le convengan a la hora adaptarlo a su situación particular.

¿Qué es *Apache 2*?

Apache 2 web server (http://httpd.apache.org/) **es uno de los servidores web más usados en la actualidad**. No es en absoluto el único que hay en el mercado ni tampoco ejerce un monopolio en cuanto a los programas de su categoría, pero su cuota de mercado es considerable y es extremadamente potente y configurable. En la siguiente imagen podemos ver su cuota de mercado frente a sus competidores el 30 de Agosto de 2023:

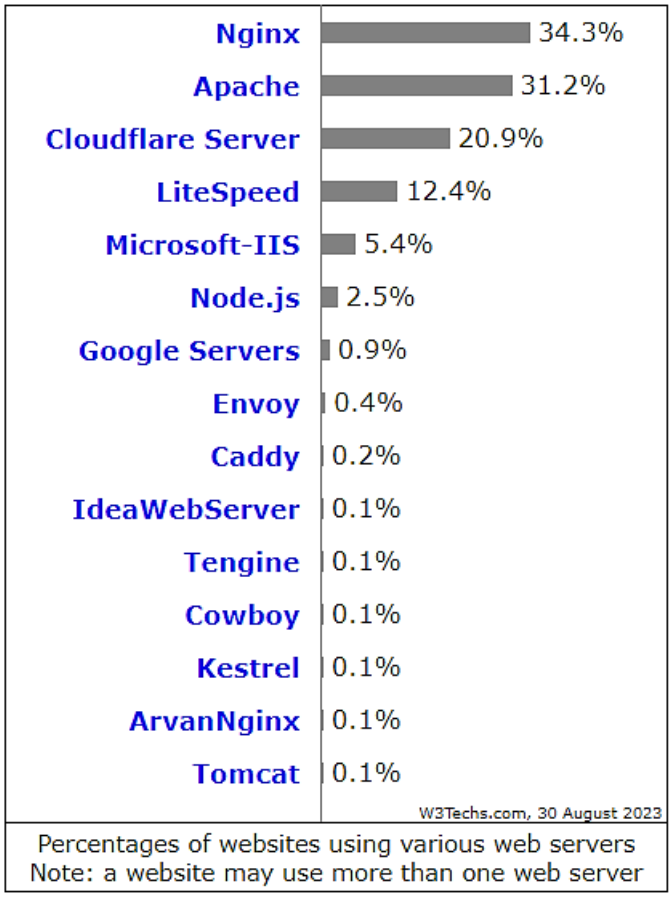


Figura 1. Cuota de mercado de Apache 2 (Fuente: http://w3techs.com/technologies/overview/web_server/all)

En lo que respecta a este libro, más que como un mero manual de nivel básico/intermedio de *Apache 2* puede verse también como una **guía de pasos a seguir a la hora de administrar un servidor cualquiera**. Aunque la ejecución de las operaciones sea diferente en cada servidor, la gran mayoría de cosas que vamos a hacer (**el fin de esas operaciones**) es algo que también deberíamos intentar conseguir en el resto de los servidores que pudiéramos manejar.

Por tanto, si finalmente tuviésemos que usar alternativas como *Nginx* o *IIS*, sería muy aconsejable buscar entre la documentación de estos cómo hacer las operaciones que vamos a describir aquí. Cada servidor web es de un fabricante diferente y su manejo es también diferente. Si bien la transición de *Apache 2* a

Nginx **no es "radical"** porque comparten una base de conceptos y módulos de funcionalidades importantes, si tenemos que elegir IIS en lugar de la versión de *Apache 2* para *Windows* sí que tendremos que aprender a implementar las operaciones descritas prácticamente desde cero, aunque el fin sea el mismo.

No obstante, tampoco podríamos haber hecho un libro de un tamaño relativamente pequeño tratando de ver con la misma profundidad los tres servidores (además de la sobrecarga de contenidos para el lector que supondría) y, obviamente, intentar ver solo superficialmente las utilidades más básicas de los tres también sería contraproducente, ya que lo aprendido apenas tendría utilidad en un entorno de despliegue real. Por tanto, creemos que la idea de usar *Apache 2* como herramienta central para explicar conceptos a un nivel detallado que luego puedan extrapolarse a otros servidores ahora y en el futuro es la orientación más adecuada en estas circunstancias.

Como hemos dicho anteriormente ***Apache 2* tiene versiones para varios SO**, destacando principalmente diversas distribuciones de *Linux* y *Windows*. En nuestro caso vamos a usar *Linux* como base, por ser un SO que permite crear máquinas virtuales de un tamaño muy pequeño, fácilmente replicables y sin problemas de licencias. De todos los *Linux* posibles, **hemos elegido una distribución** Ubuntu Server por varios motivos:

- Es una distribución con un soporte y documentación adecuados.
- La cuota de mercado de dicha distribución es aceptable, siendo una distribución muy popular.
- La versión LTS cuenta con un soporte por parte del fabricante amplio.
- Está basada en *Debian*, lo cual la dota de mucha robustez.

¿Quiere decir eso que este libro no se puede aplicar sobre otras distribuciones de *Linux* o bien sobre *Apache 2 1* en *Windows* incluso? No, no es así. En otras distribuciones de *Linux* **existirá la forma de hacer las mismas operaciones de una manera muy similar**, cambiando todo lo más el nombre de algún fichero o ruta. **En *Windows***, salvando las diferencias en cuanto la propia manera de presentar los recursos como sistemas de ficheros, etc. del propio sistema operativo frente a *Linux*, existirán los mismos ficheros, carpetas y sintaxis de las configuraciones. Aplicar las operaciones sobre *Apache 2* mencionadas aquí sobre otros *Linux* o bien *Windows* no debería ser muy complejo, dado que el servidor *Apache 2* manejado es realmente el mismo.

Y, dicho esto, es hora de empezar a trabajar con nuestro servidor web 😊

Instalación de *Apache 2* bajo sistemas operativos basados en *Debian*

Apache 2 puede instalarse como un paquete obtenible mediante `apt-get` en distribuciones basadas en *Debian* como la *Ubuntu Server* que estamos usando. Para instalarlo basta con ejecutar, con los permisos adecuados, la instalación del paquete correspondiente (`Apache 2`).

Cabe destacar que cualquier procedimiento de instalación / modificación de ficheros del sistema **debe hacerse con permisos de superusuario**, algo que podemos hacer ya que nuestra cuenta (operario) se ha creado **dentro del grupo de usuarios sudoers automáticamente**, al ser el usuario que se creó en la instalación de Ubuntu anteriormente.

1
2
3
4
5

```
operario@servidorapache:~$ sudo apt-get install apache2
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  apache2-bin apache2-data libapr1 libaprutil1 libaprutil1-dbd-sqlite3
  libaprutil1-ldap ssl-cert
Paquetes sugeridos:
  apache2-doc apache2-suexec-pristine apache2-suexec-custom apache2-utils
  openssl-blacklist
```

Figura 2.Instalación de Apache 2 mediante apt-get

Para estar seguros de que el servicio funciona correctamente, debemos reiniciar el equipo. Si una vez reiniciado el servicio no estuviera arrancado podemos forzar el arranque con `sudo service apache2 start`.

Una vez que el servidor está en marcha, **debemos asegurarnos de que es capaz de recibir peticiones HTTP de clientes externos**. Para ello, es necesario saber si los puertos adecuados para estas peticiones están abiertos, es decir, preparados para recibir conexiones. Habitualmente, y salvo que indiquemos lo contrario en la configuración de *Apache 2*, estos puertos son el `80/TCP` (HTTP) y el `443/TCP` (HTTPs). Para saber esto, lo primero es consultar el estado actual del cortafuegos de la máquina gracias al comando `ufw` (*Uncomplicated Firewall*):

```
operario@servidorapache:~$ ufw status
ERROR: Debe ser root (administrador) para ejecutar este guión
operario@servidorapache:~$ sudo ufw status
[sudo] password for operario:
Estado: activo
operario@servidorapache:~$
```

Figura 3. Consulta del estado del cortafuegos del SO

Aquí comprobamos que, si bien el cortafuegos está activo, actualmente **no deja abierto ningún puerto por el que efectuar conexiones**. La máquina es pues segura, pero inútil como servidor web. Si el cortafuegos en cambio nos dijese que está inactivo, debemos activarlo con `sudo ufw enable`.

Por el momento vamos a admitir conexiones no cifradas vía HTTP, por lo que abriremos dicho puerto con `sudo ufw allow 80/tcp`.

```
operario@servidorapache:~$ sudo ufw allow 80/tcp
Regla añadida
Regla añadida (v6)
operario@servidorapache:~$ sudo ufw status
Estado: activo

Hasta                      Acción      Desde
-----                      ------      -----
80/tcp                     ALLOW       Anywhere
80/tcp (v6)                ALLOW       Anywhere (v6)

operario@servidorapache:~$
```

Figura 4. Apertura del puerto HTTP para poder servir páginas web

Con esta operación por fin podremos tener un servidor web *Apache 2* básico en funcionamiento. Para ello podemos comprobarlo accediendo desde el propio host, mostrándonos así la página por defecto:

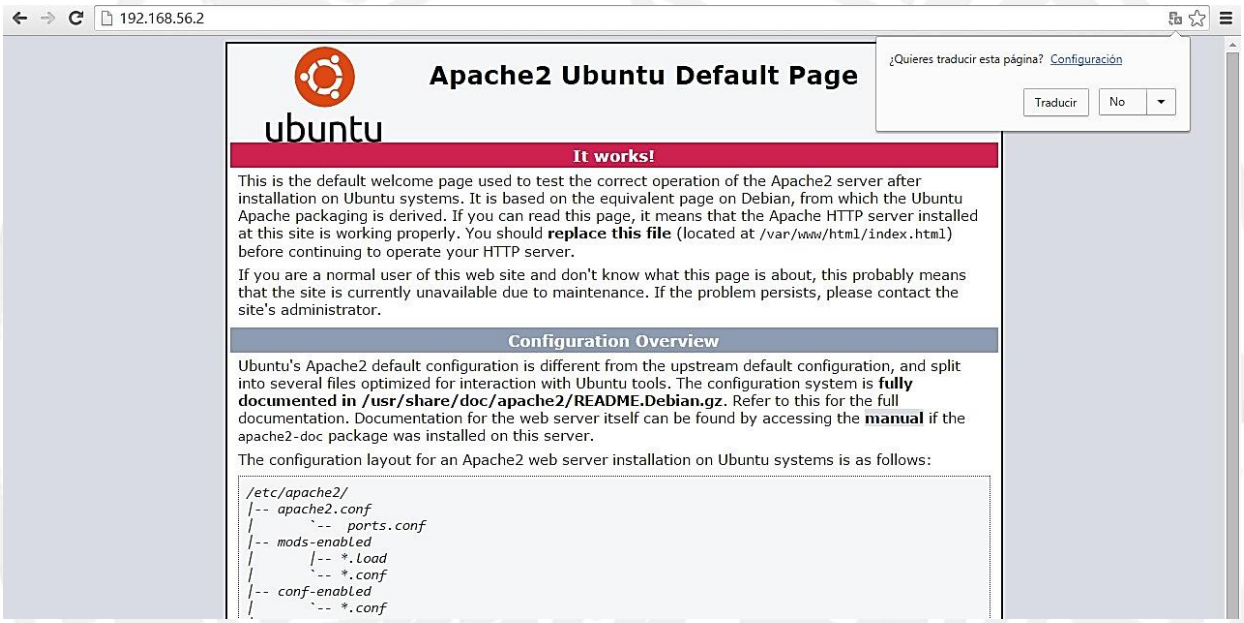


Figura 5. Página por defecto de Apache 2 en Ubuntu

Actualización del servidor

Una vez instalado el servidor y antes de pasar a describir más en detalle sus funciones es conveniente detallar algo más acerca de las actualizaciones del servidor. **Este es un proceso crítico y necesario**. Crítico porque si no lo **hacemos no podremos estar a salvo de las últimas amenazas que nuestra infraestructura pueda sufrir** ni tampoco nos beneficiaremos de las mejoras de rendimiento y seguridad que puedan tener nuevas versiones.

Pero también es crítico por otro motivo, y es que una actualización **puede romper algunas páginas que puedan estar funcionando hasta el momento sin problemas**. No es algo frecuente, pero no descartable y por tanto hay que hacer este procedimiento con una cautela muy alta, especialmente cuando se trata de una web en producción. En esta sección describiremos los aspectos clave de los procesos de actualización y daremos unos consejos acerca de cómo llevarlos a cabo.

Cómo hacer una actualización

Actualizar el servidor web y todos los módulos no es complicado si todos esos elementos se han instalado previamente mediante `apt-get`. El gestor de paquetes del SO, en caso de que haya una nueva versión del servidor o de sus módulos, se encargará de este proceso que no debería dar más complicaciones. El proceso a seguir es el mismo que ya vimos para actualizar el SO.

La cosa cambia si hemos instalado *Apache 2* (o alguno de los módulos que use) **desde su código fuente (compilándolo)**. En este caso tendremos que repetir el proceso de instalación con la nueva versión, prestando atención a las instrucciones específicas de actualización que traerá, que serán distintas para cada producto.

Un último caso ocurre cuando se ha instalado mediante `apt-get` pero el SO ya no tiene soporte. En ese caso **no recibiremos actualizaciones** y lo mejor es intentar migrar cuanto antes a una nueva versión soportada del SO. Esta es la principal razón por la que debemos siempre elegir un sistema operativo base que tenga soporte por muchos años, como el caso de las distribuciones *Ubuntu LTS* (https://wiki.ubuntu.com/LTS).

Por qué hacer una actualización

Si la actualización de *Apache 2* es un proceso que puede darnos problemas en producción, ¿por qué hacerla? Las razones dadas en la introducción de la sección (**seguridad**, **rendimiento**, **robustez**) son más que suficientes para justificar la inversión, pero además hay que resaltar un aspecto de la primera razón: es **una amenaza mayor de lo que parece**.

En una sección posterior vamos a hablar de por qué un servidor **no debe dar información ni de sí mismo ni de sus módulos** para dificultar el trabajo a los posibles atacantes. En ese momento mencionaremos páginas donde se puede ir a buscar fallos de seguridad de cualquier *software*, incluido *Apache 2* y sus módulos. Por tanto, si no actualizamos el servidor o alguno de esos módulos y nuestros atacantes averiguan que no lo hemos hecho (o no lo averiguan y simplemente prueban), pueden ir a esas páginas para ver los fallos de seguridad que tiene el software que estamos usando y actuar en consecuencia.

Por tanto, no se trata de paranoia, realmente **la información sobre vulnerabilidades de un servidor web (y de cualquier *software*) es información pública y obtenible de forma legal**. Ocultar versiones y tipos de productos (que veremos, como hemos dicho, en una sección posterior) dificulta el encontrar vulnerabilidades. Actualizar directamente impide que funcionen las vulnerabilidades conocidas.

Actualizar es por tanto un proceso continuo, una especie de carrera entre la espada (atacante) y el escudo donde la espada golpea primero y hay que ser rápido en ponerse un escudo adecuado (versión nueva) para que nadie use esa espada contra nosotros. Obviamente, actualizar no sirve **contra los errores de seguridad *0-day***, para los que no se conoce actualización que los remedie en el instante en el que se conocen. Contra eso solo vale limitar el acceso al servidor, algo que también se incluye en la asignatura a la que pertenece este libro.

Cuando hacer una actualización

Este proceso **debe hacerse siempre periódicamente**, especialmente cuando algún módulo que usemos tenga alguna vulnerabilidad detectada y documentada. Para estar al tanto de los problemas de los módulos que estamos usando hay que hacer varias cosas:

- **Tener una lista exhaustiva de módulos y otro software instalado**, la versión que está operativa y la web de su fabricante.
- **Suscribirse a las secciones de noticias / *Twitter* / otros medios de notificación de las páginas** que muestran las listas de errores conocidos que veremos posteriormente, para estar al tanto de posibles problemas graves.
- **Proceder a instalar actualizaciones de forma periódica y frecuente**, o inmediatamente si se ha detectado un error de seguridad muy grave sobre un producto que estamos usando (Ej.: el famoso *Heartbleed*).

Cómo hacer la actualización

El proceso más delicado sin duda ocurre cuando, una vez que se ha determinado que es necesario o toca actualizar, hacer la actualización y asegurarse de que todo sigue funcionando correctamente. Para que este proceso cause el menor nº de problemas posible se recomienda tener en cuenta estas consideraciones:

- **Nunca hacer una actualización directamente sobre un servidor en producción**. Causa *downtime*, y es peligroso por si como consecuencia de la actualización algo falla. Lo último que un administrador quiere en un sistema que muestra errores o no funciona.
- **Es mejor tener una réplica del servidor en producción preparada**, actualizar la réplica y comprobar que todo va bien y no hay ningún problema (rotura de paquetes, mal funcionamiento, incompatibilidades…). Una vez hecho esto se puede pasar la réplica a producción y proceder a actualizar el servidor que hasta ahora atendía las peticiones. Existen **políticas estandarizadas de actualización de infraestructuras** que puedes consultar para decidir cuál es la que más te conviene.
- **Obviamente hay excepciones**: No hay forma de disponer una réplica, la vulnerabilidad descubierta es muy crítica… En ese caso no queda más remedio que **intentar hacer la actualización a horas con el menor tráfico posible y tener suerte de que nada salga mal**. No obstante, el riesgo de romper el servicio de un servidor por una actualización está ahí y siempre debe estar presente, hay que minimizar riesgos e intentar tener el menos *downtime* posible.

Estructura de una instalación de *Apache 2*

Según la documentación oficial de *Apache 2* en *Ubuntu*: https://ubuntu.com/server/docs/how-to-install-apache2 , una instalación por defecto de *Apache 2* crea diversos directorios y ficheros en el directorio `/etc/apache2`. *Apache 2* se configura mediante directivas que se escriben en ficheros de texto `.conf` y los más importantes se crean precisamente durante la instalación. Los archivos de configuración y directorios creados en la instalación son:

- `Apache2.conf`: El **archivo de configuración principal de *Apache 2***. Contiene opciones que son globales para *Apache 2*.
- `httpd.conf`: Fue históricamente fue el archivo principal de configuración de *Apache 2*, llamado así por el servicio que originalmente era el *daemon* `httpd`. Actualmente este archivo no se usa (está vacío) **o bien no existe**, pero podría estar presente por temas de compatibilidad con aplicaciones antiguas. Todas las opciones de configuración se han movido a otros ficheros, principalmente el anterior. Es mejor que no escribas nada en él.

- `conf-available`: Esta carpeta contiene los **archivos de configuración disponibles**. Todos los archivos que se ubicaban anteriormente en `/etc/apache2/conf.d` deben moverse a `/etc/apache2/conf-available`.
- `conf-enabled`: Contiene enlaces simbólicos a ficheros del directorio `/etc/apache2/conf-available`. Son **configuraciones de módulos que serán habilitados** en el siguiente arranque de *Apache 2*. También hay ficheros de configuración aparte, como `security.conf`, que contienen configuraciones de un aspecto concreto. Este fichero por ejemplo **contiene opciones de seguridad**, por si no queremos incluirlas en el fichero `Apache2.conf` y que de esta forma la configuración quede más limpia.
- `envvars`: Archivo en el que se establecen las variables de entorno de *Apache 2*.
- `mods-available`: Este directorio contiene los **archivos de configuración para cargar módulos** y configurarlos. No todos los módulos tienen archivos de configuración específicos.
- `mods-enabled`: Mantiene enlaces simbólicos a los archivos situados en `/etc/apache2/mods-available`. Cuando se cree aquí un enlace simbólico al archivo de configuración de un módulo, éste **se activará la próxima vez que se inicie *Apache 2***.
- `ports.conf`: Almacena las **directivas que determinan los puertos TCP** por los que *Apache 2* está escuchando.
- `sites-available`: este directorio tiene **archivos de configuración para los servidores virtuales** (*Virtual Host*). Los servidores virtuales permiten que un mismo *Apache 2* **pueda servir múltiples sitios web** que tengan configuraciones separadas.
- `sites-enabled`: como `mods-enabled`, `sites-enabled` contiene enlaces simbólicos al directorio
- `/etc/apache2/sites-available`. Cuando un archivo de configuración de `sites-available` tiene un enlace simbólico en este directorio, *Apache 2* **permitirá el acceso al sitio web configurado en él** la próxima vez que se reinicie.
- `magic`: Instrucciones para determinar el tipo MIME en base a los primeros bytes de un archivo.
- `/var/www/html`: Directorio que está fuera del directorio `/etc/apache2` y que se crea **para alojar por defecto todas las webs que sirva *Apache 2***.

Además, pueden añadirse tantos archivos de configuración como necesitemos usando la directiva `Include` desde el archivo de configuración que queramos, y se pueden usar comodines para incluir muchos archivos de configuración adicionales. Por tanto, se puede colocar cualquier directiva en cualquiera de los archivos de configuración que se incluyan. Es muy importante recordar que los cambios en los archivos de configuración **son reconocidos por *Apache 2* solo cuando se inicia o se reinicia**.

El servidor también lee un archivo que contiene los tipos de documento MIME que pueden servirse, pudiendo añadir o invalidar algunos de estos tipos. El nombre de archivo es configurado por la directiva `TypesConfig` (por defecto es `/etc/mime.types`), típicamente a través de `/etc/apache2/mods-available/mime.conf`.

Organización y configuración básica de *Apache 2*

Para entender cómo trabajar con *Apache 2* a partir de ahora debemos tener en cuenta varias cosas:

- *Apache 2* **recién instalado es un programa que ofrece una funcionalidad muy básica**, que se complementa mediante módulos o ***mods***.
- Por tanto, **es un programa modular** donde el administrador puede seleccionar y **personalizar** la funcionalidad final que tiene en todo momento seleccionando el conjunto de *mods* activos.
- Los módulos pueden ser compilados como ***DSO* (*Dynamic Shared Objects*)**, de manera que existan **separados** del ejecutable principal. También pueden compilarse en el mismo momento que se compila el servidor, o bien se pueden compilar y añadir más tarde con la herramienta *Apache 2 Extension Tool* (http://httpd.apache.org/docs/current/programs/apxs.html). Finalmente, la última opción es **compilar los módulos directamente dentro del ejecutable principal de *Apache 2***.
- El soporte de *DSO* en *Apache 2* se basa en el uso del módulo `mod_so`. Este *mod* y el *mod* `core` son los únicos que no pueden ser *DSOs*, porque son necesarios precisamente para que este tipo de módulos se puedan usar. Prácticamente todo el resto módulos de *Apache 2* pueden convertirse a *DSO*. En este libro manejarán **fundamentalmente módulos de tipo *DSO***, que podremos encontrar en el directorio visto `mods-available`.
- Para activar un *DSO* se debe incluir una directiva `LoadModule` en la configuración de *Apache 2*, aunque el procedimiento más sencillo es usar la herramienta `a2enmod`, como veremos más adelante. La desactivación de un DSO se hace con la herramienta equivalente `a2dismod`. En cualquier caso, **la alteración del nº de módulos que Apache 2 tiene activos requiere el reinicio del servidor**. **Más información**: http://httpd.apache.org/docs/current/dso.html
- Cada *mod* puede tener **su propia configuración asociada**. En caso de que la tenga, se puede cambiar en un fichero de texto plano `.conf` cuyo nombre es idéntico al *mod* y que también está en el directorio `mods-available` mencionado.
- Todo fichero de configuración se crea a base de **directivas y valores**. Cada directiva tiene una serie de valores posibles y cada *mod* puede tener sus propias directivas (**Moraleja**: Si poner una directiva y al arrancar *Apache 2* dice que no se reconoce, seguramente no has cargado el *mod* que la define). *Apache 2* tiene además una serie de directivas "*core*" que pueden modificar el comportamiento del servidor en general.
- **Existe un enorme nº de directivas y de posibles valores de estas**, que hace imposible tener un control absoluto de todas. También existe un gran nº de *mods*, por lo que estas directivas crecerán a medida que se instalen más. Por suerte, **la documentación de *Apache 2* es muy completa** en lo relativo a cómo usarlas y dónde se aplican.
- Cada directiva puede tener hasta **cuatro posibles contextos de aplicación**. Estos contextos son el ámbito en el que surtirán efecto si se activan dentro del mismo. **No todas las directivas se admiten en todos los contextos**, por lo que una directiva no puede incluirse dentro de un contexto para el que no esté pensada. Debemos mirar cuidadosamente la documentación para no generar un error de configuración que impida que el servidor funcione.

Contextos de aplicación de las directivas de *Apache 2*

Los contenidos que veremos para administrar *Apache 2* siguen una serie de **principios básicos que se mantienen constantes**, sin importar la parte del servidor que estamos configurando.

- Todo se configura en **archivos de texto** `.conf`
- Toda configuración consiste en un **conjunto de directivas asociadas a sus valores**
- Cada directiva tiene un **conjunto de valores posibles** y el efecto de la directiva depende del contexto en el que se encuentren definidas.
- ¿Por qué? Pues porque los contextos donde una directiva se define determinan si afectan a todo el servidor, a un directorio concreto, a determinados tipos de ficheros, a un host virtual (que veremos qué es) o a URIs.

Esto será lo que veremos en esta sección, y es muy importante entenderlo adecuadamente antes de profundizar más en el estudio de *Apache 2*. En adelante, **a los contextos donde una directiva puede definirse les llamaremos "contenedores"**, al ser, a efectos prácticos, contenedores de directivas. Hay dos tipos de contenedores de directivas:

- Los que se evalúan sólo **cada vez que el servidor arranca o re-arranca**.
- Los que se examinan **cada vez que el cliente hace una petición** (y por tanto solo se aplicarán las directivas que contienen si la petición es sobre un elemento que encaja con la definición del contenedor). La mayoría de los contenedores son de esta clase, es más, todos los que no sean de la primera categoría lo son de la segunda.

Contenedores que se evalúan en cada arranque o reinicio

Solo hay tres contenedores de directivas que se evalúan de esta forma. Si las condiciones que establecen (que veremos a continuación) son ciertas cuando el servidor arranca, entonces todas las directivas que definen **se aplicarán a TODAS las peticiones**, ignorándose en caso contrario. Los contenedores de esta clase son:

- `IfDefine`: Aplica sus directivas contenidas si el **parámetro que acompaña a la directiva se ha definido en la línea de comandos** al arrancar *Apache 2*. Por ejemplo, si ejecutamos `Apache 2 -Dproduccion`, y definimos este contenedor:

```
<IfDefine produccion>
    Redirect "/web" "http://maquinadeproduccion.miweb.com/" </IfDefine>
```

Entonces todas las peticiones a `/web` pasarán a ser redirigidas por el servidor de producción.

- `IfModule`: Es análoga a la anterior pero solo ejecuta sus directivas **si un módulo concreto está disponible en el servidor**. Como hemos dicho, los módulos pueden ser tanto compilados como cargados con `LoadModule` antes de que se defina la directiva. Esta directiva permite crear configuraciones especiales para ciertos módulos, o bien aplicar algunas directivas si ciertos módulos están cargados. La documentación oficial muestra un ejemplo sencillo y claro de cómo funciona esta directiva, que asigna un valor a una directiva concreta solo si el módulo que la define está disponible:

```
<IfModule mod_mime_magic.c>
    MimeMagicFile "conf/magic"
</IfModule>
```

- Finalmente, la directiva `<IfVersion>` es análoga a las anteriores, solo que esta vez depende de la versión de *Apache 2* que se esté ejecutando. Es de utilidad si tenemos alguna funcionalidad que usa módulos que han sido renombrados en versiones diversas de *Apache 2* o bien queremos aprovecharnos de una determinada característica que solo esté disponible a partir de una versión dada.

```
<IfVersion >= 2.2>
      #Esto solo se ejecuta para versiones de Apache 2.2 o superiores </IfVersion>
```

Las tres directivas admiten condiciones negadas (con el carácter `!`) y permiten ser anidadas si necesitamos más complejidad.

Contenedores globales

Las directivas del contenedor `<VirtualHost>` solo se aplican a host virtuales concreto de los alojados por el servidor de *Apache 2*. Estos contenedores son útiles **cuando hay varias web alojadas en el mismo servidor**, ya que permiten tener múltiples web funcionando bajo el mismo servidor *Apache 2* y con una configuración independiente. Explicaremos más acerca de host virtuales en el futuro.

Por otro lado, los contenedores `<Proxy>` y su equivalente `<ProxyMatch>` solo aplican las directivas que contienen a aquellas web especificadas como parámetro que se accedan a través de un proxy creado gracias a `mod_proxy` (http://httpd.apache.org/docs/current/mod/mod_proxy.html).

Contenedores condicionales

La directiva `<If>` (http://httpd.apache.org/docs/trunk/mod/core.html#if) permite aplicar una configuración o no en función de una condición que devuelve un valor booleano. Por ejemplo el siguiente ejemplo aplica una serie de directivas si el cliente no es www.clientedeconfianza.com:

```
<If "!(%{HTTP_REFERER} -strmatch 'http://www.clientedeconfianza.com/*')">
    #Directivas
</If>
```

Contenedores del sistema de ficheros o de localizaciones web

Estos contenedores son los más comunes a la hora de configurar directivas en *Apache 2*. Sin embargo, es frecuente confundir el propósito de ambos:

- Los contenedores que aplican directivas sobre ficheros o directorios **usan las rutas del disco duro** tal y como lo ve el SO, en términos de ficheros y directorios físicos que existen en una ruta real.
- Los contenedores que aplican directivas sobre localizaciones web **usan las rutas existentes en el servidor web** tal y como los clientes pueden acceder a ellas. Por ejemplo, la localización `/imágenes` de la web www.miweb.com/imagenes puede corresponder a la ruta del sistema de ficheros `/var/www/html/miwebdeimagenes/uploads/imágenes`. Dicho de otra forma, una localización web **NO TIENE POR QUÉ** corresponder a una ruta física del disco duro. Es más, una localización ni siquiera debe tener una ruta física real asociada.

Contenedores del sistema de ficheros

Son los contenedores `<Directory>` y `<Files>`. Como hemos dicho, aplican directivas sobre aquellos ficheros que están dentro de un directorio del disco duro (primer caso) o bien sobre ficheros de una determinado tipo (segundo caso). **Una directiva que se aplica sobre un directorio también se aplica sobre todos los subdirectorios de este y los ficheros contenidos en todos ellos**. En el siguiente ejemplo, todas las directivas que incluyamos dentro del contenedor `/var/www/html/web1`, se aplicarán sobre el directorio, sus subdirectorios y sus archivos:

```
<Directory “/var/www/html/web1”>
    #Directivas
</Directory>
```

Las directivas dentro de un contenedor `<Files>` se aplican sobre cualquier fichero que tenga el nombre indicado independientemente de la carpeta en la que esté. Por ejemplo, esta configuración se aplicará sobre rodos los ficheros llamados `contenidoprivado.htm`:

```
<Files "contenidoprivado.html">
    #Directivas
</Files>
```

Además, ambos contenedores **pueden anidarse para conseguir más control**. En este ejemplo se aplica una directiva todos los ficheros `contenidoprivado.htm` que estén en `/var/www/html/miweb1/contenidos` y todos sus subdirectorios:

```
<Directory “/var/www/html/miweb1/contenidos">
    <Files "contenidoprivado.html">
        #Directivas
    </Files>
</Directory>
```

Por otro lado, estos contenedores pueden **usar los comodines clásicos * y ?** para referirse a nombres de carpetas o ficheros. La directiva `<Directory>` y la `<Files>` admiten expresiones regulares si les añadimos el carácter `~`, por ejemplo:

```
<Directory ~ "^/var/www/html/.*/ids/[A-Z]{3}">…</Directory>
```

Permitirá aplicar directivas sobre los directorios compuestos por 5 letras mayúsculas que se encuentren dentro del directorio `ids` que a su vez está dentro del directorio raíz de *Apache 2* para webs por defecto (`/var/www/html`).

Si necesitamos más flexibilidad, existe una versión de estos contenedores (`<DirectoryMatch>`, `<FilesMatch>`) que permiten el uso de expresiones regulares compatibles con el lenguaje *Perl* (PCRE, https://en.wikipedia.org/wiki/Perl_Compatible_Regular_Expressions), que es un formato muy ampliamente usado para expresiones regulares. Ejemplos:

```
<DirectoryMatch "/var/miweb/*/">
    #Directivas
</DirectoryMatch>
```

Con expresiones regulares se pueden hacer referencia a **varios tipos de fichero al mismo tiempo**. También se pueden usar **partes de *paths* de ficheros o URIs**:

```
<FilesMatch "\.(?i:gif|jpe?g|png)$">
    #Directivas
</FilesMatch>
```

Contenedores de localizaciones web

Se definen con la directiva `<Location>`. Por ejemplo:

```
<Location “/admin”>
      #Directivas
</Location>
```

Admite también **expresiones regulares** anteponiendo el carácter `~`, como en el caso de `<Directory>`. Por otro lado, existe un contenedor equivalente `<LocationMatch>` que también admite expresiones regulares, como los vistos en la sección anterior. El siguiente ejemplo aplica las directivas a cualquier URI que comienza por `/privado`, como podría ser `http://miweb.com/privado`, `http://miweb.com/privadoymas`, o `http://miweb.com/privado/cuentas/balance2015.html`, así como cualquier petición que comience por `/privado`.

```
<LocationMatch "^/private">
    #Directivas </LocationMatch>
```

Ficheros .htaccess

Los ficheros `.htaccess` (https://httpd.apache.org/docs/current/howto/htaccess.html) son también llamados **ficheros de configuración distribuida**. Son una forma de hacer cambios a la configuración general de *Apache 2* **directorio a directorio**. Estos ficheros no tienen un formato particular: son ficheros de texto plano con un conjunto de directivas idéntico al que puede haber en otros ficheros de configuración. Suelen encontrarse en directorios concretos, y lo que establecen **se aplica a ese directorio y a todos sus subdirectorios**.

Para que las directivas en estos ficheros funcionen **hay que hacer uso primero de la directiva** `AllowOverride` (https://httpd.apache.org/docs/current/mod/core.html#allowoverride) a nivel global sobre ese directorio. Esta directiva especifica las categorías de directivas permitidas en los ficheros `.htaccess`, o dicho de otra forma, qué se va a poder cambiar en ellos y qué no. Las categorías de directivas contempladas son las siguientes, pudiendo especificar varias en un mismo `AllowOverride` separándolas por espacios en blanco:

- `AuthConfig`: Permite el uso de directivas **relacionadas con la autorización**. Posteriormente veremos una sección sobre este tipo de directivas.
- `FileInfo`: Permite el uso de directivas que trabajan **con tipos de documentos** (tipos MIME, filtros de entrada y salida, etc.)
- `Indexes`: Permite el uso de directivas que controlan el **listado de contenidos del directorio** y qué se muestra exactamente.
- `Limit`: Permite el uso de directivas que controlan el acceso a los recursos del directorio.
- `Options`: Permite usar directivas que controlan características concretas de los directorios, con la sintaxis `OpcionX=valor`

La única razón por la que existen los ficheros `.htaccess` es para permitir a un usuario hacer ciertos cambios en la configuración del servidor de manera controlada (hemos visto que solo podrá cambiar

aquellas cosas que se le permitan previamente con `AllowOverride`) en aquellas situaciones en las que no tiene acceso a la configuración global del mismo ni puede solicitar que el administrador "general" haga los cambios que necesita.

No obstante, es importante tener claro que el uso de ficheros `.htaccess` implica una carga de trabajo adicional no desdeñable por cada petición al servidor, por lo que su rendimiento se puede ver afectado. Por ese motivo, **se debe intentar no usar nunca este tipo de ficheros**, máxime si tenemos acceso a la configuración central del servidor o podemos solicitar cambios. Todo lo que puede ir en un fichero `.htaccess` puede ir en un contenedor `<Directory>`, teniendo el mismo efecto con mejor rendimiento. Si lo que hay en el directorio es una web concreta, entonces es mejor crear un host virtual para ella. Hablaremos de host virtuales en detalle más adelante.

Como nota final contra el uso indiscriminado de estos ficheros, cabe decir que si queremos migrar nuestras web a el servidor *Nginx*, si bien existen equivalencias con muchos conceptos de *Apache 2,* no es el caso con los ficheros `.htaccess`, por lo que cualquier cosa que situemos en ellos tendremos que cambiarla al mejor lugar posible de cara a una potencial migración de contenidos. Su uso además puede llevar a introducir **problemas de seguridad** en la administración del servidor.

Anidamiento de contenedores

Apache 2 está preparado para que **determinados contenedores puedan estar dentro de otros contenedores** para aumentar así la flexibilidad y la potencia de la configuración que podemos necesitar. Por ejemplo, es posible tener contenedores `<Files>` dentro de un contenedor `<Directory>` para decir qué pasa con unos determinados ficheros que se encuentran en un directorio determinado, como hemos visto antes. Combinándolo con el uso de expresiones regulares con los equivalentes `<DirectoryMatch>`, `<FilesMatch>`, o incluso directivas `<If>` dentro de otros contenedores para casos complejos de configuración, debería ser suficiente para cubrir todas las necesidades.

No obstante, **no todas las directivas se permiten en todos los contenedores**. No obstante, todo lo que se permite en un contenedor `<Directory>` se permite también en contenedores `<DirectoryMatch>`, `<Files>`, `<FilesMatch>`, `<Location>`, `<LocationMatch>`, `<Proxy>` y `<ProxyMatch>`, salvo algunas excepciones:

- `AllowOverride` solo se permite en secciones `<Directory>`.
- Los parámetros `FollowSymLinks` y `SymLinksIfOwnerMatch` de la directiva `Options` solo funcionan en contenedores `<Directory>` o ficheros `.htaccess`.
- La directiva `Options` no puede usarse en contenedores `<Files>` o `<FilesMatch>`.

Orden de evaluación de los contenedores

Cuando los ficheros de configuración son complejos, tienen muchos contenedores y aplican muchas directivas, puede darse el caso de que **encontremos configuraciones contradictorias accidentalmente**. Esto es así porque al final una directiva puede estar aplicándose sobre una localización que ya fue configurada por otra, y en ese caso hay que tener en cuenta las reglas de precedencia, es decir, el orden de evaluación de los contenedores.

Es importante intentar **no solapar localizaciones en diferentes contenedores**, pero si lo hacemos hay que tener en cuenta el orden en el que los contenedores se evalúan, ya que unos contenedores siempre se evaluarán después de otros. Debe tenerse en cuenta que un contenedor anidado de un tipo X siempre va

a ser evaluado **después** que los contenedores del mismo tipo que no estén anidados dentro de otros. El orden de evaluación es el siguiente:

1. `<Directory>` y ficheros `.htaccess`, con las siguientes normas adicionales:
   a. Los ficheros `.htaccess` son más prioritarios que los contenedores `<Directory>` si tienen directivas que afectan a localizaciones similares.
   b. A la hora de evaluar contenedores `<Directory>`, siempre se hará de manera que el directorio más corto se evalúa antes que el más largo: `<Directory /var/www/html/>` se evaluará antes de `<Directory /var/www/html/miweb/admin/>`.
   c. Si tenemos varias secciones `<Directory>` aplicables sobre el mismo directorio, el orden de evaluación es el orden de aparición en el fichero de configuración.
2. Contenedores `<DirectoryMatch>`
3. Contenedores `<Files>` y `<FilesMatch>`
4. Contenedores `<Location>` y `<LocationMatch>`
5. Contenedores `<If>`

Además, tenemos que considerar unas reglas adicionales:

- El orden de aparición en el fichero de configuración es importante, ya que **las secciones del mismo tipo se evalúan en dicho orden por defecto**: Las secciones que aparecen más tarde **sobrescriben** a las que aparecen primero. No obstante, cada módulo puede decidir alterar este comportamiento y los contenedores pueden tener reglas de evaluación que alteren este orden (por ejemplo el tamaño de la ruta de los directorios en contenedores `<Directory>`).
- Si se **importan ficheros de configuración externos** con `<Include>` (ver sección siguiente) se tratarán como si el contenido de este estuviese exactamente en el mismo lugar donde se hace dicho `<Include>`.
- **Los contenedores y directivas dentro de una sección** `<VirtualHost>` se evalúan después de las secciones correspondientes fuera de dicho `<VirtualHost>`. El objetivo es que los host virtuales puedan sobrescribir la configuración del servidor global.
- Cuando una web se sirve con `mod_proxy`, las secciones `<Proxy>` ocupan el mismo lugar en la cadena de evaluación vista antes que las secciones `<Directory>`.

Por ejemplo, esta configuración se evaluará en orden numérico ascendente:

```
<Location "/">
    #5
</Location>

<Files "fichero_concreto.html">
    #4
</Files>

<VirtualHost *>
<Directory /var/www/html/>
         #2
</Directory>
</VirtualHost>

<DirectoryMatch "^.*b$">
    #3
```

```
</DirectoryMatch>

<Directory /var/www/html/>
    #1
</Directory>
```

Pero es sencillo cometer un error y hacer secciones de configuración que no se evaluarán nunca. Por ejemplo, como `<Location>` siempre se evalúa después que `<Directory>`, el `<Directory>` de este fichero, aunque aparezca después, no tendrá efectos porque será sobrescrito por lo dicho en la directiva `<Location>`:

```
<Location "/">
    #Directivas
</Location>

<Directory "/">
    #Directivas
</Directory>
```

Finalmente, los ficheros `.htaccess` pueden sobrescribir las directivas existentes en contenedores `<Directory>` si la configuración de `AllowOverride` se lo permite. Pero ha de tenerse en cuenta **que estos contenedores son los primeros en evaluarse**, por lo que un fichero `.htaccess` puede ser sobrescrito a su vez por los contenidos de contenedores que se evalúan más tarde en los ficheros de configuración centrales. Por ello, aunque tengamos un `AllowOverride` correcto y también un fichero `.htaccess`, sus contenidos pueden quedar reducidos a nada, como ocurre en el siguiente ejemplo:

Fichero `.htaccess`:

```
<Directory "/var/www/html/">
    #Directivas
</Directory>
…
Fichero Apache2.conf:

<Location "/">
      #Todas las directivas que se pongan aquí y que también estén en el
      #fichero .htaccess sobrescribirán a las del fichero </Location>
```

¿Qué directiva usar?

Aunque el significado de la mayoría de las directivas ya indica qué tipo de uso debemos darle, una de las dudas que puede surgir a la hora de crear configuraciones es si hacerlo sobre la estructura de directorios, con `<Directory>` y similares, o bien sobre localizaciones web, con `<Location>` o similares.

Elegir una u otra es fácil teniendo en cuenta **una máxima**: **NUNCA** aplicar una directiva `<Location>` sobre un directorio o fichero que resida físicamente en el disco duro del servidor. Este tipo de directivas solo están pensadas para páginas generadas dinámicamente o bien URIs que se generen por otros medios que veremos, pero que no correspondan exactamente a un fichero o directorio. Debe tenerse en cuenta que a medida que vayamos conociendo más aspectos del servidor nos va a ser más fácil generar diferentes URI que se refieran al mismo fichero físico, por lo que si queremos hacer una determinada configuración sobre

un fichero o directorio usando `<Location>` tenemos que estar seguros de que cubrimos todas las posibles vías de acceso al mismo, algo que es más difícil y propenso a errores u omisiones. Por ejemplo:

```
<Location "/ficherosprivados/">
    #Directivas
</Location>
```

Si la petición es www.miweb.com/ficherosprivados, entonces se aplicará la directiva. No obstante, si la petición es www.miweb.com/FicherosPrivados ya no sabemos si se aplicará o no, puesto que dependerá de si el sistema de ficheros es sensible a mayúsculas. Esto no hubiera pasado con un contenedor `<Directory>`, que **se aplicaría independientemente de cómo se acceda al mismo**. Finalmente, hay que recordar que, a pesar de lo dicho, la directiva `<Location "/">` se aplicará por lógica a todas las peticiones, independientemente de la URI accedida.

Ficheros de configuración externos

Como hemos dicho antes, toda la configuración de *Apache 2* se encuentra o bien en el fichero `Apache2.conf` o en los ficheros `.conf` de cada módulo que tenga uno asociado. Eso no quiere decir que no podamos **hacer nuestros ficheros de configuración propios** porque, como ya se comentó de pasada en secciones anteriores, podemos incluir ficheros externos dentro de un fichero de configuración.

Crear ficheros de configuración propios es sencillo y conveniente para no "sobrecargar" un fichero de configuración, haciéndolo grande y mucho menos manejable. Lo más conveniente es "modularizar" estos ficheros, de manera que **cada uno contenga la configuración de una localización o bien de un aspecto concreto del servidor** (seguridad, acceso a determinados elementos, etc.). Hecho esto, basta con crear las directivas `Include` adecuadas dentro de `Apache2.conf` y que así su contenido sea tenido en cuenta. Se debe tener en cuenta también lo que se dijo en la sección anterior acerca de que el contenido de un fichero referenciado por un `Include`, a efectos de prelación, consta como si se hubiera escrito su contenido en el mismo lugar donde se hace dicho `Include`.

Para ver un ejemplo de cómo hacer esto y de esta forma tener los medios para crear unos ficheros de configuración adecuados desde el principio, veremos un pequeño ejemplo donde se va a crear un fichero `web1.conf` donde se contendrían todas las directivas aplicables a los contenidos del directorio `/web1` que se encuentra dentro del directorio `/var/www/html` (recuerda, directorio por defecto para contener webs de *Apache 2*). El primer paso es, por supuesto, crear el fichero de configuración, por ejemplo en el mismo lugar que `Apache2.conf` (para no tener que usar rutas en el `Include`, aunque podríamos hacerlo teniendo en cuenta que la "raíz" es el propio directorio donde está `Apache2.conf`):

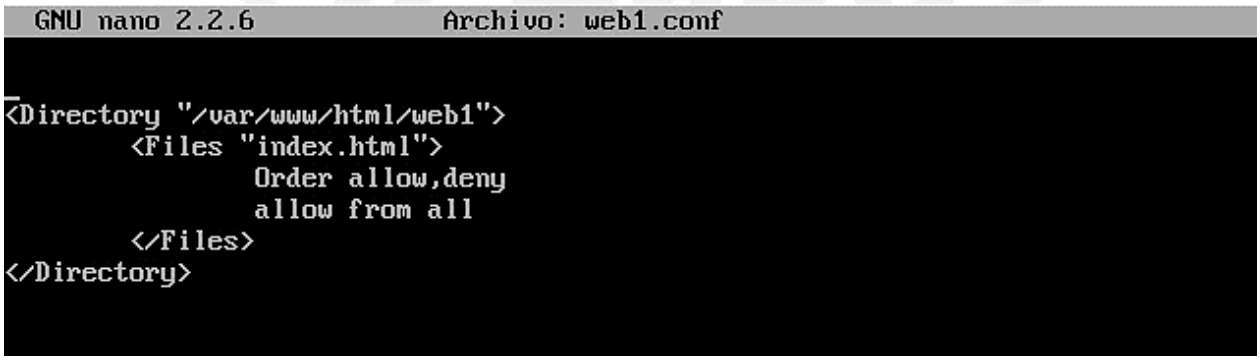


Figura 6. Fichero de configuración personalizado

Una vez hecho esto, podemos incluir en `Apache2.conf` el fichero recién creado. Lo haremos al final del todo para que la configuración escrita en el sobrescriba a alguna posible configuración sobre los mismos elementos que haya en el fichero de configuración centrar de *Apache 2* (aunque debemos adaptar esto a lo que queramos encada instalación):

```
  GNU nano 2.2.6          Archivo: apache2.conf                Modificado

LogFormat "%h %l %u %t \"%r\" %>s %O" common
LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

# Include of directories ignores editors' and dpkg's backup files,
# see README.Debian for details.

# Include generic snippets of statements
IncludeOptional conf-enabled/*.conf

# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

Include web1.conf
```

Figura 7. Inclusión de un fichero de configuración externo

Hecho esto, solo tenemos que **reiniciar *Apache 2*** para que la nueva configuración tenga efecto. Aunque ahora simplemente hemos hecho un simple cambio sobre ese directorio, esto puede extrapolarse a cualquier otro uso que necesitemos en el futuro. Unos ficheros de configuración pequeños y especializados ayudan mucho a las tareas de administración.

Módulos de *Apache 2*

Como hemos dicho, la funcionalidad de *Apache 2* **está gobernada por los módulos que se instalen en el mismo**. Existe una enorme cantidad de módulos, adaptados a diferentes funcionalidades. El listado y la descripción mostrados en esta sección se han sacado directamente de la versión más actual de la documentación de *Apache 2* (*http://httpd.apache.org/docs/current/mod/*), así como las directivas que admite cada módulo.

Aunque por el carácter de este libro y la enorme extensión y complejidad de los módulos no es posible verlos todos, si cubriremos buena parte de ellos en el resto de los capítulos. **Se han elegido los módulos más relevantes o necesarios** para realizar operaciones comunes, pero el conocimiento de la existencia de módulos concretos puede servirnos en caso de necesitar alguna funcionalidad especial o particular en el futuro.

Nombre del mod	Descripción
MPMs (Veremos que son más adelante)	
`mpm_common`	*A collection of directives that are implemented by more than one multi-processing module (*MPM*)*
`event`	*A variant of the worker* MPM *with the goal of consuming threads only for connections with active processing*

`mpm_netware`	*Multi-Processing Module implementing an exclusively threaded web server optimized for Novell NetWare*
`mpmt_os2`	*Hybrid multi-process, multi-threaded MPM for OS/2*
`prefork`	*Implements a non-threaded, pre-forking web server*
`mpm_winnt`	*Multi-Processing Module optimized for Windows NT.*
`worker`	*Multi-Processing Module implementing a hybrid multi-threaded multiprocess web server*
Módulos más conocidos de *Apache 2*	
`mod_access_compat`	*Group authorizations based on host (name or IP address)*
`mod_actions`	*Execute CGI scripts based on media type or request method.*
`mod_alias`	*Provides for mapping different parts of the host file system in the document tree and for URL redirection*
`mod_allowmethods`	*Easily restrict what HTTP methods can be used on the server*
`mod_asis`	*Sends files that contain their own HTTP headers*
`mod_auth_basic`	*Basic HTTP authentication*
`mod_auth_digest`	*User authentication using MD5 Digest Authentication*
`mod_auth_form`	*Form authentication*
`mod_authn_anon`	*Allows "anonymous" user access to authenticated areas*
`mod_authn_core`	*Core Authentication*
`mod_authn_dbd`	*User authentication using an SQL database*
`mod_authn_dbm`	*User authentication using DBM files*
`mod_authn_file`	*User authentication using text files*
`mod_authn_socache`	*Manages a cache of authentication credentials to relieve the load on backends*
`mod_authnz_fcgi`	*Allows a FastCGI authorizer application to handle Apache 2 httpd authentication and authorization*
`mod_authnz_ldap`	*Allows an LDAP directory to be used to store the database for HTTP Basic authentication.*
`mod_authz_core`	*Core Authorization*
`mod_authz_dbd`	*Group Authorization and Login using SQL*
`mod_authz_dbm`	*Group authorization using DBM files*
`mod_authz_groupfile`	*Group authorization using plaintext files*
`mod_authz_host`	*Group authorizations based on host (name or IP address)*
`mod_authz_owner`	*Authorization based on file ownership*
`mod_authz_user`	*User Authorization*
`mod_autoindex`	*Generates directory indexes, automatically, similar to the Unix ls command or the Win32 `dir` shell command*
`mod_buffer`	*Support for request buffering*
`mod_cache`	*RFC 2616 compliant HTTP caching filter.*

`mod_cache_disk`	*Disk based storage module for the HTTP caching filter.*
`mod_cache_socache`	*Shared object cache (socache) based storage module for the HTTP caching filter.*
`mod_cern_meta`	*CERN httpd metafile semantics*
`mod_cgi`	*Execution of CGI scripts*
`mod_cgid`	*Execution of CGI scripts using an external CGI daemon*
`mod_charset_lite`	*Specify character set translation or recoding*
`mod_data`	*Convert response body into an RFC2397 data URL*
`mod_dav`	*Distributed Authoring and Versioning (WebDAV) functionality*
`mod_dav_fs`	*File system provider for mod_dav*
`mod_dav_lock`	*Generic locking module for mod_dav*
`mod_dbd`	*Manages SQL database connections*
`mod_deflate`	*Compress content before it is delivered to the client*
`mod_dialup`	*Send static content at a bandwidth rate limit, defined by the various old modem standards*
`mod_dir`	*Provides for "trailing slash" redirects and serving directory index files*
`mod_dumpio`	*Dumps all I/O to error log as desired.*
`mod_echo`	*A simple echo server to illustrate protocol modules*
`mod_env`	*Modifies the environment which is passed to CGI scripts and SSI pages*
`mod_example_hooks`	*Illustrates the Apache 2 module API*
`mod_expires`	*Generation of Expires and Cache-Control HTTP headers according to user-specified criteria*
`mod_ext_filter`	*Pass the response body through an external program before delivery to the client*
`mod_file_cache`	*Caches a static list of files in memory*
`mod_filter`	*Context-sensitive smart filter configuration module*
`mod_headers`	*Customization of HTTP request and response headers*
`mod_heartbeat`	*Sends messages with server status to frontend proxy*
`mod_heartmonitor`	*Centralized monitor for mod_heartbeat origin servers*
`mod_ident`	*RFC 1413 ident lookups*
`mod_imagemap`	*Server-side imagemap processing*
`mod_include`	*Server-parsed html documents (Server Side Includes)*
`mod_info`	*Provides a comprehensive overview of the server configuration*
`mod_isapi`	*ISAPI Extensions within Apache 2 for Windows*
`mod_lbmethod_bybusyness`	*Pending Request Counting load balancer scheduler algorithm for mod_proxy_balancer*
`mod_lbmethod_byrequests`	*Request Counting load balancer scheduler algorithm for mod_proxy_balancer*

`mod_lbmethod_bytraffic`	*Weighted Traffic Counting load balancer scheduler algorithm for mod_proxy_balancer*
`mod_lbmethod_heartbeat`	*Heartbeat Traffic Counting load balancer scheduler algorithm for mod_proxy_balancer*
`mod_ldap`	*LDAP connection pooling and result caching services for use by other LDAP modules*
`mod_log_config`	*Logging of the requests made to the server*
`mod_log_debug`	*Additional configurable debug logging*
`mod_log_forensic`	*Forensic Logging of the requests made to the server*
`mod_logio`	*Logging of input and output bytes per request*
`mod_lua`	*Provides Lua hooks into various portions of the httpd request processing*
`mod_macro`	*Provides macros within Apache 2 httpd runtime configuration files*
`mod_mime`	*Associates the requested filename's extensions with the file's behavior (handlers and filters) and content (mime-type, language, character set and encoding)*
`mod_mime_magic`	*Determines the MIME type of a file by looking at a few bytes of its contents*
`mod_negotiation`	*Provides for content negotiation*
`mod_nw_ssl`	*Enable SSL encryption for NetWare*
`mod_privileges`	*Support for Solaris privileges and for running virtual hosts under different user IDs.*
`mod_proxy`	*Multi-protocol proxy/gateway server*
`mod_proxy_ajp`	*AJP support module for mod_proxy*
`mod_proxy_balancer`	*mod_proxy extension for load balancing*
`mod_proxy_connect`	*mod_proxy extension for CONNECT request handling*
`mod_proxy_express`	*Dynamic mass reverse proxy extension for mod_proxy*
`mod_proxy_fcgi`	*FastCGI support module for mod_proxy*
`mod_proxy_fdpass`	*fdpass external process support module for mod_proxy*
`mod_proxy_ftp`	*FTP support module for mod_proxy*
`mod_proxy_html`	*Rewrite HTML links in to ensure they are addressable from Clients' networks in a proxy context.*
`mod_proxy_http`	*HTTP support module for mod_proxy*
`mod_proxy_scgi`	*SCGI gateway module for mod_proxy*
`mod_proxy_wstunnel`	*Websockets support module for mod_proxy*
`mod_ratelimit`	*Bandwidth Rate Limiting for Clients*
`mod_reflector`	*Reflect a request body as a response via the output filter stack.*
`mod_remoteip`	*Replaces the original client IP address for the connection with the useragent IP address list presented by a proxies or a load balancer via the request headers.*

`mod_reqtimeout`	*Set timeout and minimum data rate for receiving requests*
`mod_request`	*Filters to handle and make available HTTP request bodies*
`mod_rewrite`	*Provides a rule-based rewriting engine to rewrite requested URLs on the fly*
`mod_sed`	*Filter Input (request) and Output (response) content using sed syntax*
`mod_session`	*Session support*
`mod_session_cookie`	*Cookie based session support*
`mod_session_crypto`	*Session encryption support*
`mod_session_dbd`	*DBD/SQL based session support*
`mod_setenvif`	*Allows the setting of environment variables based on characteristics of the request*
`mod_slotmem_plain`	*Slot-based shared memory provider.*
`mod_slotmem_shm`	*Slot-based shared memory provider.*
`mod_so`	*Loading of executable code and modules into the server at start-up or restart time*
`mod_socache_dbm`	*DBM based shared object cache provider.*
`mod_socache_dc`	*Distcache based shared object cache provider.*
`mod_socache_memcache`	*Memcache based shared object cache provider.*
`mod_socache_shmcb`	*shmcb based shared object cache provider.*
`mod_speling`	*Attempts to correct mistaken URLs by ignoring capitalization or attempting to correct various minor misspellings.*
`mod_ssl`	*Strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols*
`mod_status`	*Provides information on server activity and performance*
`mod_substitute`	*Perform search and replace operations on response bodies*
`mod_suexec`	*Allows CGI scripts to run as a specified user and Group*
`mod_unique_id`	*Provides an environment variable with a unique identifier for each request*
`mod_unixd`	*Basic (required) security for Unix-family platforms.*
`mod_userdir`	*User-specific directories*
`mod_usertrack`	*Clickstream logging of user activity on a site*
`mod_version`	*Version dependent configuration*
`mod_vhost_alias`	*Provides for dynamically configured mass virtual hosting*
`mod_watchdog`	*Provides infrastructure for other modules to periodically run tasks*
`mod_xml2enc`	*Enhanced charset/internationalisation support for libxml2-based filter modules*

HACK FIRST
HACK HARD
NO MERCY
大和
Cobra Kali

Extracción de información del servidor

Sólo por hacer una instalación inicial básica aparecen los primeros problemas de seguridad: **El servidor está dando demasiada información** sobre sí mismo, información que no es útil para un usuario "estándar" de una web, pero sí muy jugosa para un posible atacante. Como vemos en la web, la página por defecto mostrada nos dice el tipo de servidor y de SO que se está ejecutando. La información es mayor si efectuamos una conexión simple con la famosa herramienta `nmap` desde nuestro otro *Linux* cliente. El uso de la opción `-A` nos revela la versión exacta del servidor que se encuentra instalada.

```
operario@servidorubuntu:~$ nmap 192.168.23.1

Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-15 11:01 CEST
Nmap scan report for 192.168.23.1
Host is up (0.00058s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
80/tcp open  http

Nmap done: 1 IP address (1 host up) scanned in 5.45 seconds
operario@servidorubuntu:~$ nmap -A 192.168.23.1

Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-15 11:02 CEST
Nmap scan report for 192.168.23.1
Host is up (0.00062s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE VERSION
80/tcp open  http    Apache httpd 2.4.7 ((Ubuntu))
|_http-title: Apache2 Ubuntu Default Page: It works

Service detection performed. Please report any incorrect results at http://nmap.
org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 11.46 seconds
operario@servidorubuntu:~$ _
```

Figura 8. Uso de nmap para averiguar información sobre el servidor

Un cambio en las opciones de `nmap` nos permite también intentar averiguar el tipo y versión del SO instalado. Para ello, se usa la opción de `nmap -O`:

```
operario@servidorubuntu:~$ sudo nmap -O 192.168.23.1
[sudo] password for operario:

Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-15 11:03 CEST
Nmap scan report for 192.168.23.1
Host is up (0.00035s latency).
Not shown: 999 filtered ports
PORT   STATE SERVICE
80/tcp open  http
MAC Address: 08:00:27:06:1E:57 (Cadmus Computer Systems)
Warning: OSScan results may be unreliable because we could not find at least 1 o
pen and 1 closed port
Aggressive OS guesses: Linux 2.6.26 - 2.6.35 (94%), Linux 2.6.32 (93%), Linux 2.
6.32 - 3.9 (93%), Linux 3.0 - 3.9 (93%), Android 4.1.1 (92%), Linux 2.6.32 - 3.6
 (92%), Linux 2.6.22 (90%), Linux 2.6.23 - 2.6.38 (90%), Linux 2.6.32 - 2.6.39 (
90%), Linux 2.6.37 (90%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

OS detection performed. Please report any incorrect results at http://nmap.org/s
ubmit/ .
Nmap done: 1 IP address (1 host up) scanned in 24.72 seconds
operario@servidorubuntu:~$
```

Figura 9. Averiguando la versión del SO con nmap

El resultado es que, aunque parece que la herramienta tiene claro que estamos usando *Linux*, afortunadamente para nosotros no es capaz de averiguar con precisión la versión del *kernel* del mismo que realmente tenemos instalada, algo que podemos averiguar con `uname -a`.

```
operario@servidorapache:~$ uname -a
Linux servidorapache 3.16.0-40-generic #54~14.04.1-Ubuntu SMP Wed Jun 10 17:30:4
5 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
operario@servidorapache:~$ _
```

Figura 10. Averiguando la versión real del SO instalada

¿Por qué es tan importante no revelar demasiada información sobre el servidor? El motivo es que en Internet existen multitud de bases de datos que listan vulnerabilidades de cualquier aplicación:

- http://secunia.com/community/advisories/historic/
- http://osvdb.org/search/advsearch
- http://www.securityartwork.es/2015/07/21/vfeed-una-bd-de-vulnerabilidades/
- https://www.cvedetails.com/

Por ese motivo, si un atacante conoce el tipo de servidor y, sobre todo, la versión concreta del mismo que estamos usando, podrá usar dicha información para buscar los problemas conocidos del software instalado, y con ello abrimos una vía para que se le haga daño al sistema. ESTO ES APLICABLE A CUALQUIER SOFTWARE DE LA MÁQUINA, no solo a Apache 2.

Normalmente, **cuanto más puertos abiertos y servicios ofrecidos mayor peligro** tenemos de que con herramientas de esta clase un atacante pueda averiguar más datos sobre el servidor web. *¿Qué ocurriría si además del puerto 80 abriésemos el puerto 22/SSH?* Aunque el servicio no esté instalado (y por tanto el puerto no tenga un programa que escuche en él, encontrándose por tanto cerrado), `nmap` podría usarlo para averiguar más datos. Si repetimos nuestra prueba anterior:

```
operario@servidorubuntu:~$ sudo nmap -A -O -Pn 192.168.23.1

Starting Nmap 6.40 ( http://nmap.org ) at 2015-06-15 11:19 CEST
Nmap scan report for 192.168.23.1
Host is up (0.00038s latency).
Not shown: 998 filtered ports
PORT   STATE  SERVICE VERSION
22/tcp closed ssh
80/tcp open   http    Apache httpd 2.4.7 ((Ubuntu))
|_http-title: Apache2 Ubuntu Default Page: It works
MAC Address: 08:00:27:06:1E:57 (Cadmus Computer Systems)
Aggressive OS guesses: Linux 2.6.26 - 2.6.35 (93%), Linux 3.0 - 3.9 (93%), Linux
 2.6.23 - 2.6.38 (91%), Linux 2.6.32 - 2.6.39 (91%), Linux 2.6.37 (91%), Android
 4.1.1 (90%), Linux 2.6.32 (90%), Linux 2.6.22 (89%), Linux 2.6.32 - 3.9 (89%),
Linux 2.6.35 (89%)
No exact OS matches for host (test conditions non-ideal).
Network Distance: 1 hop

TRACEROUTE
HOP RTT     ADDRESS
1   0.38 ms 192.168.23.1

OS and Service detection performed. Please report any incorrect results at http:
//nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 24.32 seconds
operario@servidorubuntu:~$ _
```

Figura 11. Uso de nmap con más puertos abiertos

Vemos que no se obtiene más información que la que ya teníamos en esta ocasión, aunque debemos tener en cuenta que lo que acabamos de hacer es una mala práctica (abrir un puerto en el *firewall* que no corresponde a ningún servicio que realmente se esté ofreciendo).

Anonimización de un servidor *Apache 2*

En la actividad anterior hemos visto que el servidor, por el mero hecho de funcionar con su configuración por defecto, está proporcionando **demasiada información a posibles atacantes**. Para ello, hemos usado la herramienta `nmap` y hemos visto que mediante la simple navegación ya podemos tener información inadecuada en este contexto.

Pero este problema no queda aquí, puesto que existen otras técnicas y herramientas para averiguar más datos. Por ejemplo, el comportamiento por defecto del servidor *Apache 2* cuando navegamos a una página inexistente es también inadecuado desde el punto de vista de la seguridad:

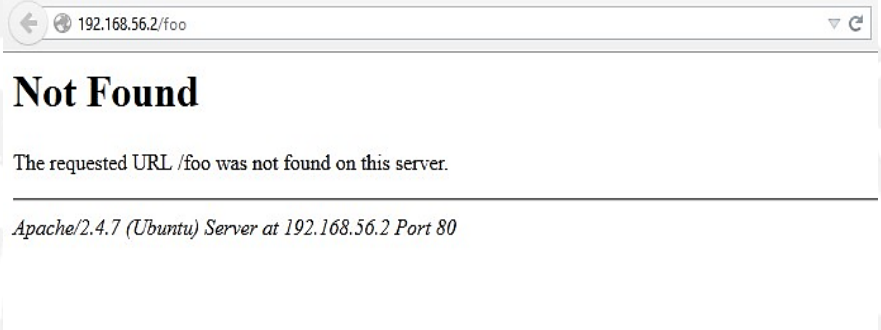


Figura 12. Página de error por defecto cuando se navega a una web inexistente

Contamos también con herramientas adicionales que nos pueden proporcionar información. Una de ellas es `whatweb` (https://github.com/urbanadventurer/WhatWeb)

```
operario@servidorubuntu:~$ sudo apt-get install whatweb
```

Figura 13. Instalación de whatweb

```
operario@servidorubuntu:~$ whatweb -v 192.168.23.1
192.168.23.1 []
URL     : 192.168.23.1
Status  :
operario@servidorubuntu:~$
```

Figura 14. Ejecución de whatweb

Sin embargo esta herramienta **está más orientada a averiguar la tecnología con la que se ha hecho una web** que a averiguar el tipo de servidor en sí. Por ello, nos será útil más adelante. Como alternativa, podemos instalar `nikto` (https://cirt.net/Nikto2):

```
operario@servidorubuntu:~$ sudo apt-get install nikto
```

Figura 15. Instalación de nikto

Siempre actualizándola antes de usarse:

```
operario@servidorubuntu:~$ sudo nikto -update
```

Figura 16. Actualización de nikto

Aunque tiene un buen nº de opciones (https://cirt.net/nikto2-docs/usage.html), simplemente con la opción `-h` ya podemos encontrar información interesante:

- Tipo y versión exacta del servidor
- Tipo del SO
- Errores de configuración: Aunque al ser una instalación por defecto no hay elementos muy reseñables (ficheros en rutas por defecto), **sí que conviene pasar esta herramienta periódicamente en busca de posibles problemas que aparezcan en el servidor con el tiempo**
- Métodos HTTP admitidos, de los que hablaremos posteriormente.

```
operario@servidorubuntu:~$ nikto -h http://192.168.23.1
- Nikto v2.1.4
---------------------------------------------------------------------------
+ Target IP:          192.168.23.1
+ Target Hostname:    192.168.23.1
+ Target Port:        80
+ Start Time:         2015-06-17 07:52:52
---------------------------------------------------------------------------
+ Server: Apache/2.4.7 (Ubuntu)
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ ETag header found on server, fields: 0x2cf6 0x5188a82e43961
+ Allowed HTTP Methods: GET, HEAD, POST, OPTIONS
```

Figura 17. Ejecución de nikto

¿Cómo arreglar este problema?

A la hora de hacer que el servidor no de información sobre sí mismo a los clientes, debemos actuar sobre las siguientes directivas de `Apache2.conf`, añadiéndolas si no existiesen en el mismo. Las directivas son `ServerTokens` y `ServerSignature`, cada una de las cuales tiene el siguiente efecto:

- `ServerTokens`: Directiva **cuyo valor recomendado es** `Prod`, al ser la opción que menos información da, y se aplica a todas las webs del servidor. Identifica **la información que *Apache 2* envía sobre sí mismo a los clientes en las peticiones**. Los posibles valores y la información que muestran son:
  - `ServerTokens Prod[uctOnly]`: `"Server: Apache 2"`
  - `ServerTokens Major`: `"Server: Apache 2/2"`
  - `ServerTokens Minor`: `"Server: Apache 2/2.0"`
  - `ServerTokens Min[imal]`: `"Server: Apache 2/2.2.15"`
  - `ServerTokens OS`: `"Server: Apache 2/2.2.15 (Unix)"`
  - `ServerTokens Full` (o sin especificar): Muestra lo mismo que OS e información de todos los módulos relevantes instalados. Ej.: `"Server:Apache 2/2.2.15 (Unix) PHP/4.2.2 MiMod/1.2"`
- `ServerSignature`: Se recomienda poner en `"Off"`
  - `"On"` añade una línea con el nombre de servidor, versión, el host virtual que envía la página y el email puesto en la configuración de *Apache 2*.

- Esto **se añadiría a los documentos generados por el servidor** (mensajes de error, salidas,…), pero no es conveniente debido a que la mejor política es la de menos información -> mejor.
- No obstante, puede ser útil si el servidor pasa por una cadena de proxys y se quiere saber el origen de cada documento que llega al cliente.

Por tanto para eliminar los datos del servidor en las cabeceras de respuesta de este tenemos que añadir la siguiente configuración en Apache2.conf (o como alternativa en `conf-enabled/security.conf`):

```
ServerSignature Off
ServerTokens Prod
```

```
 GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf          Modificado

LogFormat "%{Referer}i -> %U" referer
LogFormat "%{User-agent}i" agent

# Include of directories ignores editors' and dpkg's backup files,
# see README.Debian for details.

# Include generic snippets of statements
IncludeOptional conf-enabled/*.conf

# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

ServerSignature Off
ServerTokens Prod
```

Figura 18. Modificación de las cabeceras de respuesta para impedir dar demasiada información

Sin embargo, solo por modificar este fichero no lograremos tener esa configuración activa, sino que debemos reiniciar el servicio *Apache 2* para que los ficheros de configuración vuelvan a leerse y de esta forma *Apache 2* se reconfigure con las nuevas directivas que le hemos suministrado. Esto se puede conseguir con `sudo service apache2 restart`:

```
operario@servidorapache:/var/www/html/folder$ sudo service apache2 restart
```

Figura 19. Reinicio del servidor para aplicar una nueva configuración

RECUERDA: Cada cambio de configuración de Apache 2 DEBE CONLLEVAR UN REINICIO DEL SERVICIO (`sudo service apache2 restart`) para que estos cambios tengan efecto. Existe la opción `reload` (`sudo service apache2 reload`), pero el efecto de hacerla no es tan claro como parar y rearrancar el servidor.

Técnicamente `reload` obligaría a Apache 2 a volver a leer su configuración, pero la nueva configuración puede tardar un tiempo en hacerse efectiva o bien determinados cambios pueden no ser tenidos en cuenta, depende de cómo se haya implementado exactamente el mecanismo en la versión de Apache 2 que estemos manejando. En el resto del libro usaremos `restart` para estar seguros de que los cambios entran en funcionamiento de manera inmediata.

Hecho esto, comprobamos como el servidor ya no muestra información sobre sí mismo como antes:

Figura 20. Apache 2 sin la firma que delata su versión y otros datos críticos

Esto puede refinarse aún más habilitando el *mod* `headers` de *Apache 2*, que tiene opciones para controlar aspectos relativos a la información que el servidor envía en las peticiones. Para ello listamos el directorio `mods-enabled` de *Apache 2* y, en caso de que `headers` no esté activo, lo activamos con `sudo a2enmod headers`:

```
operario@servidorapache:/var/www/html/folder$ ls /etc/apache2/mods-enabled/
access_compat.load  authz_host.load  dir.load        negotiation.conf
alias.conf          authz_user.load  env.load        negotiation.load
alias.load          autoindex.conf   filter.load     setenvif.conf
auth_basic.load     autoindex.load   mime.conf       setenvif.load
authn_core.load     deflate.conf     mime.load       status.conf
```

Figura 21. Activación de mod_headers

Una vez activo, podemos incluir la siguiente configuración que desactiva las cabeceras `Server` y `X-Powered-By`, ambas usadas para dar información de *Apache 2*, productos instalados en el mismo, etc. Las cabeceras a desactivar dependerán de los productos que tengamos instalados en el servidor web.

```
  GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf          Modificado

# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet

ServerSignature Off
ServerTokens Prod

<IfModule mod_headers.c>
        Header unset Server
        Header always unset Server
        Header unset X-Powered-By
        Header always unset X-Powered-By
</IfModule>
```

Figura 22. Desactivación de cabeceras de las peticiones

Otro aspecto a tener en cuenta es **la página por defecto que muestra una instalación de *Apache 2***. Dada su nula utilidad y la información que muestra, debemos deshacernos de ella. Para ello las opciones son bien no mostrar nada o bien mostrar una página inocua que no dé información alguna:

```
<!--
  Modified from the Debian original for Ubuntu
  Last updated: 2014-03-19
  See: https://launchpad.net/bugs/1288690
-->
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <title>Apache2 Ubuntu Default Page: It works</title>
  <style type="text/css" media="screen">
* {
  margin: 0px 0px 0px 0px;
  padding: 0px 0px 0px 0px;
}

body, html {
  padding: 3px 3px 3px 3px;

  background-color: #D8DBE2;

  font-family: Verdana, sans-serif;
  font-size: 11pt;
  text-align: center;


operario@servidorapache:/var/www/html$ sudo rm index.html
operario@servidorapache:/var/www/html$ sudo nano index.html
```

Figura 23. Borrado de la página por defecto de Apache 2

```
 GNU nano 2.2.6            Archivo: index.html

<html>
<head>
        <title>Welcome</title>
</head>
        <body>
                <p>
                        Empty page
                </p>

                <p>
                        Welcome
                </p>
        </body>
</html>
```

Figura 24. Página inocua a efectos de seguridad

Ahora tenemos una página por defecto que no da absolutamente ninguna información acerca del servidor o SO, sólo que dicho servidor función:

Figura 25. La página por defecto ya no muestra ninguna información

Como hemos dicho, una alternativa a los cambios vistos es hacerlos en `conf-enabled/security.conf`. Este fichero incluye (aunque inicialmente desactivadas) las directivas vistas y otras que están relacionadas con la seguridad. **Se recomienda activar todas las directivas de este fichero** para lograr una configuración óptima desde el punto de vista de seguridad. Al hacer esto ya no es necesario cambiar la página inicial, puesto que toda petición de cualquier página es denegada, salvo que haya sido expresamente autorizada (veremos más sobre este aspecto posteriormente). El fichero `security.conf` debería quedar como el siguiente:

```
# Disable access to the entire file system except for the directories that
# are explicitly allowed later.
#
# This currently breaks the configurations that come with some web application
# Debian packages.
#
<Directory />
   AllowOverride None
   Order Deny,Allow
   Deny from all
</Directory>

# Changing the following options will not really affect the security of the # server, but
# might make attacks slightly more difficult in some cases.

#
# ServerTokens
# This directive configures what you return as the Server HTTP response
# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of:  Full | OS | Minimal | Minor | Major | Prod # where Full conveys the most
# information and Prod the least.
#ServerTokens Minimal
ServerTokens Prod
#ServerTokens Full

#
# Optionally add a line containing the server version and virtual host
# name to server-generated pages (internal error documents, FTP directory # listings,
# mod_status and mod_info output etc., but not CGI generated
# documents or custom error documents).
# Set to "EMail" to also include a mailto: link to the ServerAdmin.
# Set to one of:  On | Off | EMail
ServerSignature Off
#ServerSignature On

#
# Allow TRACE method
#
```

```
# Set to "extended" to also reflect the request body (only for testing and
# diagnostic purposes).
#
# Set to one of:  On | Off | extended
TraceEnable Off
#TraceEnable On

#
# Forbid access to version control directories
#
# If you use version control systems in your document root, you should # probably deny
# access to their directories. For example, for subversion:
#
<DirectoryMatch "/\.svn">
   Require all denied </DirectoryMatch>

#
# Setting this header will prevent MSIE from interpreting files as something
# else than declared by the content type in the HTTP headers.
# Requires mod_headers to be enabled.
#
Header set X-Content-Type-Options: "nosniff"

#
# Setting this header will prevent other sites from embedding pages from this
# site as frames. This defends against clickjacking attacks.
# Requires mod_headers to be enabled.
#
Header set X-Frame-Options: "sameorigin"
```

Es importante ver cómo este fichero incluye medidas contra *clickjacking* (https://www.owasp.org/index.php/Clickjacking) en el propio servidor y contra ataques derivados de interpretar los contenidos de los ficheros de una forma diferente al que se declara en las cabeceras HTTP, que también son aspectos importantes.

Al activar la configuración de este fichero `nikto` ya no nos dice que el servidor tenga problemas de seguridad conocidos, pero tampoco será capaz de servir ninguna página (por el momento):

```
operario@servidorubuntu:~$ nikto -h http://192.168.23.1
- Nikto v2.1.4
---------------------------------------------------------------------------
+ Target IP:          192.168.23.1
+ Target Hostname:    192.168.23.1
+ Target Port:        80
+ Start Time:         2015-06-17 08:31:45
---------------------------------------------------------------------------
```

Figura 26. Segunda ejecución de nikto con el servidor "anonimizado"

Figura 27. Navegación con el servidor "anonimizado"

Comprobación de que no está permitido el listado de directorios

Una de las **peores cosas** que nos puede ocurrir desde el punto de vista de la seguridad es que en alguna de las páginas web **se permita el listado de directorios**. Estas páginas no solo podrían mostrar información del servidor como hemos visto, sino que también podrían dejar al descubierto ficheros cuyo contenido no queremos que sea accesible a los usuarios (aunque en ese caso cabría pensar que el hecho de que existan esa clase de ficheros es en sí un error de seguridad). Independientemente de la pertinencia o no de los ficheros que existan, las extensiones de los ficheros que se listen determinarán las tecnologías usadas y otros productos, así como la estructura general de la web. Por ese motivo, debemos desactivar el listado de directorios a toda costa. El listado de directorios **viene activado por defecto** en una instalación de *Apache 2* sobre *Ubuntu*, como se ve en la siguiente imagen.

Figura 28. Listado de directorios

El listado de directorios está controlado en primera instancia por el módulo `mod_dir` (http://httpd.apache.org/docs/current/mod/mod_dir.html). Lo que vemos es su comportamiento por defecto que, como hemos dicho, no es adecuado desde el punto de vista de seguridad. El comportamiento de Apache 2 en este caso es sencillo: Si el directorio accedido tiene un fichero de los que están listado en la directiva `DirectoryIndex` de su configuración, este se será el fichero que se servirá.

```
  GNU nano 2.2.6    Archivo: /etc/apache2/mods-enabled/dir.conf

<IfModule mod_dir.c>
        DirectoryIndex index.html index.cgi index.pl index.php index.xhtml inde$
</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Figura 29. Ficheros de índice por defecto de mod_dir

En caso de que no lo tenga (como es nuestro caso), se ejecutará `mod_autoindex` (http://httpd.apache.org/docs/current/mod/mod_autoindex.html) y se mostrará un listado de ficheros y directorios acorde a lo que indiquen sus directivas. El resultado es un listado como el que se ve en la primera imagen, de acuerdo con lo establecido para ellos en su configuración por defecto.

```
  GNU nano 2.2.6 Archivo: .../apache2/mods-available/autoindex.conf

<IfModule mod_autoindex.c>
        # Directives controlling the display of server-generated directory list$

        #
        # IndexOptions: Controls the appearance of server-generated directory
        # listings.
        # Remove/replace the "Charset=UTF-8" if you don't use UTF-8 for your fi$
        IndexOptions FancyIndexing VersionSort HTMLTable NameWidth=* Descriptio$

        #
        # AddIcon* directives tell the server which icon to show for different
        # files or filename extensions.  These are only displayed for
        # FancyIndexed directories.
        AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip x-bzip2

        AddIconByType (TXT,/icons/text.gif) text/*
        AddIconByType (IMG,/icons/image2.gif) image/*
        AddIconByType (SND,/icons/sound2.gif) audio/*
        AddIconByType (VID,/icons/movie.gif) video/*

        AddIcon /icons/binary.gif .bin .exe
        AddIcon /icons/binhex.gif .hqx
        AddIcon /icons/tar.gif .tar
        AddIcon /icons/world2.gif .wrl .wrl.gz .vrml .vrm .iv
        AddIcon /icons/compressed.gif .Z .z .tgz .gz .zip
                                [ 96 líneas leídas ]
^G Ver ayuda ^O Guardar   ^R Leer Fich ^Y RePág.    ^K Cortar Tex^C Pos actual
^X Salir     ^J Justificar^W Buscar    ^V Pág. Sig. ^U PegarTxt   ^T Ortografía
```

Figura 30. Configuración de listado de directorios de mod_autoindex

En caso de que ninguno de los módulos esté habilitado, entonces simplemente se mostrará un error *404 Not Found*

¿Cómo arreglo este problema?

Aunque para entender bien qué está ocurriendo con las directivas indicadas en esta sección es necesario profundizar más en cómo se comporta *Apache 2*, podemos hacer determinadas acciones sencillas para solventar el problema sin que ello requiera esos conocimientos. No obstante, posteriormente describiremos todos los mecanismos involucrados en las opciones que vamos a usar, y por tanto podrá entenderse correctamente porqué lo que vamos a hacer funciona.

La forma más obvia de arreglar el problema es **asignar a la directiva `DirectoryIndex` un fichero que sepamos que existe**, de manera que sea este el que se muestre ante un acceso al directorio. Si no nos sirve ninguno de los indicados en `dir.conf` (mostrado en una imagen anterior), podemos incluir la directiva en cualquier directorio indicando un fichero dentro del mismo. De esta manera no se ejecutará `mod_autoindex`. Otra solución es prohibir el listado de directorios mediante `DirectoryIndex disabled`. En la siguiente imagen prohibimos el listado de directorios para todo el servidor:

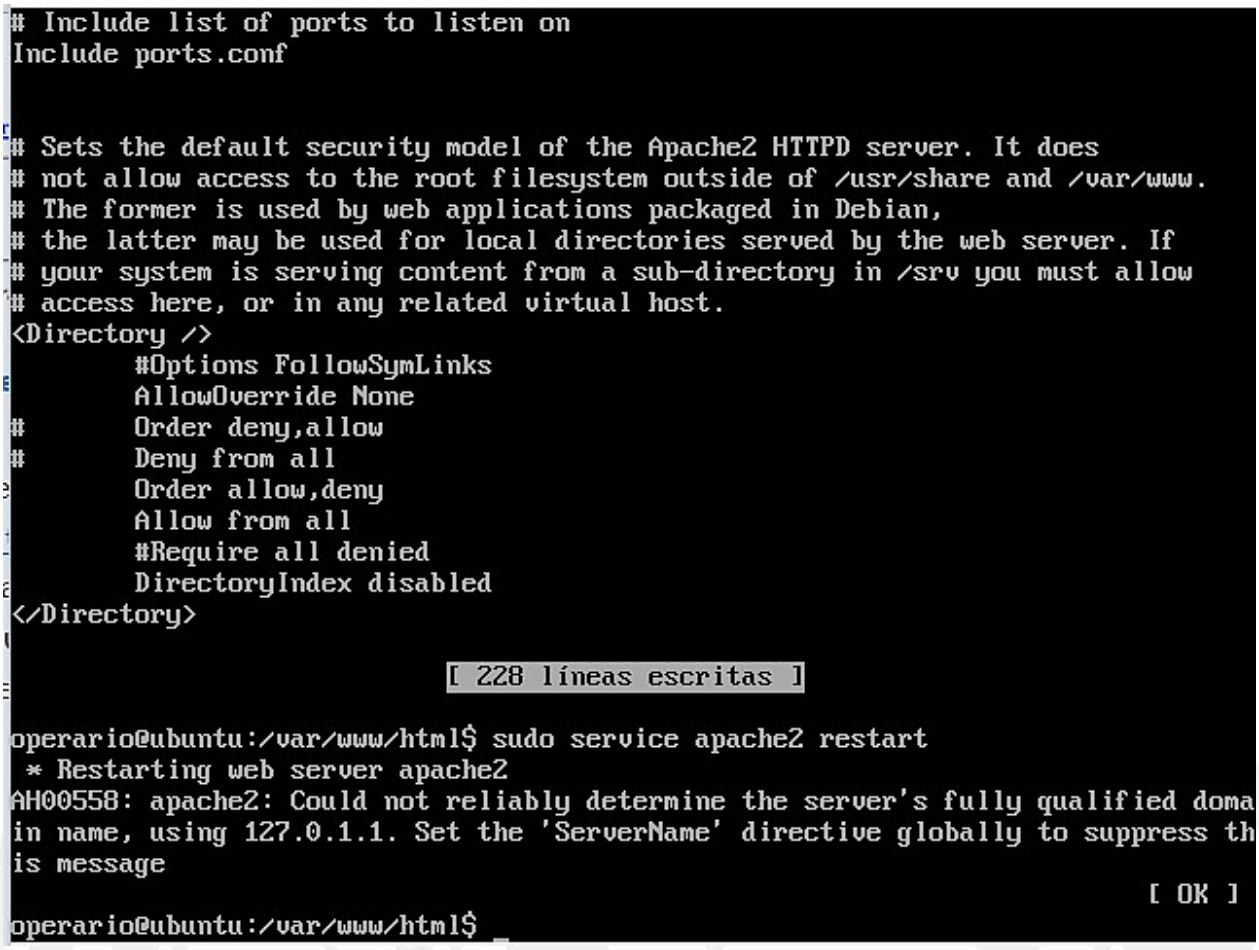


Figura 31. Deshabilitar el listado de directorios

Una última opción más drástica es prohibir la navegación a ficheros que no estén expresamente autorizados, algo que no solo desactivaría el listado de directorios por extensión sino que también evitaría el acceso accidental a ficheros que no deseamos que sean visibles en un gran nº de casos. Para esto debemos seguir los siguientes pasos, que supondrán que las *web* que se sirven están situadas en `/var/www/html` (directorio por defecto). La configuración se aplicará a todas las web que dependan del directorio `/var/www/html` y también la aplicaremos sobre el directorio raíz "/"

- Ir a la ruta `/etc/apache2/sites-enabled/`
- **Editar el archivo** `000-default` añadiendo lo siguiente (**OJO**: `deny,allow` y viceversa no llevan espacio tras la coma).
- **Quitar la directiva** `ServerAdmin` (para eliminar información)
- **Añadir** `Options -Indexes` en la configuración de los directorios mencionados. Esta opción evita que se pueda cambiar la configuración de indexación de ficheros con ficheros `.htaccess`, que veremos posteriormente.
- Luego debemos **reiniciar *Apache 2*** y probar que el acceso es correcto.

```
# This currently breaks the configurations that come with some web application
# Debian packages.
#
<Directory />
   AllowOverride None
   Order Deny,Allow
   Deny from all
   Options -Indexes
</Directory>

        #ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
        <Directory /var/www/html>
                Order allow,deny
                allow from all
                Options -Indexes
        </Directory>
```

Figura 32. Eliminación del listado de directorios

Hecho esto, vemos como el listado de directorios ya no está disponible:

Figura 33. Listado de directorios prohibido

Es conveniente recordar que en `security.conf` tenemos una configuración muy similar a esta para el directorio `/`. Si hemos usado este fichero, es conveniente hacer estas modificaciones allí, recordando siempre que no solo hemos prohibido el listado de directorios, sino que tampoco podemos navegar a ninguna página a la que no hayamos autorizado expresamente su acceso (veremos más sobre esto en la sección de autorización).

Optimización de los módulos de *Apache 2*

Como hemos dicho anteriormente, los módulos o *mods* son la forma que el servidor *Apache 2* tiene de **extender sus posibilidades y adaptarse a las diversas necesidades** que cualquier administrador puede requerir de él. Los *mods* pueden tener cometidos y configuraciones muy diferentes, pero hay un tipo de *mod* que es distinto al resto, y estos son los módulos de multiprocesamiento o MPMs.

Módulos de multiprocesamiento o MPMs

Los MPM son uno de los mayores cambios de las versiones 2.X respecto a las anteriores. Básicamente son **formas de procesar las peticiones** que *Apache 2* puede usar, y normalmente suele elegirse la más ventajosa teniendo en cuenta el sistema operativo donde *Apache 2* está funcionando u otros factores de las aplicaciones web que van a funcionar sobre este servidor. A continuación describiremos brevemente los MPM disponibles y sus características.

Las versiones 1.X de *Apache 2* usaban una arquitectura ***preforking*** para atender las peticiones. Ésta consistía en que un proceso padre hacia *fork* para generar un conjunto de procesos hijos, que eran los que finalmente servían las peticiones. El proceso padre monitorizaba a sus hijos y creaba más, o los destruía, basándose en la cantidad de peticiones que recibía, regulando así la carga de trabajo en función del número de peticiones recibido en un momento dado. No obstante, este modelo tenía el inconveniente de que **no funcionaba adecuadamente en algunas plataformas** (por su bajo rendimiento). Una de ellas era *Windows*, al no tener implementado nativamente este modelo de procesos. Por ese motivo, se creó el concepto de MPM, que está destinado a solventar este problema.

Cada MPM es responsable de **arrancar procesos del servidor y de servir peticiones** usando procesos hijos o *threads*, según su implementación. A continuación se citan algunos de los MPM más importantes disponibles. Para un listado completo de todos los existentes y como cambiar de MPM se recomienda consultar: http://httpd.apache.org/docs/current/en/mpm.html

El MPM *prefork*

Copia el modelo de *Apache 2* 1.X, **creando un *pool* de procesos hijos para servir peticiones**. Cada proceso hijo tiene un sólo *thread*, por lo que si *Apache 2* inicia 20 procesos hijos, puede servir 20 peticiones simultáneamente. En el caso de que ocurra un error grave y un proceso hijo muera, **sólo se pierde una petición**: la que está atendiendo el hijo que acaba de cesar la actividad.

Finalmente, cabe decir que el máximo y el mínimo de procesos hijos son parámetros configurables, permitiendo pues adaptar *Apache 2* a las capacidades de la máquina servidora y también fijarle un máximo de carga. Dentro de estos parámetros máximo y mínimo el servidor genera más o menos procesos en función de la carga de trabajo actual.

Más información: http://httpd.apache.org/docs/current/mod/prefork.html

El MPM *worker*

Permite a *Apache 2* **soportar hilos**, lo cual es más eficiente en determinados SO. El modelo en sí es idéntico al anterior, aunque cada proceso hijo tiene un número predefinido de *threads* y cada *thread* puede servir una petición diferente. De este modo, si tenemos 30 procesos hijos x 10 *threads* / proceso el servidor

podrá servir teóricamente hasta 300 peticiones simultaneas. Es por tanto un modelo muy escalable. En lo referente a procesos, cabe decir que éstos mueren o se crean en función del número de *threads* activos o libres que haya en un momento dado.

Más información: http://httpd.apache.org/docs/current/mod/worker.html

El MPM *winnt*

Este MPM es **especial para la plataforma *Windows*** (2008,2012,...). Es un módulo multihilo donde *Apache 2* creará un proceso padre (control) y uno hijo, que es el encargado de crear todos los hilos que sirven las peticiones. Además, este modelo usa algunas funciones nativas de *Windows*, aumentando el rendimiento de *Apache 2* sobre esta plataforma.

Más información: http://httpd.apache.org/docs/current/mod/mpm_winnt.html

El MPM *event*

Este MPM es una variante del MPM *worker*, y ha sido diseñado para servir más peticiones simultáneas. Su funcionamiento se basa en pasar una porción determinada del procesamiento de una petición a hilos de soporte secundarios que se encargarán de tratarla. Esto permite liberar los *threads* principales para que acepten nuevas peticiones, porque delegan el volumen principal del procesamiento en entidades de proceso auxiliares. La configuración de este MPM es idéntica a la del MPM en el que se basa.

Más información: http://httpd.apache.org/docs/current/mod/event.html

Instalación de MPMs

Al instalar *Apache 2* se selecciona de manera automática el que teóricamente es el mejor MPM para el SO destino, y por ello es raro que se necesite cambiarlo, aunque es posible hacerlo en caso necesario. En nuestro caso **no consideramos que sea oportuno cambiarlo**. La selección de MPM por defecto en función de la plataforma aparece en la siguiente tabla, aunque futuras versiones de Apache 2 podrían incorporar nuevos MPMs no descritos aquí:

SO	MPM seleccionado por defecto
Netware	`mpm_netware`
Unix	`prefork`, `worker` o `event` (depende de las características de la plataforma, como el tipo de procesador instalado)
Windows	`mpm_winnt`

En nuestra instalación de trabajo vemos que se ha seleccionado el MPM `event`:

```
operario@servidorapache:~$ cd /etc/apache2/mods-enabled/
operario@servidorapache:/etc/apache2/mods-enabled$ ls
access_compat.load  authz_host.load  dir.load        mpm_event.load
alias.conf          authz_user.load  env.load        negotiation.conf
```

Figura 34. MPM seleccionado por la instalación de Apache 2

Selección de los módulos en la instalación de *Apache 2*

Tras la instalación de *Apache 2*, la siguiente tarea es **seleccionar qué módulos debemos habilitar inicialmente**, en función de lo que deseemos que *Apache 2* haga. Esto debe hacerse antes de instalar módulos con funciones más avanzadas, para contar con el servidor más adecuado de partida. Para llevar a cabo esta tarea debemos tener en cuenta los siguientes criterios:

- Debemos hacer que **el nº de módulos habilitados sea el mínimo imprescindible**. Menos módulos significa más rendimiento y menos problemas de seguridad potenciales.
- Una instalación *Ubuntu* básica de *Apache 2*.4 instala y habilita una serie de módulos por defecto que describiremos a continuación. Estos módulos, como ya hemos dicho, se dividen en módulos compilados con el núcleo de *Apache 2* y DSO. Podemos encontrar una descripción de todos ellos en: http://httpd.apache.org/docs/current/mod/.

Módulos compilados con el núcleo de *Apache 2*

El listado de módulos compilados con *Apache 2* se puede obtener mediante `Apache 2 -l`. Por lo general cumplen funciones básicas e imprescindibles para el servidor, por lo que su inhabilitación no se lleva a cabo salvo que por alguna razón especial deba hacerse, lo que requeriría recompilar el binario del servidor http://httpd.apache.org/docs/2.4/install.html.

```
operario@servidorapache:/etc/apache2/mods-enabled$ apache2 -l
Compiled in modules:
  core.c
  mod_so.c
  mod_watchdog.c
  http_core.c
  mod_log_config.c
  mod_logio.c
  mod_version.c
  mod_unixd.c
operario@servidorapache:/etc/apache2/mods-enabled$
```

Figura 35. Módulos compilados con Apache 2

Los módulos compilados con *Apache 2* son habitualmente:

- `core`: Las **características centrales** de *Apache 2*, necesario en cualquier instalación. http://httpd.apache.org/docs/current/mod/core.html
- `mod_so`: Permite a *Apache 2* **cargar módulos dinámicamente en tiempo de ejecución** (*DSO*), en lugar de tener que recompilar el ejecutable cada vez que queramos añadir o quitar algún módulo. http://httpd.apache.org/docs/current/mod/mod_so.html
- `mod_watchdog`: permite a otros módulos **ejecutar tareas periódicas de forma programática**. Gracias a este módulo otros módulos pueden registrar manejadores que se ejecuten periódicamente para hacer alguna tarea. Es imprescindible para el funcionamiento en clúster de *Apache 2*. http://httpd.apache.org/docs/current/mod/mod_watchdog.html
- `http_core`: **Soporte *HTTP***, necesario en cualquier instalación. Forma parte de la funcionalidad básica de *Apache 2*.
- `mod_logconfig`: Permite **registrar las peticiones de los clientes** (*log*). Es un módulo muy flexible y configurable, pudiendo personalizar tanto el formato como el destino de la información registrada. http://httpd.apache.org/docs/current/mod/mod_log_config.html

- `mod_logio`: Permite **registrar el nº de bytes que tienen las peticiones** entrantes y salientes. Requiere el anterior para poder funcionar. http://httpd.apache.org/docs/current/mod/mod_logio.html
- `mod_version`: Este módulo está pensado para ser **usado en grandes redes y entornos de test** donde es posible que existan **varias versiones de *Apache 2*** con diferentes configuraciones. Gracias a él podemos definir un **nuevo contenedor de directivas**, `<IfVersion>`, que permite el control de directivas en función de la versión detectada. Se admiten expresiones regulares y aritméticas a la hora de determinar la versión a la que afectan esas directivas. http://httpd.apache.org/docs/current/mod/mod_version.html
- `mod_unixd`: Define y controla elementos de seguridad básicos y obligatorios para plataformas Unix/Linux: https://httpd.apache.org/docs/current/es/mod/mod_unixd.html

Módulos instalados y habilitados en una instalación por defecto

Estos son módulos tipo DSO, cargados gracias a los ficheros `.load` del directorio `mods-enabled` de *Apache 2*. En esta sección simplemente vamos a listarlos y ver una descripción breve de su utilidad; muchos de estos módulos serán descritos más en detalle cuando lleguemos a la parte de la gestión del servidor con la que están relacionados. No obstante, se muestra el enlace a la versión más moderna de su documentación completa a modo de referencia:

```
/etc/apache2/mods-enabled/authn_file.load
/etc/apache2/mods-enabled/authz_core.load
/etc/apache2/mods-enabled/authz_host.load
/etc/apache2/mods-enabled/authz_user.load
/etc/apache2/mods-enabled/autoindex.load
/etc/apache2/mods-enabled/deflate.load
/etc/apache2/mods-enabled/dir.load
/etc/apache2/mods-enabled/env.load
/etc/apache2/mods-enabled/filter.load
/etc/apache2/mods-enabled/headers.load
/etc/apache2/mods-enabled/mime.load
/etc/apache2/mods-enabled/mpm_event.load
/etc/apache2/mods-enabled/negotiation.load
/etc/apache2/mods-enabled/setenvif.load
```

Figura 36. DSO cargados por defecto en una instalación de Apache 2

- `mod_access_compat`: Proporciona **control de acceso** basado en el nombre del *host*, dirección *IP* u otras características de la petición. Permite usar las directivas "`order`", "`allow`" y "`deny`" dentro de secciones `<Directory>`, `<Files>`, y `<Location>`, además de ficheros. **Ha sido declarado obsoleto y se recomienda el uso de** `mod_authz_host` aunque, como históricamente se ha usado en gran cantidad de instalaciones de *Apache 2*, se mantiene. La instalación de este *mod* permite el uso de esa sintaxis para control de acceso **por temas de compatibilidad** con configuraciones viejas. Debería dejarse habilitado siempre que no usemos únicamente las directivas de `mod_authz_host`.
  - http://httpd.apache.org/docs/current/mod/mod_access_compat.html
- `mod_alias`: Mapea distintas partes del sistema de ficheros al árbol de documentos de la *web* y permite la **redirección de URIs**. La manipulación de URIs que permite es bastante simple, para tareas más complejas se recomienda el uso de `mod_rewrite`.

    - http://httpd.apache.org/docs/current/mod/mod_alias.html
- `mod_authXX`, `modn_authXX` o `mod_authzXX`: Necesarios para **implementar diversos tipos de autenticación, describiremos algunos más en detalle posteriormente**. Una de las cosas que pueden resultar más confusas es el esquema de nombrado de dichos módulos. En general, los nombres deben interpretarse de la siguiente forma:
    - `mod_auth_*`: Módulos que implementan un mecanismo de autenticación HTTP
    - `mod_authn_*`: Módulos que implementan un proveedor de autenticación en el *backend*
    - `mod_authz_*`: Módulos que implementan autorización (mecanismos de acceso)
    - `mod_authnz_*`: Módulos que implementan autenticación y autorización
    - Los módulos de autenticación concretos incluidos son:
        - `auth_basic`: Permite usar la **autenticación HTTP *Basic***.
            - http://httpd.apache.org/docs/current/mod/mod_auth_basic.html
        - `authn_core`: Implementa las **directivas comunes a todos los proveedores de autenticación** que permitan conceder o denegar el acceso a partes de una web.
            - http://httpd.apache.org/docs/current/mod/mod_authn_core.html
        - `authn_file`: Permite a otros módulos de autenticación, como *Basic* o *Digest* autenticar usuarios **leyendo la información de ficheros de texto plano**.
            - http://httpd.apache.org/docs/current/mod/mod_authn_file.html
        - `authz_core`: Implementa las **directivas de autorización comunes** a todos los módulos que implementen capacidades de autorización, de manera que se pueda conceder o denegar el acceso a partes de una web a todos los usuarios autenticados.
            - http://httpd.apache.org/docs/current/mod/mod_authz_core.html
        - `authz_host`: Permite implementar una forma de autenticación que concede o deniega el acceso a partes de una *web* **basándose en el nombre de *host*, *IP* u otras características de la petición** hecha por el cliente (capturadas en variables de entorno). Se puede aplicar en las mismas secciones que `mod_access_compat`.
            - http://httpd.apache.org/docs/current/mod/mod_authz_host.html
        - `authz_user`: Permite conceder o denegar el acceso a una *web* en función de si un **usuario está autenticado o es válido**, algo que depende de la directiva `Require` usada.
            - http://httpd.apache.org/docs/current/mod/mod_authz_user.html
- `mod_autoindex`: Genera **índices de directorios**, de forma similar a comandos del sistema como `ls` o `dir`
    - http://httpd.apache.org/docs/current/mod/mod_autoindex.html
- `mod_deflate`: Es un filtro de salida que permite **comprimir el contenido que el servidor envía al cliente**. Hablaremos de filtros y de este módulo más detalladamente en una sección posterior.
    - http://httpd.apache.org/docs/current/mod/mod_deflate.html
- `mod_dir`: Permite **buscar y servir un archivo por defecto** (cuyo nombre fija este módulo, pero suele ser "`index.html`", "`default.htm`",...) cuando se accede a una directorio en lugar de a un fichero concreto. En caso de no existir, el módulo `mod_autoindex` se encarga de hacer un listado del contenido de este.
    - http://httpd.apache.org/docs/current/mod/mod_dir.html

- `mod_env`: Controla el entorno (**variables de entorno**) que usan programas *CGI* o páginas que usan *Server-Side Includes*.
  - http://httpd.apache.org/docs/current/mod/mod_env.html
- `mod_filter`: Permite la implementación de **filtros de salida para contenido** en función de la clase de contenido que se va a enviar al cliente (`content-type`), de manera que cada tipo de contenido pueda procesarse de forma independiente al resto.
  - http://httpd.apache.org/docs/current/mod/mod_filter.html
- `mod_headers`: Implementa funcionalidades para **controlar y modificar las peticiones y respuestas** HTTP, modificando, reemplazando o quitando cabeceras de esta. http://httpd.apache.org/docs/current/mod/mod_headers.html
- `mod_mime`: Necesario para **fijar el juego de caracteres**, la codificación y lenguaje del contenido y los tipos *MIME* de los documentos admitidos, entre otras cosas.
  - http://httpd.apache.org/docs/current/mod/mod_mime.html
- `mpm_event`: El **MPM que se va a usar**, necesario en cualquier instalación. En función del SO que estemos usando podría aparecer otro MPM en lugar de este. Ya hemos dado detalles de estos anteriormente.
- `mod_negotiation`: Permite **seleccionar el documento de respuesta que mejor encaja** según las características del cliente dentro de un conjunto de estos que esté disponible. Esto puede hacerse mediante mapas de tipos o *Multiviews*.
  - http://httpd.apache.org/docs/current/mod/mod_negotiation.html
- `mod_setenvif`: Permite **fijar el valor de variables de entorno según** el contenido de determinados atributos de la petición.
  - http://httpd.apache.org/docs/current/mod/mod_setenvif.html
- `mod_status`: Proporciona **información acerca de la actividad del servidor y su rendimiento**.
  - http://httpd.apache.org/docs/current/mod/mod_status.html

Finalmente, todos los módulos disponibles para ser activados en *Apache 2* se pueden listar haciendo el mismo listado sobre el directorio `mods-available`. Como ya hemos dicho, la activación/desactivación se puede realizar con `a2enmod` y `a2dismod`, pasando como parámetro el nombre del módulo (sin "`mod_`"). **Más información**: http://manpages.debian.org/cgi-bin/man.cgi?query=a2enmod

Habilitando y configurando módulos

Para ver cómo se gestiona un módulo de *Apache 2* usaremos el módulo `status` que ya viene activado de serie en la instalación. Este módulo permite averiguar de forma remota, sólo a usuarios autorizados, **información del estado actual del servidor** (servicio prestado, carga de trabajo actual, etc.). Esta información se encuentra en la ruta `/server-status/`.

Una vez que tengamos el módulo activo (presente en la carpeta `mods-enabled` antes mencionada), lo que en el caso de éste concreto ya ocurre en la instalación por defecto, podemos comprobar que el acceso a la información que ofrece el mismo está muy limitado, al ser información del servidor muy sensible (solo se permite acceder a la misma desde la propia máquina que tiene el servidor, `Require local`). Para habilitar máquinas desde las cuales está permitido acceder a esa información, simplemente debemos añadir esta línea en el fichero de configuración asociado al módulo (`status.conf`): "`Require ip <ip de la máquina autorizada a usar /server-status>`". Veremos más opciones sobre la directiva `Require` posteriormente. En la siguiente imagen se ve el archivo de configuración de `mod_status`:

La existencia de la ruta `/server-status/` accesible a todo el mundo es un potencial problema de seguridad grave. Siempre debemos comprobar que esta ruta no es accesible por todo el mundo o desactivar el mod si no lo vamos a usar.

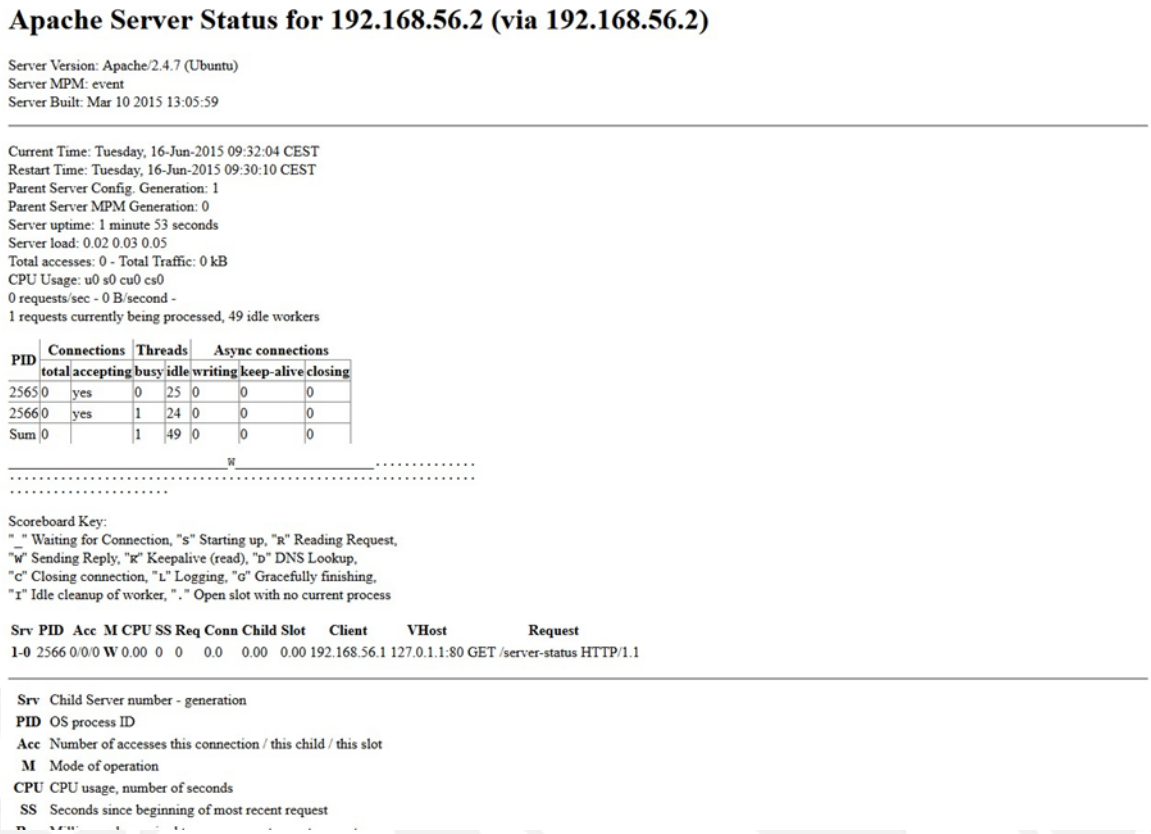
```
mainuser@ubuntuserver:/etc/apache2$ cd mods-enabled
mainuser@ubuntuserver:/etc/apache2/mods-enabled$ ls
alias.conf            authz_user.load  dir.conf          reqtimeout.conf
alias.load            autoindex.conf   dir.load          reqtimeout.load
auth_basic.load       autoindex.load   env.load          setenvif.conf
authn_file.load       cgid.conf        mime.conf         setenvif.load
authz_default.load    cgid.load        mime.load         status.conf
authz_groupfile.load  deflate.conf     negotiation.conf  status.load
authz_host.load       deflate.load     negotiation.load
mainuser@ubuntuserver:/etc/apache2/mods-enabled$ sudo nano status.conf_
```

Figura 37. Configuración de mod_status

Y a continuación añadimos la línea citada, habilitando en este caso el *host* como cliente autorizado:

```
<IfModule mod_status.c>
	# Allow server status reports generated by mod_status,
	# with the URI of http://servername/server-status
	# Uncomment and change the "192.0.2.0/24" to allow access from other hosts.

	<Location /server-status>
		SetHandler server-status
		Require local
		Require ip 192.168.56.1/24
	</Location>

	# Keep track of extended status information for each request
	ExtendedStatus On

	# Determine if mod_status displays the first 63 characters of a request or
	# the last 63, assuming the request itself is greater than 63 chars.
	# Default: Off
	#SeeRequestTail On

	<IfModule mod_proxy.c>
		# Show Proxy LoadBalancer status in mod_status
		ProxyStatus On
	</IfModule>
</IfModule>
```

Una vez reiniciado *Apache 2*, deberíamos poder ver el resultado desde la máquina autorizada (http://<ruta del servidor>/server-status/)



Apache Server Status for 192.168.56.2 (via 192.168.56.2)

Server Version: Apache/2.4.7 (Ubuntu)
Server MPM: event
Server Built: Mar 10 2015 13:05:59

Current Time: Tuesday, 16-Jun-2015 09:32:04 CEST
Restart Time: Tuesday, 16-Jun-2015 09:30:10 CEST
Parent Server Config. Generation: 1
Parent Server MPM Generation: 0
Server uptime: 1 minute 53 seconds
Server load: 0.02 0.03 0.05
Total accesses: 0 - Total Traffic: 0 kB
CPU Usage: u0 s0 cu0 cs0
0 requests/sec - 0 B/second -
1 requests currently being processed, 49 idle workers

PID	Connections		Threads		Async connections		
	total	accepting	busy	idle	writing	keep-alive	closing
2565	0	yes	0	25	0	0	0
2566	0	yes	1	24	0	0	0
Sum	0		1	49	0	0	0

Scoreboard Key:
"_" Waiting for Connection, "S" Starting up, "R" Reading Request,
"W" Sending Reply, "K" Keepalive (read), "D" DNS Lookup,
"C" Closing connection, "L" Logging, "G" Gracefully finishing,
"I" Idle cleanup of worker, "." Open slot with no current process

Srv	PID	Acc	M	CPU	SS	Req	Conn	Child	Slot	Client	VHost	Request
1-0	2566	0/0/0	W	0.00	0	0	0.0	0.00	0.00	192.168.56.1	127.0.1.1:80	GET /server-status HTTP/1.1

Srv Child Server number - generation
PID OS process ID
Acc Number of accesses this connection / this child / this slot
M Mode of operation
CPU CPU usage, number of seconds
SS Seconds since beginning of most recent request

Figura 38. Salida estándar de `mod_status`

Deshabilitando módulos

Más importante que la activación de módulos es la desactivación de estos. Un *Apache 2* **con un mínimo de módulos activos proporciona grandes ventajas en cuando a rendimiento, consumo de recursos y un nº potencialmente inferior de problemas de seguridad**. No obstante, debemos tener cuidado con los módulos que desactivamos, ya que también es cierto que a menos módulos, menos funcionalidad. Depende de la aplicación, opciones de configuración…los módulos que no son *DSO* no se pueden desactivar tan fácilmente como los que sí lo son (habría que recompilar el servidor).

La instalación por defecto en *Ubuntu*, aunque instala muy pocos módulos, puede reducirse aún más, hasta dejar un **servidor de capacidades mínimas** (lo justo para servir contenido estático). Como ya hemos dicho, la forma de habilitar o deshabilitar módulos es con los comandos `a2enmod` y `a2dismod` respectivamente, tal y como se ve en la siguiente imagen. Hay que remarcar de nuevo que hacer esto tiene un precio: el *Apache 2* del que dispondremos **tendrá muchas menos funcionalidades y facilidades**.

Por ejemplo, si bien antes al acceder al servidor se mostraba por defecto la página `index.html`, al desactivar los *mods* que listaremos se debe acceder expresamente a la página `index.html`, ya que el módulo que redirigía automáticamente a dicha página ya no está activo. Debemos por tanto tener en cuenta si las ventajas de rendimiento y seguridad que tiene hacer esto compensan las desventajas evidentes en funcionalidad.

Tras terminar esta operación, el servidor **será básicamente capaz de servir páginas estáticas con opciones mínimas de autenticación para los usuarios**. No es gran cosa por el momento, pero es una excelente base de partida para construir un servidor adaptado a nuestras necesidades. No obstante,

iremos activando algunos de los módulos desactivados en esta operación en diferentes secciones, a medida que describamos/necesitemos su funcionalidad.

Módulos preinstalados no estrictamente necesarios inicialmente:

- `access_compat`: Como veremos más adelante, si no usásemos ninguna directiva de este podríamos desactivarlo, aunque no será así en nuestro caso.
- El módulo `alias`
- Hasta que habilitemos alguna clase de autenticación más avanzada, podemos desactivar: `auth_basic`, `authn_file`, `authz_host` y `authz_user`
- El módulo `autoindex`
- `deflate`: El uso de este módulo **nos ahorra ancho de banda**, por lo que en principio lo dejaremos activo
- El módulo `dir`
- El módulo `env`
- `headers`: No lo desactivaremos porque lo hemos usado para "anonimizar" el servidor antes.
- El módulo `negotiation`
- El módulo `setenvif`
- El módulo `status`

```
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2dismod alias
[sudo] password for operario:
Module alias disabled.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2dismod auth_basic
Module auth_basic disabled.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2dismod authn_file
Module authn_file disabled.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2dismod authz_host
Module authz_host disabled.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2dismod authz_user
Module authz_user disabled.
To activate the new configuration, you need to run:
  service apache2 restart
```

Figura 39. Desactivación manual de módulos no necesarios inicialmente (I)

```
  service apache2 restart
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2dismod env
Module env disabled.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2dismod negotiation
Module negotiation disabled.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2dismod setenvif
Module setenvif disabled.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2dismod status
Module status disabled.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/etc/apache2/mods-enabled$ sudo service apache2 restart
 * Restarting web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress th
is message
                                                                    [ OK ]
operario@servidorapache:/etc/apache2/mods-enabled$
```

Figura 40. Desactivación manual de módulos no necesarios inicialmente (II) y reinicio del servidor

Vemos que ahora el nº de módulos activos en nuestro servidor **ha quedado reducido a la mínima expresión**:

```
operario@servidorapache:/etc/apache2/mods-enabled$ ls
access_compat.load  deflate.conf  headers.load  mpm_event.conf
authn_core.load     deflate.load  mime.conf     mpm_event.load
authz_core.load     filter.load   mime.load
operario@servidorapache:/etc/apache2/mods-enabled$
```

Figura 41. Apache 2 "minimizado"

Y, finalmente, comprobamos que el servidor sigue funcionando tras el reinicio:

Figura 42. Apache 2 "minimizado" en funcionamiento

Hecho esto, comprobamos que el SO + el servidor *Apache 2* "minimizado" solo consumen 46 Mb de RAM, dejando 443Mb libre (`watch -n 5 free -m`) de la MV de pruebas. http://www.linuxnix.com/2013/05/find-ram-size-inhttp://www.linuxnix.com/2013/05/find-ram-size-in-linuxunix.htmllinuxunix.html:

```
Cada 5,0s: free -m                                    Mon Jun 22 11:10:55 2015

             total       usado       libre     compart.     búffers      almac.
Mem:           489         174         315           0          28         100
-/+ buffers/cache:          46         443
Intercambio:     0           0           0
```

Figura 43. Consumo de memoria de Apache 2 "minimizado"

Para casos extremos en el que se deba obtener el mayor rendimiento posible se podría estudiar desactivar el módulo compilado `mod_log_config` (obviamente, mediante la recompilación del servidor). Esto evitará que el servidor haga log, por lo que cada petición no tendrá el coste añadido de tener que registrar en memoria secundaria información sobre la misma. La ganancia de rendimiento es pues por cada petición, y teniendo en cuenta que estamos interactuando con un dispositivo de almacenamiento que tradicionalmente es un cuello de botella, podría ser importante. No obstante, **perderemos toda la información sobre las peticiones realizadas**, **toda posibilidad de identificar que ha pasado en casos de fallos en el servidor** y por supuesto toda traza de posibles ataques que nos hayan podido realizar. Es pues una práctica muy desaconsejable, salvo razones de fuerza mayor.

Aunque este proceso lo hemos hecho para un servidor recién instalado, también puede hacerse para uno que esté en producción, pero de una forma mucho más cuidadosa por tener ya webs operativas. Si queremos hacer este proceso de optimización en un servidor actualmente en producción se recomienda tener en cuenta los siguientes aspectos:

- **Mejor hacer pruebas con una copia del servidor** y no sobre el original, porque nunca deben hacerse pruebas de esta clase con un servidor en producción dada la posibilidad de interrumpir su funcionamiento.
- **Nunca deben deshabilitarse varios módulos a la vez**, sino uno por uno y con un reinicio del servidor entre desactivación. Esto es así debido a que la desactivación de un módulo puede dar lugar a que alguna directiva ya no pueda interpretarse (el módulo desactivado era el que la procesaba) y por tanto el servidor de un fallo en el reinicio al no poder leer sus propios archivos de configuración. En caso de que esto ocurra podemos optar por cimentar la directiva problemática o volver a pensar si necesitamos el módulo desactivado.
- **Entre desactivación y desactivación conviene también que se navegue por las webs instaladas** por si la desaparición de un módulo alterase de alguna forma el comportamiento esperado de la web o determinadas secciones ya no fuesen visibles por usar una tecnología que formaba parte de un módulo desactivado.

Finalmente, aunque el procedimiento de instalación visto mediante `apt-get` simplifica mucho la instalación del servidor y la posibilidad de habilitar o deshabilitar módulos con `a2enmod` y `a2dismod` hace que la configuración del servidor se pueda hacer de manera muy flexible, para ganar más rendimiento puede estudiarse si merece la pena hacer esta primera **instalación compilando el servidor desde su código fuente para nuestro sistema**, habilitando y compilando junto con el ejecutable los módulos que sepamos que vamos a usar seguro: http://httpd.apache.org/docs/2.4/install.html

Ejecución del servidor como un usuario concreto

Una instalación estándar de *Apache 2* se inicia bajo la identidad del usuario `root`, pero una vez iniciado las peticiones a las páginas web que sirve se ejecutan bajo el usuario del sistema que está definido en la directiva `User`. Este será el usuario que se usará para atender peticiones web. El valor de esta directiva se define en el fichero `Apache2.conf`, pero dicho valor depende directamente del contenido del fichero `envvars`, que se encuentra en la misma localización que este último.

```
  GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf

KeepAliveTimeout 5


# These need to be set in /etc/apache2/envvars
User ${APACHE_RUN_USER}
Group ${APACHE_RUN_GROUP}
```

Figura 44. Configuración del usuario y grupo bajo el que Apache 2 se ejecuta

Aquí vemos como se define `www-data` como usuario por defecto y además también aparece la definición de una serie de rutas importantes para *Apache 2*:

```
  GNU nano 2.2.6         Archivo: /etc/apache2/envvars

# envvars - default environment variables for apache2ctl

# this won't be correct after changing uid
unset HOME

# for supporting multiple apache2 instances
if [ "${APACHE_CONFDIR##/etc/apache2-}" != "${APACHE_CONFDIR}" ] ; then
        SUFFIX="-${APACHE_CONFDIR##/etc/apache2-}"
else
        SUFFIX=
fi

# Since there is no sane way to get the parsed apache2 config in scripts, some
# settings are defined via environment variables and then used in apache2ctl,
# /etc/init.d/apache2, /etc/logrotate.d/apache2, etc.
export APACHE_RUN_USER=www-data
export APACHE_RUN_GROUP=www-data
# temporary state file location. This might be changed to /run in Wheezy+1
export APACHE_PID_FILE=/var/run/apache2/apache2$SUFFIX.pid
export APACHE_RUN_DIR=/var/run/apache2$SUFFIX
export APACHE_LOCK_DIR=/var/lock/apache2$SUFFIX
# Only /var/log/apache2 is handled by /etc/logrotate.d/apache2.
export APACHE_LOG_DIR=/var/log/apache2$SUFFIX
```

Figura 45. Asignación del usuario y grupo para Apache 2

Sin embargo, no debemos olvidar que cualquier programa que se ejecute bajo la identidad de `root` debe estar protegido para que ningún usuario no `root` pueda modificarlo. Esto no solo se limita a que los ficheros en si solo puedan ser modificados por `root`, sino que también debe aplicarse a los directorios y subdirectorios de la jerarquía donde están situados estos programas y los ficheros que usan para su funcionamiento cotidiano (configuración, log,…). Por suerte, **la instalación por defecto de *Apache 2* ya**

controla adecuadamente este aspecto. El ejecutable principal, situado en `/usr/sbin` tiene unos permisos correctos:

```
lrwxrwxrwx  1 root    root         11 may 13 20:34 update-grub2 -> update-grub
-rwxr-xr-x  1 root    root        241 ago 16  2011 update-grub-gfxpayload
-rwxr-xr-x  1 root    root       1691 dic 11  2013 update-info-dir
-rwxr-xr-x  1 root    root       9164 jun  1  2012 update-initramfs
-rwxr-xr-x  1 root    root       3063 sep 14  2010 update-locale
-rwxr-xr-x  1 root    root       8520 ene  6 20:28 update-mime
-rwxr-xr-x  1 root    root      31176 abr 10  2014 update-passwd
-rwxr-xr-x  1 root    root       2905 mar 24  2014 update-pciids
-rwxr-xr-x  1 root    root      16615 mar 13 20:49 update-rc.d
-rwxr-xr-x  1 root    root       1076 abr  8  2014 update-usbids
-rwxr-xr-x  1 root    root      16674 nov  7  2012 update-xmlcatalog
-rwxr-xr-x  1 root    root       1524 may 13 16:51 upgrade-from-grub-legacy
-rwxr-xr-x  1 root    root     110456 feb 17  2014 useradd
-rwxr-xr-x  1 root    root      76728 feb 17  2014 userdel
-rwxr-xr-x  1 root    root     110264 feb 17  2014 usermod
-rwsr-sr-x  1 libuuid libuuid   18904 feb 12 19:53 uuidd
-rwxr-xr-x  1 root    root       1773 dic 28  2005 validlocale
lrwxrwxrwx  1 root    root         48 jun 15 08:44 vbox-greeter -> /opt/VBoxGues
tAdditions-4.3.28/sbin/vbox-greeter
lrwxrwxrwx  1 root    root         47 jun 15 08:44 VBoxService -> /opt/VBoxGuest
Additions-4.3.28/sbin/VBoxService
-rwxr-xr-x  1 root    root       6344 feb 18  2013 vcstime
lrwxrwxrwx  1 root    root          4 jun 15 07:51 vigr -> vipw
-rwxr-xr-x  1 root    root      53808 feb 17  2014 vipw
-rwxr-xr-x  1 root    root     213952 mar 12 17:27 visudo
-rwxr-xr-x  1 root    root      14600 sep 16  2013 vpddecode
-rwxr-xr-x  1 root    root      51664 feb 25 17:57 zic
operario@servidorapache:/usr/sbin$ ls -la apache2
-rwxr-xr-x 1 root root 637496 mar 10 14:07 apache2
operario@servidorapache:/usr/sbin$ _
```

Figura 46. Permisos del ejecutable de Apache 2

Y lo mismo ocurre con el directorio `/etc/apache2`, donde están los ficheros de configuración, etc.

```
operario@servidorapache:/etc/apache2$ ls -la
total 88
drwxr-xr-x  8 root root  4096 jun 15 10:47 .
drwxr-xr-x 93 root root  4096 jul  2 20:24 ..
-rw-r--r--  1 root root  7301 jul  2 19:59 apache2.conf
drwxr-xr-x  2 root root  4096 jun 15 10:47 conf-available
drwxr-xr-x  2 root root  4096 jun 15 10:47 conf-enabled
-rw-r--r--  1 root root  1782 ene  3  2014 envvars
-rw-r--r--  1 root root 31063 ene  3  2014 magic
drwxr-xr-x  2 root root 12288 jun 15 10:47 mods-available
drwxr-xr-x  2 root root  4096 jun 22 10:54 mods-enabled
-rw-r--r--  1 root root   320 ene  7  2014 ports.conf
drwxr-xr-x  2 root root  4096 jun 15 10:47 sites-available
drwxr-xr-x  2 root root  4096 jun 15 10:47 sites-enabled
operario@servidorapache:/etc/apache2$ _
```

Figura 47. Permisos de los ficheros usados por Apache 2

Cabe destacar que **no todas las instalaciones de *Apache 2* tienen que estar en este directorio**. Para consultar cuál es, o para cambiarlo en último caso, hay que modificar el valor de la directiva `ServerRoot` en el fichero `apache2.conf`. Obviamente, no podemos poner un ruta donde no estén los ficheros que aparecen en la imagen anterior y esto solo tiene sentido si movemos físicamente todos los ficheros

usados por *Apache 2* por algún motivo. La directiva aparece por defecto comentada, pero podemos descomentarla siempre que la ruta sea válida:

```
  GNU nano 2.2.6          Archivo: apache2.conf                  Modificado



# Global configuration
#


#
# ServerRoot: The top of the directory tree under which the server's
# configuration, error, and log files are kept.
#
# NOTE!  If you intend to place this on an NFS (or otherwise network)
# mounted filesystem then please read the Mutex documentation (available
# at <URL:http://httpd.apache.org/docs/2.4/mod/core.html#mutex>);
# you will save yourself a lot of trouble.
#
# Do NOT add a slash at the end of the directory path.
#
ServerRoot "/etc/apache2"
```

Figura 48. Ruta base de Apache 2

Finalmente, tenemos que asegurarnos de que el directorio donde están los logs de *Apache 2* (por defecto `/var/log/apache2`) **solo es accesible por** `root`, y nadie más puede ni siquiera leerlo. Ya no solo es importante que **nadie pueda escribir en el log** por miedo a que pueda ocultar sus acciones, sino que tampoco es oportuno que pueda leerse ya que un fichero de log puede tener información muy relevante para un posible ataque contra la seguridad del sistema (IDs de sesión, nombres de páginas que no queremos que sean conocidos, etc.). La instalación por defecto de nuevo se asegura también de esto:

```
operario@servidorapache:/etc/apache2$ ls -la /var/log/apache2/
total 12936
drwxr-x---  2 root adm        4096 jun 15 10:47 .
drwxrwxr-x 10 root syslog     4096 jul  2 19:09 ..
-rw-r-----  1 root adm    10855943 jul  2 21:03 access.log
-rw-r-----  1 root adm     2373707 jul  2 20:00 error.log
-rw-r-----  1 root adm           0 jun 15 10:47 other_vhosts_access.log
operario@servidorapache:/etc/apache2$ _
```

Figura 49. Permisos de los log de Apache 2

Si por un defecto de la instalación, o en una reconfiguración posterior, dejamos que usuarios no `root` puedan modificar los ejecutables, o alguno de los ficheros clave necesarios para su trabajo, el nº de posibles problemas de seguridad o mal funcionamiento que podemos experimentar es muy alto. Por ejemplo:

- **El ejecutable de *Apache 2* puede reemplazarse por otro con fines maliciosos**, o bien puede ser modificado para que los tenga.
- Además de la posible **fuga de información** por lectura, permitir modificar los ficheros de log permite a cualquier usuario con malas intenciones borrar su rastro, falsearlo o bien reemplazar el fichero por un enlace simbólico a un ejecutable con fines maliciosos
- **Si ser altera la configuración de *Apache 2*** se pueden provocar malos funcionamientos, alteraciones en el servicio de contenidos o problemas de seguridad.

Otro aspecto similar es el relativo al usuario bajo el que se sirven las peticiones por defecto, que como vimos en una imagen anterior es el usuario `www-data`. Si examinamos el contenido del directorio por defecto donde se alojan las web servidas (`/var/www/html`) vemos los siguientes permisos:

```
operario@servidorapache:~$ cd /var/www
operario@servidorapache:/var/www$ ls -la
total 12
drwxr-xr-x  3 root root 4096 jun 15 10:47 .
drwxr-xr-x 13 root root 4096 jun 15 10:47 ..
drwxr-xr-x  3 root root 4096 jun 16 09:28 html
operario@servidorapache:/var/www$
```

Figura 50. Permisos de la ruta base de las web servidas

Esto permite solamente al usuario `root` añadir nuevos ficheros que formen parte de las web servidas. Además, permite a cualquiera leerlos y ejecutarlos (es decir, en el caso de carpetas, acceder a su contenido). En principio esto es suficiente para un uso normal de *Apache 2*, ya que el usuario `www-data` (usuario "normal", no `root`) puede acceder a estos contenidos con esta configuración. Vemos que los ficheros creados dentro de esta estructura tienen un esquema de permisos similar (escritura para `root`, lectura para el resto), pero sin incluir el de ejecución por no tener sentido en ficheros cuyo contenido es puramente texto:

```
operario@servidorapache:/var/www/html$ ls -la
total 16
drwxr-xr-x 3 root root 4096 jul  2 19:20 .
drwxr-xr-x 3 root root 4096 jun 15 10:47 ..
drwxr-xr-x 2 root root 4096 jun 16 07:41 folder
-rw-r--r-- 1 root root   71 jun 16 08:17 index.html
```

Figura 51. Permisos por defecto de los ficheros servidor por Apache 2

Y por supuesto, podemos acceder a su contenido:

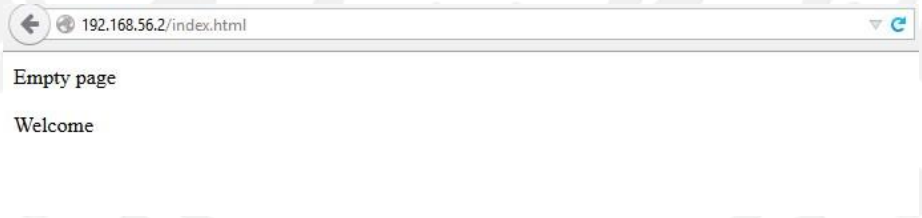


Figura 52. Sirviendo una web básica con permisos por defecto

¿Qué ocurrirá ahora si le cambiamos el usuario propietario a uno de los ficheros servidos (comando `chown`*) y eliminamos el permiso para el resto de los usuarios?* Pues que ahora el usuario que intentará acceder al fichero (`www-data`) no tiene permisos para acceder al mismo ni como el propio usuario ni como grupo, por lo que *Apache 2* no nos servirá su contenido. Con esto claramente vemos que el usuario bajo el que se hace la petición es `www-data` y no `root`, puesto que en este último caso tendría la posibilidad de leer el fichero (permiso de lectura de grupo `root`):

```
operario@servidorapache:/var/www/html$ sudo chmod o-r index.html
operario@servidorapache:/var/www/html$ ls -la
```

Figura 53. Cambio de propietario y permisos de un fichero

Figura 54. Acceso a un fichero prohibido por cambio de permisos

Por tanto, los permisos de los ficheros servidor por *Apache 2* han de controlarse cuidadosamente para poder estar seguros de que las webs se sirven con normalidad, de acuerdo a lo que se pretende.

Por otro lado, no todas las instalaciones de *Apache 2* deben tener el directorio por defecto para alojar webs mencionado. Para saber cuál es, o bien para cambiar el mismo, tenemos que ir al fichero de configuración del sitio web activo por defecto (`000-default.conf`) y consultar el valor de la directiva `DocumentRoot`:

```
 GNU nano 2.2.6 Archivo: ...apache2/sites-enabled/000-default.conf

<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and port t$
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
        # specifies what hostname must appear in the request's Host: header to
        # match this virtual host. For the default virtual host (this file) this
        # value is not decisive as it is used as a last resort host regardless.
        # However, you must set it for any further virtual host explicitly.
        #ServerName www.example.com

        #ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html
        <Directory /var/www/html>
                Order allow,deny
                allow from all
                Options -Indexes
        </Directory>

        # Available loglevels: trace8, ..., trace1, debug, info, notice, warn,
        # error, crit, alert, emerg.
        # It is also possible to configure the loglevel for particular
        # modules, e.g.
        #LogLevel info ssl:warn

        ErrorLog ${APACHE_LOG_DIR}/error.log

^G Ver ayuda ^O Guardar   ^R Leer Fich ^Y RePág.    ^K Cortar Tex^C Pos actual
^X Salir     ^J Justificar^W Buscar    ^V Pág. Sig. ^U PegarTxt  ^T Ortografía
```

Figura 55. DocumentRoot del sitio web por defecto de Apache 2

Si *Apache 2* o alguna web alojada en el mismo usase la tecnología *CGI* para la creación de páginas dinámicas, entonces tendríamos que consultar el valor de la directiva `ScriptAlias` del fichero de configuración `serve-cgi-bin.conf` para ver en qué directorio está pensado que se puedan alojar este tipo de ficheros ejecutables, al estar ya preparado para ello. No obstante, por defecto el uso de *scripts CGI* no está activo.

```
operario@servidorapache:/etc/apache2/conf-available$ cat serve-cgi-bin.conf
<IfModule mod_alias.c>
        <IfModule mod_cgi.c>
                Define ENABLE_USR_LIB_CGI_BIN
        </IfModule>

        <IfModule mod_cgid.c>
                Define ENABLE_USR_LIB_CGI_BIN
        </IfModule>

        <IfDefine ENABLE_USR_LIB_CGI_BIN>
                ScriptAlias /cgi-bin/ /usr/lib/cgi-bin/
                <Directory "/usr/lib/cgi-bin">
                        AllowOverride None
                        Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
                        Require all granted
                </Directory>
        </IfDefine>
</IfModule>
```

Figura 56. Ruta donde se guardan los programas CGI, de usarse en el servidor

Finalmente, hay que recordar que si bien esta es una forma de proteger archivos que no queremos que nadie vea dentro del directorio de la web, porque son de uso interno, **no es la solución más efectiva**. En este sentido, lo mejor es hacer que aquello que no queramos que sea accedido directamente no esté presente en los directorios donde se alojan las web de *Apache 2*. De esta forma, seremos inmunes a cualquier problema de seguridad, configuración, etc. que haga que los permisos previstos no sean efectivos.

Configuración de permisos en directorios de una web

Usuarios no root con permisos de edición sobre contenidos concretos

En este apartado vamos a exponer **una serie de ejemplos para la gestión de permisos de directorios** de la *web*. Si bien en una actividad anterior hemos visto cómo `root` era el único usuario capaz de crear nuevos archivos dentro de la estructura que aloja las webs servidas por *Apache 2*, en muchas ocasiones tener que ser `root` (o un usuario `sudoer`) para hacer esta tarea es algo que no es práctico. La cuenta de `root`, o una cuenta `sudoer`, dados sus elevados privilegios, debe ser una cuenta que **se use sólo para lo imprescindible**. Por ese motivo, tiene sentido crear un usuario específico cuyo cometido sea crear nuevo contenido para alguna de las webs alojadas, siendo este el objetivo de este ejercicio práctico. Si tuviéramos varias webs alojadas, siguiendo este procedimiento podríamos hacer que un mismo usuario pudiese modificar el contenido de todas ellas o bien crear un usuario independiente para cada una.

Para realizar la tarea descrita vamos a suponer que tenemos un grupo de usuarios especial llamado "desarrolladores web". A este grupo van a pertenecer todos aquellos usuarios que nosotros consideremos que ejercen ese rol (en nuestro ejemplo solo habrá uno, el usuario desweb1) y que tendrán permisos de escritura sobre una carpeta concreta, donde se supone que estarán los contenidos de la nueva web a servir. Para hacer esto es necesario tener un conocimiento básico de como manipular grupos y usuarios en un sistema *Linux* basado en *Debian*, algo que describiremos paso a paso.

- **Como primer paso, se creará un grupo "`desweb`"** (desarrolladores *web*) cuyos usuarios serán los únicos autorizados a manipular (modificar *webs*, añadir nuevas páginas, etc.) el contenido de un directorio `/var/www/html/scripts` que hemos creado para este fin, y en donde estará el contenido editable por estos usuarios. Nótese que para que todo funcione el resto de los usuarios podrán leer y ejecutar su contenido. Para crear el grupo `desweb` podemos usar el comando `addgroup` (`sudo addgroup desweb`). Si queremos que `root` pueda seguir introduciendo o modificando ficheros, tendría sentido añadir a dicho usuario `root` al grupo que acabamos de crear (`sudo adduser root desweb`) .

```
operario@servidorapache:/var/www/html$ sudo addgroup desweb
Añadiendo el grupo `desweb' (GID 1001) ...
Hecho.
operario@servidorapache:/var/www/html$ sudo adduser root desweb
Añadiendo al usuario `root' al grupo `desweb' ...
Adding user root to group desweb
Hecho.
```

Figura 57. Creación de un grupo de "desarrolladores web"

- Ahora crearemos el usuario `desweb1`, que pertenezca al grupo anterior. Esto puede hacerse con el comando `adduser` (`sudo adduser desweb1` para crear el usuario y `sudo adduser desweb1 desweb` para meter al usuario en el grupo anterior).

```
operario@servidorapache:/var/www/html$ sudo adduser desweb1
Añadiendo el usuario `desweb1' ...
Añadiendo el nuevo grupo `desweb1' (1002) ...
Añadiendo el nuevo usuario `desweb1' (1001) con grupo `desweb1' ...
Creando el directorio personal `/home/desweb1' ...
Copiando los ficheros desde `/etc/skel' ...
Nueva contraseña:
Vuelva a escribir la nueva contraseña:
passwd: password updated successfully
Changing the user information for desweb1
Enter the new value, or press ENTER for the default
        Full Name []:
        Room Number []:
        Work Phone []:
        Home Phone []:
        Other []:
¿Es correcta la información? [S/n] s
operario@servidorapache:/var/www/html$
operario@servidorapache:~$ sudo adduser desweb1 desweb
[sudo] password for operario:
Añadiendo al usuario `desweb1' al grupo `desweb' ...
Adding user desweb1 to group desweb
Hecho.
operario@servidorapache:~$
```

Figura 58. Creación de un usuario del grupo de desarrolladores web

Cambiamos ahora el propietario del directorio mencionado y sus contenidos para que su propietario sea el usuario que hemos añadido y su grupo, con el comando: `chown -R desweb1:desweb /var/www/html/scripts`

```
operario@servidorapache:/var/www/html$ ls -la
total 16
drwxr-xr-x 3 root root 4096 jul  2 20:30 .
drwxr-xr-x 3 root root 4096 jun 15 10:47 ..
-rw-r--r-- 1 root root   71 jun 16 08:17 index.html
drwxr-xr-x 2 root root 4096 jul  2 20:30 scripts
operario@servidorapache:/var/www/html$ sudo chown -R desweb1:desweb scripts
```

Figura 59. Cambio del propietario de un directorio concreto de la estructura de las web servidas

- Y finalmente, con el último paso lograremos el efecto deseado. Debemos pensar en lo que tenemos en este momento para saber si hemos tomado la decisión correcta para el futuro. Hemos creado un grupo `desweb` que de momento contiene un único usuario `desweb1`. Como cabe pensar que en el futuro podrían añadirse más desarrolladores web, tendría sentido conceder permisos de escritura a cualquier usuario de este grupo.

  Lo que hagamos con el usuario `desweb1` en este sentido no nos afecta demasiado, puesto que siempre tendremos los permisos de grupo para controlar lo que permitimos. Antes de asignar permisos de escritura al grupo que hemos elegido para introducir nuevos ficheros en el directorio, tenemos que considerar un último aspecto. Este aspecto es relativo a los bits `setuid` y `setgid` del directorio `scripts` que estamos tratando (https://www.gnu.org/software/coreutils/manual/html_node/Directory-Setuid-and-Setgid.html).

  Activar el bit *setgid* sobre un directorio hace que cualquier nuevo fichero que se introduzca sobre el directorio herede el grupo propietario de dicho directorio. **Activar el *setuid*** hace lo propio con el usuario, pero sólo en algunos sistemas *Linux* (y no es el caso de nuestro *Ubuntu* de pruebas). Para activar el bit *setgid* y permisos de escritura para el grupo en el directorio que estamos

manipulando habría que ejecutar `chmod -R 2575 /var/www/html/scripts`. Activar el bit *setuid* sería igual pero con la máscara de permisos `6575`. Esto es conveniente para que cualquier nuevo fichero que introduzcamos en el directorio mencionado pueda hacerse fácilmente editable por los usuarios del grupo que hemos creado al efecto con un simple cambio de permisos para el grupo si fuese necesario (ya que todos los ficheros tendrían el mismo grupo):

```
operario@servidorapache:/var/www/html$ sudo chmod -R 2575 scripts
operario@servidorapache:/var/www/html$ ls -la
total 16
drwxr-xr-x 3 root    root   4096 jul  2 20:30 .
drwxr-xr-x 3 root    root   4096 jun 15 10:47 ..
-rw-r--r-- 1 root    root     71 jun 16 08:17 index.html
dr-xrwsr-x 2 desweb1 desweb 4096 jul  2 20:30 scripts
operario@servidorapache:/var/www/html$ _
```

Figura 60. Permisos finales del directorio editable por desarrolladores

Ahora podemos crear un fichero a servir e introducirlo en el directorio:

```
 GNU nano 2.2.6         Archivo: script.simulado.txt

Este es un script simulado
```

Figura 61. Fichero a introducir por un desarrollador

Y posteriormente copiarlo a su destino. Nótese que el `sudo` es necesario porque estamos haciendo la copia con un usuario que no es del grupo `desweb` (pero si *sudoer*, y `root` pertenece al grupo). Si hiciésemos *login* como `desweb1`, y este fuese el creador del fichero, **el `sudo` no sería necesario** puesto que el directorio tiene permisos de escritura para usuarios de este grupo.

```
operario@servidorapache:~/docs$ ls -la
total 12
drwxrwxr-x 2 operario operario 4096 jul  2 21:00 .
drwx------ 3 operario operario 4096 jul  2 20:59 ..
-rw-rw-r-- 1 operario operario   27 jul  2 21:00 script.simulado.txt
operario@servidorapache:~/docs$ sudo cp script.simulado.txt /var/www/html/script
s/
operario@servidorapache:~/docs$ ls -la /var/www/html/scripts/
total 12
dr-srwsr-x 2 desweb1 desweb 4096 jul  2 21:00 .
drwxr-xr-x 3 root    root   4096 jul  2 20:30 ..
-rw-r--r-- 1 root    desweb   27 jul  2 21:00 script.simulado.txt
operario@servidorapache:~/docs$ _
```

Figura 62. Fichero puesto en servicio

Y, como los propios permisos indican, el fichero podría servirse sin problemas. Si además quisiéramos que el propio fichero fuese editable in-situ por los desarrolladores web (puesto que ahora solo se permite introducir o reemplazar ficheros en la carpeta), bastaría con activar el permiso de escritura para el grupo, como mencionamos anteriormente. No obstante, **la modificación de ficheros in situ no es una práctica aconsejable en un servidor en producción**, ya que podría ocasionar muchos problemas al servicio e incluso a la seguridad.

Figura 63. Sirviendo el fichero introducido

Debemos tener en cuenta que cada directorio podría tener una configuración independiente (probablemente no deseemos dejar permisos de lectura, solo de ejecución, a directorios con *scripts*), por lo que habrá que personalizarlo en función de cada caso. Este ejemplo por otro lado está pensado para que los contenidos se sirvan directamente desde el directorio en el que se editan. Si no fuese así, puede convenir negar todos los permisos a "otros" con `chmod -R 2570`.

Bloque 2: Opciones de Configuración y Extensión

Estructura del procesamiento de peticiones de *Apache 2*

Hasta ahora hemos visto qué eran los módulos de *Apache 2*, para qué servían y cómo era la forma de extender la funcionalidad de *Apache 2* y personalizarla de acuerdo con nuestras necesidades. Hemos visto como habilitar y deshabilitar los módulos y, de esta forma, **cómo configurar el servidor de acuerdo con nuestras necesidades** de manera básica y sencilla.

No obstante, quedarnos en esto es simplemente arañar la superficie de las posibilidades de *Apache 2* en lo relativo su funcionamiento, y un administrador debe tener un conocimiento más profundo **de lo que está ocurriendo cuando se realiza una petición**, más allá de simplemente activar o desactivar partes de un servidor. La labor de un administrador está separada de la de un desarrollador en muchas ocasiones, por lo cual no tiene que conocer en profundidad cómo programar contra el servidor para adaptarlo a necesidades concretas. No obstante, sí que debe tener una visión general de cómo se realiza esto, aunque sea simplemente para estar familiarizado con la terminología empleada a la hora de consultar un log o documento técnico que necesite para hacer alguna tarea. Esta es la labor de este bloque, que permitirá, **siempre desde el punto de vista de un administrador**, manipular *Apache 2* de una forma mucho más avanzada, realizando tareas muy diversas que proporcionen un control amplio sobre sus funcionalidades.

Para entender correctamente cómo funciona *Apache 2* es necesario tener una comprensión detallada de **cómo procesa el servidor las peticiones que le llegan**, las fases de procesamiento existentes, las diversas partes del código por las que las peticiones pasan y qué hace cada una de ellas. Además, si entendemos correctamente cómo se estructura esta parte del servidor, sabremos también donde se "**enchufan**" los diversos módulos, manejadores de contenidos (***handlers***) y filtros (***filters***) que permiten alterar/personalizar esta cadena de procesamiento en varios puntos, como veremos posteriormente.

No obstante, insistimos en que la intención es simplemente entender la estructura general desde el punto de vista de un administrador (el usuario objetivo de este libro), no lograr un conocimiento técnico profundo que nos permita programar nuevos módulos desde cero para el servidor (es decir, ser un usuario desarrollador), puesto que esto está fuera de los objetivos de este libro. Por ese motivo, en la explicación siguiente se han hecho simplificaciones y concesiones con el objetivo de entender el proceso general y no entrar en excesivos detalles. Para tener una descripción técnica aún más detallada de la cadena de procesamiento de peticiones de *Apache 2* se recomienda consultar los siguientes enlaces:

- http://httpd.apache.org/docs/2.4/developer/modguide.html
- http://www.fmc-modeling.org/category/projects/*Apache* 2/amp/3_3Extending_*Apache*.html
- http://docstore.mik.ua/orelly/*Apache*_mod/79.htm#BIN173

Peticiones a *Apache 2*

Aunque un usuario del servidor de *Apache 2* no es realmente consciente de ello, sólo por servir una simple página HTML estática en *Apache 2* ya **se está poniendo en marcha un buen nº de módulos *software*** que realizan su función de forma colaborativa para lograr servirla. La arquitectura de *Apache 2* consta de tres partes:

1. Un "**núcleo**" común a todas las versiones de *Apache 2* para cualquier plataforma
2. Una capa dependiente de la plataforma donde se despliegue (***Apache 2 Portable Runtime*** o *APR*, https://apr.apache.org/)
3. Un conjunto variable de **módulos** que sirven la petición.

Los módulos son la parte más importante a la hora de servir peticiones HTTP, puesto que **cada uno de ellos suele trabajar con un aspecto concreto de dicha petición HTTP**. De esta manera, los módulos se "encadenan" cumpliendo cada uno una función específica y complementaria al resto de módulos, resultando así más fácil de comprender y de mantener toda la estructura.

La forma más simple de ver a *Apache 2* desde el punto de vista de un administrador es que *Apache 2* es un programa que **acepta** peticiones HTTP, las **procesa** y **devuelve una respuesta** HTTP resultante.

- *¿Quién elabora la respuesta a cada petición?* Un "**generador de contenidos**", que es básicamente una "pieza" de toda la cadena de procesamiento de *Apache 2*.
- *¿Toda petición debe pasar obligatoriamente por un generador de contenidos?* Si, obligatoriamente. Se elegirá uno adecuado de acuerdo con el tipo de recurso pedido (por ejemplo `.php`, `.aspx`, …)
- *¿Y si no hay uno adecuado?* Si no hubiese uno configurado para la petición recibida, entonces **se usaría uno por defecto**, que normalmente leerá el fichero pedido del almacenamiento del servidor y lo devolverá tal cual al cliente (si tiene permiso para acceder al mismo).
- *¿Dónde encaja un generador de contenidos dentro de la arquitectura de 3 capas que vimos antes?* Es parte de la tercera, es decir, **es un módulo**. No obstante, no todos los módulos implementan generadores de contenidos (depende de su propósito).

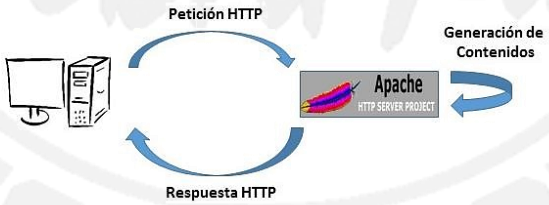


Figura 64. Esquema básico de procesamiento de Apache 2

No obstante, el esquema presentado en la figura anterior es muy simple y no muestra la flexibilidad y la potencia que ofrece *Apache 2* para poder manipular la forma en la que sirve las peticiones en función de las necesidades de los usuarios. Por ese motivo, en la siguiente sección se profundizará mucho más en este esquema.

El ciclo de procesamiento de peticiones de *Apache 2*

El proceso de servir peticiones está dividido en diferentes **fases de procesamiento**. Esto quiere decir que antes de que el contenido sea realmente generado, la petición pasa por diferentes fases que pueden modificarla o bien prohibirla (si no está autorizada), con lo que el contenido no se generaría. Un

determinado módulo posee código que puede "registrarse" para actuar en determinados puntos de la cadena de procesamiento de las peticiones (en un solo punto, en varios…según necesite para cumplir su función), por eso, en último caso, un mismo módulo podría ocuparse de gestionar aspectos durante todas y cada una de las fases de esta si quisiese, aunque no es el caso más común. A cada uno de estos "puntos de registro" para módulos se les denomina "***hooks***", y cada fase de procesamiento de la petición ofrece un determinado nº de ellos, para que así cualquiera pueda diseñar un módulo para *Apache 2* que entre en funcionamiento en el punto de procesamiento de la petición que necesite.

Por tanto, para entender la cadena de procesamiento de las peticiones de *Apache 2* es necesario tener claras sus **fases**, motivo por el cual se detallarán a continuación. Nótese que en la descripción de algunas fases se mencionarán a veces nombres de módulos que veremos posteriormente. Estos módulos actúan en su fase suscribiéndose a alguno de los *hooks* de la misma con una función concreta, de manera que la unión de los resultados de todos ellos formará la respuesta final a una petición. Esto lo hacen todos los módulos distribuidos con *Apache 2* (ya sean compilados contra el ejecutable principal o bien *DSOs*) y, obviamente, lo mismo podría hacer un módulo que desarrollemos nosotros mismos.

Fase "*Request Parsing*"

Consta de los siguientes pasos:

- La URI de la petición **se comprueba** para saber qué generador de contenidos tiene que usarse para responder a la misma. Esto se determina por la configuración de *Apache 2* y leyendo las cabeceras de la petición, comprobando de paso que lo enviado sigue la estructura de una petición HTTP correcta. A este *hook* se la denomina ***Post read request***.
- La URI de la petición **se "mapea" contra el sistema de ficheros del servidor**, es decir, se busca el fichero que el cliente ha pedido. A este *hook* de la denomina ***URI Translation***. Ha de tenerse en cuenta que este fichero puede ser un fichero estático de contenido (`.txt`, `.pdf`, `.html`,…), un *script* de alguna clase o cualquier otro tipo de fichero que el generador de contenido determinado en el paso anterior necesite. Podemos dividir este paso en dos partes:
    - ***Translation***: Implementa un determinado procesamiento de la URI, como la **decodificación de caracteres especiales** (*unescaping*) o bien la eliminación de acceso a rutas relativas (`./` o `../`, para asegurarse de que se accede a una ruta absoluta). Debe tenerse en cuenta que módulos que modifican las URIs como `mod_alias` o `mod_rewrite` pueden cambiar la URI de la petición en función de las reglas que tengan configuradas.
    - ***Map to storage***: Determina si el fichero al que se accede es un fichero existente y "legal". Si se ha activado la "negociación de contenidos" (`mod_negotiation`), entonces se usará para encontrar la versión del recurso al que se quiere acceder más adecuada para la petición realizada (en función, como veremos más adelante, de las cabeceras HTTP enviadas por el navegador)
- Tras el segundo paso llega el *hook* de ***Header parse***. Se encarga de examinar las cabeceras de la petición y quizá de cambiarlas antes de seguir con la cadena de procesamiento. Un uso común de esta fase es bloquear peticiones de robots de búsqueda antes de que supongan más carga de trabajo al servidor (al bloquearse en una fase muy temprana de procesamiento de la petición).

Fase "*Security*"

Consta de los siguientes pasos, una vez concluidos con éxito los anteriores:

- El primer paso es comprobar **si la petición puede tener acceso al contenido** que ha solicitado (*hook* de ***Access control***). Los módulos registrados en este *hook* determinan esto en función de la IP del cliente o de otros parámetros, como veremos más adelante con módulos como `mod_access_compat` o `mod_authz_host`.
- En el caso de que el acceso al recurso esté permitido en el paso anterior, **se comprueba la identidad de usuario y su clave** enviadas en la petición (*hook* ***Authentication***). Obviamente, para que esto ocurra tiene que existir un módulo registrado en dicho *hook*, con la particularidad de que **sólo se permite a un módulo hacer esta comprobación** en cada petición (no pueden existir varios módulos de esta clase registrados en el *hook* para una petición a un mismo recurso).
- Este paso solo se supera con éxito si el acceso al recurso requiere una determinada identidad de usuario válida y las credenciales se envían con la petición usando el método de autenticación correcto, según lo configurado para dicho recurso. Veremos posteriormente más en detalle módulos de autenticación.
- Si la identidad de usuario suministrada es válida, lo siguiente a comprobar es **si dicha identidad de usuario tiene realmente acceso al recurso**. De igual forma, solo se permite un módulo registrado en este *hook* por petición a un recurso determinado. Este el *hook* llamado ***Authorization***.

Fase "*Preparation*"

- Una vez que se ha garantizado el acceso a un recurso particular gracias a todas las fases anteriores y a los módulos registrados para dar servicio en los diferentes *hooks* de las mismas, se pasa a **determinar el tipo MIME del recurso pedido**. La elección de un tipo MIME tiene efecto a la hora de seleccionar el generador de contenidos que va a dar respuesta a la petición finalmente. Nótese que el tipo MIME seleccionado puede cambiarse posteriormente en las siguientes fases de la cadena. A este hook se le conoce como ***MIME type checking*****,** y módulos como `mod_mime` o `mod_mime_magic` actúan en este momento preciso de la cadena de procesamiento de la petición.
- Llegados a este punto ya sabemos qué recurso se ha pedido, qué podemos acceder a él y cuál es tu tipo MIME (un fichero de texto, una imagen, etc.). Antes de pasar a generar el contenido, ahora que Apache 2 ya ha recopilado toda la información necesaria para ello y sabe que es lo que se está pidiendo, existe un *hook* que da una última oportunidad a los módulos habilitados de Apache 2 para hacer algún cambio a la cabecera de la petición. A este *hook* se le denomina ***Fixup***.

Fase "*Handler*"

- *Hook* ***Insert filter***: Este *hook* permite a los módulos especificar **filtros de entrada** o de **salida** para las peticiones. Los filtros de entrada complementan el procesamiento de las cabeceras HTTP recibidas antes de que el contenido de respuesta propiamente dicho sea generado. Los filtros de salida actúan sobre el contenido de la respuesta una vez generado, para hacer alguna clase de transformación o modificación sobre el mismo antes de ser enviado al cliente. Veremos más sobre la funcionalidad de los filtros más adelante en esta sección.
- *Hook* ***Content Handler*** (manejador de contenidos): **Este es el *hook* más importante en el ciclo de servicio de peticiones**, porque el código que se suscriba al mismo es el encargado de leer la información del recurso pedido, procesarla y enviar la petición de respuesta al cliente (pasando previamente el contenido generado por el conjunto de filtros de salida antes de enviarlo al cliente, como veremos).

Fase "Request Logging"

Una vez que el manejador de contenidos, previo paso por los filtros de salida, han enviado la petición al cliente, la petición se manda a un log (*hook* ***Logging handler***) que guarda información acerca del recurso servido y otros datos importantes que puedan ser necesarios para su consulta posterior, con la intención de tener siempre claro qué se ha hecho y cuando. Obviamente, cada uno de los *mods* registrados en las fases anteriores pueden hacer log de sus operaciones (en un fichero independiente o en el mismo que *Apache 2*, según lo configuremos) o bien saltar directamente a esta fase si el módulo ha detectado algún error en la petición, para generar una página de error adecuada (*hook* ***Error page generation***).

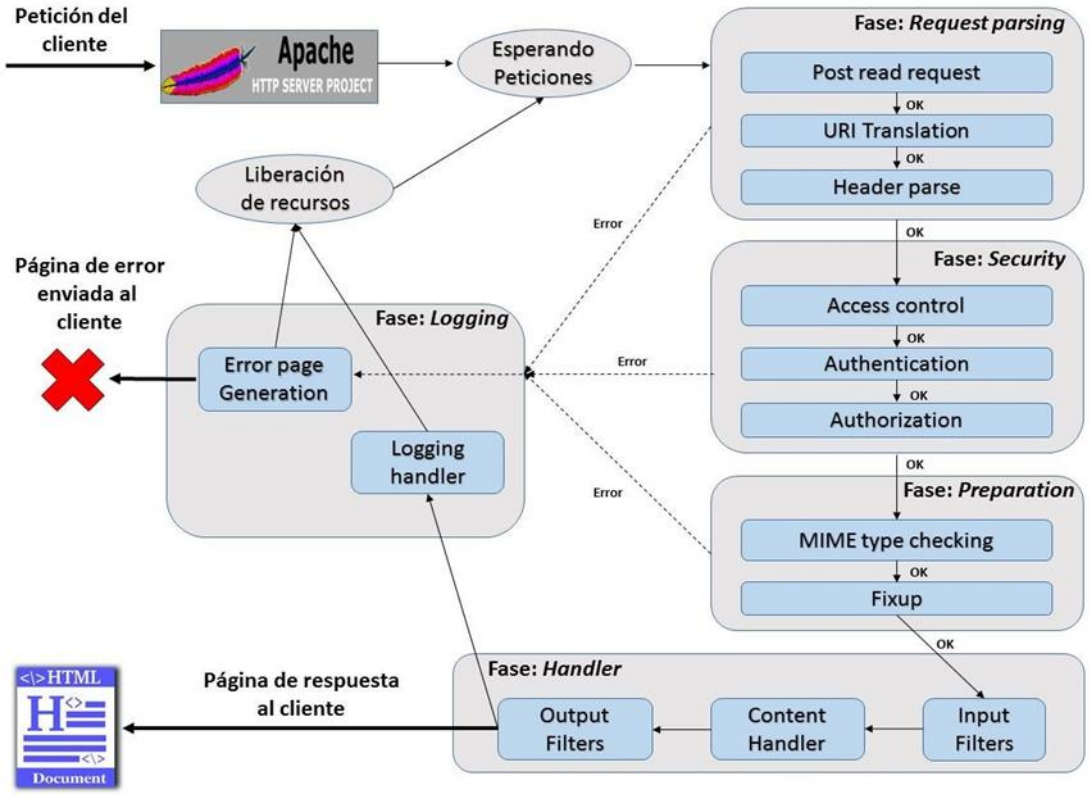


Figura 65. Cadena de procesamiento de peticiones de Apache 2 detallada

A modo de resumen, las fases de *Request parsing*, *Security* y *Preparation* son las denominadas **fases de proceso de metadatos**, mientras que la fase restante es la de **proceso de datos**. Esta última fase, a través de los filtros de entrada y de salida (que son implementados por determinados módulos), se consigue una implementación mucho más eficiente y flexible que una aproximación basada en que un solo módulo se ocupe de la generación de todo el contenido, al poder así reutilizar partes del procesamiento (los filtros) para cualquier generador de contenidos que usemos. Esta última fase es precisamente la que describiremos en el siguiente punto.

Generación de contenidos, filtros y tipos MIME

Un aspecto crucial a la hora de generar la respuesta a cualquier petición es **llamar al manejador de contenidos adecuado**. A la hora de elaborar la respuesta que verá el cliente, **solo se puede llamar a un único generador de contenidos** por tipo de recurso pedido. Por ejemplo, si pedimos una página `.php` al final habrá un único módulo encargado de procesar esta clase de páginas, que debe ser el encargado de elaborar el HTML de la respuesta a partir del código de la página pedida.

No obstante, los filtros descritos anteriormente son la clave para permitir que varios módulos puedan colaborar para elaborar la respuesta que sirva una determinada petición. Esto es posible porque en la fase donde se determina el tipo MIME del recurso pedido *Apache 2* puede tener asociado a cada tipo MIME un conjunto de **filtros** en un orden determinado, encargados de manipular los datos de la petición. Por tanto, cada tipo MIME no solo está asociado con un determinado manejador de contenidos, sino también con un conjunto de filtros en un orden dado, que puede ser diferente al de otros tipos MIME (diferente tanto al tipo de filtros que se usa como al orden en el que los usa). Los filtros asociados a un tipo MIME pueden estar implementados en otros módulos diferentes al que genera los contenidos propiamente dichos a partir de la petición hecha.

Se recuerda que todos los módulos están programados para "suscribirse" a los *hooks* disponibles que deseen y de esta forma incorporarse cuando se activan a la cadena de procesamiento de las peticiones donde ellos consideren oportuno. A la "cadena" de filtros asociada a un tipo MIME concreto (es una cadena porque la salida de un filtro será la entrada del siguiente) se la conoce como ***output filter chain***.

A nivel programático, todos los filtros implementan un interfaz común y son intercambiables en cuanto a su orden por ese motivo. Además, para optimizar el procesamiento de la información**, los filtros parten los datos de las peticiones en diferentes partes**, siendo estas partes las que usan todos los filtros para trabajar en lugar de que un filtro tenga que escribir toda la petición completa, una vez que la ha manipulado, en la entrada del siguiente para que este último pueda comenzar a trabajar. Una vez que un filtro trata una parte de los datos asignada le pasa dichos datos ya manipulados por él al siguiente, manteniendo el orden pero sin que se requiera procesar completamente la petición por un filtro N para que el N+1 entre en funcionamiento.

Los filtros de entrada o de salida pueden clasificarse además en las siguientes categorías en función de lo que hacen:

- **Filtros para modificar contenidos o recursos**: Ejemplos de este tipo de filtros son los *Server Side Includes*, `deflate` (compresión de los contenidos antes de enviarlos al cliente) o los que implementan lenguajes del servidor como PHP o .NET, que reciben los contenidos de la petición y los modifican. Estos son los que nos interesan principalmente como administradores y por tanto serán los que veremos en la siguiente sección, que estará dedicada a ver diferentes ejemplos de cómo **modificar la cadena de peticiones en varios puntos** de la misma gracias a un conjunto de módulos.
- **Filtros de protocolo y de codificación**: Los primeros implementan el **comportamiento de un determinado protocolo** (como HTTP), y se usan para soportar futuras revisiones de los protocolos de manera modular. Los segundos modifican la codificación de los datos de las peticiones o de sus respuestas para adaptarlos a las codificaciones soportadas por el cliente. No veremos ejemplos de los mimos, por estar fuera del ámbito de este libro.
- **Filtros de conexión y de red**: Los primeros se ocupan de **cómo establecer y liberar conexiones**, implementando los pasos necesarios para ello. Por ejemplo, la implementación de una conexión HTTPS. También pueden alterar el contenido si fuese necesario. El mejor ejemplo es la implementación del cifrado de la petición que describiremos en el bloque se seguridad. Finalmente, los filtros de red son los que interactúan con el SO para establecer conexiones de red. Permiten soportar protocolos más allá del TCP/IP si fuera necesario. No veremos ejemplos de los mimos, por estar fuera del ámbito de este libro.

Otros *hooks* predefinidos en *Apache 2*

Anteriormente hemos visto que los módulos pueden intervenir en la cadena de procesamiento de las peticiones a través de los *hooks*, es decir, puntos en los que los módulos que tenemos habilitados se registran para que, una vez que la petición pase por un punto determinado, este pueda manipularla de alguna forma. Aunque este esquema es muy potente y flexible, lo cierto es que sólo hemos visto una parte de los *hooks* existentes, concretamente aquellos que afectan al procesamiento de los datos de la petición, que son los que nos interesan desde el punto de vista de la administración.

No obstante, **existen más *hooks* que se definen en otros puntos**, de los cuales veremos una breve descripción a continuación ya que ayudan a entender mejor cómo funciona *Apache 2*. Concretamente, los *hooks* existentes pueden dividirse en tres categorías:

- Manejo y procesamiento de directivas de configuración
- Arranque, reinicio o finalización del servidor
- Recepción y procesamiento de peticiones HTTP (los segundos son los vistos anteriormente)

Hooks de manejo y procesamiento de directivas de configuración

Cada vez que arrancamos o reiniciamos *Apache 2* (por ejemplo para cambiar el conjunto de módulos habilitados, como vimos anteriormente), **el servidor lee y procesa los diferentes ficheros de configuración que tiene registrados**. Cada módulo habilitado puede tener su propio conjunto de directivas particulares, por lo que *Apache 2* llamará a las funciones definidas por ese módulo para tratar con sus directivas cada vez que encuentre una directiva perteneciente a un módulo concreto. Un módulo puede registrarse en los siguientes *hooks* de estas clases:

- ***Hooks* para crear una configuración a nivel de servidor**: Si un módulo tiene opciones de configuración a nivel de servidor (afectando a todo el servidor, no a una localización concreta) se registra en este *hook*. Se llama una vez para todo el servidor y también una vez por cada host virtual (concepto que veremos más adelante).
- ***Hooks* para crear una configuración por cada directorio**: Un módulo puede tener unas configuraciones independientes por ruta o directorio de diversas webs alojadas en el servidor. Si un módulo se registra en este *hook*, tendrá que llamarse al menos una vez por cada directorio que tenga directivas que sean de ese módulo.
- ***Hooks* para manejadores de comandos**: Cada módulo tiene una tabla de comandos, que establece una correspondencia entre una directiva definida por dicho módulo y la función que la implementa, procesando sus parámetros. También se usa para determinar en qué circunstancias se admite una directiva o cuando lo que establece se ve anulado por otra directiva más global.

En relación con lo anterior, hay que destacar que los ficheros de configuración de organizan de forma jerárquica, y hemos visto que se definen reglas que establecen en qué casos un parámetro más específico puede sobrescribir la configuración de otro de más nivel. Para esto se definen los siguientes *hooks*:

- ***Hooks* para gestionar la configuración del servidor**: Si, durante la creación de la configuración del servidor, algunos host virtuales queda con alguna opción sin establecer, los módulos registrados aquí permiten rellenar estas opciones con la configuración del servidor principal. Estos *hooks* se llaman una vez por cada host virtual.
- ***Hooks* para gestionar la configuración de los directorios**: Se usan para mezclar la configuración de los directorios con la del propio servidor. Si existen diferencias entre ambas, en el sentido de que una directiva establezca cosas distintas a ambos niveles, estos manejadores son los

encargados de calcular cual es el valor de la directiva a partir de ambos datos. Debido a ello, estos manejadores se llaman una vez al arrancar el servidor y otra por cada petición procesada por el mismo. En este punto es donde los ficheros `.htaccess` se toman en consideración.

Debido a esto, está claro por qué entonces cada vez que hacemos un cambio de configuración en los ficheros `.conf` del servidor o habilitamos/deshabilitamos un módulo es necesario el reinicio de *Apache 2*, ya que de no hacerlo no se pasaría por el proceso de leer la configuración modificada de nuevo.

Hooks de arranque, reinicio o apagado del servidor

Como vimos anteriormente en la descripción de los MPMs, *Apache 2* es un programa multitarea. Salvo en la inicialización, donde una sola tarea inicializa el servidor y lee la configuración, el servidor atiende peticiones mediante el uso de tareas concurrentes. En función del MPM usado, pueden existir puntos donde ciertos módulos requieran una inicialización adicional por cada tarea ejecutada, además de cuando dicha tarea finalice, con la intención de liberar sus recursos. Los *hooks* incluidos en esta categoría son:

- ***pre_config***: Este *hook* se ejecuta una vez leída la configuración de los ficheros pero antes de que sea procesada tras un arranque o reinicio.
- ***open_logs***: *Hook* usado en el caso de que un módulo tenga que abrir ficheros de log.
- ***post_config***: Ejecutado por el servidor tras leer y procesar la configuración en un arranque o reinicio del servidor.
- ***pre_mpm***: Ejecutado antes de lanzar las tareas que sirven peticiones, es decir, antes de ejecutar la política de tareas del MPM que *Apache 2* va a usar.
- ***child_init***: *Hook* que se llama cada vez que se crea un proceso hijo, usado para hacer operaciones justo antes de empezar a servir peticiones.

Hooks de recepción y procesamiento de peticiones HTTP

Finalmente, aunque hemos descrito en detalle los *hooks* disponibles para los módulos a la hora de procesar peticiones HTTP, es conveniente destacar que antes de que éstos se ejecuten hay una serie de *hooks* adicionales que se ejecutan: los de la recepción de la petición HTTP. Éstos son:

- ***create connection***: Se ejecuta cuando se crea una conexión TCP.
- ***pre connection***: Permite preparar el servidor antes del procesamiento de las peticiones. Puede usarse, por ejemplo, para registrar filtros de entrada.
- ***process connection***: Ejecuta el protocolo correcto (por defecto HTTP)
- ***create request***: Prepara los recursos para leer la petición y enviar la respuesta

Y, tras este último, a continuación vendrían los ***hooks* de procesamiento de la petición** vistos anteriormente, comenzando por el paso de ***post read request***.

Content Handlers definidos por Apache 2

Como hemos visto, un *content handler* permite a *Apache 2* realizar una serie de acciones para **generar un contenido de respuesta cuando se solicita un fichero de un tipo concreto**. Como hemos dicho, los ficheros que *Apache 2* puede servir tienen un *handler* que depende del tipo (extensión) del fichero. Por defecto normalmente este *handler* implícito simplemente sirve el fichero tal cual, pero algunos tipos de archivos requieren un tratamiento especial, y aquí entran en juego los *handlers* que el usuario cree explícitamente. Un *handler* explícito se puede especificar de acuerdo con el tipo MIME del fichero o bien a su localización

dentro del sistema de ficheros (en cuyo caso se aplica a los ficheros de la localización, sin tener en cuenta el tipo).

Los *content handlers* disponibles pueden formar parte de la distribución estándar propio servidor o bien ser incluidos como parte de un módulo que hayamos instalado. No obstante, una tercera vía para añadirlos es a través de una directiva `Action` (http://httpd.apache.org/docs/current/mod/mod_actions.html). Esta directiva es parte del módulo `mod_action`, que nos permite especificar al servidor qué tipo de manejador ejecutar para un fichero de un tipo MIME concreto de manera manual, en lugar de usar la asociación que dicho tipo MIME pudiera tener definida previamente.

A continuación se listan los ***handlers* que vienen incluidos en la distribución estándar de la última versión de *Apache 2*** disponible en estos momentos, junto con un enlace al módulo que los define para consultar más información de estos en caso necesario:

- ***default-handler***: Envía el fichero usando el *handler* por defecto para tratar contenido estático (definido en `core`, http://httpd.apache.org/docs/current/en/mod/core.html).
- ***send-as-is***: Envía el fichero con cabeceras HTTP tal y como le ha llegado al servidor (`mod_asis`, http://httpd.apache.org/docs/current/en/mod/mod_asis.html).
- ***cgi-script***: Trata el fichero como un *script CGI* (`mod_cgi`, http://httpd.apache.org/docs/current/en/mod/mod_cgi.html).
- ***imap-file***: Trata el fichero como un mapa de imágenes (`mod_imagemap`, http://httpd.apache.org/docs/current/en/mod/mod_imagemap.html).
- ***server-info***: Extrae la información de configuración del servidor (`mod_info`, http://httpd.apache.org/docs/current/en/mod/mod_info.html).
- ***server-status***: Extrae el informe de estado del servidor (`mod_status`, http://httpd.apache.org/docs/current/en/mod/mod_status.html). Lo vimos anteriormente.
- ***type-map***: Trata el fichero como una correspondencia de tipos para la negociación de contenidos (`mod_negotiation`, http://httpd.apache.org/docs/current/en/mod/mod_negotiation.html). Veremos un ejemplo más adelante.

Filtros de entrada y salida predefinidos por Apache 2

Como hemos visto al describir el ciclo de procesamiento de peticiones de *Apache 2*, un filtro es un procesamiento concreto que se aplica a los datos que el servidor recibe o envía. Los datos enviados por los clientes al servidor son procesados por **filtros de entrada**, mientras que los datos enviados por el servidor al cliente, una vez generada la respuesta, se procesan por los **filtros de salida**. A los datos se les pueden aplicar varios filtros, y el orden en que se aplica cada filtro se puede especificar de forma explícita, lo que puede dar lugar a procesamientos de la información realmente complejos en caso necesario.

Actualmente, **se incluyen con la distribución de *Apache 2*** los siguientes filtros seleccionables por el usuario:

- `mod_include`: Permite el uso de *Server-Side Includes*. Veremos un ejemplo.
- `mod_deflate`: Permite la compresión de las peticiones. Veremos un ejemplo.
- `mod_ext_filter`: permite definir programas externos como filtros. El siguiente enlace proporciona más información acerca de cómo programarlos.
    - **Más información**: http://httpd.apache.org/docs/current/mod/mod_ext_filter.html
- `mod_filter`: Permite la configuración de filtros "inteligente", permitiendo que la cadena de filtros vista anteriormente pueda configurarse dinámicamente en función del contexto. Por

ejemplo, *Apache 2* se puede configurar para procesar distintos `Content-type` mediante diferentes cadenas de filtros, incluso aunque el `Content-type` no sea conocido de antemano.

De esta forma, si vamos a servir una página HTML con imágenes, podremos establecer de forma dinámica qué filtros procesarán el HTML y qué filtros procesarán las imágenes de forma separada. Cualquier filtro se podrá introducir en la cadena de filtros y ejecutarse en cualquier momento de la ejecución. En definitiva, la principal ventaja que aporta es dotar de aún más flexibilidad a la cadena de filtros que el servidor web implementa. **Más información**: http://httpd.apache.org/docs/current/mod/mod_filter.html

- **`mod_charset_lite`**: Permite al servidor cambiar el conjunto de caracteres usado en las respuestas antes de enviárselas al cliente.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_charset_lite.html
- **`mod_reflector`**: Permite enviar la petición al cliente de nuevo, pero tras pasar por toda la cadena de filtros de salida. A efectos prácticos, esto permite convertir cualquier filtro de salida en un servicio HTTP. Permite implementar algo similar a un servicio web, usando para ello los filtros de salida de *Apache 2*.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_reflector.html
- **`mod_buffer`**: Permite crear buffers para los filtros de entrada o de salida, lo que en ciertas circunstancias puede mejorar el rendimiento del servidor.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_buffer.html
- **`mod_data`**: Este módulo convierte la respuesta enviada al cliente en una URI de datos RFC2397 (http://tools.ietf.org/html/rfc2397), adecuadas para enviar conjuntos de datos pequeños de cualquier tipo en una URI.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_data.html
- **`mod_ratelimit`**: Limita el ancho de banda disponible para los clientes, simulando una velocidad máxima de conexión en Kb/s.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_ratelimit.html
- **`mod_reqtimeout`**: Permite definir un tiempo y un ratio de datos máximo para mantener una conexión activa. De no alcanzarse el servidor lo notificaría (*Timeout*) al cliente.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_reqtimeout.html
- **`mod_request`**: Filtro que permite que el cuerpo de las peticiones HTTP no se elimine en determinadas peticiones a ficheros, ya que el comportamiento por defecto de *Apache 2* es eliminarlas en esos casos.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_request.html
- **`mod_sed`**: Permite modificar el contenido (no las cabeceras HTTP) de las peticiones usando los mismos comandos que el programa `sed` (http://www.gnu.org/software/sed/manual/sed.txt) del SO *Solaris*. Puede actuar tanto como filtro de entrada como de salida.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_sed.html
- **`mod_substitute`**: Permite hacer sustituciones en el cuerpo de las respuestas HTTP, reemplazando cadenas de texto o bien mediante el uso de expresiones regulares.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_substitute.html
- **`mod_xml2enc`**: Permite, entre otras cosas, detectar automáticamente la codificación de los datos y de esta forma que sean procesados correctamente por librerías que procesen XML, convirtiendo los datos a Unicode (UTF-8) cuando sea necesario y modificando cabeceras HTTP cuando convenga hacerlo.
  - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_xml2enc.html

- **`mod_proxy_html`**: Es un filtro de salida que permite reescribir enlaces HTML cuando hay un proxy entre el cliente y el servidor, de manera que estos enlaces sigan funcionando.
    - **Más información**: http://httpd.apache.org/docs/current/en/mod/mod_proxy_html.html

Además de estos, *Apache 2* tiene algunos filtros más para la manipulación de bytes. También debe tenerse en cuenta que existen filtros de terceros (ver http://modules.apache.org) cuya funcionalidad es muy diversa, adaptados a casi cualquier necesidad que podamos necesitar:

- Cualquier tipo de procesamiento y transformación de HTML y XML
- Subida de ficheros
- Procesamiento de imágenes
- Mecanismos de seguridad para proteger aplicaciones potencialmente vulnerables
- Búsqueda y reemplazo de texto
- …

Uso de filtros

Existen dos formas para usar filtros: la **forma simple** y la **dinámica**. Se recomienda usar una de ellas e intentar no mezclar en la medida de lo posible ambas, porque podría dar lugar a errores difíciles de depurar. Los filtros de entrada solo pueden configurarse de forma simple.

En la siguiente sección, donde veremos varios ejemplos de uso de filtros, usaremos la forma simple, que es más que suficiente cuando lo que necesitamos es una cadena de filtros estática conocida. Para usar la **forma simple**, basta con indicar en la localización pertinente alguna de las siguientes directivas (descritas en detalle en http://httpd.apache.org/docs/current/en/mod/core.html las dos primeras y en http://httpd.apache.org/docs/current/en/mod/mod_mime.html el resto):

- **`SetInputFilter`**, **`SetOutputFilter`**: Sustituye los filtros de entrada y de salida por los indicados en estas directivas (separados por un espacio en blanco)
- **`AddInputFilter`**, **`AddOutputFilter`**: Permite asociar extensiones de fichero concretas a filtros de entrada y de salida que procesarán estos ficheros, respectivamente.
- **`RemoveInputFilter`**, **`RemoveOutputFilter`**: Hacen la operación contraria a las dos anteriores.

La **forma dinámica** consiste en usar `mod_filter`, como mencionamos anteriormente. Las directivas que controlan esta forma de uso son (http://httpd.apache.org/docs/current/en/mod/mod_filter.html):

- **`FilterChain`**: Permite configurar toda la cadena de filtros indicando nombres de filtros precedidos por un carácter que indica qué hacer con ellos (añadirlos, quitarlos, …)
- **`FilterDeclare`**: Permite declarar un filtro "inteligente" de salida, especificando un nombre y un tipo (opcional) de entre los vistos anteriormente (de contenido, de protocolo…)
- **`FilterProvider`**: Registra un proveedor para un filtro "inteligente" declarado con la directiva anterior. El proveedor será llamado solo si la expresión booleana (http://httpd.apache.org/docs/current/expr.html) proporcionada como parámetro devuelve cierto.
- **`FilterProtocol`**: Controla qué tipo de cambios puede hacer un filtro a los datos.

El siguiente ejemplo ilustra el uso de estas directivas. En él, podemos ver como se declara un filtro dinámico (**`SUSTITUCION`**), cuyo proveedor (el filtro que realmente hará alguna acción útil) es **`mod_substitute`** (**`SUBSTITUTE`**, http://httpd.apache.org/docs/current/mod/mod_substitute.html). Este filtro solo se ejecutará en la cadena de filtros si alguna de las expresiones que acompañan las

declaraciones de `FilterProvider` se evalúan a cierto (ver los comentarios de cada ejemplo). Su objetivo es extremadamente simple, y es que allí donde en el cuerpo de la respuesta aparezca "*index*" aparezca en su lugar "*Indice*". Todo ello se hará si se accede a un fichero que esté dentro de la carpeta `/var/www/html/subs`, o su localización equivalente `/subs`.

```
…
#Los filtros dinámicos pueden declararse fuera o dentro de un contenedor
#En nuestro ejemplo se declaran fuera del contenedor que los usa, pero serviría
#también dentro de estos contenedores Directory o Location, que se refieren ambos a la
#misma localización
#<Directory "/var/www/html/subs">
#<Location "/subs/"> FilterDeclare SUSTITUCION
#Se aplicaría sobre todos los ficheros que no fueran imágenes, audio o video
#FilterProvider SUSTITUCION SUBSTITUTE "%{CONTENT_TYPE} !~ m#^(image|audio|video)#i"
#Se aplicaría sobre todos los ficheros cuyo tipo de contenido fuese html o texto plano
#FilterProvider SUSTITUCION SUBSTITUTE "%{CONTENT_TYPE} =~ m#^.*(html|plain).*#i"
#Se aplicaría sobre todos los ficheros cuya extensión fuese .htm
FilterProvider SUSTITUCION SUBSTITUTE "%{REQUEST_URI} =~ m#^.*htm#i"
#El filtro puede cambiar los datos de la petición y su resultado no es cacheable
FilterProtocol SUSTITUCION SUBSTITUTE change=yes;cache=no
#Se puede aplicar a ambos tipos de contenedores
#<Location "/subs/">
<Directory "/var/www/html/subs">
      FilterChain SUSTITUCION
      Substitute s/index/indice/ni
</Directory>
#</Location>
…
```

Para que este ejemplo funcione correctamente debe desactivarse `mod_deflate`, que está activo por defecto de manera simple en la mayoría de las instalaciones. `mod_substitute` no funcionará si existe `mod_deflate` activado de forma simple, aunque podría intentar usarse conjuntamente con `mod_substitute` de forma dinámica siguiendo las instrucciones de la siguiente dirección: http://httpd.apache.org/docs/2.4/mod/mod_filter.html. No obstante, este es un buen ejemplo de por qué no debe mezclarse la forma simple y la dinámica de usar filtros sobre los mismos elementos.

Modificación de la cadena de procesamiento de peticiones

A continuación vamos a ver ejemplos de cómo manipular la cadena de procesamiento de peticiones HTTP usando módulos predefinidos por *Apache 2* o bien instalando algún módulo nuevo. Esto permite ver diferentes acciones que permiten manipular partes de la cadena de procesamiento de peticiones descrita en varios puntos, desde el punto de vista de la administración.

Tipos MIME (*mod_mime*)

Como mencionamos al hablar de la fase de ***MIME type checking*** de la cadena de procesamiento de peticiones, la función de este módulo (http://httpd.apache.org/docs/current/mod/mod_mime.html) es asignar metadatos a los recursos pedidos, que indiquen el tipo de contenido que podemos esperar encontrar. Esto se hace mediante un mapa que asigna a las extensiones de los ficheros solicitados en las URIs valores para diversas clases de metadatos. Concretamente, la extensión de los ficheros solicitados por un cliente puede determinar:

- El tipo de medio que es el contenido (texto, imágenes, video,…): `Content-Type`
- El conjunto de caracteres usado en el mismo
- Su idioma: `Content-Language`
  Cómo se codifica el contenido: `Content-Encoding`

Esta información se adjunta en las peticiones HTTP a los datos en sí, y se usa para seleccionar cómo se procesa el contenido, es decir, como se "negocia" la manera de servir el contenido cuando hay varias posibilidades. Esto es función específica del *mod* `mod_negotiation` (http://httpd.apache.org/docs/current/mod/mod_negotiation.html)

Las directivas `AddCharset` (añadir conjunto de caracteres), `AddEncoding` (añadir codificación del contenido), `AddLanguage` (añadir idioma del contenido) y `AddType` (añadir tipo MIME del contenido) se deben usar para crear asociaciones entre extensiones de ficheros y sus metadatos (cada una de ellas aplicada al tipo de metadato correspondiente puesto entre paréntesis). Finalmente, la directiva `TypesConfig` se puede usar además para establecer un fichero del que puedan leerse mapeos entre extensiones de fichero y tipos de contenido.

Apache 2 está configurado para servir todos los tipos *MIME* comunes, los cuales están registrados por el propio SO y se leen del fichero `/etc/mime.types` cuando el servidor arranca (en el fichero de configuración de `mod_mime` puede verse al principio la directiva `TypesConfig /etc/mime.types`). El contenido de este fichero es como el que aparece en la siguiente imagen:

Figura 66. Fichero /etc/mime.types

Toda la información de este módulo se encuentra en: `/etc/apache2/mods-enabled/mime.conf` y se recomienda consultar su documentación oficial para ver todas las opciones disponibles. Aquí destacaremos **algunos de los contenidos más importantes** de este fichero:

- Directivas `AddType`, que asocia una extensión con su tipo de contenido (Ej.: `AddType image/gif .gif, AddType application/x-java-jnlp-file .jnlp`). Esta es la mejor forma de añadir nuevos tipos *MIME* a *Apache 2*. Cuando se instalan determinados módulos de *Apache 2* es posible que dicha instalación añada nuevos tipos automáticamente.
- Directivas `RemoveType`, que tienen la función contraria. Para hacer efectivo un cambio cualquiera (`AddType`, `RemoveType`) en `mime.conf` es necesario reiniciar *Apache 2*, como hemos indicado anteriormente. Las directivas `RemoveType` se procesan después de las `AddType`, teniendo pues prioridad.
- Directivas `AddLanguage`, que definen los idiomas soportados por el servidor.
- Definición de **filtros** (*Filters*) y **manejadores** (*Handlers*) asociados a determinadas extensiones de fichero y que son los encargados de procesar los contenidos de determinados esos ficheros. Esta es la forma de lograr asociar los módulos adecuados para procesar determinadas clases de ficheros, y es la implementación del ***output filter chain*** y del manejador de contenidos descrita anteriormente. No obstante, normalmente, salvo que desarrollemos nosotros mismos un módulo o bien decidamos instalar un módulo de terceros desde el código fuente manualmente, la instalación de un módulo ya hace los cambios oportunos a `mime.conf` para que los ficheros de un determinado tipo sean tratados por el módulo correcto.

Eliminación de un tipo MIME

Como hemos dicho los tipos MIME indican al navegador cómo procesa ciertos contenidos asignándole metadatos. Un ejemplo claro de lo que ocurre si no tuviéramos bien definido un tipo MIME o simplemente no la tuviéramos, es que el contenido podría ser interpretado de forma errónea, obteniendo resultados no esperados o incluso problemas de seguridad. A modo de ejemplo, vamos a eliminar el tipo *MIME* de los ficheros `.shtml` (vistos posteriormente), para ver si el navegador los sigue interpretando como *HTML* (`text/html`) o no:

```
  GNU nano 2.2.6            Archivo: mime.conf

        #
        # For type maps (negotiated resources):
        # (This is enabled by default to allow the Apache "It Worked" page
        #  to be distributed in multiple languages.)
        #
        AddHandler type-map var

        #
        # Filters allow you to process content before it is sent to the client.
        #
        # To parse .shtml files for server-side includes (SSI):
        # (You will also need to add "Includes" to the "Options" directive.)
        #
#       AddType text/html .shtml
        RemoveType .shtml
#       AddOutputFilter INCLUDES .shtml

</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Figura 67. Eliminación explícita de un tipo MIME

Al eliminar el tipo MIME de estos ficheros y tener más prioridad el `RemoveType` que el `AddType`, el navegador pasa a interpretarlos tal cual, en modo texto plano:

192.168.56.2/ssi/index.shtml

```
<html>
<head>
Empty page
</head>
<body>

<p>Welcome</p>

<p>
        Wednesday, 08-Jul-2015 08:16:05 CEST
</p>
</body>
</html>
```

Figura 68. Resultado de eliminar el tipo MIME de un tipo de fichero concreto

Ejemplo de uso de *content handlers*: Negociación de contenidos

En este ejemplo vamos a ver en el uso de `mod_negotiation` para lograr un sistema de internacionalización automático de las webs alojadas que no requiera programación, es decir, independiente del lenguaje del servidor usado. Se basa en la capacidad de *Apache 2* de seleccionar una página u otra en función del valor de la cabecera `Content-Language` que nos envíe el cliente en función de su configuración. Este es uno de los posibles usos de este *mod*, aunque sus funcionalidades pueden usarse para otras tareas relacionadas

con qué versión de un recurso existente debe servirse. (**Más información**: http://httpd.apache.org/docs/current/content-negotiation.html).

El primer paso es, por supuesto, asegurarnos de que `mod_negotiation` está activo:

```
operario@servidorapache:/var/www/html/negotiation$ sudo a2enmod negotiation
Enabling module negotiation.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/var/www/html/negotiation$ sudo service apache2 restart
 * Restarting web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress th
is message
                                                                        [ OK ]
operario@servidorapache:/var/www/html/negotiation$ _
```

Figura 69. Activación de mod_negotiation

Hecho esto, desarrollaremos dos versiones de una misma página de ejemplo `index.html`, una en español y otra en inglés. Para distinguirlas, asignaremos una doble extensión de fichero a cada una de ellas, indicando el idioma en el que están escritas, tal y como se ve en la imagen siguiente. Posteriormente crearemos un fichero que se llamará de igual forma que la página a servir, pero con una doble extensión `.var`. Nótese que esta extensión de fichero es la asociada al manejador de contenidos `type-map` (ver imagen de la sección "***Eliminación de tipos MIME***") que es la que hemos dicho que usa el módulo `mod_negotiation` para implementar su funcionalidad.

```
operario@servidorapache:/var/www/html/negotiation$ ls
index.html.en  index.html.es  index.html.var
operario@servidorapache:/var/www/html/negotiation$ ls -la
total 20
```

Figura 70. Preparación de recursos para mod_negotiation

Para este ejemplo no es necesario preparar páginas web complejas, sino una página con un mínimo de contenido traducido a ambos idiomas:

```
operario@servidorapache:/var/www/html/negotiation$ ls
index.html.en  index.html.es  index.html.var
operario@servidorapache:/var/www/html/negotiation$ ls -la
total 20
drwxr-xr-x  2 root root 4096 jul  8 07:39 .
drwxr-xr-x 15 root root 4096 jul  8 07:34 ..
-rw-r--r--  1 root root   71 jul  8 07:37 index.html.en
-rw-r--r--  1 root root   76 jul  8 07:38 index.html.es
-rw-r--r--  1 root root  147 jul  8 07:42 index.html.var
operario@servidorapache:/var/www/html/negotiation$ cat index.html.en
<html>
<head>
Empty page
</head>
<body>
<p>Welcome</p>
</body>
</html>
operario@servidorapache:/var/www/html/negotiation$ cat index.html.es
<html>
<head>
Pagina vacia
</head>
<body>
<p>Bienvenido</p>
</body>
</html>
operario@servidorapache:/var/www/html/negotiation$
```

Figura 71. Páginas web en dos idiomas a servir

Lo más importante es el contenido del fichero `.var`, que es el que usará `mod_negotiation` para realizar su función. En la siguiente imagen podemos observar cómo le decimos al *mod* que para la URI `index.html`, si recibe una petición en el que la cabecera `Content-Language` valga `en` y la cabecera `Content-Type` valga `text/html` entonces sirva el documento `index.html.en` (y lo mismo en el caso de español):

```
operario@servidorapache:/var/www/html/negotiation$ cat index.html.var
URI: index.html

Content-language: en
Content-type: text/html
URI: index.html.en

Content-language: es
Content-type: text/html
URI: index.html.es
```

Figura 72. Contenido del fichero .var usado por mod_negotiation

Y finalmente, añadimos un manejador de contenidos con la directiva `AddHandler` a la ruta específica que tiene los tres ficheros que hemos visto. En la siguiente imagen podemos ver como asignamos a los ficheros `.var` el manejador de contenidos `type-map`:

```
<Directory /var/www/html/negotiation>
        AddHandler type-map .var
</Directory>
```

Figura 73. Activando mod_negotiation para una ruta concreta del servidor

Aunque obviamente el `Content-type` es HTML, no sabemos cuál va a ser a priori el valor de `ContentLanguage`. Esta cabecera la establece el navegador que estamos usando, en función del idioma que el usuario haya establecido como primario para dicho navegador. En el siguiente ejemplo vemos que el navegador se encuentra en español y que la petición del fichero `.var` al final hace que se sirva el fichero HTML en español automáticamente:

Figura 74. mod_negotiation sirviendo el contenido en español

¿Qué ocurrirá si cambiáramos el idioma primario del navegador por inglés? En ese caso el navegador enviaría otro valor en la cabecera `Content-Language`, y si repetimos la petición anterior el resultado debería cambiar.

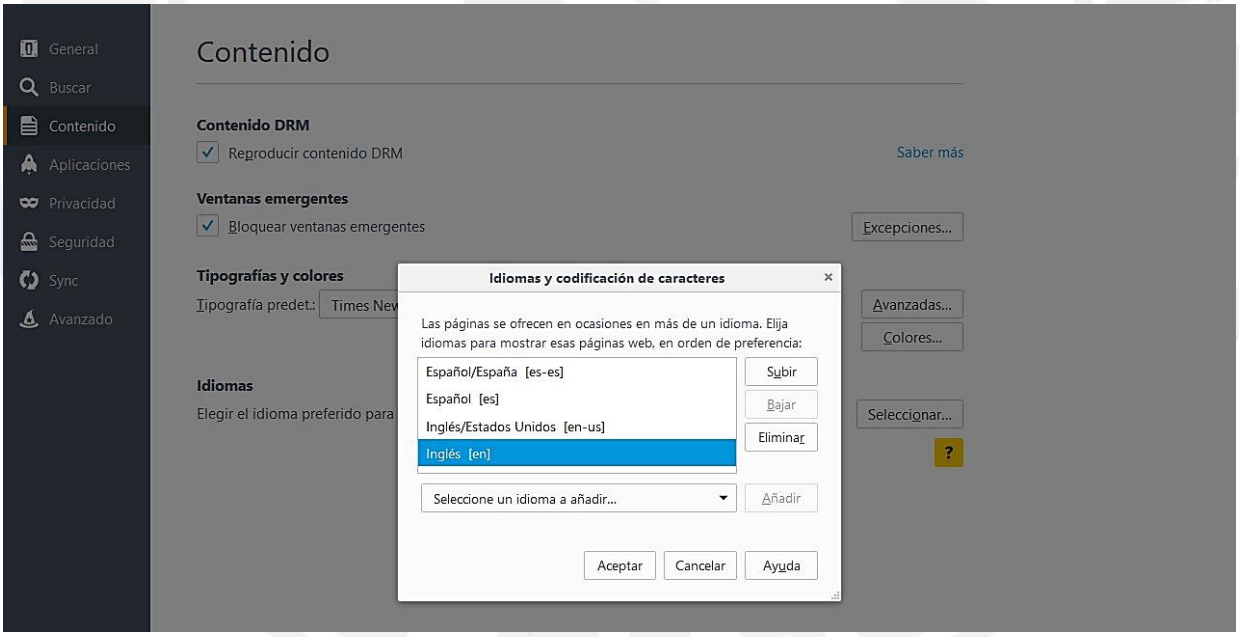


Figura 75. Subiendo "Inglés" al primer puesto de la lista de idiomas a usar por el navegador a la hora de seleccionar los contenidos

Como efectivamente ocurre:

Figura 76. mod_negotiation sirviendo contenido en inglés

No obstante, el hacer una petición a un fichero `.var` en lugar de a una página `.html` **no es muy natural**. Lo ideal sería solicitar el fichero `.html` y que el propio *mod* decidiese cuál de las versiones existentes es la más adecuada, sin mostrar el detalle de implementación del fichero `.var`. Para ello, `mod_negotiation` define las ***Multiviews*** que, una vez activas para la localización donde estamos usando el *mod,* permiten hacer precisamente lo que hemos descrito. En la siguiente imagen vemos como se han activado (`+Multiviews`) las *Multiviews* para la ruta donde tenemos el contenido internacionalizado y cómo hemos configurado que se usen los *content handlers* definidos (en nuestro caso únicamente el `type-map`) para determinar el fichero final a servir.

Una de las opciones que tiene `MultiviewsMatch` es `Any`. No se recomienda el uso de esta opción por motivos de seguridad, ya que puede permitir que se sirvan ficheros `.old` o `.bak` que tengan versiones anteriores de una determinada web, lo que puede suponer un riesgo para la seguridad del sistema.

```
  GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf


<Directory /var/www/html/env2>
        <RequireAll>
                Require env ACCESO_ESPECIAL
                Require ip 192.168.56.1
        </RequireAll>
</Directory>

<Directory /var/www/html/otherenv>
        <RequireAll>
                Require env ACCESO_ESPECIAL
                Require ip 192.168.56.1
        </RequireAll>
</Directory>

<Directory /var/www/html/negotiation>
        Options +Multiviews
        AddHandler type-map .var
        MultiviewsMatch Handlers
</Directory>
```

Figura 77. Activando las Multiviews

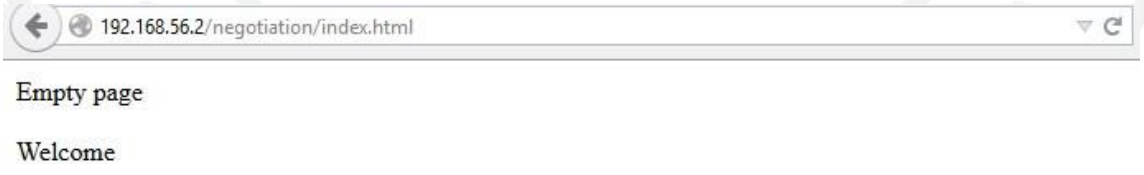


Figura 78. Sirviendo automáticamente el contenido más adecuado gracias a las Multiviews

Ejemplo de creación de un *content handler*: Instalación de *mod_python*

Anteriormente hemos dicho que los manejadores de contenidos pueden provenir **de dos fuentes**: los que vienen **por defecto** con las instalación del servidor (como el que usamos en el ejemplo anterior) o bien los que instalamos **con módulos de terceros**. Este ejemplo trata precisamente de este segundo caso: instalar un manejador de contenidos que permita la ejecución de ficheros que tengan código escrito en el lenguaje *Python* (ficheros `.py`). Sin la instalación de este *mod*, los ficheros `.py` serán interpretados **como ficheros**

de texto plano o bien no serán servidos (dependiendo de la configuración de *Apache 2*), en lugar de procesarse para generar un HTML de respuesta adecuado a su contenido.

Para hacer este ejemplo, lo primero es instalar el módulo de terceros `mod_python`, que registrará en los *hooks* oportunos todas las funciones necesarias para procesar ficheros *Python* automáticamente:

```
operario@servidorapache:/var/www/html/python$ sudo apt-get install libapache2-mo
d-python
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Los paquetes indicados a continuación se instalaron de forma automática y ya no
son necesarios.
  keyutils libecryptfs0 libnspr4 libnss3 libnss3-1d libnss3-nssdb
  linux-headers-3.13.0-54 linux-headers-3.13.0-54-generic
  linux-headers-3.16.0-30 linux-headers-3.16.0-30-generic
  linux-image-3.16.0-30-generic linux-image-extra-3.16.0-30-generic
Use 'apt-get autoremove' to remove them.
Paquetes sugeridos:
  libapache2-mod-python-doc
Se instalarán los siguientes paquetes NUEVOS:
  libapache2-mod-python
0 actualizados, 1 se instalarán, 0 para eliminar y 0 no actualizados.
Necesito descargar 106 kB de archivos.
Se utilizarán 475 kB de espacio de disco adicional después de esta operación.
Des:1 http://es.archive.ubuntu.com/ubuntu/ trusty/main libapache2-mod-python amd
64 3.3.1-11ubuntu2 [106 kB]
```

Figura 79. Instalación de mod_python

Antes de seguir, conviene comprobar que el *mod* está funcionando adecuadamente con un test sencillo. Para ello, creamos un programa *Python* muy simple llamado `prueba.py` que reciba los contenidos de la petición en el objeto `req` de la función `handler` (nombre predefinido para la función que implementa la gestión de los contenidos) y devuelva una simple cadena de texto:

```
 GNU nano 2.2.6            Archivo: prueba.py

from mod_python import apache

def handler(req):
        req.content_type = 'text/plain'
        req.write("mod_python funciona!")
        return apache.OK
```

Figura 80. Fichero .py de prueba

Ahora hacemos simplemente que cualquier petición de un fichero `.py` en una ruta determinada sea procesada por el módulo `mod_python`, y que además use el programa `prueba.py` para generar el documento de respuesta. Nótese que la última opción es solo para que el *mod* nos muestre información de los posibles errores extendida, y por ese motivo solo tiene sentido en una máquina de desarrollo y no en una de producción.

```
 GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf          Modificado

        Options +Includes
        AddOutputFilter INCLUDES .shtml
        XBitHack on
</Directory>


<Directory /var/www/html/python>
        AddHandler mod_python .py
        PythonHandler prueba
        PythonDebug On
</Directory>
```

Figura 81. Configuración de mod_python para una ruta concreta

Solo nos resta probar a acceder al fichero `.py` y comprobar que el *content handler* efectivamente se ejecuta:

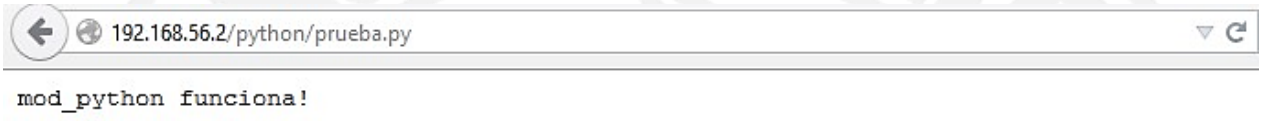


Figura 82. Prueba de funcionamiento de mod_python

No obstante, este ejemplo no es muy útil ya que cualquier acceso a cualquier fichero `.py` dentro de este directorio siempre devolvería la ejecución de la función `handler` del fichero visto. Un ejemplo más avanzado consistiría en que al acceder a un fichero `.py` se leyese una página web existente y se devolviese la misma modificada de alguna forma mediante código *Python*. En el siguiente ejemplo cargamos una página HTML fija, pero podemos usar variables de entorno y otras partes de las cabeceras de las peticiones para decidir qué fichero cargamos para modificarlo. El ejemplo concreto consiste en modificar la página cargada para añadir una fecha y hora a la página servida y para ello crearemos el fichero `time_handler.py`:

```
 GNU nano 2.2.6          Archivo: time_handler.py                 Modificado



from mod_python import apache
import datetime

def handler(req):
    lines = open('/var/www/html/python/pagina.html', 'r').readlines()
    req.content_type = 'text/html'

    for line in lines:
        if "</body>" in line:
            req.write("<p>Esto lo ha escrito Python</p>")
            str_time = datetime.datetime.now().time().isoformat()
            req.write("<p>"+str_time+"</p>")
            pass

        req.write(line)

    return (apache.OK)
```

Figura 83. Programa Python que añade fecha y hora a una página HTML cargada

Para que funcione simplemente tenemos que cambiar el programa *Python* usado para generar los contenidos en la localización anterior:

```
 GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf

        Options +Includes
        AddOutputFilter INCLUDES .shtml
        XBitHack on
</Directory>


<Directory /var/www/html/python>
        AddHandler mod_python .py
        PythonHandler time_handler
        PythonDebug On
</Directory>
```

Figura 84. Uso del programa Python time handler

Y finalmente, vemos cómo funciona:

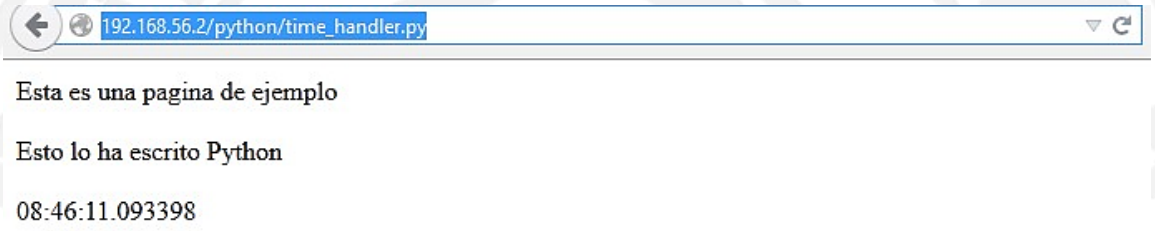


Figura 85. Ejecución del programa Python time_handler

Habilitar páginas .psp (*Python Script Page*)

Para finalizar con este ejemplo vamos a ver como permitir el uso de páginas `.psp`, que permiten introducir código *Python* en cualquier parte de estas entre secciones `<%` y `%>`. El proceso para soportarlas es muy parecido al anterior, solo que tenemos que activar, en la localización donde vayamos a servir estas páginas (en este caso el directorio `/python2` del directorio donde se sirven por defecto las páginas web, aunque en instalaciones más modernas solo podremos usar ***Python 3*** para crear las páginas), `mod_python` sobre la extensión `.psp` e indicar que el código que se va a usar para atender las peticiones se encuentra en `mod_python.psp`, fichero que forma parte de la instalación de `mod_python`.

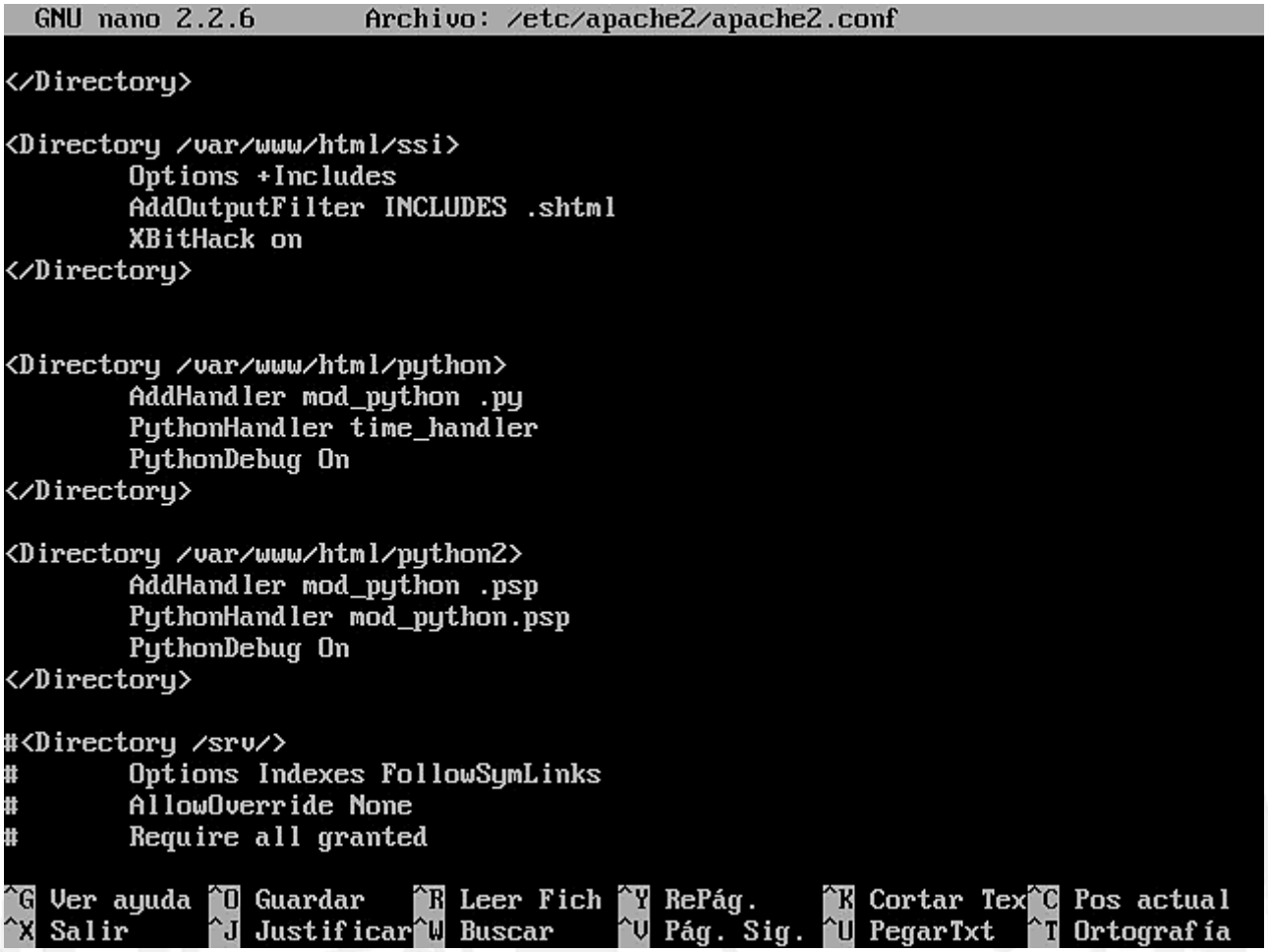


Figura 86. Activación del soporte de páginas .psp

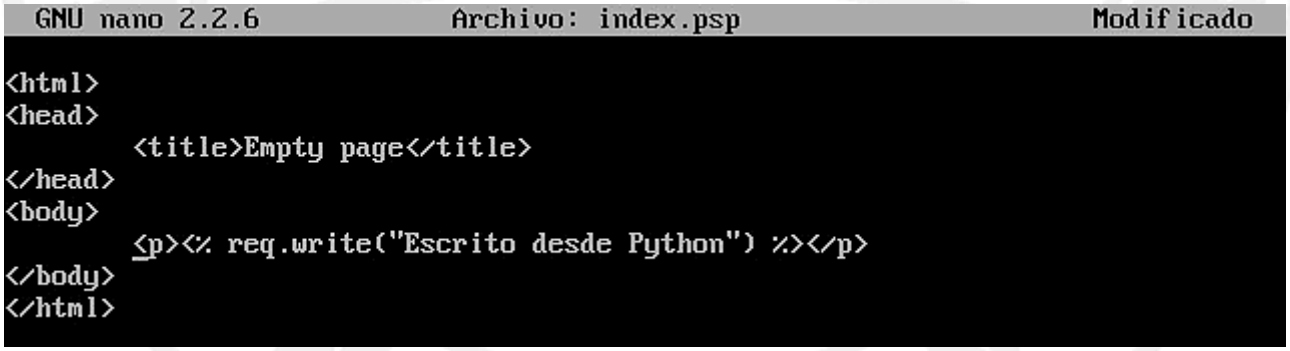


Figura 87. Ejemplo de página .psp

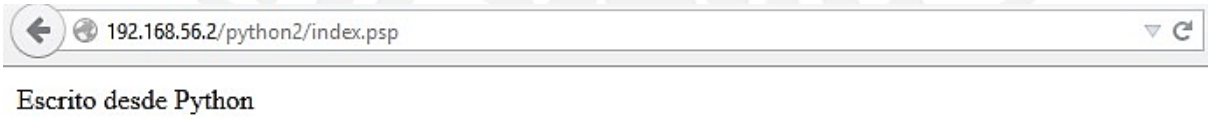


Figura 88. Página .psp servida correctamente

Debemos insistir en que la directiva `PythonDebug On` solo tiene sentido en máquinas de desarrollo, ya que no queremos que en una máquina de producción se mande información de la traza de ejecución al cliente: no es adecuado y además puede suponer un problema de seguridad.

Finalmente, debe quedar claro que esto es solo una introducción al uso de *Python* con *Apache 2*, ya que profundizar más estaría fuera del ámbito de labores de administración. Para profundizar más en el tema, se recomienda consultar este enlace: https://modpython.org/live/current/doc-html/contents.html

Ejemplo de filtro de salida: SSI (*Server Side Includes*)

Los SSI (*Server Side Includes*) son un **conjunto de directivas** que se incluyen en las páginas HTML y que se evalúan en el servidor antes de que se sirvan al cliente. Permiten incluir determinado contenido dinámico en dichas páginas **sin el uso de ninguna tecnología particular para este fin** (PHP, .NET, etc.). El uso de SSI **puede suponer un riesgo de seguridad** y por tanto, como veremos posteriormente, es conveniente intentar **desactivarlos siempre**. No obstante, en algunas ocasiones se usa para añadir pequeñas cantidades de información dinámicamente a una página que es mayoritariamente estática.

Para habilitar SSI hay que añadir la siguiente directiva en la localización donde tengamos las páginas que incluyan esta tecnología: `Options +Includes`. De esta forma *Apache 2* recorrerá las páginas en esa localización y buscará si existen directivas SSI que procesar y calcular. No obstante, hacer esto no es muy adecuado porque las páginas que incluyen directivas SSI suelen tener una extensión propia (`.shtml`), por lo que lo más aconsejable es indicar a *Apache 2* que estos ficheros son los únicos que deben ser procesados en busca de directivas SSI. Esto se hace de la siguiente forma:

```
AddType text/html .shtml
AddOutputFilter INCLUDES .shtml
```

No obstante, esto obligaría a cambiar los nombres y las referencias a estas páginas en el caso de que existieran previamente. Por ello, se puede recurrir a la directiva `XBitHack` (con `XBitHack On`), que indica a *Apache 2* que busque directivas SSI en todas las páginas que tengan el bit de permisos de ejecución activo (aunque no sean `.shtml`). Nótese que ambos métodos consiguen exactamente lo mismo, pero el segundo no obliga a hacer cambios en webs existentes.

Vamos a ver los pasos necesarios para hacer todas estas operaciones. Lo primero es habilitar el módulo de *Apache 2* que permite el uso de SSI (`mod_include`):

```
operario@servidorapache:/var/www/html/ssi$ sudo a2enmod include
Considering dependency mime for include:
Module mime already enabled
Enabling module include.
To activate the new configuration, you need to run:
  service apache2 restart
operario@servidorapache:/var/www/html/ssi$
```

Figura 89. Habilitando mod_include

Hecho esto, indicamos que vamos a permitir SSI sobre los ficheros de un directorio concreto, solo para ficheros `.shtml`. Nótese que hemos activado a modo de muestra `XBitHack` aunque no se va a usar en el ejemplo al no haber páginas `.html` con el bit de ejecución activo:

```
  GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf

        </RequireAll>
</Directory>

<Directory /var/www/html/negotiation>
        Options +Multiviews
        AddHandler type-map .var
        MultiviewsMatch Handlers
</Directory>

<Directory /var/www/html/ssi>
        Options +Includes
        AddOutputFilter INCLUDES .shtml
        XBitHack on
</Directory>
```

Figura 90. Activando SSI en un directorio concreto

Y a continuación creamos una página que use SSI. En este ejemplo simplemente mostramos la hora local del servidor en la página de respuesta al cliente:

```
  GNU nano 2.2.6            Archivo: index.shtml

<html>
<head>
Empty page
</head>
<body>

<p>Welcome</p>

<p>
        <!--#echo var="DATE_LOCAL" -->
</p>
</body>
</html>
```

Figura 91. Fichero de ejemplo que usa SSI

Antes de probar si funciona correctamente, conviene destacar que la directiva `AddOutputFilter` anterior no era realmente necesaria porque ya estaba incluida a nivel global, como vemos en la siguiente imagen:

```
 GNU nano 2.2.6              Archivo: mime.conf

        #
        # For type maps (negotiated resources):
        # (This is enabled by default to allow the Apache "It Worked" page
        #  to be distributed in multiple languages.)
        #
        AddHandler type-map var

        #
        # Filters allow you to process content before it is sent to the client.
        #
        # To parse .shtml files for server-side includes (SSI):
        # (You will also need to add "Includes" to the "Options" directive.)
        #
        AddType text/html .shtml
        AddOutputFilter INCLUDES .shtml

</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Figura 92. SSI activo a nivel global

Como el fichero `mime.conf` ya incluye una asociación de SSI para `.shtml` podría quitarse en este caso, pero dado que los SSI pueden suponer un problema de seguridad (como veremos en breve) lo más oportuno sería **eliminarlo a nivel general** y activarlo solo en aquellas localizaciones donde realmente se vaya a usar SSI. En cualquier caso, vemos que funciona correctamente:

Figura 93. SSI en funcionamiento

En un ejemplo anterior hemos visto que si eliminamos a nivel general el tipo MIME de los ficheros `.shtml`, el contenido de dichos ficheros pasa a tratarse como texto plano en lugar de como HTML. Sin embargo, los SSI que pudieran incluir estas páginas funcionarían (se ve la fecha), porque los tenemos activos a nivel de directorio (a pesar de haberlos borrado a nivel general). Es el contenido del fichero de respuesta una vez calculado en el servidor lo que no se interpreta correctamente. Lo importante es recordar que **eliminar el tipo MIME de estos ficheros no impide pues que los SSI se ejecuten**.

Para más información se pueden consultar los siguientes enlaces:

- http://httpd.apache.org/docs/current/howto/ssi.html
- http://www.htmlgoodies.com/beyond/webmaster/article.php/3473341
- http://www.freewebmasterhelp.com/tutorials/ssi

Ejemplo de habilitación de un filtro de entrada

Como ejemplo sencillo de un filtro de entrada vamos a activar `mod_buffer` (http://httpd.apache.org/docs/current/en/mod/mod_buffer.html). Este módulo lee las peticiones y/o las respuestas en RAM y las empaqueta **para que ocupen la menor cantidad de memoria posible**, evidentemente con un coste de CPU. Puede tener un impacto positivo en el rendimiento general de *Apache 2*, pero conviene monitorizar si esto es realmente así o no en cada caso concreto. Para activar `mod_buffer` vamos por ejemplo al fichero de configuración del host virtual por defecto `000-default.conf` y añadimos la siguiente línea:

```
        AddInputFilter BUFFER text/html
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Figura 94. Activación de mod_buffer

Ejemplo de creación de un filtro de salida manual

Como vimos en la sección dedicada a describir el ciclo de gestión de peticiones de *Apache 2*, los filtros de entrada y de salida **son una cadena**. Veremos un ejemplo ahora de cómo **añadir un filtro desarrollado por nosotros en *Python* a dicha cadena**, con la idea de ilustrar lo fácil que es alterar el comportamiento de *Apache 2* mediante los mecanismos vistos.

El primer paso para hacer esto es determinar la localización y el tipo de fichero sobre el que queremos añadir un filtro adicional que procese sus contenidos. En este caso vamos a usar la localización anterior para ficheros *Python* y procesar los ficheros `.py` con un filtro adicional. Nuestro objetivo es que el contenido generado por el *content handler* `mod_python` se procese por un filtro de salida también desarrollado en *Python* que ponga todo el texto en mayúsculas. Para ello, modificamos la configuración del directorio de la siguiente forma:

```
  GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf

</Directory>

<Directory /var/www/html/ssi>
        Options +Includes
        AddOutputFilter INCLUDES .shtml
        XBitHack on
</Directory>

<Directory /var/www/html/python>
        AddHandler mod_python .py
        PythonHandler time_handler
        PythonOutputFilter mayusculas MAYUS
        AddOutputFilter MAYUS .py
        PythonDebug On
</Directory>
```

Figura 95. Añadiendo un filtro de salida para ficheros .py

Vemos que hemos creado un filtro de salida en *Python* llamado `MAYUS` a partir del programa `mayúsculas.py` (la extensión se supone implícita) que veremos posteriormente. Una vez creado ese filtro lo añadimos a la cadena de filtros de salida con `AddOutputFilter`.

Hecho esto nos queda simplemente desarrollar el filtro en sí. La siguiente imagen muestra el programa que implementa el filtro en su función `outputfilter`:

```
 GNU nano 2.2.6          Archivo: mayusculas.py

from mod_python import apache

def outputfilter(filter):
        s = filter.read()
        while s:
                filter.write(s.upper())
                s = filter.read()

        if s is None:
                filter.close()
```

Figura 96. Código del filtro de salida para ficheros .py

Y por supuesto, probamos su funcionamiento:

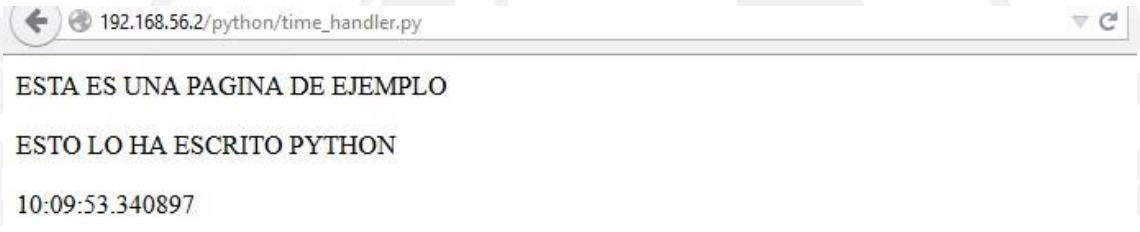


Figura 97. Ejemplo de funcionamiento del filtro de salida

Nótese como están funcionando al mismo tiempo el `Handler` para ficheros `.py` y el filtro de salida. Usando la misma aproximación vista podríamos hacer filtros mucho más avanzados que sustituyeran términos, elementos, modificasen determinada información o hicieran cualquier clase de modificación a la página de respuesta que quisiésemos, usando para ello toda la potencia del lenguaje *Python*. Podemos usar este enlace como referencia para la creación de filtros más avanzados: https://modpython.org/live/current/doc-html/pythonapi.html#overview-of-a-filter-handler

Ejemplo de instalación de un filtro predefinido: *mod_deflate*

Para finalizar con esta serie de ejemplos que hablan de cómo modificar la cadena de peticiones de *Apache 2*, vamos a activar un filtro de salida muy útil e incluido con la instalación del servidor: `mod_deflate` (http://httpd.apache.org/docs/current/mod/mod_deflate.html). Este filtro sirve para **comprimir los datos de las peticiones y aprovechar mejor el ancho de banda**, de forma que se transmita la misma información de una manera mucho más eficiente. Esto se puede conseguir de la siguiente forma:

- Activar `mod_deflate` con el comando `a2enmod deflate`. La última versión de *Apache 2* ya viene con el activo y por tanto esto no es necesario en la mayoría de los casos.
- Reiniciar *Apache 2* si finalmente se ha activado un mod nuevo, como ya hemos venido haciendo anteriormente.

- Añadir las siguientes líneas al fichero `000-default.conf` del directorio `sites-available` (o en el fichero de configuración del sitio web donde quisiésemos aplicarlo). Estas líneas activan el máximo nivel de compresión (`9`) y además establecen un log particular (`deflate_log`) en la ruta dedicada a guardar *logs* por defecto (`/var/log/apache2/`) para este *mod* donde se escribe la información en un determinado formato, indicando el ratio de compresión alcanzado. Para ello se usan las variables del *mod* `Input`, `Output` y `Ratio`:

```
#Tipos MIME de los documentos procesados por Apache 2 que el filtro comprimirá
AddOutputFilterByType DEFLATE text/html text/plain text/css text/javascript
DeflateCompressionLevel 9
DeflateFilterNote Input input_info
DeflateFilterNote Output output_info
DeflateFilterNote Ratio ratio_info

#Formato del log de deflate que vamos a usar
LogFormat '"%r" %{output_info}n/%{input_info}n (%{ratio_info}n%%)' deflate   #Logs de las
peticiones realizadas y de su procesamiento por deflate
CustomLog /var/log/apache2/deflate_log deflate
```

```
 GNU nano 2.2.6 Archivo: ...ache2/sites-available/000-default.conf

        # For most configuration files from conf-available/, which are
        # enabled or disabled at a global level, it is possible to
        # include a line for only one particular virtual host. For example the
        # following line enables the CGI configuration for this host only
        # after it has been globally disabled with "a2disconf".
        #Include conf-available/serve-cgi-bin.conf

AddOutputFilterByType DEFLATE text/html text/plain text/css text/javascript
        DeflateCompressionLevel 9
        DeflateFilterNote Input input_info
        DeflateFilterNote Output output_info
        DeflateFilterNote Ratio ratio_info
```

Figura 98. Habilitando mod_deflate para ciertos tipos de fichero

- Este filtro comprime el contenido en los tipos *MIME* comentados en su primera línea, aunque podríamos añadir más, como `text/xml`. En cualquier caso, tras hacer una serie de peticiones vamos como el contenido se sirve de la misma forma que se hacía anteriormente para el cliente, pero en el log se puede ver el ratio de compresión alcanzado:

```
operario@servidorapache:/var/www/html/python2$ cat /var/log/apache2/deflate_log
"GET /index.html HTTP/1.1" 57/71 (80%)
"GET /index.html HTTP/1.1" 57/71 (80%)
"GET /index.html HTTP/1.1" 57/71 (80%)
"GET /index.html HTTP/1.1" 57/71 (80%)
"GET /index.html HTTP/1.1" 57/71 (80%)
operario@servidorapache:/var/www/html/python2$
```

Figura 99. Comprobación del log de mod_deflate

Figura 100. mod_deflate es transparente al cliente

Si la versión de *Apache 2* usada ya viene con `mod_deflate` activo por defecto, entonces poseerá una configuración básica en el fichero `deflate.conf` que permita usarlo en todo el servidor, que por defecto es ésta:

```
operario@servidorapache:/var/www/html/python2$ cat /etc/apache2/mods-enabled/def
late.conf
<IfModule mod_deflate.c>
        <IfModule mod_filter.c>
                # these are known to be safe with MSIE 6
                AddOutputFilterByType DEFLATE text/html text/plain text/xml

                # everything else may cause problems with MSIE 6
                AddOutputFilterByType DEFLATE text/css
                AddOutputFilterByType DEFLATE application/x-javascript applicati
on/javascript application/ecmascript
                AddOutputFilterByType DEFLATE application/rss+xml
                AddOutputFilterByType DEFLATE application/xml
        </IfModule>
</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
operario@servidorapache:/var/www/html/python2$ _
```

Figura 101. Configuración de mod_deflate a nivel de servidor

En este caso, simplemente podemos añadirle las líneas de la configuración anterior que especifican los tipos de ficheros comprimidos (si quisiéremos cambiarlos) y el nivel de compresión, quedando pues la configuración así:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/mods-enabled/deflate.conf

<IfModule mod_deflate.c>
        <IfModule mod_filter.c>
                # these are known to be safe with MSIE 6
                AddOutputFilterByType DEFLATE text/html text/plain text/xml
                DeflateCompressionLevel 9

                # everything else may cause problems with MSIE 6
                AddOutputFilterByType DEFLATE text/css
```

Figura 102. Cambio de configuración de mod_deflate a nivel de servidor

Por tanto, ya no haría falta añadir lo mismo a nivel de host virtual, ya que esto se aplicaría en todos ellos. Lo que si haremos es crear un log particular para ese sitio web concreto, de forma que pueda guardarse como actúa al *mod* de forma independiente para el mismo. En la siguiente imagen vemos cómo queda la configuración modificada para ello:

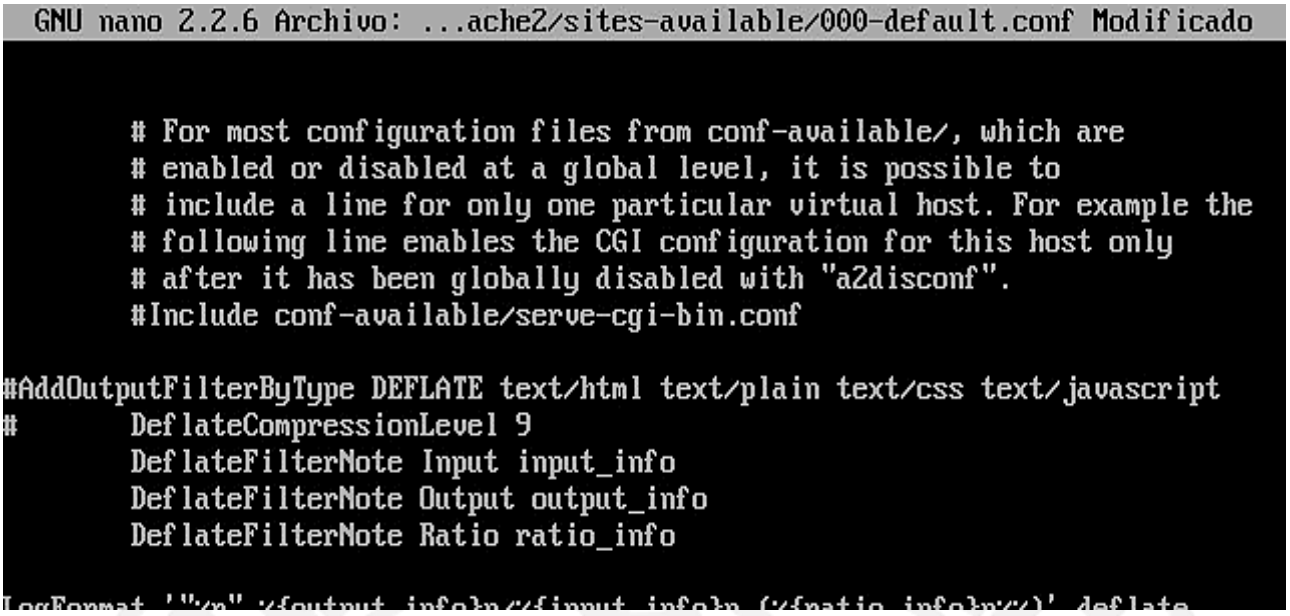


Figura 103. Unión de la configuración de mod_deflate global y particular para un virtual host

Obviamente, *Apache 2* sigue funcionando sin ningún problema

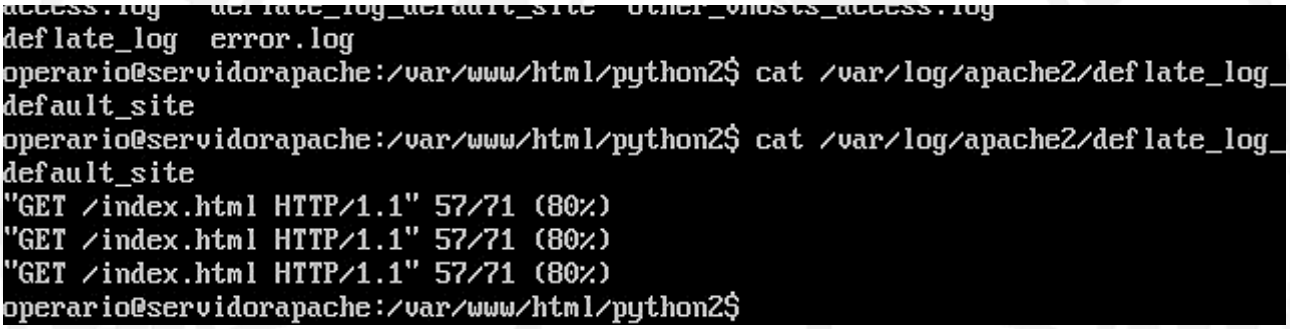


Figura 104. Comprobación de funcionamiento de la unión de configuración global + particular

En una página con un mayor nº de contenidos, podríamos ver como `mod_deflate` no intenta comprimir aquellos ficheros cuyo tipo MIME no hemos indicado. En general hay ficheros que no es posible comprimir mucho, y por tanto no merece la pena incluirlos en este *mod* porque no tendríamos ningún beneficio real en ello. Estos ficheros son por ejemplo imágenes `.gif`, `.jpg`, etc., ficheros ya comprimidos `.zip`, `.rar` o multimedia, entre otros. En general, los mejores resultados se obtendrán en aquellos ficheros cuyo contenido sea texto plano. Una adecuada configuración de este aspecto tendrá muchos beneficios para nuestro servidor.

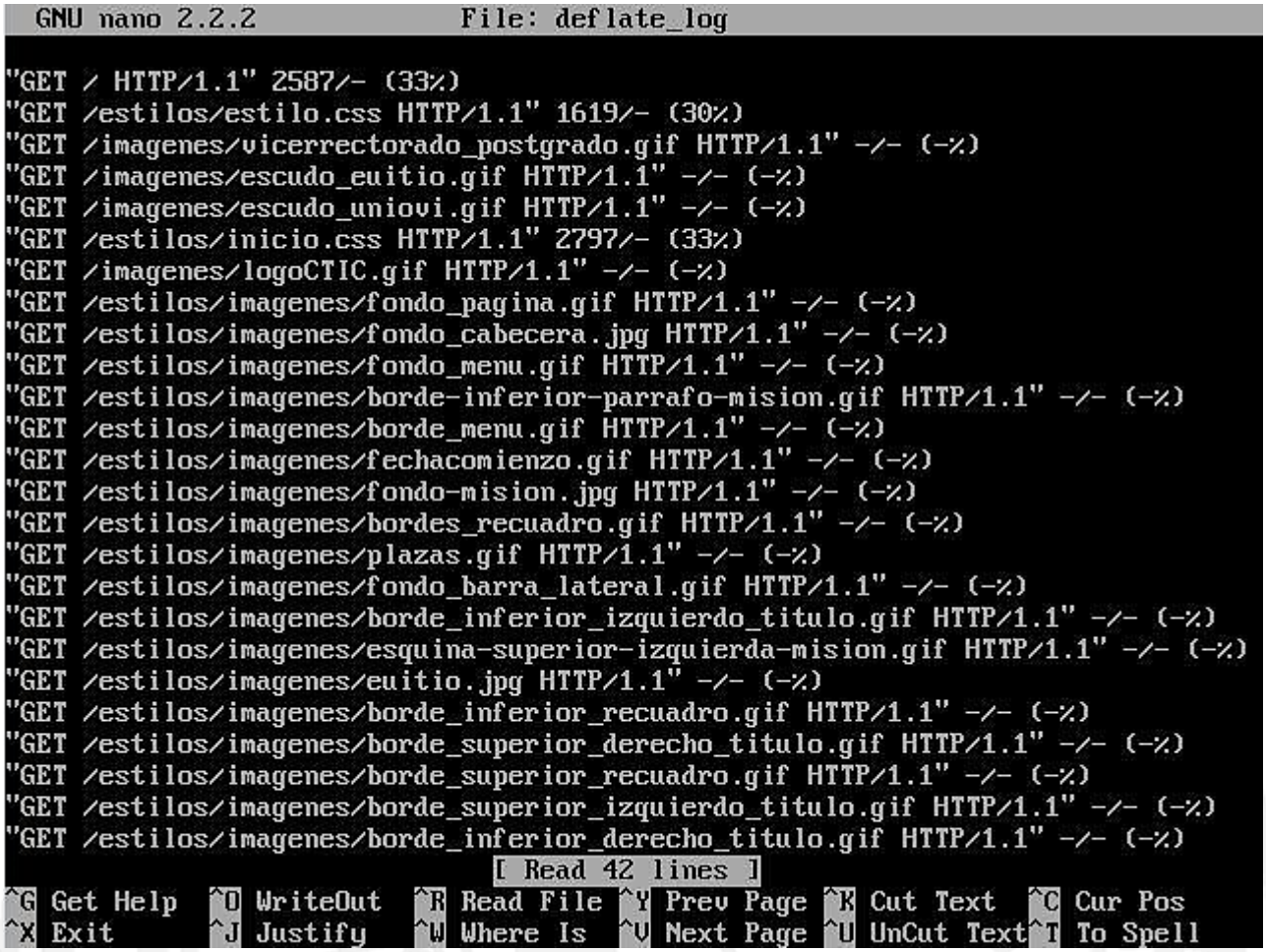


Figura 105. Funcionamiento selectivo de mod_deflate

Finalmente, en este enlace: http://www.howtoforge.com/linux_*Apache* 2_mod_gzip se describe cómo usar un módulo de propósito similar, `mod_gzip`.

Aspectos básicos de ciberseguridad en *Apache 2*

Desactivación de los enlaces simbólicos, CGI *y Server Side Includes*

Problemas de seguridad con *Server Side Includes*

Ya hemos visto lo que son los SSI y su utilidad para generar páginas con un determinado contenido dinámico. No obstante, debemos tener en cuenta que el uso de SSI **puede suponer un problema de rendimiento y también de seguridad**. Los motivos son varios:

- Si no se configura adecuadamente, como vimos en la sección anterior, **todos los ficheros servidos por *Apache 2* deben examinarse para comprobar si tienen directivas SSI incluidas**. Si los ficheros examinados de esta forma son muchos, la **penalización de rendimiento** puede ser significativa. Conviene pues, como vimos anteriormente, limitar las extensiones de ficheros habilitadas para SSI e intentar centralizar en un solo sitio los ficheros que usan estas directivas.
- Los ficheros que usan SSI tienen **los mismos riesgos de seguridad que los programas CGI** que veremos posteriormente. Mediante SSI es posible **la ejecución de cualquier *script* o programa bajo la identidad del usuario que ejecute la petición** (por defecto `www-data`). Esto es posible porque entre los comandos SSI posibles están `exec cmd` y `exec cgi`, que permiten precisamente esta acción. Aunque la página esté pensada para la ejecución de un comando o CGI concretos, no podemos descartar que bien por una entrada maliciosa de un usuario o por otros vectores de ataque se cambie el fichero a ejecutar o se haga alguna modificación maliciosa de estos comandos.
- Para intentar evitar estos problemas, se recomienda:
    - Si se cumplen las condiciones adecuadas que veremos en la siguiente sección, habilitar `SuExec` para ejecutar los scripts SSI bajo una identidad de usuario adecuada para ellos (de esa forma se pueden limitar los privilegios de ejecución de las páginas SSI todo lo que sea necesario).
    - Hacer como en la sección anterior y **limitar el uso de SSI a ficheros** `.shtml`, en lugar de dejar que puedan usarse en páginas `.html` o `.htm` "normales". Esto no solo limita el conjunto de ficheros que puedan suponer un riesgo, sino que además sobrecarga menos el servidor.
    - **Deshabilitarlos completamente o de forma limitada**. En el siguiente ejemplo veremos cómo deshabilitar completamente el uso de SSI mediante la directiva `Options -Includes` (que evitará que funcionen incluso aunque añadamos un filtro de procesamiento explicito para páginas que usen SSI) o bien habilitarlos pero prohibiendo los comandos que permiten la ejecución de programas externos (los dos `exec` vistos), con `Options +IncludesNOEXEC`.

      Nótese que aunque esta última directiva impide el uso de estos dos comandos peligrosos, si permite incluir ficheros CGI externos con `#include virtual` (http://www.htmlgoodies.com/beyond/webmaster/article.php/3473341).

```
<Directory "/var/www/html/ssi">
        Options -Includes
        AddOutputFilter INCLUDES .shtml
</Directory>
```

Figura 106. Deshabilitando SSI en un directorio que tiene esta clase de contenido

En la imagen anterior vemos la forma de **deshabilitar SSI incluso aunque tengamos un filtro declarado para este tipo de contenidos**. La página con SSI no da un error al hacer esto, sino que no mostraría las líneas que incluyesen directivas SSI porque su sintaxis es en forma de comentarios HTML (no obstante, los comentarios con directivas SSI aparecerían en el código fuente de la página que se le envía al cliente). Por otro lado, usar `+IncludesNOEXEC` (siguiente imagen) no impide la ejecución de SSI que no los incluya. Esta opción simplemente elimina un posible foco de ataque importante, no la tecnología al completo:

```
<Directory "/var/www/html/ssi">
        Options +IncludesNOEXEC
        AddOutputFilter INCLUDES .shtml
</Directory>
```

Figura 107. Habilitando SSI pero prohibiendo comandos exec

Problemas de seguridad con programas CGI y otro contenido dinámico

¿Qué es un programa CGI?

Hasta ahora hemos mencionado de pasada estas siglas sin pararnos a describir en detalle en qué consisten. No obstante, ahora que vamos a hablar detalladamente de los problemas que pueden traer, es el momento adecuado para hacerlo. En general, cuando se instala una extensión de cualquier servidor *web*, **se tiene que elegir una tecnología para que dicho servidor pueda comunicarse con la extensión** y de esta forma coordinarse para llevar a cabo la labor que dicha extensión tiene asignada. Una tecnología típica para ello es ***CGI (Common Gateway Interface)***, que es un protocolo estándar para la comunicación entre una aplicación externa y un servidor de información (en este caso un servidor *web*). Esta tecnología es muy común y está soportada tanto por *Apache 2* como por *IIS*, además de otros servidores.

Básicamente, una aplicación *CGI* **es un ejecutable (proceso) externo cuyo código procesa las variables de entorno y su entrada estándar**, generando unos determinados resultados **que envía a su salida estándar** y que finalmente son los que llegan al cliente. *CGI* es por tanto una posible forma de generar contenido dinámico para una aplicación *web*, puesto que el cliente recibe un resultado que puede depender de la información que le haya pasado a la aplicación *CGI* inicialmente.

La programación de aplicaciones *CGI* **no está ligada a ningún lenguaje concreto**, aunque muchos lenguajes (como *C*) incluyen bibliotecas de funciones para procesar las variables de entorno y otras ayudas para la creación de *CGI*. *Apache 2* permite la ejecución de programas siguiendo las reglas descritas por este protocolo de comunicación gracias a un *mod* denominado `mod_cgi` (http://httpd.apache.org/docs/current/mod/mod_cgi.html). Este *mod* incluye el *handler* `cgi-script`, que trata a cualquier fichero existente sobre una localización donde sea aplicable este *handler* como un

programa CGI (proporcionando información en su entrada estándar y procesando su salida). El proceso seguido por ese *mod* es:

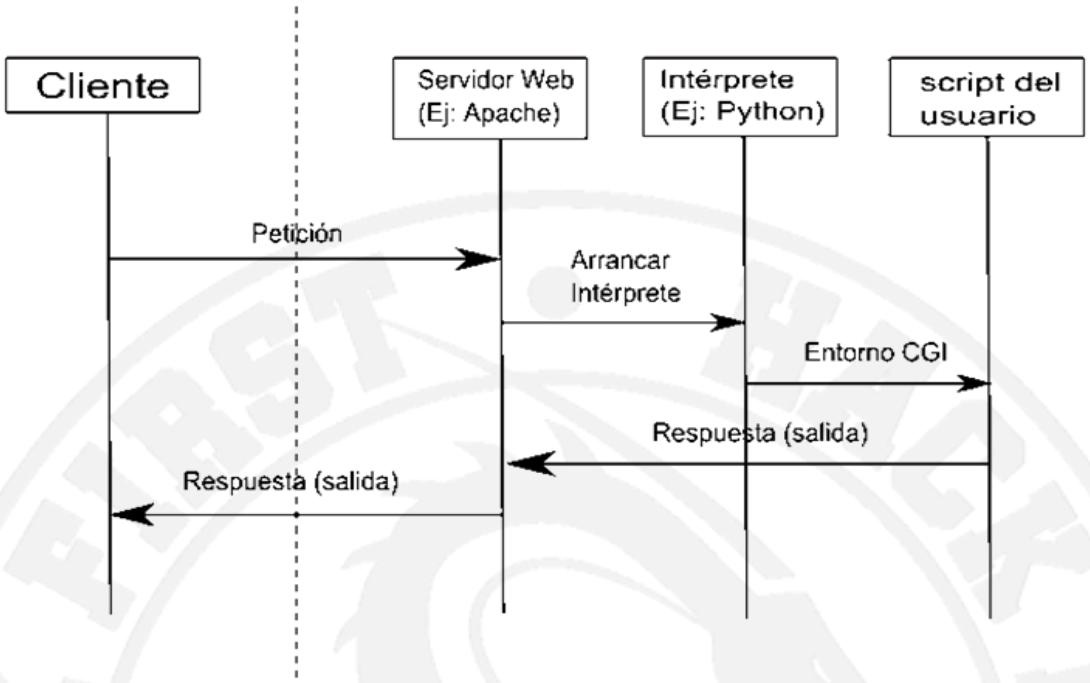


Figura 108. Diagrama de secuencia: Proceso de una petición CGI

FastCGI es otra tecnología con el mismo propósito que *CGI*, pero que a diferencia de ésta usa un único proceso de ejecución continua para manejar más de una petición de los usuarios. A pesar de esta diferencia fundamental de funcionamiento, ***FastCGI* trata de mantener un modo de funcionamiento muy similar al de *CGI*** ya visto, es decir, la funcionalidad de ambas tecnologías es prácticamente idéntica. El objetivo fundamental de *FastCGI* es mantener la simplicidad del modelo de programación de las aplicaciones *CGI* **eliminando una importante penalización de rendimiento** que posee, causada porque cada petición se atiende en un proceso independiente. Dado que crear un proceso es una operación costosa, *FastCGI* obtiene un rendimiento notablemente superior a *CGI*, suponiendo pues una mejora de este último.

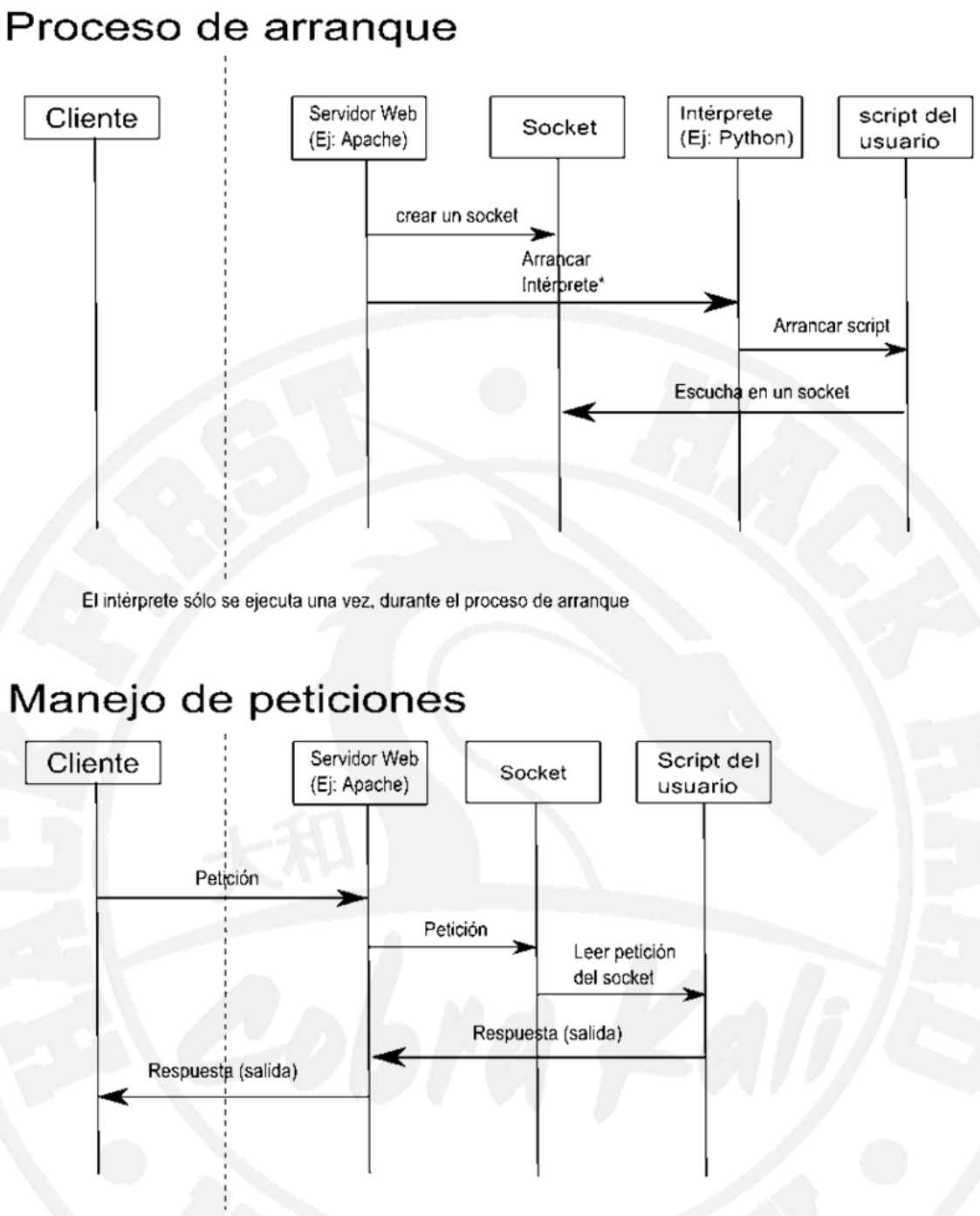


Figura 109. Diagrama de secuencia: Proceso de una petición FastCGI

Podemos ampliar la información acerca de *CGI* y *FastCGI* en los siguientes enlaces:

- https://en.wikipedia.org/wiki/Common_Gateway_Interface
- http://en.wikipedia.org/wiki/FastCGI
- http://www.fastcgi.com/

¿Qué problemas de seguridad pueden dar estos programas?

Debe tenerse en cuenta que el mecanismo seguido por esta forma de servir contenidos dinámicos es muy potente y a la vez muy peligroso: a partir de una entrada que nos viene del cliente, esta se procesa y se pasa a un programa, que **a modo de "caja negra"** usa esa entrada para producir la salida que se envía al cliente finalmente. El servidor no sabe ni comprueba nada de lo que hace el programa externo con el que

se comunica mediante CGI, por lo que queda a merced de este. Por ese motivo, hay que tomar una serie de precauciones a la hora de usar esta tecnología:

- Lo primero y más importante es que si usamos esta clase de programas **debemos tener confianza plena en los autores de estos**, ya que si tuviesen alguna clase de vulnerabilidad o error, seríamos víctimas de este. Es especialmente peligroso que un programa CGI ejecute comandos del sistema operativo para cumplir con su función, por el mismo motivo que lo era en el caso de SSI.
- **Restringir la existencia de programas CGI a unos directorios concretos**, de forma que se sepa en todo momento donde pueden residir los mismos. Esto además tiene la ventaja de que podemos habilitar permisos sólo en esos directorios para que los desarrolladores de scripts los suban, como vimos en una sección anterior.
- **Todos los programas CGI se ejecutan bajo la misma identidad de usuario**, por lo que podrían entrar en conflicto (de forma accidental o a propósito) entre ellos. Si queremos evitar que todos los scripts se ejecuten bajo la misma identidad de usuario, una opción es el uso de `suEXEC`. Esta tecnología se basa en un programa `setuid` "envoltorio" al que llamará el servidor cada vez que reciba una petición adecuada (es decir, una petición que solicite un programa CGI o una página que use SSI y que el administrador haya configurado para que se ejecuten bajo una identidad de usuario diferente a la que tiene el servidor).

  Si se recibe una petición de esta clase, el servidor *Apache 2* llamada a ese envoltorio proporcionándole el programa CGI o SSI a ejecutar, así como el usuario y grupos bajo cuya identidad debe ejecutarlos. No obstante, el envoltorio, antes de ejecutar el programa solicitado, ejecuta un proceso en múltiples pasos muy estricto para asegurarse de que la petición no tiene efectos maliciosos. El proceso completo puede consultarse en el siguiente enlace: http://httpd.apache.org/docs/current/suexec.html. Si uno de los pasos del proceso fallase, la ejecución del programa CGI o SSI se cancelaría. Solo pasando todas las pruebas descritas se permitiría la ejecución del programa solicitado y por tanto se serviría la petición.
- El uso de `suEXEC` es **MUY delicado** y desde la propia documentación de *Apache 2* incluso de recomienda que no se use si no se está completamente seguro de saber qué se está haciendo y si es necesario. Se puede usar a nivel de host virtual o bien en páginas web personales de usuario (se dedicará posteriormente una sección a este tipo de páginas). Concretamente, para poder usarlo se deben tener en cuenta las siguientes directrices:
  - *Apache 2* debe ejecutarse en un sistema operativo *Unix/Linux* que soporte las operaciones `setuid` y `setgid`.
  - El usuario que intente emplear `suEXEC` debe tener conocimientos de administración, al menos para comprender el efecto de las dos operaciones anteriores sobre la configuración particular del sistema
  - Debe usarse un `suEXEC` original, no modificada.
  - `suEXEC` **NO** es parte de la instalación por defecto de *Apache 2*. Si se necesita usar, debe instalarse y configurarse cuidadosamente.

Finalmente, si se decide usar esta tecnología, es frecuente emplear la directiva `ScriptAlias` (http://httpd.apache.org/docs/current/mod/mod_alias.html#scriptalias) para simplificar y unificar la configuración de los programas que la usan, asumiendo que todos van a estar en una ruta concreta. Esta directiva **marca un directorio concreto como contenedor de programas CGI** que sean procesados por el `mod_cgi` mencionado. Por ejemplo:

```
ScriptAlias "/cgi/" "/miweb/scripts/"
```

De esta forma una petición a `http://www.miweb.com/cgi/programa.rb` hará que el servidor ejecute el *script* `/miweb/scripts/programa.rb` (asumiendo que hay un *mod* que implementa el lenguaje Ruby (`.rb`) instalado para admitir la tecnología CGI para comunicarse con él). Esto es equivalente a:

```
Alias "/cgi/" "/miweb/scripts/"
<Location "/cgi" >
    SetHandler cgi-script
    Options +ExecCGI
</Location>
```

Nótese como se puede usar `/cgi` junto con `<Location>` para crear una configuración común a todos los *scripts*.

Aunque determinados *mods* que habilitan a *Apache 2* a interpretar *scripts* en lenguajes variados puedan no usar por defecto la tecnología CGI para que *Apache 2* se comunique con ellos, esta tecnología lleva mucho tiempo en el mercado, y por tanto muchos de ellos suelen tener una forma de permitir comunicación vía CGI manteniendo toda su funcionalidad.

En el caso del `mod_python` ya visto, podemos lograr esto con el *handler* `mod_python.cgihandler` (http://modpython.org/live/mod_python-3.2.8/doc-html/handhttp://modpython.org/live/mod_python-3.2.8/doc-html/hand-cgi.htmlcgi.html). En el caso de que necesitásemos por alguna razón usar `SuEXEC`, podemos usar la siguiente guía: https://www.digitalocean.com/community/tutorials/how-to-use-suexec-in-*apache*-to-run-cgi-scriptshttps://www.digitalocean.com/community/tutorials/how-to-use-suexec-in-apache-to-run-cgi-scripts-on-an-ubuntu-vpson-an-ubuntu-vps

Seguridad con otras formas de crear páginas dinámicas

Como ya hemos dicho, existen determinados *mods* que h**abilitan tecnologías variadas para la creación de páginas dinámicas** (`mod_php`, `mod_perl`, `mod_tcl`, `mod_python`…) que se instalan por defecto de una forma embebida en el servidor y que por tanto se ejecutan bajo la misma identidad de usuario que sirve las peticiones.

Por ese motivo, los programas elaborados en estas tecnologías pueden acceder a todo aquello a lo que también puede acceder el servidor (incluidos ficheros pertenecientes a otros sitios web), salvo que el propio *mod* imponga restricciones de seguridad al respecto. No obstante, esto escapa a nuestro control y depende de la voluntad de los programadores de este. No obstante, herramientas como `mod_security` (que veremos posteriormente) permiten mejorar la seguridad de esta clase de contenidos sin tener que depender al 100% de lo que los programadores de estos *mods* hayan querido o no querido hacer.

Problemas de seguridad con enlaces simbólicos

Es frecuente que las aplicaciones web tengan algún directorio destinado a alojar los contenidos que sus usuarios puedan querer subir, como parte de la funcionalidad de este. Para que esto sea posible, este directorio debe tener por tanto permisos de escritura sobre el usuario bajo el que se ejecutan las peticiones.

Tener esta funcionalidad es peligroso por varios motivos:

1. **Los ficheros subidos por un usuario A pueden ser accesibles por otro usuario B solo por conocer su ruta**. Esto puede ser un problema de privacidad y arruinar completamente el propósito de la

página. Una forma de controlar este aspecto es mediante la programación explícita de controles de acceso en el código de la página web, impidiendo el acceso directo a estos ficheros escribiendo su URI. Para impedir el acceso directo a los ficheros subidos algo muy típico es que el directorio donde se suben esté fuera de la carpeta raíz de *Apache 2* (`/var/www/html`), para que sea imposible generar una URI que acceda al mismo. Los scripts usan los permisos del sistema de ficheros local, por lo que el hecho de que puedan leerlos es fácilmente configurable.

2. Los ficheros subidos por un usuario **pueden ser de un tipo no permitido** o, aún peor, contenido ejecutable. Esto ya no es solo malo por corromper el propósito de la web, sino los resultados de su ejecución. Si un usuario sube un fichero `.php` y luego accede al mismo, la ejecución de ese código subido bajo la identidad del usuario que sirve las peticiones puede tener consecuencias graves: engaños al usuario para robar información, acceso a ficheros de otra manera inaccesibles, o bien descargar en el cliente programas ejecutables con código malicioso, entre otros. Nótese que la solución al problema anterior también soluciona éste.
3. Finalmente, otra de las posibilidades es que un atacante **suba un enlace simbólico a un fichero**. Estos enlaces pueden designar un fichero cualquiera del sistema de ficheros (`/etc/passwd`, ficheros de configuración…) de manera que acceder al mismo sea equivalente a ver su contenido o a descargarlo, con todo el perjuicio que eso puede conllevar. Esto puede evitarse indicándole a *Apache 2* que en una determinada localización no siga posibles enlaces simbólicos que pueda contener, para que así este vector de ataque no pueda llevarse a cabo, independientemente de los permisos que tengan los ficheros "destino" de estos enlaces.

Para ilustrar este último punto, vamos a ver qué fácil es causar un problema de seguridad solo por el hecho de tener los enlaces simbólicos activos. Vamos a activar los enlaces simbólicos en el directorio `/var/www/html/ln`, por ejemplo:

```
<Directory "/var/www/html/ln">
        Options +FollowSymLinks
</Directory>
```

Figura 110. Habilitar los enlaces simbólicos

Ahora vamos a crear un enlace simbólico al fichero central de configuración de *Apache 2* que tome la forma de un fichero de texto simple. Esto mismo podría subirlo un usuario malicioso a una ruta donde se le permitan subir ficheros de un determinado tipo (para lo cual no tendría problema, ya que el enlace simbólico podría tener la extensión que fuese necesaria):

```
operario@ubuntu:/var/www/html/ln$ sudo ln -s /etc/apache2/apache2.conf fichero.t
xt
operario@ubuntu:/var/www/html/ln$ ls
fichero.txt
operario@ubuntu:/var/www/html/ln$
```

Figura 111. Creación de un enlace simbólico a la configuración de Apache 2

Comprobamos ahora que acceder al enlace simbólico nos da información muy comprometedora para la seguridad del servidor (el fichero `apache2.conf`, con todos los detalles de configuración):

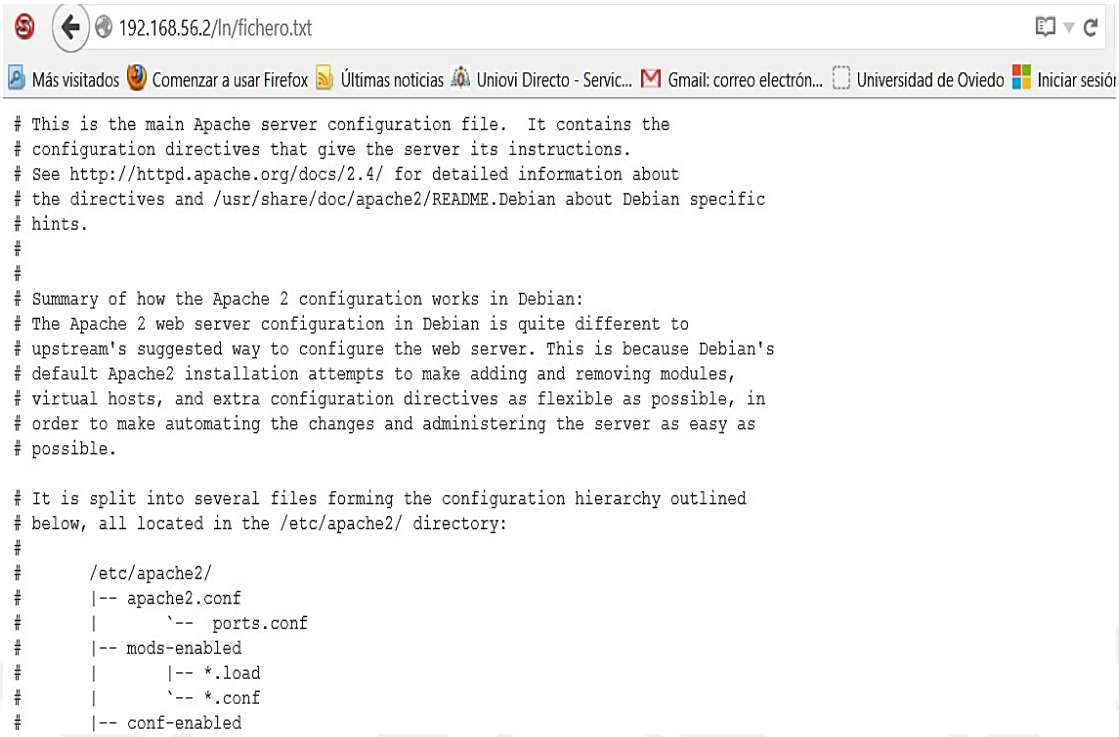


Figura 112. Fichero Apache2.conf a través de un enlace simbólico

Podemos ver un caso aún peor si el enlace simbólico es al fichero de *passwords* de *Linux*:

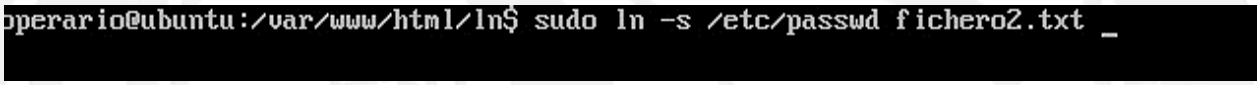


Figura 113. Enlace simbólico a /etc/passwd

Que por ser legible para todos los usuarios (incluido obviamente `www-data`), puede ser leído, conociendo así toda la lista de usuarios que el sistema tiene:

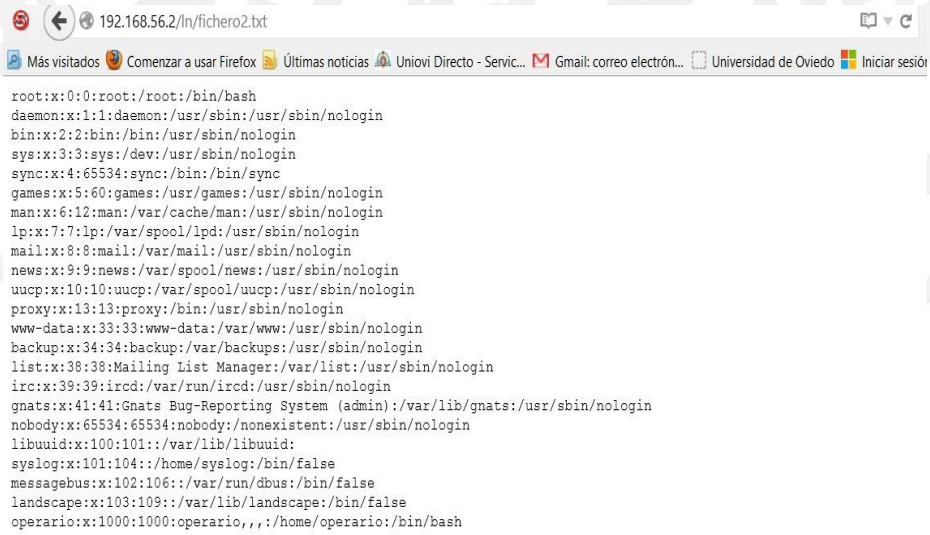


Figura 114. Fichero /etc/passwd revelado a través de un enlace simbólico

Estos son solo dos ejemplos de potenciales problemas de seguridad que podrían ocurrirnos por permitir seguir enlaces simbólicos. Si no hacemos uso de ellos, lo mejor es desactivarlos (no solo a nivel de directorio particular como aquí, sino a nivel de todo el servidor), como se ve en la siguiente imagen:

```
<Directory "/var/www/html/ln">
        Options -FollowSymLinks
</Directory>
```

Figura 115. Prohibición de enlaces simbólicos

Hecho esto, aunque consiguieran subirnos un enlace simbólico al servidor, no podría seguirse:

Figura 116. Acceso a un enlace simbólico en una localización que no los permite

Limitación del tamaño de las peticiones y medidas anti-DoS básicas

Los ataques de denegación de servicio contra servidores web son hoy en día muy frecuentes. El propósito de esta clase de ataques es más amplio de lo que puede parecer en un principio:

- Obviamente, el primer efecto es **evitar que una empresa dé servicio a sus clientes**.
- Si lo primero no es posible, entonces se **trata de hacer que la calidad de ese servicio sea lo menos aceptable posible**.
- Con los dos puntos anteriores, se consigue también una **pérdida de reputación y de imagen** del cliente, cuyo perjuicio es mayor del que puede parecer en un principio.
- Si nuestro servicio de hosting es un proveedor en nube que nos cobra por uso (algo muy frecuente) estos "picos" de tráfico inútil (porque no nos reporta ningún beneficio) **pueden dar lugar a pérdidas económicas importantes** si no se gestiona adecuadamente o el proveedor no ofrece una medida de protección efectiva contra ellos.

Por tanto, un ataque *DoS* tiene perjuicios económicos, de servicio y reputación potencialmente devastadores para una empresa. Por ello, estos problemas deben ser tratados en varios frentes:

1. El primero **contratar un servicio de hosting que ofrezca protección contra estos ataques** en aquellos puntos de la comunicación con nuestro servidor que probablemente escapen a nuestro control como administradores. Estos elementos podrían ser *firewalls* o detectores de intrusiones (IDS) que limiten el nº de conexiones que se realizan en un determinado periodo de tiempo desde una IP o red concreta. Obviamente, en este aspecto es probable que no podamos hacer gran cosa si no tenemos acceso a estas infraestructuras, pero en todo caso tratar estos temas escapa de los objetivos de este libro. ***Cloudflare*** (https://www.cloudflare.com/es-es/) y ***TransparentEdge*** (https://www.transparentedge.eu/) son dos de estos proveedores.
2. Segundo, **instalar un software específico que nos proteja** en la medida de lo posible contra estos ataques, como veremos posteriormente con `mod_evasive`.
3. Y tercero, y en combinación con el resto, **modificar ciertos parámetros del servidor** para que este se comporte de la mejor forma posible ante subidas súbitas de tráfico.

Este último será el aspecto que veremos en esta sección, describiendo los parámetros que se pueden configurar al respecto y su utilidad, siempre teniendo en cuenta que **es una capa más de protección** y no una forma definitiva de solventar estos problemas, sobre todo teniendo en cuenta que ante un ataque de *DoS* bien orquestado es prácticamente imposible evitar que nos afecte aunque sea en menor medida. Las directivas a modificar son:

- `RequestReadTimeout`: Establece un ti**empo máximo que el cliente puede tardar en enviar su petición una vez establecida la conexión**. Esto intenta solventar las denegaciones de servicio causadas por la técnica de establecer conexiones válidas y enviar el resto de la petición con un retardo pre-programado para así saturar el nº de conexiones máximas aceptables por el servidor. Nótese que las conexiones SSL deben tener un valor mayor por el tiempo que se tarda en negociar el cifrado. **Necesita tener activo el** `mod_reqtimeout` y puede establecerse a nivel de todo el servidor o por `VirtualHost`. Se recomienda bajar manualmente su valor lo más posible, siempre que ninguno de nuestros clientes legítimos nos informe de problemas de *timeout* durante el trabajo normal con la web.

- **Más información**: http://httpd.apache.org/docs/current/mod/mod_reqtimeout.html#requestreadtimeout

- `TimeOut`: Tiempo máximo que *Apache 2* **esperará a que determinadas acciones terminen** en el transcurso del ciclo de servicio de peticiones que hemos visto. Se recomienda establecer su valor entre unos 10-20 segundos, en función del historial de ataques *DoS* de la web.
  - **Más información**: http://httpd.apache.org/docs/current/mod/core.html#timeout
- `KeepAliveTimeout`: Por temas de rendimiento (crear una conexión es una operación costosa), cuando un cliente crea una conexión para hacer una petición, dicha conexión **se mantiene abierta** (*keep alive*) por si el cliente enviase más peticiones posteriormente, de manera que no se abra una conexión nueva por petición. Esto se controla con la directiva `KeepAlive`, recomendándose que esté activa (`On`) en la mayor parte de casos (http://httpd.apache.org/docs/current/mod/core.html#keepalive).

  No obstante, mantenerlas abiertas indefinidamente puede traernos problemas de rendimiento y de denegación de servicio (máximo de conexiones alcanzado), por lo que este parámetro nos permite determinar cuánto tiempo mantenemos una conexión abierta. Se recomienda fijar su valor a 2, aunque algunos sitios con un tráfico muy elevado o que están siendo sometidos en estos instantes a un *DoS* llegan a desactivar este parámetro (a costa del rendimiento, claro).
  - **Más información**: http://httpd.apache.org/docs/current/mod/core.html#keepalivetimeout
- `LimitRequestBody`, `LimitRequestFields`, `LimitRequestFieldSize`, `LimitRequestLine`, y `LimitXMLRequestBody` son directivas para **limitar el tamaño total de las peticiones HTTP** o bien de algunas de sus partes (tamaño máximo de una petición HTTP, nº máximo de campos en la cabecera de la petición HTTP, tamaño máximo de estos los campos, tamaño máximo de la combinación de método HTTP + URI + protocolo y tamaño máximo de una petición en formato XML, respectivamente).

  Fijar un valor de 0 significa que **no hay un tamaño máximo** en muchas de ellas, algo que no se recomienda porque enviar peticiones muy grandes es una de las formas más típicas de colapsar un servidor. Los valores de estas directivas deben estudiarse cuidadosamente para adaptarse a las peticiones de los clientes esperables, reduciéndolos lo más posible. Por ejemplo, si la petición más grande que un cliente puede hacer es subir un fichero cuyo tamaño máximo son 5 Mb, el tamaño de las peticiones debe fijarse algo por encima de este límite, ya que nunca será necesario enviar más información. La misma política ha de seguirse con peticiones XML. Valores recomendables para otras primitivas son 40 para `LimitRequestFields` y 4096 para `LimitRequestFieldSize` y `LimitRequestLine`.
  - **Más información**: http://httpd.apache.org/docs/current/mod/core.html#limitrequestbody
- La directiva `AcceptFilter` es una optimización que permite **pasar parte de la carga de trabajo de *Apache 2* al SO**, mejorando el rendimiento. Está activa por defecto en muchos SO, pero conviene comprobar que no se ha desactivado si el SO lo soporta. Diversos SO soportan distintos valores para esta directiva.
  - **Más información**: http://httpd.apache.org/docs/current/mod/core.html#acceptfilter
- La directiva `MaxRequestWorkers` permite establecer **el máximo nº de conexiones simultáneas** que se pueden establecer. Debe fijarse manualmente en función de la memoria disponible para el servidor, teniendo en cuenta que debería evitarse a toda costa que el servidor tuviese que hacer swap para atender sus peticiones, ya que la penalización de rendimiento que eso supondría es muy grande.

- **Más información**: http://httpd.apache.org/docs/current/mod/mpm_common.html#maxrequestworkers

Finalmente, se deben tener en cuenta o**tras consideraciones de seguridad**

- Si otros módulos de terceros que atienden peticiones tienen directivas de *timeout*, debe intentar optimizarse su valor también como hemos hecho con las mencionadas.
- Si el MPM usado por *Apache 2* no es `event` (el que se usa en *Ubuntu* por defecto en muchos *Linux*), conviene consultar y optimizar el valor de otras directivas que ajustan el rendimiento de esos MPM, información que podemos encontrar aquí: http://httpd.apache.org/docs/current/es/mod/mpm_common.html
- Como al final el nº de peticiones atendidas **es proporcional al rendimiento del servidor**, en último caso conviene comprobar que estamos cumpliendo con todas las recomendaciones oficiales para mejorar el rendimiento del mismo: http://httpd.apache.org/docs/current/misc/perfhttp://httpd.apache.org/docs/current/misc/perf-tuning.htmltuning.html

Exceptuando la directiva que forma parte de la funcionalidad de `mod_reqtimeout`, el resto de las directivas vistas pueden configurarse en el fichero de configuración central de *Apache 2* `apache2.conf`. Algunas ya están presentes con un valor determinado, otras no lo están pero podríamos añadirlas.

Es difícil establecer un valor "bueno" único para cada una de estas primitivas porque hay muchos factores de las que dependen: SO, tipo de MPM, cantidad de peticiones que se reciben, tipo de webs servidas, etc. Conviene pues estudiar cada caso y configurarlas de acuerdo con las necesidades concretas de un determinado servidor.

```
  GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf

Timeout 300

#
# KeepAlive: Whether or not to allow persistent connections (more than
# one request per connection). Set to "Off" to deactivate.
#
KeepAlive On

#
# MaxKeepAliveRequests: The maximum number of requests to allow
# during a persistent connection. Set to 0 to allow an unlimited amount.
# We recommend you leave this number high, for maximum performance.
#
MaxKeepAliveRequests 100

#
# KeepAliveTimeout: Number of seconds to wait for the next request from the
# same client on the same connection.
#
KeepAliveTimeout 5
```

Figura 117. Las directivas que permiten mitigar problemas de DoS se encuentran en Apache2.conf

Configuración y uso del *log*

Para hacer un seguimiento de las peticiones que llegan a *Apache 2* a lo largo del tiempo hay que revisar sus archivos de *log*. Una instalación por defecto de *Apache 2* típicamente tiene los siguientes archivos de log:

- `/var/log/apache2/access.log`: Registro de los accesos que se han hecho a páginas web que alojamos en el servidor
- `/var/log/apache2/error.log`: Registro general de errores de *Apache 2*: errores de módulos, configuración, módulos, bloqueos diversos, problemas de seguridad potenciales detectados, etc.

Algunos módulos de *Apache 2* **crean sus propios ficheros de log**. También es posible cambiar los ficheros de log por defecto o habilitar otros en función de las funcionalidades que activemos en el servidor. En cualquier caso, tener activo un *log* es primordial para saber qué clase de actividades ocurren en nuestro servidor y si mantenemos un nivel de seguridad adecuado.

La existencia de un *log* adecuadamente configurado **es imprescindible en caso de ataques**, ya que el registro de las actividades es completamente necesario para saber que ha pasado, cuándo, cómo y si, por tanto, puede considerarse una actividad maliciosa. En caso de que hay alguna clase de ataque, **el análisis de los ficheros de *log*** (bien por nuestra parte o de las autoridades pertinentes) es una herramienta imprescindible para esclarecer lo que ha pasado. El *log* de peticiones de *Apache 2* es muy detallado. La siguiente imagen muestra el contenido típico del mismo:

```
 GNU nano 2.2.2                File: access.log

192.168.56.1 - - [29/Jul/2010:11:09:40 +0200] "GET / HTTP/1.1" 200 484 "-" "Moz$
192.168.56.1 - - [29/Jul/2010:11:09:41 +0200] "GET /favicon.ico HTTP/1.1" 404 5$
192.168.56.1 - - [29/Jul/2010:11:09:42 +0200] "GET /favicon.ico HTTP/1.1" 404 5$
192.168.56.1 - - [29/Jul/2010:11:09:44 +0200] "GET /favicon.ico HTTP/1.1" 404 5$
192.168.56.1 - - [29/Jul/2010:12:25:41 +0200] "GET /login HTTP/1.1" 404 501 "-"$
192.168.56.1 - - [29/Jul/2010:12:25:45 +0200] "GET / HTTP/1.1" 304 209 "-" "Moz$
```

Figura 118. Log de Apache 2

Por tanto, los ficheros de *log* deberían ser analizados frecuentemente para buscar información interesante en determinados contextos, especialmente si se tienen indicios de que ha habido un ataque o actividad maliciosa. Además, existen búsquedas conocidas para identificar determinados problemas o intentos de ataque. Por ejemplo:

- Una orden como ésta: `grep -c "/jsp/source.jsp?/jsp/ /jsp/source.jsp??" access_log` muestra el número de ataques que tratan de explotar la vulnerabilidad *"Apache 2 Tomcat Source.JSP Malformed Request Information Disclosure Vulnerability"* (http://www.securityfocus.com/bid/4876/exploit).
- Esta orden: `grep "client denied" error_log | tail -n 20` muestra los 20 últimos clientes a los que se le ha denegado el acceso, que producirán en el *log* líneas como:

```
[Thu Jul 11 17:18:39 2002] [error] [client foo.bar.com] client denied by server
configuration: /usr/local/apache2/htdocs/.htpasswd
```

- Para averiguar si un cliente ha intentado acceder a un archivo "sensible", como el fichero `mywebsite.conf`, buscaremos en el log líneas como: `miweb.com - - [11/Jul/2015:01:59:13`

`+0200] "GET /mywebsite.conf HTTP/1.1`. Lo mismo puede aplicarse a cualquier archivo "sensible" de *Apache 2* o de la aplicación (en función de las tecnologías usadas para la implementación de la web). Los intentos de acceso a ficheros que no deberían ser accedidos por ningún cliente legítimo (como configuraciones, carpetas no públicas conocidas como `/admin`, etc.) son prueba inequívoca de que alguien está intentando "buscar las cosquillas" a nuestro sistema.

Estos son solo algunos ejemplos del tipo de búsquedas que se pueden hacer en ficheros de log. No obstante, a no ser que busquemos cosas concretas, analizar logs "a mano" puede ser muy tedioso. Es aconsejable usar **herramientas automatizadas**, como `Awstats` (http://awstats.sourceforge.net/), que analizan los logs en busca de información que pueda identificar un posible ataque o bien un comportamiento malicioso por parte de algún usuario.

Análisis de *log* con *awstats*

`awstats` es una herramienta de análisis de log y que muestra unos resultados mucho más detallados, complejos y útiles. A continuación vamos a ver un ejemplo básico de uso de `awstats`, pero el nº de opciones que posee es muy superior al que vamos a ver aquí y, por tanto, en caso de usar la herramienta para algo más que un mero análisis de seguridad a petición del cliente (por ejemplo, para hacer una monitorización de accesos continuada) se tiene que recurrir a consultar su documentación oficial para lograrlo. No obstante, para profundizar más en este tema se recomienda consultar los siguientes enlaces:

- http://blogging.dragon.org.uk/installing-awstats-on-ubuntu-14-04-lts/
- https://www.howtoforge.com/community/threads/howto-run-awstats-from-the-commandhttps://www.howtoforge.com/community/threads/howto-run-awstats-from-the-command-line.49603/line.49603/

El primer paso es instalar la herramienta y todos los paquetes necesarios para su funcionamiento, como se ve en la siguiente imagen:

```
operario@ubuntu:~$ sudo apt-get install awstats libgeo-ipfree-perl libnet-ip-per
l
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  libnet-xwhois-perl
Paquetes sugeridos:
  libnet-dns-perl liburi-perl
Se instalarán los siguientes paquetes NUEVOS:
  awstats libgeo-ipfree-perl libnet-ip-perl libnet-xwhois-perl
0 actualizados, 4 se instalarán, 0 para eliminar y 3 no actualizados.
Necesito descargar 1.403 kB de archivos.
Se utilizarán 4.492 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n] _
```

Figura 119. Instalación de awstats

Una vez instalada la herramienta, podemos consultar su fichero de configuración para `/etc/awstats/awstats.conf` para ver una descripción muy completa de todas las opciones que incluye a través de sus comentarios. No obstante, en nuestras pruebas lo único que cambiaremos será la opción `SiteDomain` para indicar cuál es la web que queremos analizar. Es importante tener en cuenta que en configuraciones más complejas, donde puedan existir múltiples webs por servidor, es posible tener una

configuración independiente de `awstats` por cada web alojada copiando el fichero de configuración central a otro llamado `awstats.<nombre de dominio de la web>.conf` y cambiándolo independientemente del resto. Para que el resto de los comandos que vamos a ver funcionen con una web particular solo es necesario añadir la opción `-config=<nombre del dominio>` a cada uno de ellos:

```
 GNU nano 2.2.6       Archivo: /etc/awstats/awstats.conf          Modificado

# server name, used to reach the web site.
# If you share the same log file for several virtual web servers, this
# parameter is used to tell AWStats to filter record that contains records for
# this virtual host name only (So check that this virtual hostname can be
# found in your log file and use a personalized log format that include the
# %virtualname tag).
# But for multi hosting a better solution is to have one log file for each
# virtual web server. In this case, this parameter is only used to generate
# full URL's links when ShowLinksOnUrl option is set to 1.
# If analyzing mail log, enter here the domain name of mail server.
# Example: "myintranetserver"
# Example: "www.domain.com"
# Example: "ftp.domain.com"
# Example: "domain.com"
#
SiteDomain="192.168.56.2"
```

Figura 120. Preparando la configuración de awstats

La primera vez que arrancamos `awstats` conviene hacer una actualización de sus datos como se ve en la siguiente imagen:

```
operario@ubuntu:~$ sudo /usr/lib/cgi-bin/awstats.pl -update
```

Figura 121. Primer arranque de awstats

```
Other options:
  -debug=X     to add debug informations lesser than level X (speed reduced)

Now supports/detects:
  Web/Ftp/Mail/streaming server log analyzis (and load balanced log files)
  Reverse DNS lookup (IPv4 and IPv6) and GeoIP lookup
  Number of visits, number of unique visitors
  Visits duration and list of last visits
  Authenticated users
  Days of week and rush hours
  Hosts list and unresolved IP addresses list
  Most viewed, entry and exit pages
  Files type and Web compression (mod_gzip, mod_deflate stats)
  Screen size
  Ratio of Browsers with support of: Java, Flash, RealG2 reader,
                     Quicktime reader, WMA reader, PDF reader
  Configurable personalized reports
  276 domains/countries
  752 robots
  5 worm's families
  80 operating systems
  211 browsers (592 with phone browsers database)
  248 search engines (and keyphrases/keywords used from them)
  All HTTP errors with last referrer
  Report by day/month/year
  Dynamic or static HTML or XHTML reports, static PDF reports
  Indexed text or XML monthly database
  And a lot of other advanced features and options...
New versions and FAQ at http://www.awstats.org
operario@ubuntu:~$
```

Figura 122. awstats actualizado

Para ver los datos que nos proporciona la herramienta hay varias opciones. La herramienta admite ser programada mediante cron para que se ejecute cada X tiempo. Además admite ver todos sus datos vía web a través del propio servidor que aloja las webs de usuario normales (normalmente a través de una ruta como `http://<nombre del dominio>/awstats`).

No vamos a usar ninguna de estas opciones, especialmente porque **la segunda supone un riesgo de seguridad muy importante** al dejar visibles estadísticas y datos de uso muy privados de nuestro servidor. En caso de querer usar esta opción se recuerda que el propio fichero de configuración de la herramienta tiene una opción para restringir las IPs desde las que se pueden ver los datos que genera, al estilo de lo que hicimos con `mod_status`.

En nuestro caso vamos a usar la opción de generar manualmente las estadísticas y su análisis en páginas HTML autogeneradas. Las crearemos sobre un directorio cualquiera que luego podemos consultar off-line, lejos del acceso de cualquier usuario (aunque en nuestro ejemplo, por simplicidad, lo generemos sobre la carpeta donde se alojan las webs, esto no debe hacerse **NUNCA** en un entorno real). Para generar la información, basta con ejecutar el siguiente programa distribuido conjuntamente con `awstats`:

```
operario@ubuntu:/var/www/html$ sudo /usr/share/awstats/tools/awstats_buildstatic
pages.pl -config=awstats.conf -update -awstatprog=/usr/lib/cgi-bin/awstats.pl -d
ir=/var/www/html
```

Figura 123. Generando las estadísticas de awstats

Vemos como el destino es la carpeta raíz para alojar web de *Apache 2*, que insistimos que solo se hace por simplicidad y no debemos hacerlo nunca. La herramienta genera gran cantidad de información:

```
nks -output=session
Build urldetail page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -static
links -output=urldetail
Build urlentry page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -staticl
inks -output=urlentry
Build urlexit page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -staticli
nks -output=urlexit
Build osdetail page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -staticl
inks -output=osdetail
Build unknownos page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -static
links -output=unknownos
Build browserdetail page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -st
aticlinks -output=browserdetail
Build unknownbrowser page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -s
taticlinks -output=unknownbrowser
Build downloads page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -static
links -output=downloads
Build refererse page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -static
links -output=refererse
Build refererpages page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -sta
ticlinks -output=refererpages
Build keyphrases page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -stati
clinks -output=keyphrases
Build keywords page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -staticl
inks -output=keywords
Build errors404 page: "/usr/lib/cgi-bin/awstats.pl" -config=awstats.conf -static
links -output=errors404
21 files built.
Main HTML page is 'awstats.awstats.conf.html'.
operario@ubuntu:/var/www/html$ _
```

Figura 124. Información generada por awstats

Y finalmente podemos ver claramente que la herramienta es capaz de hacer un análisis de los datos de nuestros logs muy completo y exhaustivo, muy útil para identificar potenciales problemas:

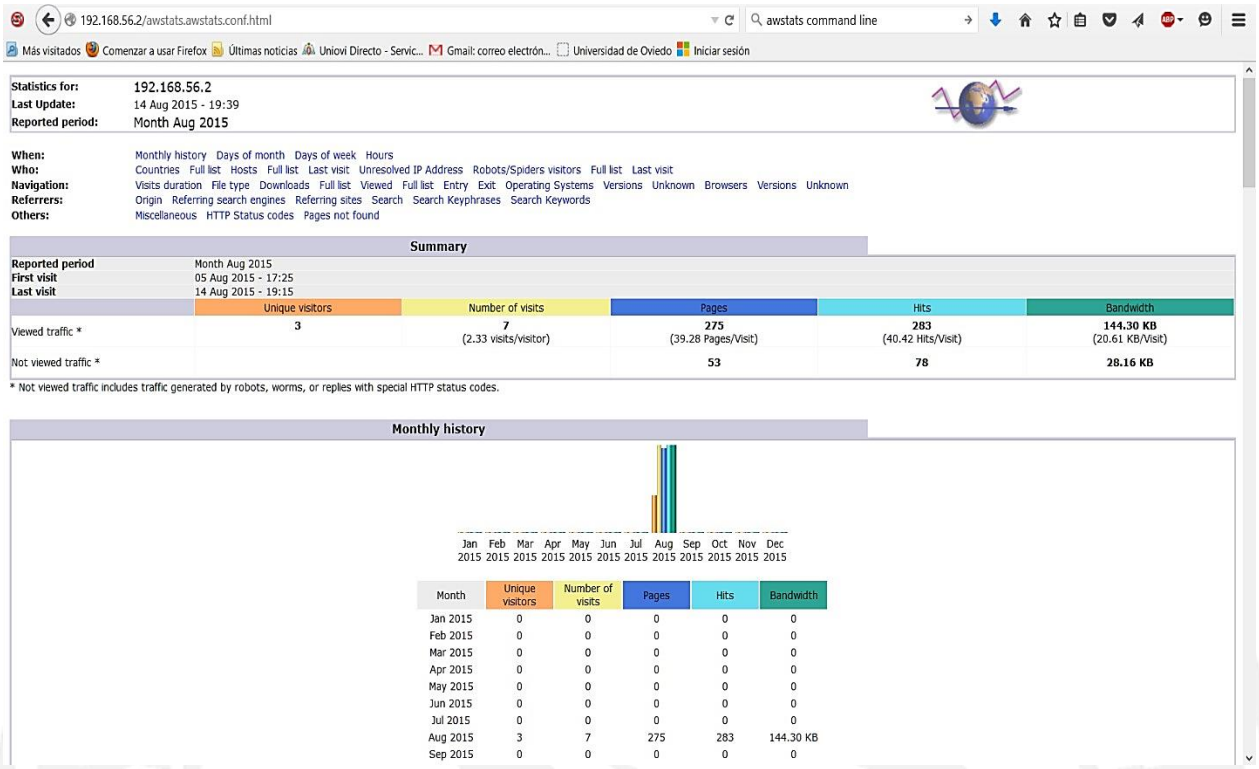


Figura 125. Información de awstats

Otros programas para analizar *logs*

Otras herramientas similares a *Awstats* son:

- ***EventLog Analyzer*** (https://demo.eventloganalyzer.com/): Herramienta que ofrece funciones completas para el análisis de los logs del servidor web Apache 2. Puede auditar la actividad del servidor, detectar la actividad sospechosa de los usuarios, recibir alertas en tiempo real y obtener informes y gráficos detallados.
- ***GoAccess*** (https://goaccess.io/): Herramienta de código abierto que permite el análisis y la representación visual de estadísticas web en tiempo real. Evalúa los distintos formatos de archivos log de servidores web y servicios en la nube como *Apache 2*, *Nginx*, *Amazon S3* o *CloudFront* y presenta los resultados gráficamente en un panel de control o en formato HTML, JSON o CVS.

Generación de páginas de error personalizadas

Apache 2 tiene la capacidad de **mostrar páginas de error genéricas** ante todo tipo de errores HTTP 400 y 500. Estas páginas **muestran más información técnica de la debida a los usuarios**, lo cual no está bien desde el punto de vista de la seguridad, pero es cierto que al mismo tiempo no son muy agradables a la vista de un usuario estándar.

Por ese motivo, es muy común que cualquier desarrollador/administrador tenga que desarrollar páginas de error **más amigables**, internacionalizadas o bien que sigan la guía de estilos de la web o de la empresa, y en ese caso también es necesario que haya una forma de indicar al servidor qué página mostrar ante determinados errores HTTP además de lo que se haga mediante programación en la propia página. En esta sección vamos a describir cómo usar esta característica, pudiéndose encontrar toda la información relativa a la misma en http://httpd.apache.org/docs/current/custom-error.html.

A la hora de generar páginas de error personalizadas, lo primero que debemos tener en cuenta es que se permite crear una página de error por cada posible código de error HTTP de la serie 400 o 500 que necesitemos mediante la directiva `ErrorDocument`. Esta directiva puede usarse a nivel de servidor global, host virtual o incluso directorio individual. Para poder usarlo en ficheros `.htaccess` tendríamos que asegurarnos de que el directorio que contiene dicho fichero tiene la directiva `AllowOverride +FileInfo`.

La **sintaxis** de la directiva es la siguiente:

```
ErrorDocument <código de error HTTP 4xx o 5xx> <acción>
```

Posibles acciones:

- Una URI local a la que redirigir en caso de error (si la acción comienza por /).
- Una URI remota a la que redirigir en caso de error.
- Un texto a mostrar (entre comillas dobles si es más de una palabra)

Ejemplos:

```
ErrorDocument 500 "Sayonara, Baby"
ErrorDocument 500 /errores/registro_de_fallos
ErrorDocument 500 http://servidordeerrores.org/error_500_estandar.html ErrorDocument 404
/errores/404_not_found.html
```

Veamos un sencillo ejemplo mostrando un simple mensaje cuando se accede a una página que no existe. Para ello cambiamos la configuración de la siguiente forma:

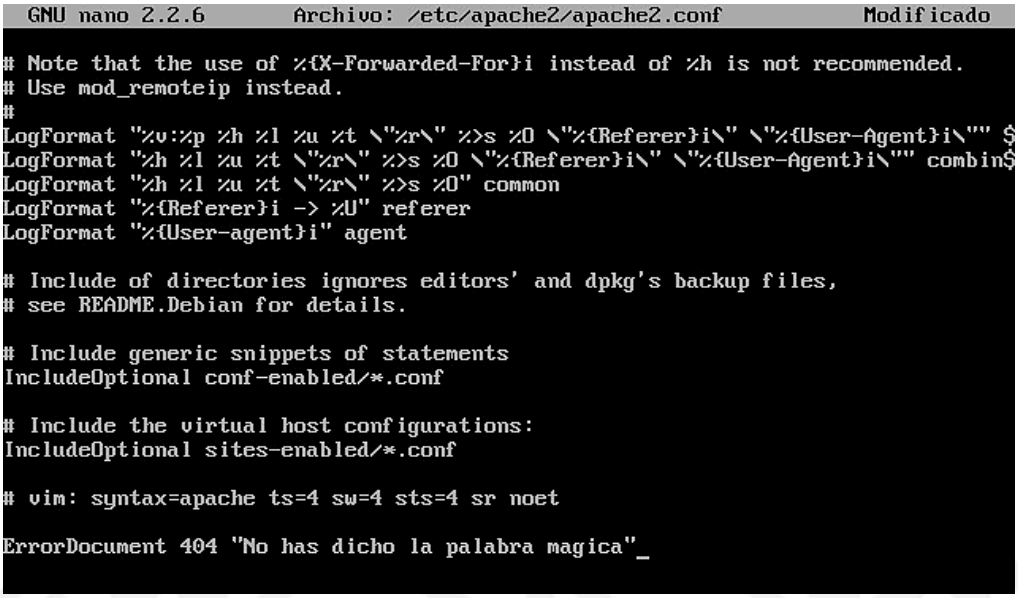


Figura 126. Error 404 personalizado

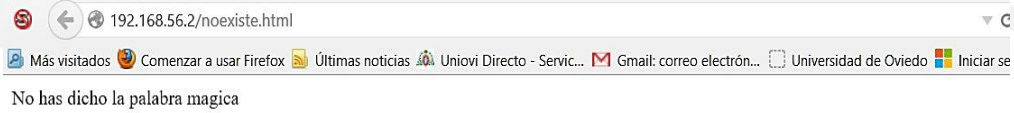


Figura 127. Resultado de un error 404 personalizado

Variables de entorno

Para ampliar el mecanismo básico de generación de páginas de **error podemos hacer uso de una serie de variables de entorno** cuyo valor se establece para así poder generar páginas de error más personalizadas. Es importante recordar que esto puede hacerse si la URI del error es local, **estas variables NUNCA se establecen para URIs externas**, por posibles implicaciones de seguridad.

Por otro lado, las variables se generan a partir de variables existentes, copiando su valor y anteponiendo a su nombre la palabra `REDIRECT_`. Las variables accesibles (y un ejemplo de sus valores) son:

Variable	Ejemplo de valor
REDIRECT_HTTP_ACCEPT	*/*, image/gif, image/jpeg, image/png
REDIRECT_HTTP_USER_AGENT	Mozilla/5.0 (Windows; U; Windows NT 6.1; rv:2.2) Gecko/20110201
REDIRECT_PATH	=.:/bin:/usr/local/bin:/sbin
REDIRECT_QUERY_STRING	id=250;q=search
REDIRECT_REMOTE_ADDR	192.168.10.1
REDIRECT_REMOTE_HOST	operario.miweb.com
REDIRECT_SERVER_NAME	www.miweb.com
REDIRECT_SERVER_PORT	443
REDIRECT_SERVER_SOFTWARE	Apache 2/2.4
REDIRECT_URL	/errores/procesar_error.py

Hay que tener en cuenta por otro lado que de todas estas variables las únicas que seguro que tienen un valor son `REDIRECT_URL`, `REDIRECT_STATUS` y `REDIRECT_QUERY_STRING` (si la URI de la petición tiene parámetros). El resto tendrán valor si existían previamente a la generación del error.

De esta forma podremos generar páginas de error con cualquier tecnología de desarrollo de páginas dinámicas que tengamos instalada en el servidor usando los valores de estas variables de entorno. A continuación veremos un ejemplo con Python, instalado previamente para otras funciones.

```
print "Content-type: text/html\n";
printf "Se ha producido un error. Status: %s\n", $ENV{"REDIRECT_STATUS"};
```

Una última cosa a tener en cuenta es que las páginas de error generadas dinámicamente deben incluir una cabecera `Status` para que el navegador sepa que la página que está viendo es consecuencia de un código de error. El valor para poner en esta cabecera lo podemos leer de la variable `REDIRECT_STATUS` mencionada antes.

Finalmente, es importante destacar que si tenemos activos `mod_include`, `mod_alias` y `mod_negotation` podemos **generar páginas de error en varios idiomas**. *Apache 2* viene de serie con páginas de error traducidas a 16 idiomas, que podemos usar si incluimos el fichero de configuración apropiada `localizederror-pages.conf` del directorio `conf-enabled`. *Apache 2* está preparado para incluir por defecto todos los ficheros `.conf` de `conf-enabled`, por lo que esto ya se está haciendo.

Si no se han usado páginas de error internacionalizadas hasta el momento es debido a que el fichero en cuestión **se encuentra completamente comentado**. Estos documentos de error hacen uso de las variables vistas anteriormente, por lo que podemos informar al usuario de más datos si lo creemos oportuno. También valen como "plantilla" a partir de la cual desarrollar nuestros propios documentos de error. Para usar estas páginas internacionalizadas basta pues por quitar los comentarios la sección que las gestiona, como se ve en la siguiente imagen:

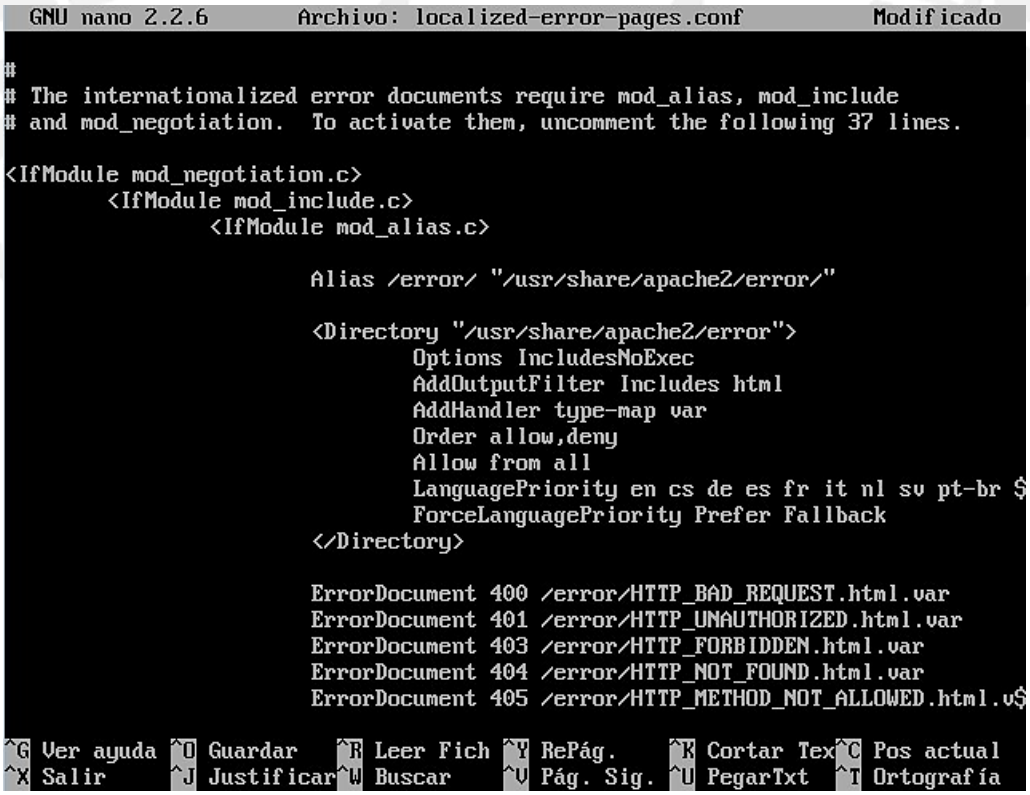


Figura 128. Activación de las páginas de error internacionalizadas

Y como vemos, *Apache 2* incluirá este fichero gracias a su configuración por defecto. Esto quiere decir que si no funcionasen las páginas el motivo será muy probablemente que no están activados los tres módulos necesarios mencionados.

```
# Include generic snippets of statements
IncludeOptional conf-enabled/*.conf

# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf
```

Figura 129. Inclusión de todos los ficheros de configuración de conf-enabled

Finalmente, podemos ver como efectivamente *Apache 2* nos muestra ahora una página de error internacionalizada en español dada la configuración de nuestro navegador de pruebas:

Figura 130. Página de error internacionalizada

Bloque 3: Mejora de la Seguridad del Servidor

Configuración de restricciones de acceso

Una de las principales labores de todo administrador web es determinar qué contenidos son accesibles y qué contenidos no lo son en una determinada web alojada. *Apache 2*, obviamente, proporciona medios para realizar esta tarea que tenemos que aprender a usar adecuadamente. El problema, es que actualmente conviven dos formas de hacerlo, el **antiguo y obsoleto** `mod_access`, que actualmente puede usarse gracias a la implementación de este existente en `mod_access_compat`, y el **más moderno** `mod_authz_host`, que en la versión 2.4+ ha evolucionado de un esquema más similar a su antecesor a uno nuevo y distinto que estudiaremos a continuación.

Por otro lado, si debemos elegir entre ambos a la hora de hacer una instalación nueva siempre es más aconsejable usar el último, pero existen multitud de servidores que todavía usan el esquema clásico de `mod_access`, por lo que hace necesario que lo revisemos.

Módulo *mod_access_compat*

Este módulo proporciona **control de acceso** basado en el nombre del *host*, dirección *IP* u otras características de la petición del cliente (capturadas mediante variables de entorno). Las directivas de `mod_access_compat` pueden aplicarse dentro de secciones `<Directory>`, `<Files>`, `<Location>` y en ficheros `.htaccess`, permitiendo así limitar el acceso al nivel que deseemos. De esta forma, el acceso se puede controlar sólo en ciertas partes del servidor o en el servidor completo. La gestión del acceso se hace mediante las siguientes directivas:

- `Allow` y `Deny`: Especifican a qué clientes se les permite o no acceder a esa parte del servidor.
- La palabra `from` seguida de una IP, red, nombre de host, valorable de entorno o `all` especifica el ámbito de aplicación de `Allow` y `Deny` a dichos elementos. Usar `all` significa aplicar la directiva a todos los clientes. Veremos ejemplos posteriormente.
- `Order`: Controla el estado de acceso por defecto y el orden de evaluación:
  - `Order allow,deny`:
    - Primero se evalúan todas las directivas `Allow`
    - Al menos una debe encajar, en caso contrario se deniega el acceso
    - Después se evalúan todas las directivas `Deny`, si una encaja se deniega el acceso
    - Por último, todas las peticiones que no encajen con ninguna directiva anterior se deniegan por defecto (**lista de permitidos**)
  - `Order deny,allow`:
    - Primero se evalúan todas las directivas `Deny`
    - Si alguna encaja la petición se deniega, salvo que encaje también en una directiva `Allow`
    - Todas las peticiones que no encajen con ninguna directiva son **permitidas** (**lista de bloqueo**, ¡CUIDADO!)

- **Directiva `Satisfy`**: Si un sitio está protegido por el mecanismo que estamos describiendo y por *password* proveniente de un módulo de autenticación (que veremos posteriormente), especifica si se deben satisfacer ambas restricciones (autenticación o autorización) o solo una de ellas para permitir el acceso al recurso pedido. Podemos encontrar más información en: http://httpd.apache.org/docs/current/mod/mod_access_compat.html#satisfy

Finalmente, hay que comentar que en general las directivas de control de acceso vistas se aplican a todos los métodos *HTTP* (`GET`, `PUT`, `POST`, etc.). Es posible no obstante restringir métodos concretos con secciones `<Limit>`: http://httpd.apache.org/docs/current/mod/core.html#limit

La política de restricción de *IPs* puede y debe trabajar en colaboración con otras herramientas de bloqueo (*firewalls*, *ACLs* de *routers*, filtrado *TCP/IP*,…) para una mejor protección contra el acceso de clientes no autorizados.

Ejemplos de uso

A continuación se presentan una serie de ejemplos que ilustran las diferentes opciones de `mod_access_compat` y el uso de sus directivas. Estos ejemplos están pensados para aplicarse sobre el fichero `000-default.conf` (la configuración del *virtual host* por defecto), aunque se podrán mover a todo el servidor (`apache2.conf`), otro fichero de configuración incluido o a otro *host* virtual que queramos:

Una *web* a la que sólo queremos dejar acceder a clientes de la red `156.35.X.X`, con máscara en notación CIDR:

```
<Location />
Order deny,allow
Deny from all
Allow from 156.35.0.0/16
</Location>
```

Restringir acceso sólo a una subred local entre máquinas (`5.0.0.X`), con máscara en notación CIDR:

```
<Location />
Order deny,allow
Deny from all
Allow from 5.0.0.0/16  </Location>
```

Prohibir el acceso sólo a un *host* cuya *IP* es `192.168.56.1`, permitiéndoselo a los demás:

```
<Location />
Order allow,deny
Deny from 192.168.56.1
Allow from all  </Location>
```

Conceder acceso solo a un *host* cuya *IP* es `5.0.0.1`, prohibiéndolo a los demás:

```
<Location/>
Order deny,allow
Deny from all
Allow from 5.0.0.1
</Location>
```

Conceder acceso en función del nombre de dominio del cliente que accede

```
Allow from departamentos.contabilidad.org
Allow from .org universidad.edu
```

Como vemos los nombres de dominio pueden encajar completamente o bien solo la terminación de estos. Veremos la política aplicada para esto con `mod_authz_host` posteriormente, ya que se usa la misma.

Permitir a varias direcciones IP concretas

```
Allow from 192.168.10.4 192.168.4.205
```

Permitir a IPs parciales

```
Allow from 10.1
Allow from 10 200.20 192.168.20
```

Se comienza a comparar por los primeros bytes de las IP

Acceso permitido por red/máscara

```
Allow from 18.1.0.0/255.255.0.0
```

Acceso permitido por IPv6:

```
Allow from 2001:db8::a00:20ff:fea7:ccea
Allow from 2001:db8::a00:20ff:fea7:ccea/10
```

Acceso por presencia (o no presencia) de variables de entorno

Para implementar este tipo de acceso debemos tener en cuenta

- Para poder definir una variable de entorno se deben usar las directivas de `mod_setenvif`
- SI se especifica `Allow from env=<nombre de la variable de entorno>` la petición se permite si la variable ha sido definida
- Si en cambio se especifica `Allow from env=!<nombre de la variable de entorno>` la petición se permite si la variable NO ha sido definida
- Para definir o no una variable se pueden usar cabeceras HTTP como `Referer` o `User-Agent`, teniendo en cuenta que esta información puede ser cambiada por el cliente con algún fin malicioso (por ejemplo para saltarse precisamente esta restricción). Esto hace que esta forma de autorizar deba usarse con mucha precaución por temas de seguridad
- Veremos un ejemplo concreto con `mod_autz_host`, que usa la misma aproximación de autorización con variables de entorno.

```
SetEnvIf User-Agent ^Firefox eres_un_mozilla
<Directory "/mozilla_only">
    Order Deny,Allow
    Deny from all
    Allow from env=eres_un_mozilla
</Directory>
```

Y para ilustrar en nuestra infraestructura de pruebas cómo funciona este módulo, pondremos en práctica los siguientes ejemplos sencillos.

Permitir el acceso a todo el mundo salvo a la IP de la máquina host al fichero `contenido.txt` del directorio `ac_deniedfor1`:

- Comprobamos que podemos acceder al contenido:

Figura 131. Acceso al contenido de un directorio sin control de permisos

```
operario@servidorubuntu:~$ wget 192.168.23.1/ac_deniedfor1/contenido.txt
--2015-07-06 09:23:38--  http://192.168.23.1/ac_deniedfor1/contenido.txt
Conectando con 192.168.23.1:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 22 [text/plain]
Grabando a: "contenido.txt"

100%[======================================>] 22          --.-K/s   en 0s

2015-07-06 09:23:38 (5,09 MB/s) - "contenido.txt" guardado [22/22]

operario@servidorubuntu:~$ cat contenido.txt
Contenido de la ruta

operario@servidorubuntu:~$ _
```

Figura 132. Acceso al contenido de un directorio sin control de permisos vía wget

- Editamos un fichero `.conf` para establecer la restricción (en este caso `apache2.conf`, por ejemplo) y reiniciamos *Apache 2*:

```
 GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf          Modificado

<Directory />
        Options FollowSymLinks
        AllowOverride None
        Require all denied
</Directory>

<Directory /usr/share>
        AllowOverride None
        Require all granted
</Directory>

<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>

<Directory /var/www/html/ac_deniedfor1>
        Order allow,deny
        Allow from all
        Deny from 192.168.56.1
</Directory>
```

Figura 133. Modificación de permisos con mod_access_compat para prohibir una IP concreta

- Comprobamos que el acceso está permitido sólo para las máquinas adecuadas

Figura 134. Acceso prohibido por mod_access_compat

```
operario@servidorubuntu:~$ rm contenido.txt
operario@servidorubuntu:~$ wget 192.168.23.1/ac_deniedfor1/contenido.txt
--2015-07-06 09:29:24--  http://192.168.23.1/ac_deniedfor1/contenido.txt
Conectando con 192.168.23.1:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 22 [text/plain]
Grabando a: “contenido.txt”

100%[======================================>] 22          --.-K/s   en 0s

2015-07-06 09:29:24 (3,84 MB/s) - “contenido.txt” guardado [22/22]

operario@servidorubuntu:~$ cat contenido.txt
Contenido de la ruta

operario@servidorubuntu:~$ _
```

Figura 135. mod_access_compat permitiendo una sola IP aislada

Permitir el acceso sólo a la IP de la máquina host al fichero contenido.txt de la carpeta ac_allowedfor1:

- Editamos un fichero `.conf` para establecer la restricción (en este caso `apache2.conf`, por ejemplo) y reiniciamos *Apache 2*:

```
 GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf          Modificado


<Directory /usr/share>
        AllowOverride None
        Require all granted
</Directory>

<Directory /var/www/>
        Options Indexes FollowSymLinks
        AllowOverride None
        Require all granted
</Directory>

<Directory /var/www/html/ac_deniedfor1>
        Order allow,deny
        Allow from all
        Deny from 192.168.56.1
</Directory>

<Directory /var/www/html/ac_allowedfor1>
        Order deny,allow
        Deny from all
        Allow from 192.168.56.1
</Directory>_
```

Figura 136. Acceso autorizado sólo a una IP particular

- Comprobamos que el acceso está permitido sólo para las máquinas adecuadas

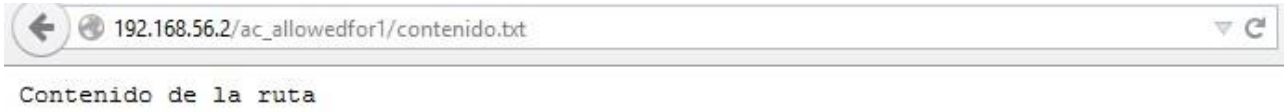


Figura 137. Acceso permitido al cliente concreto

```
operario@servidorubuntu:~$ wget 192.168.23.1/ac_allowedfor1/contenido.txt
--2015-07-06 10:06:07--  http://192.168.23.1/ac_allowedfor1/contenido.txt
Conectando con 192.168.23.1:80... conectado.
Petición HTTP enviada, esperando respuesta... 403 Forbidden
2015-07-06 10:06:07 ERROR 403: Forbidden.

operario@servidorubuntu:~$
```

Figura 138. Acceso prohibido al resto de clientes

Módulo *mod_authz_host*

Como hemos dicho anteriormente, el *mod* `mod_authz_host` es la forma más moderna que tiene *Apache 2* de controlar el acceso a una web o parte de una web, siendo una mejor alternativa a `mod_access_compat`. Aunque comparte algunos conceptos con él, la forma de funcionar es bastante distinta, lo que requiere que sea estudiado independientemente. No obstante, los elementos a partir de los cuales podemos conceder o denegar un acceso (IPs, host,…) son prácticamente los mismos.

Todas las opciones de autorización disponibles por este *mod* se controlan usando la directiva `Require`. Al igual que en el *mod* anterior, esta directiva se puede aplicar a directorios (`<Directory>`), tipos de ficheros (`<Files>`) o rutas (`<Location>`). Por supuesto, también puede aplicarse a ficheros (`.htaccess`) para controlar el acceso a localizaciones concretas el servidor. El acceso se controla bien por IP o bien por nombre de host. Las restricciones de acceso se aplican normalmente a cualquier método HTTP usado. No obstante, en el caso de que se necesitase sólo restringir alguno de los métodos, puede usarse la sección `<Limit>` ya mencionada para introducir en ella las directivas que se aplicarán sólo a los métodos concretos que queramos.

En general tenemos que usar la directiva `Require` con cuidado para determinar si un cliente tiene o no acceso a un determinado recurso de la forma más precisa posible. Como norma general debemos tener en cuenta a la hora de usarla:

- Si nos restringimos al uso más básico de la misma, una directiva `Require` va acompañada siempre de una IP, nombre de host o bien la directiva `local`.
- Si se usan módulos de autenticación adicionales, es posible que la directiva `Require` admita más clases de parámetros. Por ejemplo, en `mod_authz_core` se definen los parámetros `all`, `env`, `method` y `expr`.
- Las últimas versiones de *Apache 2* **permiten usar expresiones para especificar el nombre del host** requerido. Las expresiones se evalúan al arrancar *Apache 2*, no por cada petición. El uso de expresiones de *Apache 2* es un tema que requiere un estudio adicional y que no vamos a evaluar más allá de mostrar algún ejemplo, pero puede consultarse toda la documentación de las mismas aquí: http://httpd.apache.org/docs/current/expr.html

- `Require` puede llevar `not` como parámetro adicional, cuyo significado será el contrario al `Require` normal (el elemento puesto detrás es aquello que el cliente no puede ser para otorgar el acceso a la parte protegida).
- Todo lo que no encaje con lo especificado en la directiva `Require` tendrá denegado el acceso. `Require` es pues una implementación de la política de listas de elementos permitidos, que ya hemos visto que es la más segura. Por tanto, si algo tiene que ser de acceso público, no se debe poner ninguna directiva `Require`.

Restricción por IP

- El acceso se puede controlar por IPs aisladas, fragmentos de IP o bien rangos de IP
- La sintaxis es `Require ip <ips autorizadas>`.

Ejemplos de restricción de IPs concretas:

```
Require ip 192.168.10.3
Require ip 192.168.10.3 192.168.10.4
```

Ejemplos de restricción de IPs especificadas parcialmente:

Estos son ejemplos de IPs especificadas parcialmente para restringir de forma precisa el acceso a determinadas subredes (`192.168.XX.XX`, `192 192.150.XX` y `192.140.20.XX`, respectivamente):

```
Require ip 192.168
Require ip 192 192.150 192.140.20
```

Ejemplo de restricción a una red con mascara de red:

Se puede especificar una máscara de red explícitamente o bien con notación CIDR, también admitida:

```
Require ip 192.168.15.0/255.255.255.0 Require ip 192.168.0.0/16
```

Obviamente, el uso de direcciones IPv6 también está admitido en todos los casos anteriores:

```
Require ip 2001:db8::a00:20ff:fea7:ccea
Require ip 2001:db8::a00:20ff:fea7:ccea/10
```

A continuación veremos un ejemplo con nuestra infraestructura de pruebas. Pero, ante todo, lo primero que tenemos que hacer es asegurarnos de que el *mod* está activo y activarlo en caso contrario. La siguiente imagen muestra que el *mod* no se encuentra en `mods-enabled`, por lo que lo habilitamos con `a2enmod` y reiniciamos *Apache 2* para que podamos hacer uso de sus directivas.

```
operario@servidorapache:/etc/apache2/mods-enabled$ ls
access_compat.load  deflate.conf  headers.load  mpm_event.conf
authn_core.load     deflate.load  mime.conf     mpm_event.load
authz_core.load     filter.load   mime.load
operario@servidorapache:/etc/apache2/mods-enabled$ sudo a2enmod authz_host
Considering dependency authz_core for authz_host:
Module authz_core already enabled
Enabling module authz_host.
To activate the new configuration, you need to run:
  service apache2 restart
```

Figura 139. Activación del mod authz_host

El ejemplo consiste en que solo la IP `192.68.56.1` podrá acceder a la carpeta `ip_required` y además también sólo esta IP tendrá denegado el acceso a la carpeta `ip_required2` (permitiéndose al resto):

- Modificamos `apache2.conf`:

```
 GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf          Modificado

        Deny from all
        Allow from 192.168.56.1
</Directory>

<Directory /var/www/html/ip_required>
        Require ip 192.168.56.1
</Directory>

<Directory /var/www/html/ip_required2>
        <RequireAll>
                Require all granted
                Require not ip 192.168.56.1
        </RequireAll>_
</Directory>
```

Figura 140. Autorizaciones a una IP concreta y denegación de una IP concreta

Nótese que para poder hacer uso de `Require not` hemos tenido que usar un contenedor de directivas y además acompañarla de una directiva `Require` que permita el acceso a algún cliente. Esto es un requisito que se explicará algo más adelante.

- Comprobamos accesos:

Figura 141. Cliente autorizado mediante mod_authz_host

Figura 142. Cliente prohibido mediante mod_authz_host

Y como en el directorio `ip_required2` solo hemos prohibido a una IP específica, el acceso por parte de los otros clientes debería hacerse sin problemas:

```
operario@servidorubuntu:~$ wget 192.168.23.1/ip_required/contenido.txt
--2015-07-06 10:30:23--  http://192.168.23.1/ip_required/contenido.txt
Conectando con 192.168.23.1:80... conectado.
Petición HTTP enviada, esperando respuesta... 403 Forbidden
2015-07-06 10:30:23 ERROR 403: Forbidden.

operario@servidorubuntu:~$ wget 192.168.23.1/ip_required2/contenido.txt
--2015-07-06 10:30:29--  http://192.168.23.1/ip_required2/contenido.txt
Conectando con 192.168.23.1:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 22 [text/plain]
Grabando a: “contenido.txt.1”

100%[======================================>] 22          --.-K/s   en 0s
```

Figura 143. Cliente autorizado mediante mod_authz_host al no estar expresamente prohibido

Restricción por *host*

Este tipo de restricciones permite controlar el acceso en función del nombre del host que intenta acceder a *Apache 2*, siendo de nuevo una política de lista blanca. Para su funcionamiento en un entorno real normalmente se requiere la presencia de un servidor DNS que resuelva IPs a nombres DNS y viceversa, por lo que en una infraestructura de pruebas como la que estamos usando tendremos que usar el fichero `/etc/hosts`, como veremos posteriormente. Los siguientes ejemplos muestran cómo usar la directiva `Require` de esta forma:

```
Require host redlocal.org
Require host informatica.universidad.edu universidad.edu
```

Como criterio de comparación de nombres de host se requiere que se cumpla **una** de estas condiciones. Las mismas condiciones pueden aplicarse en `mod_access_compat` cuando trabajamos con nombres de host:

- Los nombres de host encajan **exactamente** con uno de los especificados.
- Los nombres de host acaban exactamente con uno de los especificados. Por ejemplo, con el primer ejemplo encajaría `administración.redlocal.org`, pero no `administracionredlocal.org`.

Para que esta directiva funcione, es necesario que el DNS usado para resolver las peticiones **sea capaz de aceptar una petición DNS inversa** (*reverse DNS lookup*) sobre la IP del cliente que envía la petición. La petición DNS inversa es la que obtiene el nombre de *host* asociado y, si tiene éxito, entonces se comparará la IP del cliente con el resultado de hacer una petición DNS estándar (*forward DNS lookup*) usando el nombre DNS obtenido en la primera petición. Si ambas peticiones son coherentes (y, obviamente, el nombre del *host* está autorizado), el acceso se permitirá.

La consecuencia de esto es que si por alguna razón el servidor DNS que usemos no está preparado para resolver peticiones DNS inversas, entonces no podremos usar estas restricciones. Nótese como el *mod* hace una doble comprobación de la identidad del cliente para ver la coherencia de DNS – IP con la intención de evitar posibles suplantaciones de clientes.

No obstante, en nuestra infraestructura de pruebas no tenemos un servidor *DNS* establecido para nuestra red de máquinas, lo cual requeriría la instalación de un servicio más en alguno de los servidores o mejor de una máquina *Linux* independiente que cumpla con esta función (http://www.thegeekstuff.com/2014/01/install-dns-server/). Lo primero va en contra de nuestra política de minimizar los servicios activos en cada servidor y dedicar cada servidor a una sola función. Lo segundo requeriría una máquina más, complicando nuestra infraestructura de pruebas y usando más recursos, además de distraernos de nuestro objetivo principal.

Por ello, usaremos una alternativa más "barata", que es el **fichero** `/etc/hosts`. Este fichero permite establecer una tabla que mapee direcciones IP a nombres de *host*, y es especialmente útil ente máquinas que se encuentren en redes locales. La sintaxis es muy simple: cada línea tiene una dirección IP seguida de una lista de nombres de host asociados, el primero de los cuales tiene que ser el nombre de dominio completo.

Lo positivo de usar este fichero es que funciona incluso aunque exista un *DNS* operativo en la red y este no se encuentre operativo por algún motivo, ya que se consulta siempre antes que el propio *DNS*. Este hecho es lo que hace que el fichero pueda tener otros usos más creativos, como por ejemplo redirigir todo el tráfico que se haga desde el cliente a determinados hosts de vuelta al host local, lo que permite evitar que se establezca cualquier clase de conexión con dichos hosts. Por tanto, el uso de esta técnica puede limitar la navegación a determinados lugares desde los clientes que tengan ese `/etc/hosts` o bien bloquear las peticiones que se hagan a servidores de anuncios u otros contenidos no deseados.

No obstante, el principal inconveniente de este fichero es que para que funcione **todas las máquinas involucradas en la comunicación deben tener exactamente el mismo**, porque si no se pueden producir incoherencias y problemas de acceso de unas máquinas a otras. La distribución y actualización de este fichero no es sencilla cuando hay un nº de máquinas elevado, por lo que en entornos reales no suele usarse. No obstante, nos valdremos de este fichero para probar el bloqueo por nombre de host, ya que podemos usarlo solo con nuestros dos clientes *Linux*. Para ello, seguiremos los siguientes pasos:

- Lo primero, como hemos dicho, es configurar los ficheros `/etc/hosts` de ambas máquinas *Linux* de nuestra infraestructura para que cada IP tenga un nombre de host asociado:

```
  GNU nano 2.2.6          Archivo: /etc/hosts                Modificado

127.0.0.1       localhost
127.0.1.1       servidorapache

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

192.168.23.2    clientelinux1
192.168.56.1    hostwindows
```

Figura 144. Fichero /etc/hosts del servidor Apache 2

```
  GNU nano 2.2.6          Archivo: /etc/hosts                  Modificado

127.0.0.1       localhost
127.0.1.1       servidorubuntu

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters

192.168.23.1 servidorapache1
```

Figura 145. Fichero /etc/hosts del cliente Linux

- Y luego modificaremos `apache2.conf` para que solo el `clientelinux1` pueda tener acceso al directorio `host_required`:

```
<Directory /var/www/html/host_required>
        Require host clientelinux1
</Directory>


#<Directory /srv/>
#       Options Indexes FollowSymLinks
#       AllowOverride None

^G Ver ayuda ^O Guardar   ^R Leer Fich ^Y RePág.    ^K Cortar Tex^C Pos actual
^X Salir     ^J Justificar^W Buscar    ^V Pág. Sig. ^U PegarTxt  ^T Ortografía
```

Figura 146. Uso de Require host

- Comprobamos finalmente que solo el host que hemos autorizado puede acceder a esos contenidos:

```
operario@servidorubuntu:~$ wget 192.168.23.1/host_required/contenido.txt
--2015-07-06 10:39:48--  http://192.168.23.1/host_required/contenido.txt
Conectando con 192.168.23.1:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 22 [text/plain]
Grabando a: "contenido.txt.2"

100%[======================================>] 22          --.-K/s   en 0s

2015-07-06 10:39:48 (4,73 MB/s) - "contenido.txt.2" guardado [22/22]

operario@servidorubuntu:~$ cat contenido.txt.2
Contenido de la ruta

operario@servidorubuntu:~$
```

Figura 147. Acceso al contenido por el cliente Linux

Figura 148. Acceso prohibido a otros host

Require local

Especificar `local` en la directiva `Require` (`Require local`) impone una serie de condiciones que deben cumplirse para permitir el acceso:

La dirección del cliente encaja con el patrón `127.0.0.0/8`

- La dirección del cliente finaliza además en 1
- La dirección del cliente y del servidor de la conexión son la misma

Esto es realmente útil cuando parte de las páginas que sirve *Apache 2* solo deben ser accesibles desde la propia máquina que las sirve, como pueden ser aquellas dedicadas a tareas de mantenimiento o administración, etc. Un ejemplo son módulos o productos de tercero de monitorización y análisis que solo deben estar accesibles a usuarios autorizados como los `mod_status` o `awstats` vistos.

No obstante, como ya hemos repetido varias veces, la mejor política de seguridad es que si algo no debe ser accesible, lo mejor es que no esté presente si es posible. En las siguientes imágenes vemos como añadimos una directiva de este tipo al directorio `/only_local`, de forma que solo podemos acceder a él desde el propio servidor y no desde el otro *Linux* ni desde el host de nuestro sistema de pruebas:

```
 GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf          Modificado

<Directory /var/www/html/ip_required>
        Require ip 192.168.56.1
</Directory>

<Directory /var/www/html/ip_required2>
        <RequireAll>
                Require all granted
                Require not ip 192.168.56.1
        </RequireAll>
</Directory>

<Directory /var/www/html/host_required>
        Require host clientelinux1
</Directory>

<Directory /var/www/html/only_local>
        Require local
</Directory>
```

Figura 149. Acceso permitido solo a la máquina local

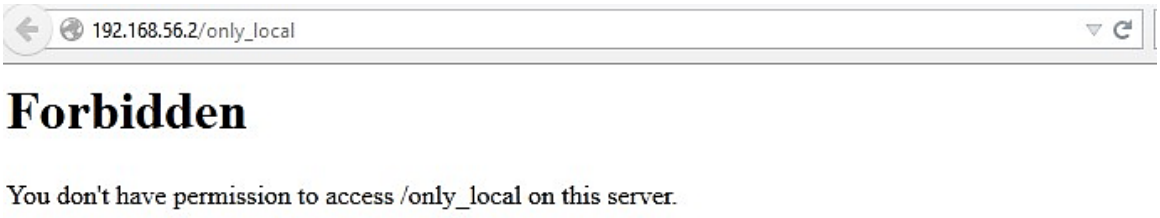


Figura 150. Cualquier cliente externo tiene el acceso prohibido por Require local (I)

```
100%[=====================================>] 22          --.-K/s   en 0s

2015-07-06 10:39:48 (4,73 MB/s) - "contenido.txt.2" guardado [22/22]

operario@servidorubuntu:~$ cat contenido.txt.2
Contenido de la ruta

operario@servidorubuntu:~$ wget 192.168.23.1/only_local/contenido.txt
--2015-07-06 17:29:53--  http://192.168.23.1/only_local/contenido.txt
Conectando con 192.168.23.1:80... conectado.
Petición HTTP enviada, esperando respuesta... 403 Forbidden
2015-07-06 17:29:53 ERROR 403: Forbidden.

operario@servidorubuntu:~$ _
```

Figura 151. Cualquier cliente externo tiene el acceso prohibido por Require local (II)

```
operario@servidorapache:~$ wget 192.168.23.1/only_local/contenido.txt
--2015-07-06 17:30:42--  http://192.168.23.1/only_local/contenido.txt
Conectando con 192.168.23.1:80... conectado.
Petición HTTP enviada, esperando respuesta... 200 OK
Longitud: 22 [text/plain]
Grabando a: "contenido.txt"

100%[=====================================>] 22          --.-K/s   en 0s
```

Figura 152. El propio servidor sí puede acceder a los contenidos

Require all

Es el sustituto de `Allow from all` y `Deny from all` de `mod_access_compat`. Vimos un ejemplo de uso anteriormente.

```
Require all granted
Require all denied
```

Require env

Se usa para que solo se permite el acceso en función de si una variable de entorno cuyo nombre especificamos está o no definida. Veremos ejemplos un poco más adelante.

```
Require env zona_de_administradores
```

```
<RequireAll>
Require not env ZONA_PROTEGIDA  </RequireAll>
```

Require method

Limita el acceso a peticiones que usen determinados métodos HTTP:

```
Require method GET HEAD
```

Require expr

Controla el acceso en función de la evaluación de una expresión, pudiendo usar para ello variables de entorno. Por ejemplo:

```
Require expr %{HTTP_USER_AGENT} != 'Googlebot'
Require expr %{REQUEST_URI} !~ m#^/(admin|private)/#
```

La última expresión básicamente especifica que se necesita que la URI de la petición tenga una sección llamada `/admin` o `/private`. Como en una expresión regular no se admite el carácter / directamente, la solución es pasar de `Require expr %{REQUEST_URI} !~ /expr/` a `Require expr %{REQUEST_URI} !~ m#expr#`

Consideraciones cuando se usan *proxies*

Esta clase de autorización **puede verse dificultada por el uso de *proxies***. Ha de tenerse en cuenta que el uso de un proxy hará que a *Apache 2* le llegue como dirección del cliente la dirección del *proxy*, por lo que si la dirección del proxy está admitida, todas las máquinas que se conecten a través de él también lo estarán.

Existe una posible solución a este problema mediante el uso del *mod* `mod_remoteip`: http://httpd.apache.org/docs/current/mod/mod_remoteip.html . Este *mod* básicamente permite que la dirección IP de un cliente se sobrescriba con una dirección IP existente en alguna cabecera del *User Agent* de la petición. La cabecera en concreto que se usa para determinar la IP del cliente se especifica con su directiva `RemoteIPHeader`. Una vez configurado este *mod*, la IP sobrescrita se usará por la directiva `Require ip` vista, y también por otros módulos conocidos como `mod_status` o `mod_log_config`, entre otros.

No obstante, esto debe aplicarse con extremo cuidado, puesto que no hay que olvidar que el *User Agent* que envía un cliente puede ser completamente falsificado (al fin y al cabo es una información que parte del cliente y sobre la que no tenemos ningún control). Por ese motivo, lo mejor es habilitar este *mod* en *host* intermedios entre el cliente y el servidor (*proxies*), y usar el resto de las directivas del *mod* para configurar con mucho más detalle qué se permite y qué no.

Contenedores RequireAll, RequireAny, RequirenOne

Una de las aportaciones más interesantes de este módulo de autenticación son estos tres contenedores de directivas de autorización. Por defecto, si no se engloban en ningún contenedor específico, las directivas `Require` se tratan como si estuvieran dentro de un contenedor `RequireAny`, es decir, que en el caso de que haya vayas directivas `Require` aplicadas sobre una localización concreta, con que una de ellas sea válida para el cliente, se permitirá el acceso. El siguiente ejemplo muestra este comportamiento,

aunque es necesario habilitar el *mod* `setenvif` para que funcione la directiva `SetEnvIf` (http://httpd.apache.org/docs/current/mod/mod_setenvif.html):

```
SetEnvIf REQUEST_URI "^/(admin|private)/" PROTECTED
<RequireAny>
    Require env PROTECTED
    Require ip 192.168.10
</RequireAny>
```

En este caso el acceso se permite si la variable de entorno `PROTECTED` está establecida (lo que ocurre si el acceso es a los directorios `/admin` o `/private`) o bien si la petición viene de una IP que comiencen por `192.168.10`. No obstante, podemos hacer la siguiente variante:

```
SetEnvIf REQUEST_URI "^/(admin|private)/" PROTECTED
<RequireAll>
    Require env PROTECTED
    Require ip 192.168.10
</RequireAll>
```

Donde básicamente se dice que el acceso a las URIs anteriores solo se puede realizar desde las IPs que comienzan por `192.168.10`. Los contenedores se pueden anidar de forma indefinida para formar reglas de acceso todo lo complejas que se necesiten. También debe tenerse en cuenta que las directivas `Require not` deben estar siempre dentro de una directiva `RequireAll` y acompañadas al menos de otra directiva que permita el acceso, ya que en caso contrario al reiniciar *Apache 2* este nos dará un error de configuración.

Con ambos contenedores pueden hacerse ejemplos todo lo avanzados que se quiera. A continuación mostramos dos ejemplos sencillos, pero pueden mezclarse (y anidarse) todas las variantes de `Require` vistas para tener un control fino de lo que se permite o lo que no:

```
<RequireAll>
    Require ip 192.168.10.0/24
    Require not ip 192.168.10.66
</RequireAll>
```

Y un ejemplo similar con nombres de host:

```
<RequireAll>
    Require host redlocal.miempresa.org
    Require not host invitados.redlocal.miempresa.org
</RequireAll>
```

Como ejemplo ilustrativo final de varios de los conceptos vistos sobre nuestra infraestructura de pruebas vamos a hacer que cuando la petición contenga una parte de la URI que empiece por `/env`, entonces se defina la variable de entorno `ACCESO_ESPECIAL` y mediante contenedores `RequireAll` haremos que los directorios con ese patrón (en nuestro ejemplo `/env1` y `/env2`) sean accesibles sólo desde la IP del host (`192.168.56.1`).

Para completar el ejemplo pondremos el mismo contenedor de directivas sobre el directorio `/otherenv`, usando las mismas restricciones. El acceso a este directorio será imposible, puesto que la variable de entorno `ACCESO_ESPECIAL` no se definirá nunca al no encajar `/otherenv` con el patrón establecido en la directiva `SetEnvIf (/env)`. Esto sirve para ilustrar dos cosas: como *Apache 2* puede definir variables de

entorno en función de las peticiones que se van haciendo y además lo fácil que puede ser definir una configuración contradictoria que nos lleva a problemas de acceso que podrían ser difíciles de arreglar.

El primer paso es definir (de nuevo en `apache2.conf`, por ejemplo) la variable de entorno y los requisitos de acceso a los directorios mencionados:

```
  GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf

SetEnvIf REQUEST_URI "/env" ACCESO_ESPECIAL

<Directory /var/www/html/env1>
        <RequireAll>
                Require env ACCESO_ESPECIAL
                Require ip 192.168.56.1
        </RequireAll>
</Directory>

<Directory /var/www/html/env2>
        <RequireAll>
                Require env ACCESO_ESPECIAL
                Require ip 192.168.56.1
        </RequireAll>
</Directory>

<Directory /var/www/html/otherenv>
        <RequireAll>
                Require env ACCESO_ESPECIAL
                Require ip 192.168.56.1
        </RequireAll>
</Directory>

#<Directory /srv/>
#       Options Indexes FollowSymLinks

^G Ver ayuda ^O Guardar   ^R Leer Fich ^Y RePág.    ^K Cortar Tex^C Pos actual
^X Salir     ^J Justificar^W Buscar    ^V Pág. Sig. ^U PegarTxt   ^T Ortografía
```

Figura 153. Definición de variable de entorno y acceso en función de esta, combinado con una restricción de IP

Hecho esto y, como siempre, reiniciado *Apache 2*, probamos a acceder desde el cliente autorizado, obteniendo el resultado esperado:

Figura 154. Acceso restringido por variable de entorno, aunque no por IP (al requerirse ambos)

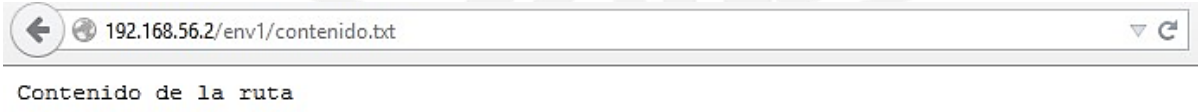


Figura 155. Acceso permitido por variable de entorno e IP (I)

Figura 156. Acceso permitido por variable de entorno e IP (II)

Prohibición para servir tipos de archivo concretos

Para terminar con esta sección vamos a ver una última medida de seguridad a implementar **de forma transversal a todas las vistas relativas al control de acceso a los recursos**. Independientemente de hosts, IPs y variables de entorno que regulen el acceso a determinados recursos, existirán **determinados tipos de archivos que NUNCA deberían servirse al público** y que puedan estar presentes en una carpeta accesible públicamente por ser imposible eliminarlos de dicha localización. Esto puede ocurrir por la forma de trabajar de la tecnología de desarrollo que estamos usando, restricciones impuestas por el servicio de hosting u otros motivos. Por tanto, ante la imposibilidad de mover los archivos a una localización más segura, vamos a ver la forma de prohibir su visualización de forma global.

Para ello basta con combinar el uso del contenedor `<Files>` con alguno de los módulos de autenticación vistos. A modo de ejemplo, suponiendo que existan ficheros `.exe` accesibles por alguna razón en un directorio de acceso público, vamos a prohibir el acceso a los mismos a nivel de todo el servidor. Lo primero es crear un fichero `.exe` y comprobar que es accesible:

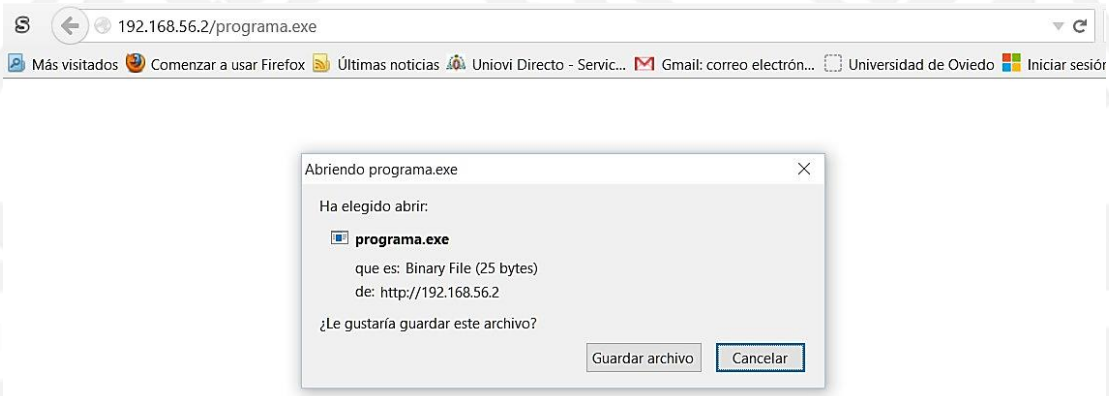


Figura 157. Acceso a un fichero ejecutable

Como vemos, el comportamiento por defecto del servidor ante este tipo de archivos es permitir su descarga por parte del cliente. Esto puede traer más problemas de los que parece en un principio, puesto que, por ejemplo, `dlls`, `exes` y otros ejecutables pueden ser decompilados y de esta forma saber mucha información acerca de una aplicación web, incluso claves que se encuentren embebidas en el propio código fuente u otros detalles importantes.

Ahora prohibiremos el acceso a los `.exe` con la siguiente directiva (en `apache2.conf` para que se aplique a nivel global):

```
 GNU nano 2.2.6        Archivo: /etc/apache2/apache2.conf          Modificado

IncludeOptional conf-enabled/*.conf

# Include the virtual host configurations:
IncludeOptional sites-enabled/*.conf

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet


<Files ~ "\.exe$">
        Require all denied
</Files>
```

Figura 158. Restricción de acceso global a un tipo de fichero

Y tras reiniciar el servidor comprobamos que ahora el acceso a estos ficheros no es posible:

Figura 159. Acceso prohibido a ficheros ejecutables

Esto puede repetirse con cualquier tipo de fichero (ejecutable, de configuración, de la plataforma de desarrollo, etc.) que consideremos no accesible. No obstante, ha de tenerse en cuenta el orden de evaluación de los contenedores y que no estemos sobrescribiendo esta configuración con ningún contenedor `<Location>`, por ejemplo, más prioritario que `<Files>`. No obstante, esta es una capa más de seguridad que conviene implementar **juntamente con el resto de los controles de acceso** que implementemos.

Autenticación basada en el servidor

La identidad de usuario que accede a los recursos del servidor es un aspecto de capital importancia. Obviamente, cualquier administrador va a necesitar que, en función de qué identidad de usuario, se determine para el cliente se permita acceder o no a determinados ficheros, según la configuración del servidor. Una de las cosas que puede ser más confusa a la hora de trabajar con este aspecto es **qué ocurre cuando no hay ninguna clase de método de autenticación especificado para un recurso**. En este caso también se usa una identidad de usuario, que como hemos visto por defecto es `www-data`. A este usuario podemos llamarle el "usuario anónimo de Internet", concepto que también existe en otros servidores.

Si nuestro interés es tener otra identidad de usuario, entonces tenemos que hablar de métodos de autenticación soportados por *Apache 2*. **Inicialmente, se proporciona soporte para dos**: *Basic* y *Digest*. Otros tipos de autenticación se soportan mediante la instalación de módulos específicos que implementen métodos de autenticación diversos, incluso adaptados a interactuar con programas concretos (Ej.: *MySQL*). Un listado de módulos que implementen autenticaciones podemos encontrarlo en: https://modules.apache.org/browse.lua?tag=security&pos=0 (buscar módulos con la palabra "*auth*").

En esta sección no es posible describir **la gran variedad de métodos de autenticación soportados** por *Apache 2* mediante módulos, ya que cada módulo tiene su forma particular de activarse y usarse. Por ello, nos limitaremos a describir **cómo implementar autenticación básica y *Digest*** y luego hablar de un par de casos típicos de autenticación adicionales. Se da a su vez la circunstancia de que muchos tipos de autenticación son *front-ends*, que trabajan con *Basic* y *Digest* "por debajo" realmente. También es importante recordar que para que un método de autenticación sea realmente seguro es imprescindible el uso de TLS (próxima sección) para cifrar el tráfico, dado que el uso de uno de estos dos métodos hace que se envíe información importante en texto plano o con un método de codificación vulnerable. Si queremos profundizar más en el tema de autenticación en *Apache 2* podemos consultar el siguiente enlace: http://httpd.apache.org/docs/current/howto/auth.html

Finalmente, veremos que para el uso de autenticación *Basic* o *Digest* hacen falta ficheros adicionales con información de usuarios y sus claves. Estos ficheros de autenticación **NUNCA** deben estar en el *path* designado por `DocumentRoot`, ya que podrían descargarse por un usuario cualquiera y de esta manera lograr romper nuestro sistema de autenticación.

Autenticación básica

Al usar esta autenticación el servidor solicita al cliente un nombre de usuario y clave **que se envían sin cifrar por la red desde el cliente al servidor**. De hecho, la información introducida se codifica en ***base64***, que no es un método de cifrado sino una codificación que permite aceptar en la información transmitida caracteres que no pertenecen al estándar *ASCII* sin romper el protocolo *HTTP*, y que se decodifica muy fácilmente. Por tanto, la información es casi como si se enviase en texto plano.

Debe quedar claro que si usamos este tipo de autenticación tenemos que emplear métodos adicionales para hacerla segura. Esto consiste en cifrar el tráfico por otros medios (*IPSec*, TLS), para que nadie no autorizado pueda acceder al contenido de nuestras transmisiones. El empleo de comunicaciones cifradas se examinará en la próxima sección.

Tras la primera autenticación *Basic* válida, cuando se vuelva a requerir autorización para acceder a un recurso, el servidor enviará el identificador de usuario y su clave de nuevo, al ser necesarios para proceder

a la autorización, volviéndose a mandar dicha información sin cifrar. Esto ocurre porque *HTTP* es un protocolo sin estado y no hay soporte para guardar información entre peticiones, es decir, entre la petición en la que se introdujo usuario y clave por primera vez y las siguientes peticiones que requieran esta información. Este proceso se hace de la siguiente forma:

- Ante una petición de autenticación del servidor, los *browsers* están preparados **para volver a enviar las credenciales que poseen de una petición anterior** (son los *browsers* pues quienes suplen la carencia de estado de *HTTP*). Nótese que las credenciales sólo se mandan en caso de que el servidor pida autentificación (el recurso este protegido) y no en caso contrario (es decir, las credenciales no se mandan siempre).
- Si éstas siguen siendo válidas para el nuevo recurso pedido, **no hará falta que el usuario las introduzca de nuevo y el recurso se podrá obtener**. Esto sin duda mejora la usabilidad del sitio (no tenemos que solicitar la clave cada vez que accedamos a un recurso protegido), pero hace que la clave se envíe por un canal no seguro múltiples veces

Por tanto, **siempre deberemos cifrar la información en aquellas partes del sitio donde se requiera autenticación básica** si queremos seguridad para las credenciales que estamos enviando. En caso contrario, estaremos incurriendo en un problema serio de seguridad. A pesar de esta grave desventaja, la autenticación básica tiene tres ventajas: es parte de *HTTP 1.0* (por lo que está soportada por todos los *browsers* actuales), es sencilla y puede pasar a través de *firewalls* y *proxys*.

Para ilustrar cómo implementar la autenticación *Basic* con *Apache 2*, vamos a poner un ejemplo sencillo. Este ejemplo consistirá en implementar en el directorio `/web1` una autenticación básica para usuarios que pertenezcan a un grupo concreto. Veremos dos posibles casos, reflejo de posibles situaciones reales con las que nos podamos encontrar.

Caso 1. Si tenemos acceso a los ficheros de configuración de *Apache 2*

El primer paso es incluir las siguientes líneas en el fichero `apache2.conf`, que indicarán a *Apache 2* que se desea este tipo de autenticación sobre el directorio indicado.

```
<Directory "/var/www/html/web1">
        AuthType Basic
        AuthName "El contenido de esta web es privado"
        AuthUserFile /etc/apache2/authusers
        AuthGroupFile /etc/apache2/authgroups
        Require group privado
</Directory>
```

Figura 160. Configuración de la autenticación básica

Una de las cosas nuevas que tiene el contenido anterior es una referencia a **un fichero** `authgroups`. Este fichero **debe contener un listado de usuarios** que han sido creados expresamente para esta clase de autenticación (lo veremos a continuación) y que han sido distribuidos en una serie de grupos, que también aparecen en el mismo. Esto permite crear varias áreas de acceso en diversas partes de la *web,* donde un mismo grupo pueda tener diferentes privilegios o bien sólo algunos puedan acceder a los contenidos. Evidentemente, cuando se comprueben las credenciales de un usuario para conceder el acceso a un lugar protegido, dicho usuario debe pertenecer a uno de los grupos autorizados. La directiva del ejemplo anterior "`Require group private`" hace que solo los usuarios de este grupo (que como veremos posteriormente serán `alfa` y `bravo`) tengan acceso a este directorio vía *web*.

El contenido del fichero `authgroups` se puede crear simplemente con un editor de texto como el `nano`, y debe hacerse en la carpeta indicada anteriormente en el `apache2.conf`. El contenido es muy sencillo y debe seguir el patrón `<nombre de grupo>: <lista de usuarios>`, tal y como se ve a continuación. Este fichero se referencia con la directiva `AuthGroupFile`, pero para poder usarla necesitamos incluir el módulo `authz_groupfile`:

Figura 161. Fichero authgroups

De igual forma que el fichero anterior, el fichero `authusers` contiene una lista de nombres de usuario y una *hash* de su *password,* que son comprobadas cada vez que se requiera autorización. Si añadiésemos a la configuración vista de `apache2.conf` la directiva "`Require valid-user`" dentro del contenedor del directorio que estamos usando, entonces indicaríamos que para entrar en una parte de la *web* protegida por este tipo de autenticación simplemente es necesario conocer las credenciales de cualquier usuario de este fichero. Normalmente se usa si no existen grupos de usuarios con accesos diferenciados,

Antes de continuar describiendo la implementación de autenticación básica, debemos tener en cuenta que **el usuario sobre el que *Apache 2* se ejecuta por defecto se denomina** `www-data`. Esto influye a la hora de poder acceder a los archivos con usuarios y claves que estamos usando, ya que este usuario podría no tener privilegios para ello. Para que funcione correctamente, o bien los ponemos en una localización independiente del resto de las *web* dedicada exclusivamente a estos fines, con unos permisos adecuados, o bien los creamos en el directorio de instalación de *Apache 2* y damos permiso de ejecución sobre el directorio `/etc/apache2` a "otros" (`chmod o+x /etc/apache2`, si no lo tuviese ya), para que al menos el usuario por defecto pueda acceder a su contenido y de esta manera acceder a los ficheros de permisos. Por motivos de seguridad sería preferible la primera opción, pero en el ejemplo usaremos la segunda al ser más sencilla.

Finalmente, debemos construir las *password* de los usuarios que se mencionan en el fichero de grupos. Esto se hace con la siguiente secuencia de comandos, invocando un comando `htpasswd` por cada usuario registrado. Al llamar a esta orden se solicitará por entrada estándar una clave para **cada** usuario. Es posible que haya que instalar el paquete `apache2-utils` con `apt-get` para poder usar el programa `htpasswd`:

```
touch /etc/apache2/authusers
htpasswd /etc/apache2/authusers alfa
htpasswd /etc/apache2/authusers bravo
htpasswd /etc/apache2/authusers charlie
```

```
mainuser@ubuntuserver:/etc/apache2$ sudo touch /etc/apache2/authusers
mainuser@ubuntuserver:/etc/apache2$ sudo htpasswd /etc/apache2/authusers alfa
New password:
Re-type new password:
Adding password for user alfa
mainuser@ubuntuserver:/etc/apache2$ sudo htpasswd /etc/apache2/authusers bravo
New password:
Re-type new password:
Adding password for user bravo
mainuser@ubuntuserver:/etc/apache2$ sudo htpasswd /etc/apache2/authusers charlie
New password:
Re-type new password:
Adding password for user charlie
mainuser@ubuntuserver:/etc/apache2$ _
```

Figura 162. Creación de passwords para usuarios en autenticación Basic

Para probar la autenticación una vez terminado este proceso, es necesario **reiniciar y probar a acceder** desde nuestro cliente con usuarios válidos y no válidos para probar que está funcionando correctamente. Esto es lo que ocurre cuando accedemos a una localización protegida:

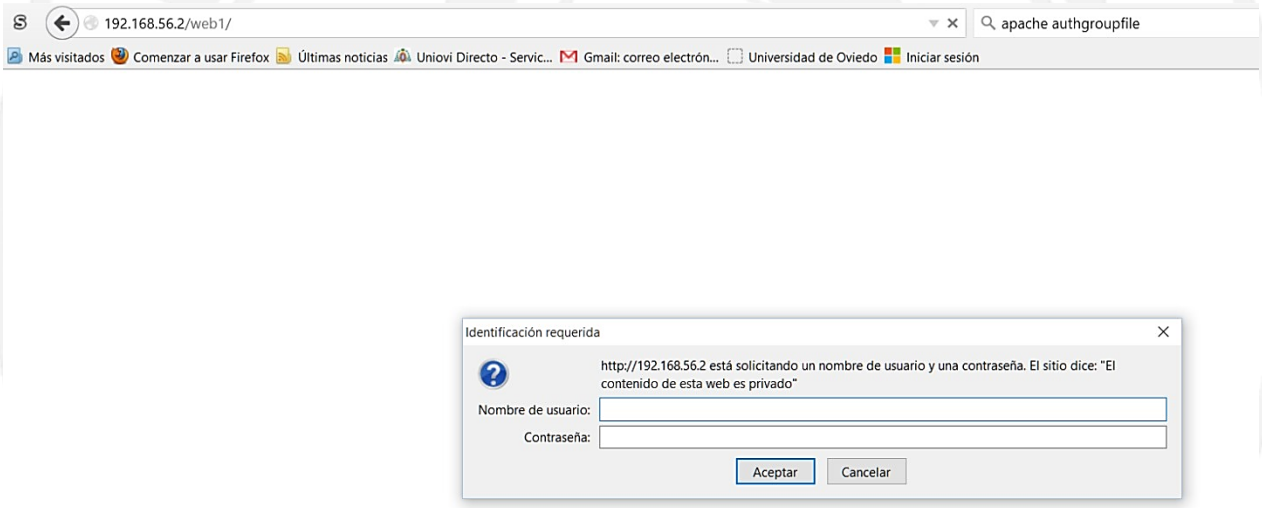


Figura 163. Ejemplo de browser solicitando credenciales para autenticación Basic

La siguiente imagen muestra el mensaje de un navegador en el caso de que los datos introducidos no sean válidos:

Figura 164. Respuesta ante credenciales inválidas en autenticación Basic

Caso 2. Si no tenemos acceso a los ficheros de configuración de Apache 2

¿Qué ocurre si queremos establecer una autenticación de esta clase y no tenemos acceso a los ficheros de configuración de Apache 2? (como por ejemplo para *web* personales de las que describimos con `mod_userdir`). En este caso no nos queda más remedio que usar archivos `.htaccess`. Con ellos es posible hacer exactamente lo mismo que antes, pero para este segundo ejemplo vamos a hacerlo para un solo usuario. Lo que vamos a hacer consta de los siguientes pasos:

- Lo primero es recordar que el uso de archivos `.htaccess` **puede estar prohibido** por una configuración anterior de *Apache 2* (`AllowOverride None`). Si esto ocurriese, no podríamos seguir con estos pasos. Para poder hacer todo lo que se plantea a continuación, es necesario dejar que *Apache 2* permita usar estos archivos, como mínimo para fijar aspectos de autenticación. Esto puede lograrse editando el fichero `000-default` (*site* por defecto, o bien el host virtual que estemos usando en este momento) o `apache2.conf` y cambiar en la directiva `<Directory/>` la entrada `AllowOverride` de `None` a `AuthConfig`. Si hemos tenido que hacer esto, debemos reiniciar *Apache 2* posteriormente.

```
<Directory />
        AllowOverride AuthConfig
        Order allow,deny
        Allow from all_
</Directory>
```

Figura 165. Permitiendo establecer mecanismos de autenticación en directorios particulares

- Ir al directorio *web* personal del usuario del sistema que queramos usar para esta prueba (en nuestro caso `operario`).
- Crear en ese directorio un fichero `.htaccess` con este contenido. En este caso solo vamos a permitir la entrada a un único usuario (`pepe`), pero podríamos repetir el ejemplo visto anteriormente sin problemas:

```
 GNU nano 2.2.6          Archivo: .htaccess                    Modificado

AuthType Basic
AuthName "Este es mi directorio y es privado"
AuthUserFile /home/operario/public_html/.htauthusers
Require valid-user
_
```

Figura 166. Fichero .htaccess con datos de autenticación

- Ahora simplemente debemos crear los usuarios y *passwords* como vimos anteriormente:

```
touch .htauthusers
htpasswd .htauthusers pepe
```

```
operario@ubuntu:~/public_html$ touch .htauthusers
operario@ubuntu:~/public_html$ htpasswd .htauthusers pepe
New password:
Re-type new password:
Adding password for user pepe
operario@ubuntu:~/public_html$
```

Figura 167. Creación de passwords para autenticación Digest

- Y solo restaría reiniciar la máquina y probar la configuración.

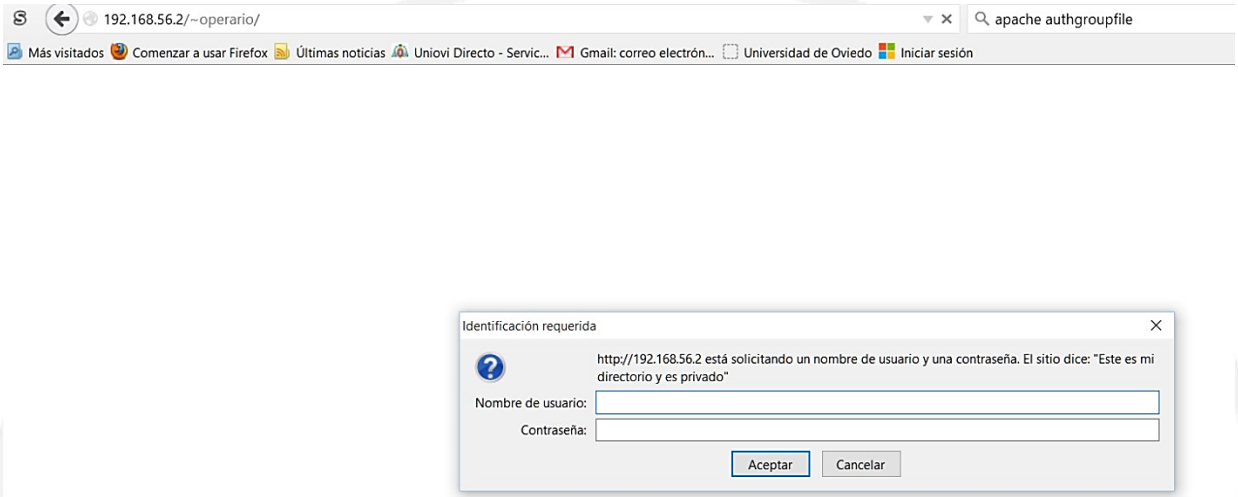


Figura 168. Autenticación Basic en un directorio particular

Autenticación *Digest*

Al igual que en el método de autenticación anterior, al usar autenticación *Digest* para acceder a un recurso, al cliente se le solicitan credenciales de usuario válidas. En este caso, sin embargo, **se manda su nombre en forma de texto plano y una *hash* de la clave**. Este método de autenticación tiene las siguientes características:

- Es soportado por los *browsers* más extendidos.
- Puede "pasar" a través de *proxys* y *firewalls*.
- Usa *nonces* (datos aleatorios) de cliente y servidor para limitar la capacidad de hacer ataques de *replay* y criptoanálisis de la clave enviada.
- No se envía la *password* en texto plano, sino que se envían los datos que se obtienen como resultado de la siguiente función: *MD5* (`nombreusuario:dominio:password`) . *MD5* es una función *hash* criptográfica. Más información: http://en.wikipedia.org/wiki/MD5 .
- **Es vulnerable a ataques *Man in the Middle***, que puedan interferir en la comunicación y solicitar a los clientes las credenciales con un nivel de seguridad menor. Además, el algoritmo *MD5* es parcialmente inseguro, por lo que no podemos confiar en la seguridad de este método. Por tanto, al igual que con el método anterior, tendremos que usar TLS o *IPSec* para tener un nivel de seguridad adecuado en la transmisión.

A la hora de implementar autenticación *Digest* sobre algún contenido servido por *Apache 2* debemos seguir una serie de pasos muy similares a los dados para la autenticación *Basic* vista anteriormente. En este caso haremos que el directorio `/web1` tenga ahora esta clase de autenticación. Los pasos a seguir son:

- Habilitar este tipo de autenticación activando el módulo correspondiente. Esto se puede hacer con el comando `a2enmod auth_digest` y reiniciando *Apache 2* posteriormente.

```
operario@ubuntu:~$ sudo a2enmod auth_digest
Considering dependency authn_core for auth_digest:
Module authn_core already enabled
Enabling module auth_digest.
To activate the new configuration, you need to run:
  service apache2 restart
operario@ubuntu:~$
```

Figura 169. Activando el mod auth_digest

- Modificamos el fichero `apache2.conf` para cambiar el tipo de autenticación usada ahora en `/web1`:

```
<Directory "/var/www/html/web1">
        AuthType Digest
        #Esto mismo hay que indicarlo al ejecutar htdigest
        AuthName "Privado"
        AuthUserFile /etc/apache2/authusersdigest
        AuthGroupFile /etc/apache2/authgroupsdigest
        Require group privado
</Directory>
```

Figura 170. Ejemplo de autenticación Digest

- El parámetro `AuthName "Privado"` es una descripción del equipo al que se quiere acceder mediante esta clase de autenticación. Esta información se usa para calcular las hash de las claves enviadas y que será mostrada al usuario cuando se conecte al servidor. **Más información**: http://en.wikipedia.org/wiki/Digest_access_authentication
- Crear luego en una localización adecuada, según lo dicho para la autenticación básica, un fichero `authusersdigest` con `touch` y las *passwords* de los usuarios así como un `authgroupsdigest` (aunque realmente nos serviría el mismo que hemos usado en la autenticación básica). Esto puede hacerse, para un usuario llamado "`alfa`", con el siguiente comando: `htdigest -c authuserdigest Privado alfa`. Nótese que hemos especificado el valor de la directiva `AuthName` en la llamada. En nuestro caso lo haremos para los tres usuarios definidos en el fichero de grupos:

```
operario@ubuntu:/etc/apache2$ sudo htdigest authusersdigest Privado alfa
Adding user alfa in realm Privado
New password:
Re-type new password:
operario@ubuntu:/etc/apache2$ sudo htdigest authusersdigest Privado bravo
Adding user bravo in realm Privado
New password:
Re-type new password:
operario@ubuntu:/etc/apache2$ sudo htdigest authusersdigest Privado charlie
Adding user charlie in realm Privado
New password:
Re-type new password:
operario@ubuntu:/etc/apache2$
```

Figura 171. Creación de una clave para un usuario con autenticación Digest

- Una vez terminado esto, solo resta probarlo desde el cliente, igual que hemos hecho antes. Comprobaremos de nuevo que el usuario `charlie` no puede acceder a la carpeta mientras que los otros usuarios, al pertenecer al grupo privado, si pueden hacerlo. Lo mismo se podría hacer con usuarios aislados sino usamos grupos.

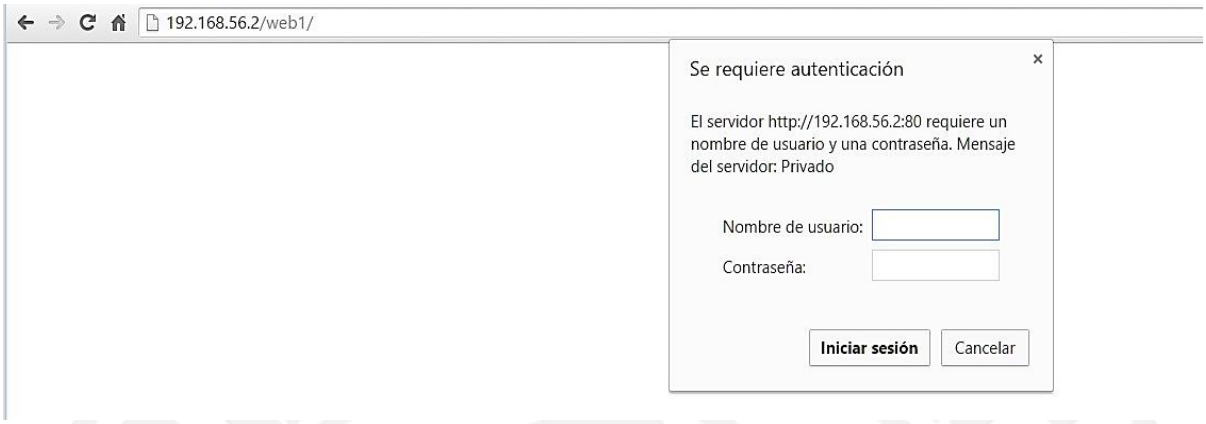


Figura 172. Petición de credenciales con autenticación Digest

Autenticación con los usuarios del sistema operativo

La autenticación con medios externos variados se ejecuta a través de un módulo denominado `mod_auth_external` (https://code.google.com/p/mod-auth-external/). Dentro de este módulo se encuentra un programa llamado `pwauth`, cuya función es ejecutar tareas de autenticación cuando otros programas le llamen para saber si un nombre de usuario y *password* son válidos. Mediante este programa podemos conseguir comprobar si la identidad de un usuario del sistema es válida sin necesidad de ser `root`, ya que si intentásemos hacer este proceso como `root` en alguna parte del procedimiento de comprobación del usuario estaríamos incurriendo en un problema de seguridad.

Un escenario típico de uso de `pwauth` es proporcionar un mecanismo para realizar autenticación web *Basic* como la vista pero de fuentes externas diversas (en nuestro caso veremos un ejemplo con los usuarios del sistema operativo en breve), pero también tiene aplicaciones para programar formularios de autenticación (*Form-Based Authentication*), para autenticación *SMTP* en servidores de E-Mail y otros. Relativo a la primera aplicación podemos citar las siguientes fuentes de usuarios/*passwords* admitidas y que todavía están en activo:

- Base de datos de usuarios del sistema
- MySQL, aunque en este caso es mejor usar el módulo `mod_authn_dbd` (http://httpd.apache.org/docs/current/mod/mod_authn_dbd.html que admite varios tipos de bases de datos).
- Servidores *RADIUS* (https://en.wikipedia.org/wiki/RADIUS)
- Servidores *LDAP*
- Servidores *IMAP* para email (https://en.wikipedia.org/wiki/Internet_Message_Access_Protocol)

Entre las características más interesantes de `pwauth` tenemos:

- **Soporte de *Lastlog***, actualizando el fichero que guarda cuando fue la última vez que un usuario entró en sesión

- Soporte para que **se invoque solo por un usuario de una lista predeterminada**, evitándose así que cualquier usuario del sistema invoque `pwauth`
- Impide que determinadas **cuentas cuyo id esté por debajo de uno dado** hagan *login*, para que no se puedan usar usuarios del sistema.
- Si un usuario ha expirado o bien su clave ha expirado, `pwauth` **no permitirá hacer *login*** con el mismo.
- Si se intenta hacer *login* **con una cuenta que no ha sido creada para permitir hacer *login* interactivo** en el sistema (cuentas para un servicio, por ejemplo), `pwauth` lo detecta e impide el *login*. De esta forma se evita intentar autenticarse con una cuenta de un usuario que no ha sido creado para interactuar con el sistema.
- **Implementa retardos** ante un nº sucesivo de *logins* inválidos, mitigando así ataques de averiguación de *passwords* por fuerza bruta.
- **Guarda datos acerca del nº de intentos** de *login* inválidos.

Aunque `pwauth` tiene varios mecanismos de seguridad embebidos, muchos autores consideran que autenticar usuarios de una web contra la base de datos del servidor es siempre una mala idea por posibles problemas de seguridad. Aunque realmente hoy en día la forma de autenticar usuarios dominante es **con un servicio de autenticación de terceros** (*Google*, *Microsoft*...), con lo cual no guardas contraseñas, o el clásico esquema basado en programarlo en la aplicación con una librería de autenticación (como OAuth2, https://oauth.net/2/) + BD, no se puede descartar el uso de este esquema de autenticación en webs administrativas o de empresa en redes locales, donde los usuarios son conocidos y potencialmente limitados (salvo intrusiones).

A continuación veremos un ejemplo de cómo poner en marcha esta clase de autenticación sobre los contenidos del directorio `/web2`. El primer paso, por supuesto, es activar `mod-authnzexternal`:

```
operario@ubuntu:/etc/apache2$ sudo apt-get install libapache2-mod-authnz-externa
l pwauth_
```

Figura 173. Activando mod_authnz_external

También necesitaremos el módulo `mod_authz_unixgroup` para permitir autenticar por grupos de usuarios además de mediante usuarios individuales, ya que veremos ambas funcionalidades en este ejemplo:

```
operario@ubuntu:/etc/apache2$ sudo apt-get install libapache2-mod-authz-unixgrou
p
```

Figura 174. Instalación de mod_authz_unixgroup

Finalmente habilitamos este último módulo, aunque en una instalación normal ya debería haberse habilitado automáticamente

```
operario@ubuntu:/etc/apache2$ sudo a2enmod authnz_external authz_unixgroup_
```

Figura 175. Habilitando mod_authz_unixgroup

Hecho esto, simplemente configuramos la autenticación para el directorio que hemos dicho como se ve en la imagen siguiente, diciendo que el proveedor de autenticación es externo, que se llama `pwauth` (el programa que hemos descrito) y que para acceder al contenido necesitamos que el usuario sea operario.

```
 GNU nano 2.2.6 Archivo: ...apache2/sites-enabled/000-default.conf Modificado

#       Options +ExecCGI -Multiviews +SymLinksIfOwnerMatch


        <IfModule mod_authnz_external.c>
                AddExternalAuth pwauth /usr/sbin/pwauth
                SetExternalAuthMethod pwauth pipe

                <Directory /var/www/html/web2>
                        AuthType Basic
                        AuthName "Directorio para operarios"
                        AuthBasicProvider external
                        AuthExternal pwauth
                        Require user operario
                </Directory>_
        </IfModule>
</VirtualHost>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Figura 176. Configuración de autenticación con usuarios del SO

Finalmente probamos que efectivamente se nos pide un usuario y clave:

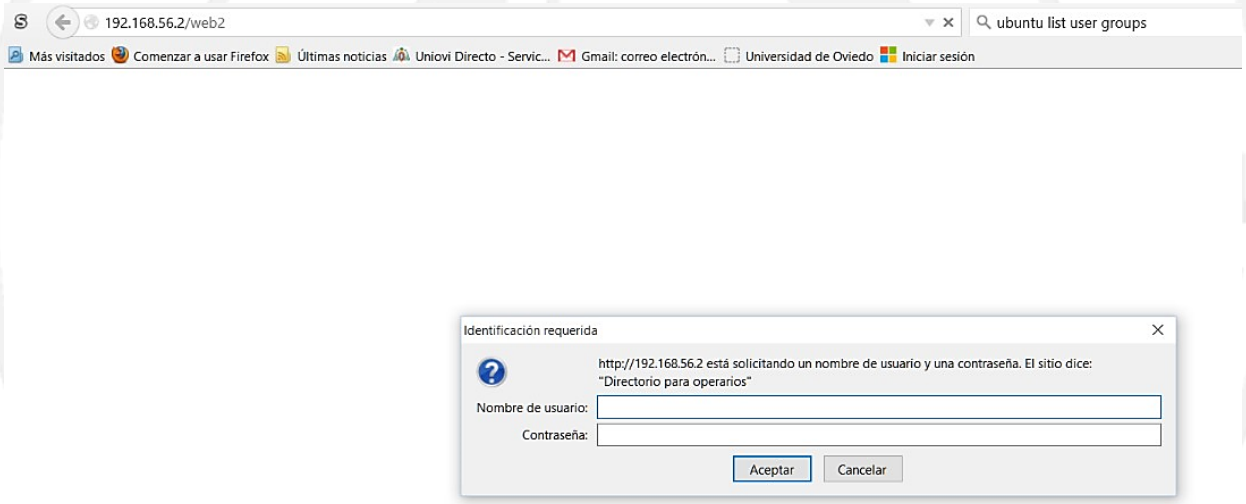


Figura 177. Solicitud de autenticación para usuarios del sistema

No obstante, debemos ser consciente del grave problema de seguridad que supone este ejemplo: estamos enviando la contraseña del usuario de Linux en texto plano a través del canal de comunicación. Si este fuese interceptado, estaríamos revelando la clave del usuario del SO, en lugar de una clave independiente y que no tiene por qué ser la misma como pasaba cuando usábamos autenticación *Basic* y *Digest*. Si implementamos este esquema en una web en producción **es imperativo que las comunicaciones estén cifradas**: necesitamos TLS, algo que veremos en la próxima sección.

Para finalizar este ejemplo, en la siguiente imagen podemos ver que para usar grupos en lugar de usuarios los cambios a la configuración son mínimos. Para entender el ejemplo debe recordarse que por defecto **cuando se crea un usuario en *Linux* el usuario pertenece a un grupo llamado como él mismo**. La siguiente imagen muestra los cambios a realizar, que básicamente son la última línea:

```
        <IfModule mod_authnz_external.c>
                AddExternalAuth pwauth /usr/sbin/pwauth
                SetExternalAuthMethod pwauth pipe_

                <Directory /var/www/html/web2>
                        AuthType Basic
                        AuthName "Directorio para operarios"
                        AuthBasicProvider external
                        AuthExternal pwauth

                        Require unix-group operario
                </Directory>
        </IfModule>
```

Figura 178. Autenticación con grupos del SO

Autenticación con un LDAP

Finalmente, algo más frecuente y aconsejable que la opción anterior es la autenticación contra un servidor *LDAP* que contenga usuarios de una organización o similar. Esto también es posible mediante módulos, de *Apache 2* concretamente gracias a `mod_authnz_ldap` (http://httpd.apache.org/docs/current/en/mod/mod_authnz_ldap.html). Se pueden encontrar instrucciones de cómo hacer esto en los siguientes enlaces:

- https://techexpert.tips/es/apache-es/apache-autenticacion-ldap-en-active-directory/
- https://asirfvg.wordpress.com/2014/11/26/autentificacion-con-ldap-en-apache/
- https://www.sahw.com/wp/archivos/2009/06/08/autenticacion-en-apache-mediante-mod_authnz_ldap-soporte-openldap/

Habilitación del tráfico cifrado

Transport Layer Security o TLS, actualmente en su versión 1.3, es un método para **cifrar el tráfico que circula por una red**. Es un estándar industrial que se usa típicamente para cifrar tráfico *HTTP*, aunque también puede usarse con otros protocolos. Debe usarse siempre que se mande información sensible entre cliente y servidor (credenciales, datos personales, autenticación *Basic* y *Digest*…) y el método de transmisión que se use no incluya ya ninguna forma de cifrado sobre los datos enviados. Por este motivo, usar TLS es imprescindible para hacer seguras autenticaciones como las vistas en la sección anterior.

TLS *es la evolución del protocolo SSL. Curiosamente, muchas veces se usa su antigua denominación aunque realmente se esté hablando de* TLS

TLS usa certificados y cifrado de clave pública para identificar a los interlocutores y crear transmisiones seguras. Debido a que usa certificados, TLS puede usarse también para asegurarnos de la identidad de un interlocutor, evitando suplantaciones. A grandes rasgos, el proceso que se sigue para cifrar tráfico por TLS es el siguiente:

- Un cliente **pide un recurso mediante *https*** (suponemos que el servidor está configurado para permitirlo, algo que veremos cómo se hace posteriormente).
- Ante esta petición, el servidor **devuelve su certificado** TLS. Entonces el cliente hace lo siguiente sobre dicho certificado:
  - Comprueba que **no está caducado**.
  - Comprueba que **el nombre del servidor** al que se conecta es el mismo que el que tiene el certificado (evita suplantación de sitios *web*).
  - Comprueba que **el certificado ha sido emitido por una autoridad certificadora (*CA*) de confianza**, es decir, comprueba que el certificado es creíble:
    - Los navegadores tienen una **lista de *CAs* de confianza** (como *Verisign*, *Thawte, etc.*). Como los navegadores confían en ellas, confían en los certificados que emiten y los dan por válidos.
    - Estas *CAs* se supone que **verifican** que quienes les piden certificados **son usuarios válidos y legítimos**, por lo que son entidades de confianza (como si fuesen notarios).
    - Por tanto, un navegador **no confiará por defecto en otras *CA* desconocidas**. No obstante, podemos montar una propia (de la empresa concreta) y configurar al navegador para que confíe en sus certificados manualmente. Esto suele hacerse sobre todo en *intranets, webs* no públicas o empresas que no quieran hacer una inversión económica por este concepto.
- Si el certificado no cumple con algún requisito, **avisa al usuario de los problemas encontrados** y tendrá que decidir si continúa con la operación.
- En caso de que sea válido o el usuario continúe, el navegador extrae la clave pública del servidor de su certificado TLS.
- Posteriormente **genera una clave de sesión** (*session key*) aleatoria y la cifra con dicha clave pública, devolviendo este dato al servidor.
- El servidor **descifra la *session key* con su clave privada**. Las futuras comunicaciones entre cliente y servidor usarán cifrado simétrico usando esta *session key* como clave.
- **El cifrado simétrico es más rápido que el basado en clave pública**, por lo que se usa en el resto de la comunicación por temas de rendimiento.

Aunque como método para asegurar la confidencialidad de las transmisiones TLS es muy adecuado, su uso plantea ciertos problemas / limitaciones:

- Cualquier información **que se transmita como parte de la URI de la petición** (la *query string*) **no se cifra**, ya que la URI es necesaria para localizar el servidor de destino y no puede cifrarse. Por tanto, si usamos la *query string* para pasar información sensible, cualquiera podrá leerla.
- La información se transmite cifrada, pero **estará descifrada tanto en la máquina cliente como en la servidora**, para que éstos pueden manejarla correctamente. Si alguno de ellos se compromete, el atacante podrá leerla igualmente.
- **Existen problemas potenciales a la hora de usar múltiples *webs* con TLS sobre la misma *IP*** si existen varios host virtuales que se distinguen por su nombre, ya que la información para distinguir un host de otro está en las cabeceras HTTP y estas están cifradas cuando llegan al servidor, por lo que no se podrá leer su contenido y por tanto no se podrá seleccionar en su debido momento el certificado del host virtual adecuado.

  Existen dos soluciones a este problema: que las webs se ejecuten cada una con su propia IP (asignando varias IPs a la misma tarjeta de red, por ejemplo) o bien usando *SNI* (*Server Name Identification*) a través de la directiva `SSLStrictSNIVHostCheck` (**Más información**: https://wiki.apache.org/httpd/NameBasedSSLVHostsWithSNI).

Dicho esto, podemos pasar a ver un ejemplo. Para activar el uso de *SSL* en *Apache 2* necesitamos completar los siguientes pasos:

- Instalar el paquete instalable de *Apache 2* `ssl-cert`.
- **Generar el certificado** con las herramientas proporcionadas por dicho paquete.
- En nuestro caso será ***self-signed*** (autofirmado), ya que de esta forma es gratuito y nos sirve para ilustrar lo que queremos. Si queremos enviar un certificado a una *CA* de confianza, habría que contratar la emisión por una CA específica y seguir sus instrucciones o bien usar, de forma gratuita, ***Let's Encrypt*** (visto después)
- **Usar TLS sobre la web completa** (hoy en día se debe erradicar el uso de HTTP en webs en producción)
- Determinar si queremos acceso a una misma localización mediante HTTP y HTTPS o bien solo mediante HTTPS. Esto se puede hacer añadiendo la directiva `SSLRequireSSL` al contenedor que englobe la localización sobre la que estamos aplicando cifrado. De esta forma, forzamos a que un sitio sólo pueda ser accedido de forma segura. De permitir HTTP, **debe redirigir automáticamente al sitio HTTPs** si estamos en producción.

Debe tenerse en cuenta que al elegir un certificado autofirmado éste **dará errores en el navegador del cliente** por no ser emitido por una *CA* (*Certification Authority*) de confianza. No obstante, este hecho **no impide que el cifrado funcione**, aunque el cliente podría elegir no confiar en nuestra página por ese motivo, al no ser la autoridad certificadora o *CA* una reconocida o en la que confíe. Además, es posible que en algunas ocasiones el uso de `mod_security` (que veremos en la siguiente sección) pueda dar algún problema con el uso de TLS, siendo necesario modificar sus reglas en caso de que esto ocurra para ver la directiva que da el conflicto (que aparecerá en el archivo de *log* de este *mod*).

Dicho todo esto, ya podemos hacer un ejemplo para activar la comunicación cifrada, en este caso solo sobre el directorio `/web1`:

- Activar el mod que gestiona TLS (llamado `mod_ssl`, http://httpd.apache.org/docs/current/mod/mod_ssl.html) con el comando `a2enmod ssl` y reiniciar *Apache 2* para que el mod entre en funcionamiento.

```
root@ubuntu:/home/operario# sudo a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.
See /usr/share/doc/apache2/README.Debian.gz on how to configure SSL and create s
elf-signed certificates.
To activate the new configuration, you need to run:
  service apache2 restart
root@ubuntu:/home/operario#
```

Figura 179. Activando mod_ssl

- Crear un directorio `/ssl` dentro de la estructura de directorios de *Apache 2* para contener los certificados generados. El nombre no tiene por qué ser ese, es simplemente un ejemplo.
- Crear el certificado autofirmado que va a usar *SSL* con el siguiente comando, que usará el nombre del directorio creado anteriormente: `/usr/sbin/make-ssl-cert /usr/share/sslcert/ssleay.cnf /etc/apache2/ssl/apache.pem`

```
operario@ubuntu:~$ sudo /usr/sbin/make-ssl-cert /usr/share/ssl-cert/ssleay.cnf /
etc/apache2/ssl/apache.pem_
```

Figura 180. Creación de un certificado para el servidor

- Este proceso podría **preguntar algunos datos para generar el certificado** además del propio nombre de la máquina. Básicamente son datos que facilitan la localización del emisor de este. Es conveniente contestar con localizaciones y direcciones reales si se nos pidiesen. Si el certificado se fuese a usar con **múltiples nombres DNS**, conviene especificarlos todos aquí, siguiendo las propias instrucciones que nos da la herramienta de configuración.

Figura 181. Nombre de la máquina para generar el certificado

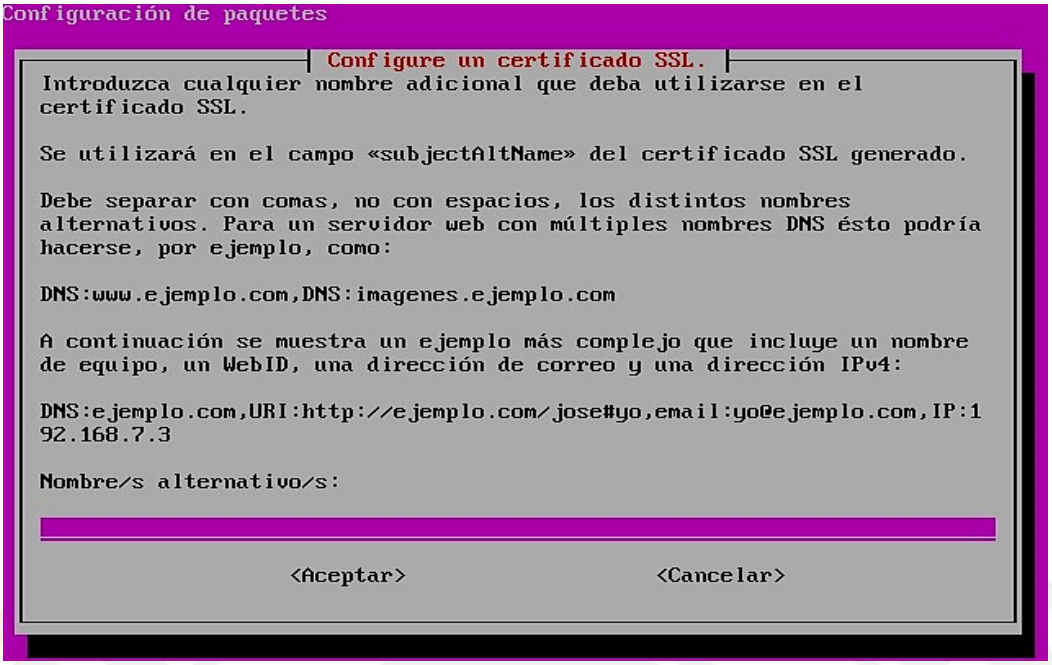


Figura 182. Nombres de dominio sobre los que se aplica el certificado

Hecho esto, podemos optar por dos formas de hacer uso de TLS. La primera y más sencilla es hacer uso de un host virtual por defecto preparado para ello que se proporciona automáticamente las últimas versiones de *Apache 2*, y que se llama `default-ssl`. Esto puede hacerse activándolo con `sudo a2ensite defaultssl` y reiniciando *Apache 2*. A efectos prácticos, esto permitirá usar TLS sobre el *host* virtual por defecto directamente, pudiendo acceder mediante HTTPS a todo el contenido que ya estuviese en él.

```
operario@ubuntu:~$ sudo a2ensite default-ssl
Enabling site default-ssl.
To activate the new configuration, you need to run:
  service apache2 reload
```

Figura 183. Activando el host virtual por defecto para TLS

Figura 184. Aviso originado por usar un certificado autofirmado

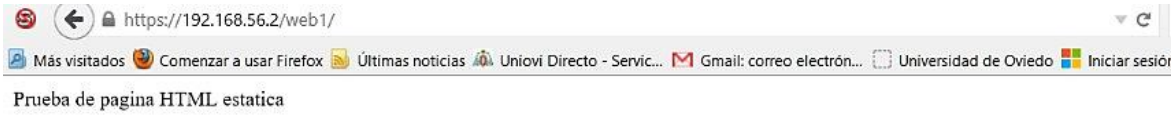


Figura 185. Página web bajo HTTPS

Aunque todavía no hemos visto en detalle los que son los host virtuales, sí que hemos mencionado que son la forma de tener varias webs en un mismo servidor. Si queremos **activar TLS sobre otro *host* virtual que no sea el de por defecto**, simplemente necesitamos configurar TLS en uno de los *host* virtuales que tengamos y que queramos que tenga esa posibilidad de uso. Si por ejemplo tenemos un *host* virtual en la *IP* `5.0.0.4`, podemos añadir estas líneas al fichero de configuración particular de dicho *host* virtual. De esta forma podrán convivir bajo el mismo servidor webs que usen TLS y webs que no lo usen:

```
<VirtualHost 5.0.0.4:443>

ServerAdmin webmaster@localhost

DocumentRoot /var/www/site2

ErrorLog /var/log/apache2/site2-error_log

TransferLog /var/log/apache2/site2-access_log

SSLEngine on
SSLCertificateFile /etc/apache2/ssl/Apache 2.pem

SSLProtocol all
SSLCipherSuite HIGH:MEDIUM

</VirtualHost>
```

Comprobación de revocación de certificados

Vistos los conceptos básicos, no podemos terminar esta sección sin mencionar una serie de detalles adicionales que se deben tener en cuenta, sobre todo en webs en producción.

Lo primero es tener en cuenta que, si estamos usando un certificado emitido por una CA, es posible que **debido a un ataque o compromiso de la información de dicha CA este certificado haya sido revocado**. Hay una forma de obligar al servidor a chequear si su certificado ha sido revocado y es activando el uso de *Online Certificate Status Protocol* (*OCSP*). Este es un mecanismo que determina si un certificado de servidor ha sido revocado. Concretamente `mod_ssl` puede usar una implementación de este llamada *OCSP Stapling*, que consiste en que el servidor mantiene el estado *OCSP* de sus certificados (es decir, si está o no revocado) y envía dicho estado a los clientes que se comunican con él.

¿Cómo sabe el servidor si su certificado ha sido revocado? La mayor parte de certificados emitidos por una CA comercial tienen junto con el resto de su información **la dirección de un servidor *OCSP*** mantenido por la propia CA que generó el certificado. `mod_ssl` se comunica con dicho servidor para obtener así el estado actual de su certificado. Nótese que de esta forma los clientes no se comunican con la CA (por lo que esta no puede trazar las páginas web que los clientes visitan) y se optimiza el rendimiento del proceso ya que se elimina dicha comunicación y el estado *OCSP* del certificado se reutiliza para todos los clientes.

Habilitar *OCSP* es sencillo, aunque para poder hacerlo como es lógico necesitamos que el certificado del servidor haya sido emitido por una CA que tenga un servidor *OCSP*, no sirve con nuestro ejemplo de certificado autofirmado:

```
SSLUseStapling On
SSLStaplingCache "shmcb:logs/ssl_stapling(32768)"
```

Uso de suites de cifrado lo más seguras posible

Ante la gran cantidad de ***exploits*** (por ejemplo el conocido *Heartbleed*) basados en realizar un mal uso de TLS o bien aprovecharse de vulnerabilidades presentes en versiones antiguas de este protocolo que pudieran usarse debido a una mala configuración, es importante asegurarnos de dos aspectos importantes que describiremos a continuación.

El primer aspecto es asegurarnos de que **no estamos usando versiones de SSL/TLS viejas**. `mod_ssl` podría permitir métodos de cifrado SSL/TLS antiguos en aras a lograr una mayor compatibilidad con clientes que usen SO también antiguos. Sin embargo, estos métodos de autenticación **tienen problemas de seguridad conocidos** y permitir su uso es un riesgo potencial que no debemos permitir. Para deshabilitar métodos de cifrado vulnerables de entre los soportados por `mod_ssl` se añaden líneas como esta a donde tengamos habilitado el cifrado: `SSLProtocol All -SSLv2 -SSLv3`. `All` significa habilitar el resto de *las suites* de cifrado, pero es algo peligroso porque a medida que evoluciona el *mod* se van "cayendo" de la lista versiones muy viejas y aparecen versiones nuevas.

Es posible **que algunos clientes pierdan el acceso a la web** si hacemos esto por no soportar métodos de cifrado más modernos. Esto es realmente preferible a ser vulnerable a un ataque, salvo que el nº de clientes con problemas fuese muy elevado, en cuyo caso habría que pensar en hacer una actualización de software de los clientes más que en aumentar la vulnerabilidad del servidor.

El segundo aspecto es usar algoritmos de cifrado seguros, especialmente aquellos que soporten PFS (***Perfect Forward Secrecy***). PFS es una técnica que hace que cada sesión de comunicación con TLS que haya entre cliente y servidor use una clave simétrica de comunicaciones nueva en lugar de reutilizar una anterior. De esta manera, si dicha clave es robada, la comunicación solo estará comprometida un periodo mínimo, por lo que no podrá usarse dicha clave para averiguar los contenidos transmitidos posteriormente.

En https://raymii.org/s/tutorials/Strong_SSL_Security_On_apache2.html podemos encontrar un estudio de los ataques a los mecanismos de cifrado más comunes existentes actualmente y se recomienda su lectura para entender la gravedad del problema y las posibles acciones a tomar. En nuestro caso, para saber cuál es la configuración recomendada en cada momento, podemos usar la que nos recomiendan en esta web poniendo las versiones adecuadas del software que usemos: https://ssl-config.mozilla.org/#server=apache&version=2.4.41&config=intermediate&openssl=1.1.1k&guideline=5.7

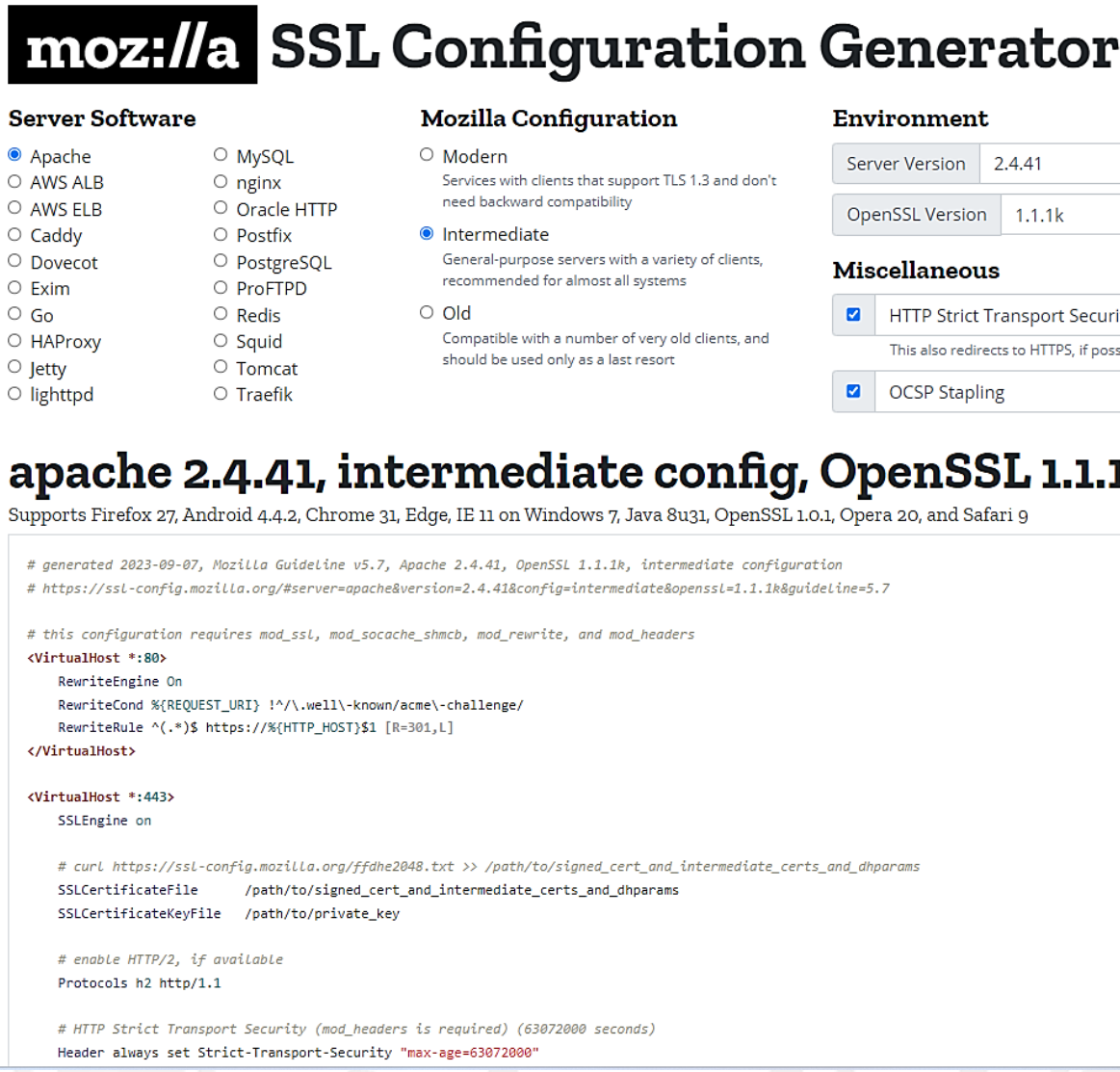


Figura 186. Configuración recomendada por Mozilla para Apache 2

Y añadirla luego al fichero correspondiente del *VirtualHost*, copiando y pegando las opciones que nos proporciona.

Figura 187. Permitiendo solo suites y protocolos de cifrado seguros

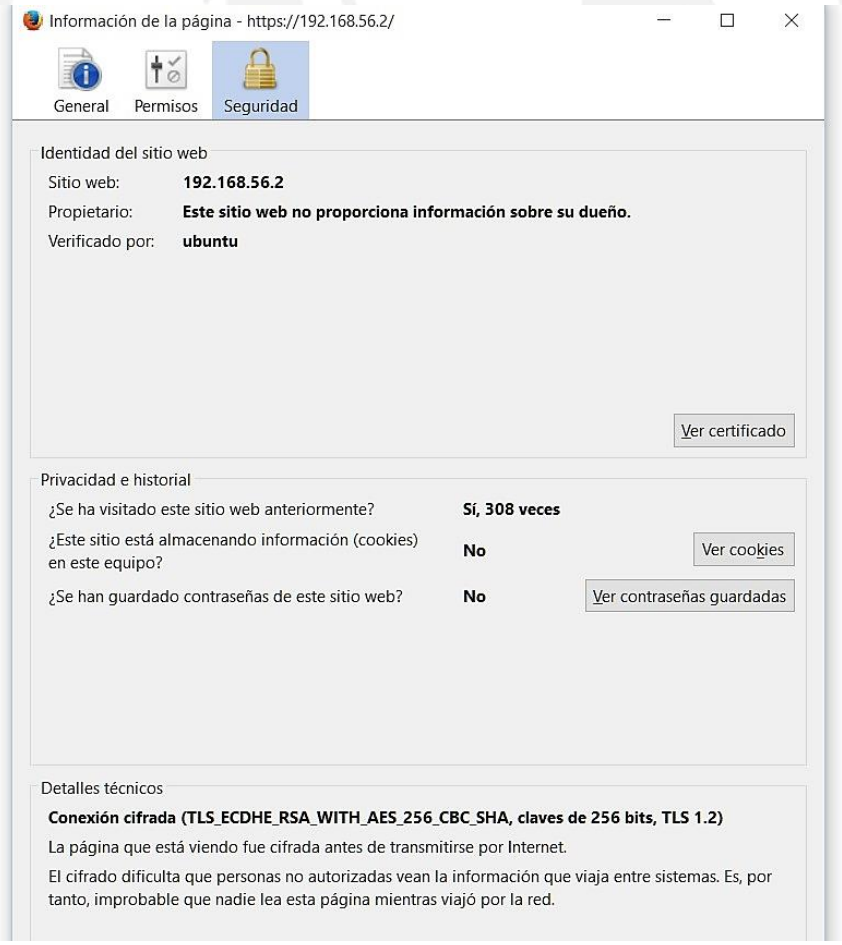


Figura 188. Vista en el navegador de las características de cifrado

Como se ve en la imagen anterior, nuestra página usa TLS 1.2, contra el que no se conoce ningún ataque en estos momentos.

Paso a producción: *Let's Encrypt* y *Certbot*

Como dijimos anteriormente, *Let's Encrypt* (https://letsencrypt.org/es/) es **una forma gratuita** de conseguir un certificado aceptado por todos los navegadores actuales que nos permita implementar cifrado de forma adecuada **en un servidor en producción** (hace falta tener un dominio asignado). Si tenemos acceso *shell* al servidor, esta CA usa el protocolo ACME que, a través de un programa llamado `certbot`, te permite instalar el certificado de forma muy sencilla siguiendo unos pasos documentados en su web oficial: https://letsencrypt.org/es/getting-started/. Además, se ocupa de su renovación automática, despreocupándonos de mantener el certificado si todo va bien. Esta imagen muestra un ejemplo:

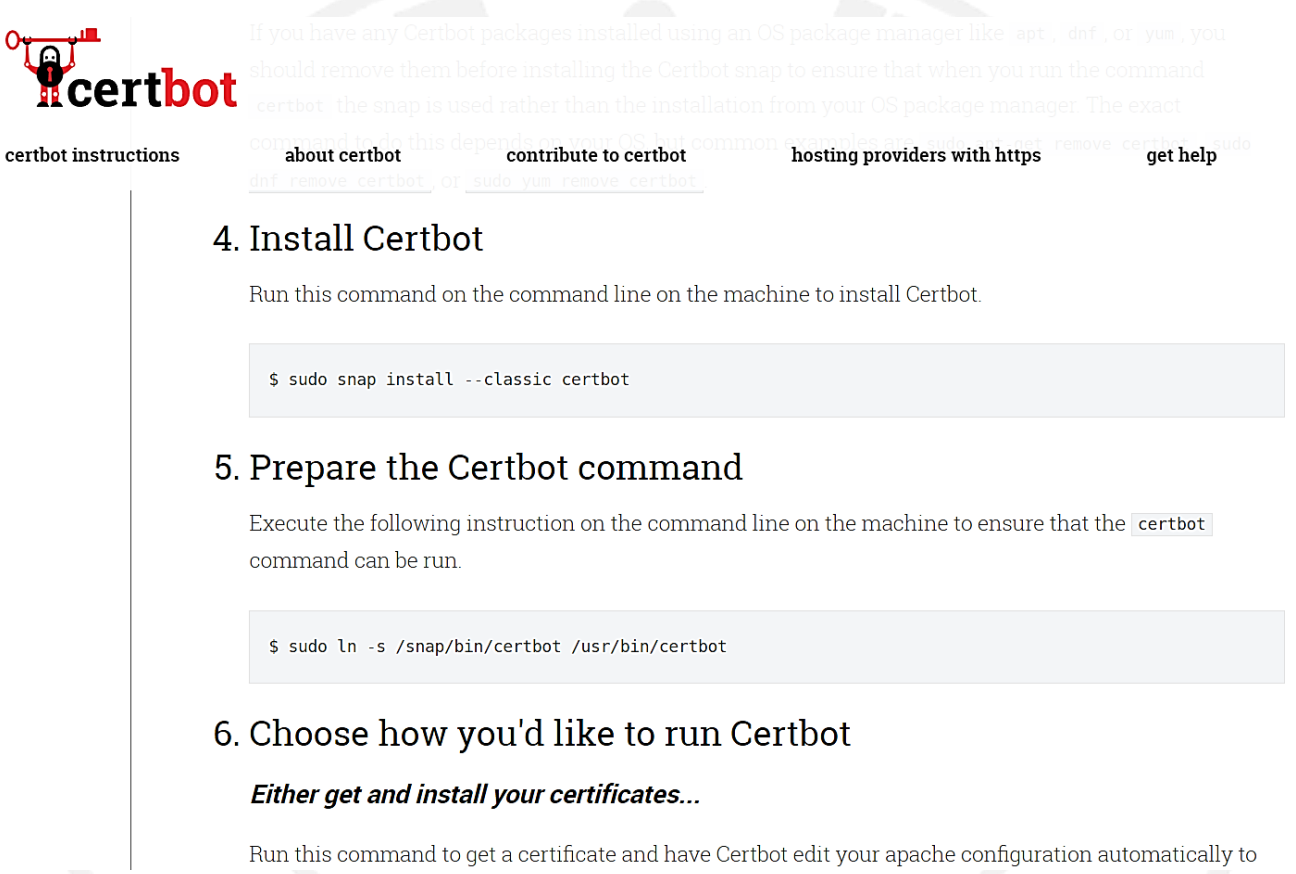


Figura 189. Instrucciones concretas para certbot tras decirle a la web que tenemos un Apache 2 en un Ubuntu 20.04

Si no tenemos acceso *shell* al servidor hay otras alternativas e **integraciones** con proveedores de nube conocidos en la misma página que podemos usar para instalar el certificado.

Más información

Obviamente si necesitamos usos más específicos de TLS nos encontraremos con más preguntas a resolver y con configuraciones más complejas que se adapten a casos particulares. Es importante pues tener unas referencias de dónde se pueden consultar qué hacer en situaciones más específicas. *Apache 2* ha pensado en eso y ha diseñado una FAQ que responde a muchas preguntas y situaciones que pudieran darse, y que se recomienda consultar si fuese necesario realizar alguna operación adicional no cubierta en este apartado. Dicho FAQ puede consultarse aquí: http://httpd.apache.org/docs/current/ssl/ssl_howto.html.

Por otro lado, si queremos tener una estimación de la seguridad de un certificado que estemos usando en cualquier momento podemos usar la utilidad presente en el siguiente enlace: https://raymii.org/s/software/OpenSSL_Decoder.html

Instalación y puesta en marcha de WAFs

`mod_security` es un *firewall* de aplicaciones web (***Web Application Firewall*** o *WAF*). Esta clase de programas pueden venir distribuidos de varias formas: como una máquina virtual preconfigurada (*appliance*) o como un complemento del servidor, siendo este último nuestro caso. El objetivo de esta clase de programas es básicamente aplicar un conjunto de reglas a las peticiones HTTP que le llegan.

Normalmente, estas reglas están orientadas a identificar y parar multitud de ataques conocidos a las webs, como son los conocidos *XSS* y *SQL Injection*. En definitiva, si conseguimos tener un conjunto de reglas adecuado podremos parar automáticamente muchas clases diferentes de ataques, independientemente de si la aplicación implementa medidas de seguridad o no y del lenguaje en el que la aplicación web esté programada. Como la mejor aproximación de cara a la seguridad es la defensa implementada en varias capas, `mod_security` debería ser un complemento para mejorar la seguridad y su instalación no debe ser una excusa para "relajar" este aspecto en cualquier web alojada en un servidor que lo tenga instalado.

Nosotros veremos la instalación de `mod_security` sobre *Apache 2*, pero un proceso similar puede hacerse sobre otros servidores web como *Ngnix* o *IIS*. Hay que tener en cuenta también que, por el propio planteamiento del programa, la potencia de este dependerá exclusivamente **de la calidad de su sistema de reglas**. Por suerte, el *mod* se distribuye con un conjunto de reglas que están pensadas para parar ataques comunes como *XSS*, *SQL Injection*, troyanos, peticiones corruptas o maliciosas de diferentes tipos, robo de sesiones, entre otros.

Instalación de *mod_security 2*

Para instalar el *mod* hay que hacer `apt-get install libapache2-modsecurity`. El *mod* debería aparecer, como de costumbre, en `mods-enabled`. No obstante, antes de reiniciar el servidor conviene renombrar el fichero de configuración recomendado que viene con la instalación `/etc/modsecurity/modsecurity-recommended.conf` a `/etc/modsecurity/modsecurity.conf`.

```
operario@ubuntu:~$ sudo apt-get install libapache2-modsecurity
[sudo] password for operario:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes extras:
  libapache2-mod-security2 liblua5.1-0 modsecurity-crs
Paquetes sugeridos:
  lua geoip-database-contrib ruby
Se instalarán los siguientes paquetes NUEVOS:
  libapache2-mod-security2 libapache2-modsecurity liblua5.1-0 modsecurity-crs
0 actualizados, 4 se instalarán, 0 para eliminar y 3 no actualizados.
Necesito descargar 565 kB de archivos.
Se utilizarán 4.143 kB de espacio de disco adicional después de esta operación.
¿Desea continuar? [S/n]
```

Figura 190. Instalación de mod_security

```
.2.8-1 [265 kB]
Descargados 565 kB en 1seg. (513 kB/s)
Seleccionando el paquete liblua5.1-0:amd64 previamente no seleccionado.
(Leyendo la base de datos ... 58521 ficheros o directorios instalados actualment
e.)
Preparing to unpack .../liblua5.1-0_5.1.5-5ubuntu0.1_amd64.deb ...
Unpacking liblua5.1-0:amd64 (5.1.5-5ubuntu0.1) ...
Seleccionando el paquete libapache2-mod-security2 previamente no seleccionado.
Preparing to unpack .../libapache2-mod-security2_2.7.7-2_amd64.deb ...
Unpacking libapache2-mod-security2 (2.7.7-2) ...
Seleccionando el paquete libapache2-modsecurity previamente no seleccionado.
Preparing to unpack .../libapache2-modsecurity_2.7.7-2_all.deb ...
Unpacking libapache2-modsecurity (2.7.7-2) ...
Seleccionando el paquete modsecurity-crs previamente no seleccionado.
Preparing to unpack .../modsecurity-crs_2.2.8-1_all.deb ...
Unpacking modsecurity-crs (2.2.8-1) ...
Configurando liblua5.1-0:amd64 (5.1.5-5ubuntu0.1) ...
Configurando libapache2-mod-security2 (2.7.7-2) ...
apache2_invoke: Enable module security2
 * Restarting web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress th
is message
                                                                    [ OK ]
Configurando libapache2-modsecurity (2.7.7-2) ...
Configurando modsecurity-crs (2.2.8-1) ...
Processing triggers for libc-bin (2.19-0ubuntu6.6) ...
operario@ubuntu:~$ sudo cp /etc/modsecurity/modsecurity.conf-recommended /etc/mo
dsecurity/modsecurity.conf
```

Figura 191. Renombrado del fichero de configuración tras la instalación

Hecho esto, ya podemos reiniciar el servidor y empezar a configurarlo. Otra pista de que el *mod* está trabajando correctamente es la aparición de un nuevo fichero de log en `/etc/apache2/log` llamado `modsec_audit.log`.

```
operario@ubuntu:~$ sudo service apache2 restart
 * Restarting web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress th
is message
                                                                    [ OK ]
operario@ubuntu:~$ ls -la /var/log/apache2/
total 180
drwxr-x---  2 root adm     4096 ago 17 17:36 .
drwxrwxr-x 10 root syslog  4096 ago 14 21:32 ..
-rw-r-----  1 root adm    87105 ago 17 15:40 access.log
-rw-r-----  1 root adm    77225 ago 17 17:36 error.log
-rw-r-----  1 root root       0 ago 17 17:36 modsec_audit.log
-rw-r-----  1 root adm        0 ago  4 11:07 other_vhosts_access.log
operario@ubuntu:~$ _
```

Figura 192. Fichero de log de mod_security

Configurar *mod_security 2*

Es importante recalcar de nuevo el tema de las reglas de este *mod*. Sin reglas, este *mod* **no es capaz de hacer absolutamente nada**. Y no solo es necesario contar con reglas, sino que además es necesario decirle al *mod* que queremos usarlas y cuales concretamente. Por defecto, la configuración del *mod* en

cuanto a las acciones a tomar cuando se comprueban las reglas tiene el valor `DetectionOnly`. Esto significa que si las reglas identifican un patrón en las peticiones que encuentra sospechoso **el módulo no ejecutará la acción especificada por la regla** ante la presencia de este patrón (parar la petición, etc.), solamente tomará nota de este en el log.

¿Por qué se hace esto? Porque hay que reconocer que `mod_security` es un módulo muy potente y complejo, pero que **su puesta en marcha puede alterar el funcionamiento normal de webs que están en producción**. De hecho, es frecuente que `mod_security` deba ir adaptándose a medida que se van introduciendo más webs o cambiando las existentes en el servidor. Para probar el funcionamiento del *mod* como un *WAF* real que ejecute acciones basadas en las reglas que posee, solamente tenemos que ir al fichero de configuración `/etc/modsecurity/modsecurity.conf` y cambiar la línea `SecRuleEngine DetectionOnly` por `SecRuleEngine On`. Por los motivos que hemos dicho, hacer esto en producción directamente es arriesgado, conviene probarlo primero en un servidor de pruebas por si interfiriese con el funcionamiento de alguna web.

Una vez puesto en marcha hay que prestar atención a otras directivas:

- `SecRequestBodyAccess`: Permite a `mod_security` examinar con sus reglas las peticiones entrantes. Si ponemos `On` como valor de esta directiva, `mod_security` podrá examinar **la totalidad de la petición** que le llega (cuerpo incluido, lo que incluye parámetros `POST`), y eso es normalmente lo deseable para detectar mejor los posibles ataques. No obstante, esto aumenta el consumo de RAM y, si el rendimiento es un problema, entonces quizá pueda considerarse desactivarlo, aunque estaremos perdiendo efectividad. Para controlar el máximo de RAM consumido por esta característica podemos consultar y configurar al valor más bajo posible de las directivas:
  - `SecRequestBodyLimit`: Tamaño máximo de los datos enviados en una petición `POST`. Si se supera, el *mod* responderá al cliente con un error `413 Request Entity Too Large`. La clave para poder bajar el valor lo más posible es saber si la aplicación tiene subida de ficheros entre sus características: si no lo tiene, podemos calcular un valor muy bajo y optimizar la memoria consumida.
  - `SecRequestBodyNoFilesLimit`: Es el tamaño anterior quitando el tamaño de los posibles ficheros subidos. De igual forma, si no hay necesidad de subir ficheros podemos reducir mucho su valor.
  - La última directiva relativa a las peticiones y que puede afectar enormemente al rendimiento es `SecRequestBodyInMemoryLimit`. Determina qué parte de la petición se mantiene en RAM y qué parte se guarda en el *swap*. Claramente, la velocidad del disco duro afectará enormemente al rendimiento si la petición se va parcialmente al *swap*, pero la cantidad de RAM disponible también afecta a esta decisión.
- `SecResponseBodyAccess`: Es el equivalente a la anterior, **pero con las respuestas en lugar de con las peticiones**. Poner esta directiva a `On` permite al `mod_security` leer y almacenar los cuerpos de las respuestas a las peticiones para que sus reglas puedan examinarlos, aumentando a cambio el consumo de recursos y el tamaño del log. Tiene sentido si necesitamos protección contra fugas de datos. No obstante, la mayoría de las veces los administradores se preocupan más de lo que les llega que de lo que sale de sus servidores, por lo que es más frecuente encontrarse esta directiva a `Off` que la anterior.

  Otra forma de optimizar más esta opción es usar `SecResponseBodyMimeType` para restringir los tipos MIME de las respuestas que **SI** queremos que `mod_security` examine, como podrían ser por ejemplo `text/html` o `null`, significando este último valor cualquier tipo MIME desconocido por

no especificarse en la cabecera de la respuesta correspondiente (lo que podría indicar la existencia de una petición maliciosa).

Como puede verse, hay múltiples directivas que controlan el rendimiento del *mod* y que deben configurarse en función del tipo de webs servidas y también de los recursos disponibles. `mod_security` es un *mod* que es muy potente pero también puede consumir una cantidad ingente de recursos, por lo que su puesta en marcha requiere un examen pormenorizado de lo que se va a hacer en el servidor, de lo que queremos que consuma el *mod* y de otros factores, como las funcionalidades usadas por las webs servidas o de los permisos de ciertos directorios. Todas esas directivas y sus valores por defecto se pueden encontrar en `/etc/modsecurity/modsecurity.conf` (el fichero que hemos renombrado anteriormente), como se ve en la siguiente imagen:

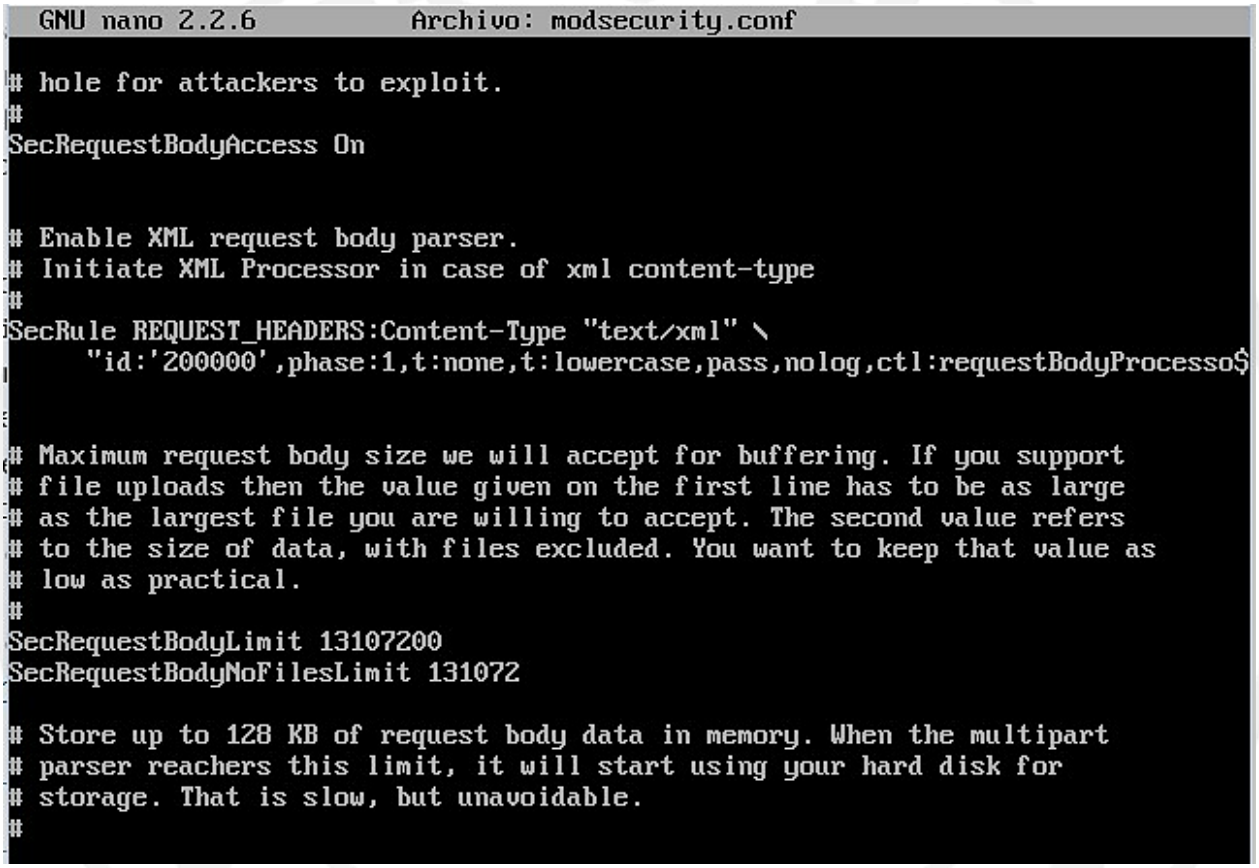


Figura 193. Directivas de mod_security

Nosotros para las pruebas que realizaremos dejaremos estas directivas en su valor recomendado. Para terminar esta sección, cabe decir que conocer en profundidad `mod_security` es un tema que se sale de los objetivos de este libro, pero se puede profundizar más sobre aspectos de instalación en: https://www.feistyduck.com/library/modsecurity-handbook-free/online/ch03-configuration.html y también sobre cómo mejorar el rendimiento del *mod* en: https://www.trustwave.com/Resources/SpiderLabs-Blog/ModSecurity-Performance-Recommendations/

Prueba del módulo

La mejor forma de probar el *mod* es intentar simular un ataque sobre el servidor y ver si el *mod* lo identifica y lo para. Para ello haremos un sencillo ejemplo de *SQL Injection* con una página PHP de prueba preparada al efecto y que situaremos en el directorio `/php`. Para su ejecución es necesario instalar el soporte de PHP

en el servidor siguiendo estas instrucciones: https://ubuntu.com/server/docs/programming-php. La página de pruebas será la siguiente:

Fichero `/var/www/html/php/login.php`:

```
<html>
	<head>
		<title>Pagina de login de pruebas</title>
	</head>
	<body>
	<?php
	if(isset($_POST['login']))
	{
		$usuario = $_POST['nombreusuario'];
		$password = $_POST['password'];
		/*Simulamos una consulta a la BD*/
  $consulta = "SELECT * FROM Usuarios WHERE nombre='$usuario' AND
password='$password'";
		print "<p>Se ejecutaria la sentencia SQL: $consulta</p>";
	}
	else
	{
	?>
		<form action="" method="post">
			Nombre: <input type="text" name="nombreusuario"/><br/>
			Password: <input type="password" name="password"/><br/>
			<input type="submit" name="login" value="login"/>
		</form>
	<?php
	}
	?>
	</body>
</html>
```

La página muestra un formulario de *login* sencillo y el intento de acceso muestra la sentencia SQL que se ejecutaría si realmente tuviéramos una BD operativa. Nótese que la sentencia SQL se construye mediante concatenación de *strings* en lugar de `PreparedStatements` o similar. **Nunca se recomienda hacer lo primero si queremos evitar problemas de *SQL Injection*.** Probamos que la web funciona accediendo a ella e introducimos datos válidos:

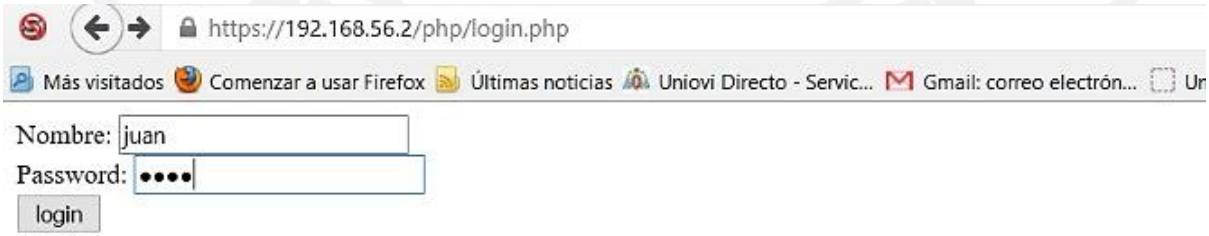


Figura 194. Página de login para pruebas y datos válidos

Figura 195. Respuesta de la página ante una entrada válida

Introducimos ahora un **intento clásico de *SQL Injection*** en el *login* y dejamos en blanco la *password*: `' or true --`

Figura 196. Ataque clásico de SQL Injection

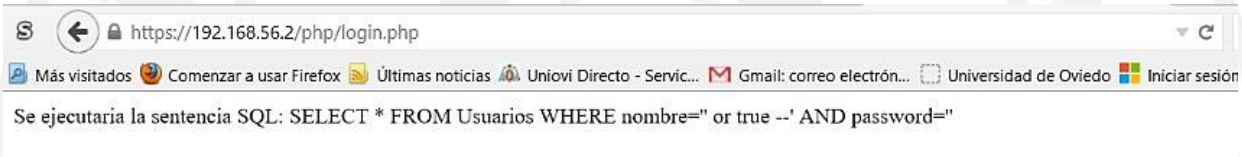


Figura 197. Resultado del SQL Injection

En la anterior imagen puede verse como la segunda sentencia es extremadamente peligrosa puesto que permitiría hacer un ataque clásico de *SQL Injection* entrando como el usuario juan sin conocer su clave. Además, la página no contiene ningún mecanismo de seguridad integrado en su código, por lo que podríamos hacer cualquier tipo de *SQL Injection* que se nos antojase. Con este marco de pruebas, ahora ya estamos listos para poner en marcha `mod_security` y sus reglas y comprobar que funciona.

Incluyendo las core rule set y poniendo en marcha el motor de detección

Como decíamos antes, para facilitarnos el trabajo con `mod_security` se incluye este conjunto de reglas muy completo y pensado para todo tipo de usos y que resulta, sobre todo, mucho más completo que cualquier conjunto que podamos preparar nosotros manualmente, ahorrándonos también tener que aprender la sintaxis de las reglas de `mod_security` en detalle. Este conjunto de reglas se llama *CRS* (***Core Rule Set***) y se encuentra en `/usr/share/modsecurity-crs/`. La documentación de estas puede consultarse `/usr/share/doc/modsecurity-crs/`. CRS detecta una enorme cantidad de posibles problemas de seguridad potenciales y sin duda contar con él puede salvarnos de muchos intentos de ataques. Entre los problemas identificados podemos destacar:

- Protección HTTP: Detecta violaciones del protocolo.
- *Blacklisting* de IPs en tiempo real: Usando servidores de reputación de IPs de terceros
- Detección de malware web: Identifica contenido malicioso comprobándolo con la API de *Google Safe Browsing*.
- Protección DoS: Proporciona mecanismos de defensa contra ataques de saturación de peticiones o contra ataques basados en mandar peticiones con retardos preprogramados.

- Detección de ataques web comunes
- Detección de *bots*, *crawlers*, *scanners* y otro tipo de herramientas maliciosas similares.
- Integración con antivirus para examinar los ficheros subidos
- Identificación de datos sensibles: Como datos de tarjetas de crédito, bloqueando su posible robo.
- Protección contra troyanos
- Identifica problemas de configuración de las aplicaciones
- Detecta y oculta mensajes de error enviados por el servidor para que no den demasiada información.
- Previene de que se puedan servir ficheros con código fuente de la aplicación, por ejemplo por la presencia de *backups* o versiones viejas de una determinada página.

Activar las reglas básicamente consiste en decirle al *mod* que busque ficheros de reglas en un determinado directorio. Para ello simplemente tenemos que editar el fichero `/etc/apache2/mods-enabled/security2.conf` y añadir las siguientes directivas dentro del bloque `<IfModule security2_module>`:

```
<IfModule security2_module>
        # Default Debian dir for modsecurity's persistent data
        SecDataDir /var/cache/modsecurity

        # Include all the *.conf files in /etc/modsecurity.
        # Keeping your local configuration in that directory
        # will allow for an easy upgrade of THIS file and
        # make your life easier
        IncludeOptional /etc/modsecurity/*.conf

        Include /usr/share/modsecurity-crs/*.conf_
        Include /usr/share/modsecurity-crs/activated_rules/base/*.conf
</IfModule>
```

Figura 198. Indicando a mod_security el conjunto de reglas a usar

El directorio `activated_rules` tiene una función similar al directorio `mods-enabled` de *Apache 2* y por tanto en el declararemos, mediante enlaces simbólicos las reglas de `mod_security` que queremos aplicar en nuestra instalación. Ha de tenerse en cuenta que todas las reglas que vienen con el módulo se encuentran en estos tres directorios:

```
/usr/share/modsecurity-crs/base_rules
/usr/share/modsecurity-crs/optional_rules
/usr/share/modsecurity-crs/experimental_rules
```

Nosotros haremos un enlace simbólico en `activated_rules` llamado `base` que apunte al directorio `base_rules` del cuadro anterior, para que encaje con la configuración que vimos en la imagen anterior. De esta manera aplicaremos todas la reglas básicas que vienen con el *mod*.

```
operario@ubuntu:/usr/share/modsecurity-crs$ sudo ln -s /usr/share/modsecurity-cr
s/base_rules/ /usr/share/modsecurity-crs/activated_rules/base
operario@ubuntu:/usr/share/modsecurity-crs$ _
```

Figura 199. Activando todas las reglas básicas de mod_security

En la siguiente imagen podemos ver el listado de ficheros que contienen las reglas que se van a aplicar, cada uno con todas las reglas relacionadas con una determinada funcionalidad:

```
operario@ubuntu:/usr/share/modsecurity-crs/activated_rules$ cd base
operario@ubuntu:/usr/share/modsecurity-crs/activated_rules/base$ ls
modsecurity_35_bad_robots.data
modsecurity_35_scanners.data
modsecurity_40_generic_attacks.data
modsecurity_50_outbound.data
modsecurity_50_outbound_malware.data
modsecurity_crs_20_protocol_violations.conf
modsecurity_crs_21_protocol_anomalies.conf
modsecurity_crs_23_request_limits.conf
modsecurity_crs_30_http_policy.conf
modsecurity_crs_35_bad_robots.conf
modsecurity_crs_40_generic_attacks.conf
modsecurity_crs_41_sql_injection_attacks.conf
modsecurity_crs_41_xss_attacks.conf
modsecurity_crs_42_tight_security.conf
modsecurity_crs_45_trojans.conf
modsecurity_crs_47_common_exceptions.conf
modsecurity_crs_48_local_exceptions.conf.example
modsecurity_crs_49_inbound_blocking.conf
modsecurity_crs_50_outbound.conf
modsecurity_crs_59_outbound_blocking.conf
modsecurity_crs_60_correlation.conf
operario@ubuntu:/usr/share/modsecurity-crs/activated_rules/base$
```

Figura 200. Ficheros de reglas básicas de mod_security

Si por alguna razón solo quisiéramos activar unos pocos conjuntos de reglas, basta con hacer un `Include` a `/activated_rules` y luego enlaces simbólicos a las reglas que queremos dentro de `/activated_rules`:

```
cd /usr/share/modsecurity-crs/activated_rules/
ln -s /usr/share/modsecuritycrs/base_rules/modsecurity_crs_41_sql_injection_attacks.conf .
```

Cualquier cambio a las reglas necesita un reinicio de *Apache 2* como de costumbre. En cualquier caso, si hemos hecho bien todo el proceso, ahora la petición con el vector de ataque de *SQL Injection* debería devolver un error `403 Fobidden` (`SecRuleEngine On`) o bien escribir la incidencia en el log (`SecRuleEngine DetecionOnly`):

```
 GNU nano 2.2.6    Archivo: /var/log/apache2/modsec_audit.log

--c2949117-A--
[17/Aug/2015:18:33:43 +0200] VdIM538AAQEAADAFSigAAAAD 192.168.56.1 64531 192.16$
--c2949117-B--
POST /php/login.php HTTP/1.1
Host: 192.168.56.2
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefo$
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: https://192.168.56.2/php/login.php
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 50

--c2949117-C--
nombreusuario=%27+or+true+--&password=&login=login
--c2949117-F--
HTTP/1.1 200 OK
X-Powered-By: PHP/5.5.9-1ubuntu4.11
Content-Length: 199
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html

--c2949117-E--
```

Figura 201. SQL Injection detectado por mod_security en modo DetectionOnly (I)

```
  GNU nano 2.2.6    Archivo: /var/log/apache2/modsec_audit.log

        <head>
                <title>Pagina de login de pruebas</title>
        </head>
        <body>
        <p>Se ejecutaria la sentencia SQL: SELECT * FROM Usuarios WHERE nombre=$
</html>

--c2949117-H--
Message: Warning. Pattern match "^[\\d.:]+$" at REQUEST_HEADERS:Host. [file "/u$
Message: Warning. Pattern match "(^[\"'`\xc2\xb4\xe2\x80\x99\xe2\x80\x98;]+|[\"$
Message: Warning. Pattern match "(?i:\\bor\\b ?(?:\\d{1,10}|[\\'\"][^=]{1,10}[\$
Message: Warning. Operator GE matched 5 at TX:inbound_anomaly_score. [file "/us$
Apache-Handler: application/x-httpd-php
Stopwatch: 1439829223306052 3612 (- - -)
Stopwatch2: 1439829223306052 3612; combined=2225, p1=263, p2=1783, p3=1, p4=94,$
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.7.7 (http://www.modsecurity.org/); OWASP_CRS$
Server: Apache/2.4.7 (Ubuntu)
Engine-Mode: "DETECTION_ONLY"

--c2949117-Z--
```

Figura 202. SQL Injection detectado por mod_security en modo DetectionOnly (II)

Dado que de esta forma el ataque tiene lugar igualmente, podemos pasar a una forma más activa de protección cambiando la manera de funcionar del módulo a **On**:

```
  GNU nano 2.2.6    Archivo: /etc/modsecurity/modsecurity.conf      Modificado

# -- Rule engine initialization ----------------------------------------

# Enable ModSecurity, attaching it to every transaction. Use detection
# only to start with, because that minimises the chances of post-installation
# disruption.
#
#SecRuleEngine DetectionOnly
SecRuleEngine On_

# -- Request body handling ---------------------------------------------

# Allow ModSecurity to access request bodies. If you don't, ModSecurity
# won't be able to see any POST parameters, which opens a large security
# hole for attackers to exploit.
#
SecRequestBodyAccess On


# Enable XML request body parser.
# Initiate XML Processor in case of xml content-type
#
SecRule REQUEST_HEADERS:Content-Type "text/xml" \
     "id:'200000',phase:1,t:none,t:lowercase,pass,nolog,ctl:requestBodyProcesso$
```

Figura 203. Activando mod_security como WAF

La respuesta al ataque de *SQL Injection* será ahora muy diferente:

Figura 204. SQL Injection interceptado y evitado por mod_security

Y, por supuesto, en el log de `mod_security` se registrará la incidencia de manera diferente:

```
  GNU nano 2.2.6    Archivo: /var/log/apache2/modsec_audit.log


--72034a78-C--
nombreusuario=%27+or+true+--&password=&login=login
--72034a78-F--
HTTP/1.1 403 Forbidden
Vary: accept-language,accept-charset
Accept-Ranges: bytes
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
Content-Language: es

--72034a78-H--
Message: Access denied with code 403 (phase 2). Pattern match "(^[\"'`\xc2\xb4\$
Action: Intercepted (phase 2)
Apache-Handler: type-map
Stopwatch: 1439848691481488 3026 (- - -)
Stopwatch2: 1439848691481488 3026; combined=972, p1=315, p2=592, p3=0, p4=0, p5$
Producer: ModSecurity for Apache/2.7.7 (http://www.modsecurity.org/); OWASP_CRS$
Server: Apache/2.4.7 (Ubuntu)
Engine-Mode: "ENABLED"
```

Figura 205. Log de mod_security ante petición denegada

El mismo procedimiento podemos hacerlo con el resto de las reglas de los otros directorios. No obstante, conviene tener muy presente que, **a mayor nº de reglas activas, mayor es la probabilidad de que nos encontremos con un problema de incompatibilidad** con nuestra web de pruebas, por lo que conviene ir activándolas poco a poco y siempre primero en modo `DetectionOnly`. En todo caso, cualquier potencial problema que tengamos aparecerá en el fichero de log del *mod*, con lo que podemos identificar el problema y modificar o comentar la regla que lo causa si no hay forma de adaptar nuestra aplicación para que deje de producirlo.

A modo de ejemplo, en nuestro sistema de pruebas sí que hay una incompatibilidad con `mod_security`, ya que este *mod* considera que un acceso vía IP a un host es un posible intento de ataque (escaneo de máquinas, ataques automáticos a rangos de IPs). En un entorno real esto no es frecuente y se accede más bien bajo un nombre de dominio. Como en nuestro entorno de pruebas sabemos que los accesos usando directamente la IP son normales por no haber incluido un DNS en nuestra infraestructura, debemos consultar el *log* y ver qué regla es la que impide dichos accesos, como se ve en la siguiente imagen:

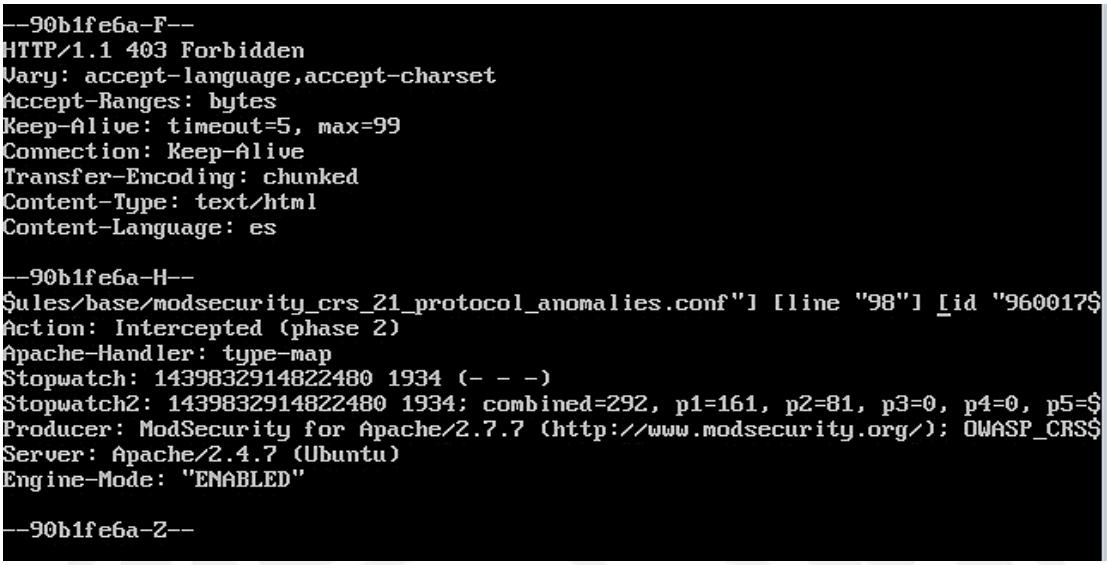


Figura 206. Regla que impide el acceso directo vía IP en el log de mod_security

Vamos pues al fichero de reglas `modsecurity_crs_21_protocol_anomalies.conf` y buscamos la regla `960017` para desactivarla y permitir el funcionamiento normal de nuestra web:

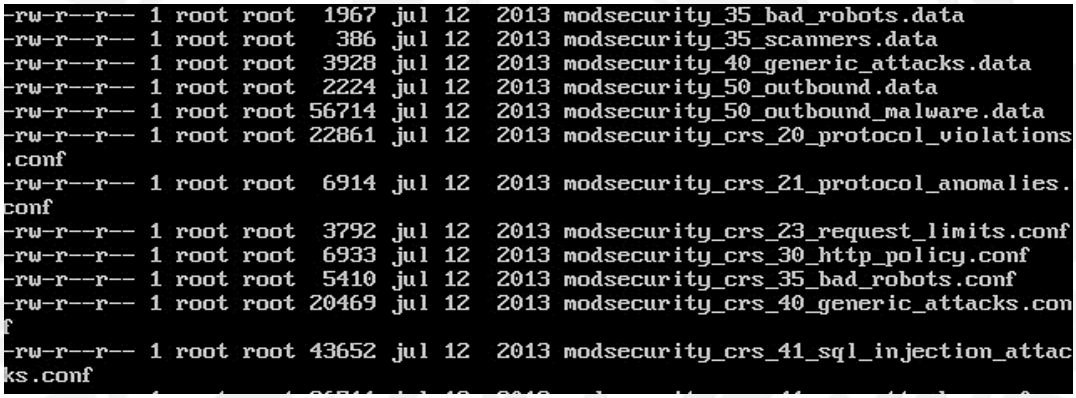


Figura 207. Localizando el fichero de reglas adecuado

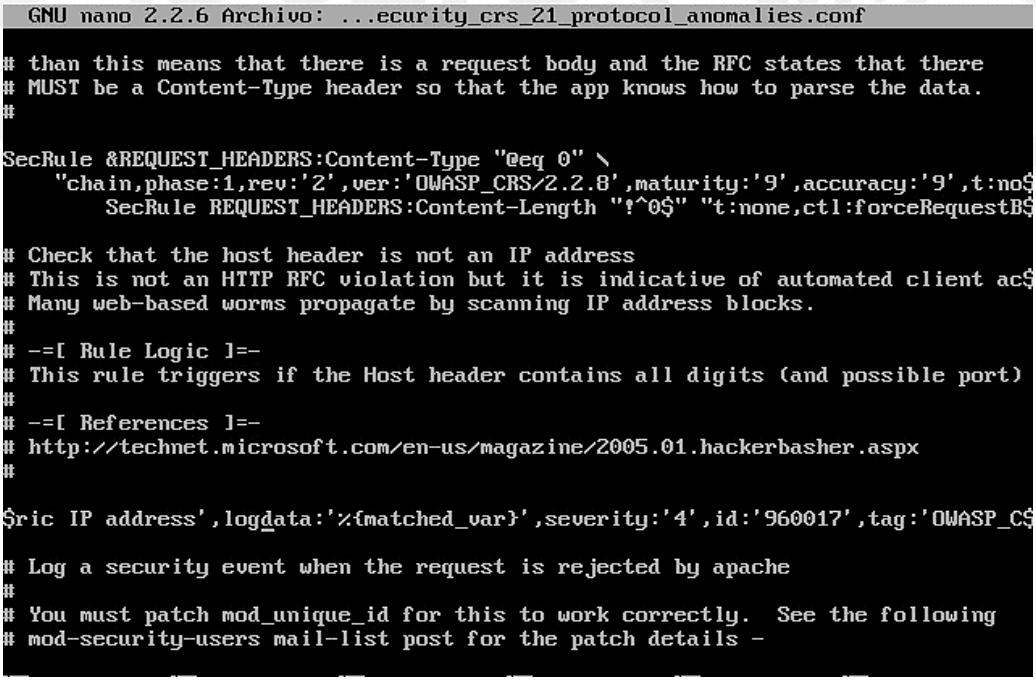


Figura 208. Detectando la regla problemática por ID

```
SecRule &REQUEST_HEADERS:Content-Type "@eq 0" \
    "chain,phase:1,rev:'2',ver:'OWASP_CRS/2.2.8',maturity:'9',accuracy:'9',t:no$
        SecRule REQUEST_HEADERS:Content-Length "!^0$" "t:none,ctl:forceRequestB$

# Check that the host header is not an IP address
# This is not an HTTP RFC violation but it is indicative of automated client ac$
# Many web-based worms propagate by scanning IP address blocks.
#
# -=[ Rule Logic ]=-
# This rule triggers if the Host header contains all digits (and possible port)
#
# -=[ References ]=-
# http://technet.microsoft.com/en-us/magazine/2005.01.hackerbasher.aspx
#

#SecRule REQUEST_HEADERS:Host "^[\d.:]+$" "phase:2,rev:'2',ver:'OWASP_CRS/2.2.8$
```

Figura 209. Comentando la regla problemática

Una vez reiniciado *Apache 2* los accesos deberían comportase exactamente como lo hacían antes de instalar el mod.

Escribiendo reglas propias para mod_security

Aunque no es nuestra intención profundizar mucho en este aspecto, sí que es interesante ver una introducción a **cómo hacer nuestras propias reglas** para aquellos casos en los que queramos controlar algún aspecto de seguridad personalizado no incluido en los conjuntos de reglas vistos. Como ejemplo, podemos probar una forma de censurar determinadas palabras introducidas en un campo de texto en una determinada página de la web. Para ello, crearemos una página PHP de ejemplo que tenga una *textarea* y que mande sus contenidos al servidor como esta:

Fichero `/var/www/html/php/comentarios.php`:

```
<html>
	<head>
		<title>Pagina de comentarios de pruebas</title>
	</head>
	<body>
	<?php
	if(isset($_POST['comentarios']))
	{
		$comentario = $_POST['comentarios'];		print "<p>Su comentario fue: $comentario</p>";
	}
	else
	{
	?>
		<form action="" method="post">
		<p>
			Haga su comentario:
		</p>
		<input type="text" name="comentarios"/><br/>
		<input type="submit"/>
		</form>
	<?php
	}
```

```
        ?>
        </body>
</html>
```

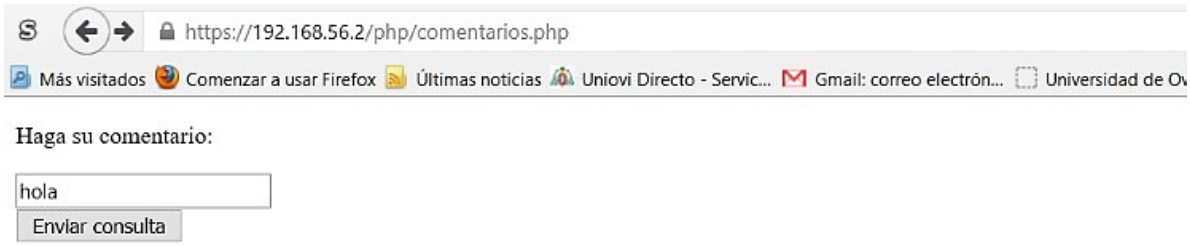


Figura 210. Página de comentarios de pruebas

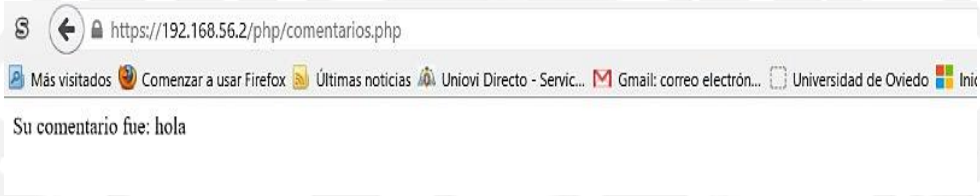


Figura 211. Respuesta de la página de comentarios

Ahora solo tenemos que crear una regla adecuada en un fichero `.conf` que sepamos que se va a cargar cuando el *mod* lo haga. Siguiendo la configuración dada anteriormente, bastaría con crearlo dentro del directorio `/etc/modsecurity/`. Llamaremos a este fichero `mis_reglas.conf`. Para crear la regla debemos tener en cuenta que la sintaxis básica es:

```
SecRule <variables> <operador> [<acciones>]
```

Las variables, operadores y acciones posibles son muchos y no entraremos en detalle de todos ellos aquí, viendo simplemente algunos para tener clara la forma de trabajar del *mod*. No obstante, se puede profundizar mucho en cómo hacer reglas propias para el *mod* en estos dos enlaces:

- **Escritura de reglas**: https://support.kemptechnologies.com/hc/en-us/articles/204301909https://support.kemptechnologies.com/hc/en-us/articles/204301909-Technical-Note-Rule-Writing-GuideTechnical-Note-Rule-Writing-Guide
- **Ejemplos de acciones comunes**: https://www.feistyduck.com/books/modsecurityhttps://www.feistyduck.com/books/modsecurity-handbook/modsecurity-rule-writing-workshop.pdfhandbook/modsecurity-rule-writing-workshop.pdf

En nuestro caso diseñaremos la regla para bloquear comentarios con palabras prohibidas en nuestro fichero `comentarios.php` usando los siguientes elementos:

- **Variables**:
  - `REQUEST_FILENAME`: El nombre del fichero pedido
  - `REQUEST_METHOD`: Método HTTP usado (GET, POST…)
  - `REQUEST_BODY`: El contenido de la petición
- **Acciones**:
  - `Id:'<identificador>'`: Identificador único de la regla

- `Chain`: Operador lógico and (esta regla y la siguiente deben cumplirse para que la petición sea denegada)
- `Deny`: Prohibir (`503 Forbidden`) la petición
- `Log`: Escribir la petición prohibida en el log
- `Msg:'<mensaje>'`: Mensaje a escribir en el servidor

- **Operadores**:
  - `@rx`: Encaje de patrones usando expresiones regulares (de la misma forma que vimos en `mod_rewrite`)
  - `?i`: Encaje de patrones no sensible a mayúsculas

Con todos elementos, componemos la siguiente cadena de reglas que cumple con lo que hemos especificado:

```
SecRule  REQUEST_FILENAME  "comentarios.php"  "id:'1337133',chain,deny,log,msg:'Detectada
palabra censurada en el comentario'"
SecRule REQUEST_METHOD "POST" chain
SecRule REQUEST_BODY "@rx (?i:(malo|basura|inadecuado|ineficiente))"
```

Una vez guardado el fichero y reiniciado *Apache 2*, si introducimos alguna de las palabras mencionadas tendríamos que ver un error de `503 Forbidden` (modo `SecRuleEngine on`) o bien una entrada en el log del módulo (modo `SecRuleEngine DetectionOnly`), señal de que nuestra cadena de reglas está funcionando correctamente. En la siguiente imagen vemos el segundo caso:

```
 GNU nano 2.2.6    Archivo: /var/log/apache2/modsec_audit.log

--88d51812-E--
 <html>
       <head>
              <title>Pagina de comentarios de pruebas</title>
       </head>
       <body>
       <p>Su comentario fue: basura</p>        </body>
</html>

--88d51812-H--
Message: Warning. Pattern match "(?i:(malo|basura|inadecuado|ineficiente))" at $
Apache-Handler: application/x-httpd-php
Stopwatch: 1439834642523833 3068 (- - -)
Stopwatch2: 1439834642523833 3068; combined=1541, p1=292, p2=1081, p3=2, p4=124$
Response-Body-Transformed: Dechunked
Producer: ModSecurity for Apache/2.7.7 (http://www.modsecurity.org/); OWASP_CRS$
Server: Apache/2.4.7 (Ubuntu)
Engine-Mode: "DETECTION_ONLY"

--88d51812-Z--
```

Figura 212. Detectando palabra prohibida en la página de comentarios

Desactivando `mod_security` en determinadas localizaciones:

Ya hemos dicho que la potencia y complejidad de `mod_security` es elevada y que es fácil **que interfiera con la funcionalidad de alguna web que tengamos instalada**. Por ello, repetimos que es bueno que antes de activar el módulo para que haga intercepciones es buena idea que permanezca un tiempo en modo

`DetectionOnly` para analizar qué clase de peticiones intercepta durante el uso normal de la web y de esta forma ver si interfiere con ella o podemos usarlo tranquilamente. De cualquier manera, si fuese necesario podemos desactivar `mod_security` completamente a nivel de host virtual, directorio y otros contenedores de reglas o bien solo desactivar determinadas reglas de este. Si bien esto puede solucionar determinados problemas, debemos tener en cuenta que el precio que estamos pagando es disminuir la seguridad de una parte de la web. Los siguientes ejemplos muestran cómo hacer esto:

- **Desactivar `mod_security` para todo un host virtual**: Esto puede hacerse incluyendo esto dentro del contenedor `<VirtualHost>` adecuado

```
<IfModule security2_module>
SecRuleEngine Off
</IfModule>
```

- **Desactivar `mod_security` para un directorio o localización**:

```
<Directory "/var/www/wp-admin">
<IfModule security2_module>  SecRuleEngine Off
</IfModule>
</Directory>
```

- **Desactivar una regla concreta por Id** en un contenedor cualquiera:

```
<LocationMatch "/wp-admin/update.php">
<IfModule security2_module>
SecRuleRemoveById 13371337
</IfModule>
</LocationMatch>
```

Nótese que en los ejemplos aparece el contenedor de directivas condicional `<IfModule>` para estar seguros de que el *mod* se encuentra activo antes de operar con él.

Enmascarando el tipo de servidor

Además de comprobar las peticiones y bloquear las que puedan suponer un problema, `mod_security` puede usarse para otras cosas, como por ejemplo **enmascarar completamente el tipo de servidor usado**. Anteriormente vimos que la manipulación de la directiva `ServerTokens` controlaba cuanta información daba el servidor sobre sí mismo. No obstante, al final siempre sabíamos que era un servidor *Apache 2*. `mod_security` nos da la oportunidad de directamente poner lo que queramos como respuesta del servidor, para que de esta manera no sea tan obvio saber cuál estamos usando.

Otra variante sería **poner la respuesta de un servidor diferente al nuestro** (*IIS*, *Nginx*) para así despistar más a los posibles atacante. Para probar esta funcionalidad primero escanearemos el servidor para ver la respuesta que da. En la siguiente imagen podemos ver como efectivamente se informa del tipo de servidor, versión y SO (configuración por defecto de *Apache 2*):

```
operario@ubuntu:/usr/share/modsecurity-crs/optional_rules$ sudo nmap -v -sV 127.
0.0.1

Starting Nmap 6.40 ( http://nmap.org ) at 2015-08-18 00:18 CEST
NSE: Loaded 23 scripts for scanning.
Initiating SYN Stealth Scan at 00:18
Scanning localhost (127.0.0.1) [1000 ports]
Discovered open port 443/tcp on 127.0.0.1
Discovered open port 80/tcp on 127.0.0.1
Completed SYN Stealth Scan at 00:18, 2.36s elapsed (1000 total ports)
Initiating Service scan at 00:18
Scanning 2 services on localhost (127.0.0.1)
Completed Service scan at 00:19, 12.05s elapsed (2 services on 1 host)
NSE: Script scanning 127.0.0.1.
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0000040s latency).
Not shown: 998 closed ports
PORT    STATE SERVICE  VERSION
80/tcp  open  http     Apache httpd 2.4.7 ((Ubuntu))
443/tcp open  ssl/http Apache httpd 2.4.7 ((Ubuntu))

Read data files from: /usr/bin/../share/nmap
Service detection performed. Please report any incorrect results at http://nmap.
org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 14.56 seconds
           Raw packets sent: 1061 (46.684KB) | Rcvd: 2124 (89.212KB)
operario@ubuntu:/usr/share/modsecurity-crs/optional_rules$
```

Figura 213. Escaneo del servidor con nmap con la configuración por defecto

Para solucionar el problema basta con editar el fichero `security.conf` de `conf-enabled` ya visto anteriormente e incluir las dos líneas siguientes:

```
# ServerTokens
# This directive configures what you return as the Server HTTP response
# Header. The default is 'Full' which sends information about the OS-Type
# and compiled in modules.
# Set to one of:  Full | OS | Minimal | Minor | Major | Prod
# where Full conveys the most information, and Prod the least.
#ServerTokens Minimal
#ServerTokens OS
ServerTokens Full
SecServerSignature Soy_un_IIS_de_verdad_de_la_buena
```

Figura 214. Enmascarando el servidor con mod_security y SecServerSignature

Al hacer esto `nmap` es incapaz de identificar el servidor que estamos usando:

```
Starting Nmap 6.40 ( http://nmap.org ) at 2015-08-18 00:56 CEST Nmap scan report for
localhost (127.0.0.1)
Host is up (0.000026s latency).
Not shown: 998 closed ports PORT    STATE SERVICE    VERSION
80/tcp  open  http?
443/tcp open  ssl/https?
2 services unrecognized despite returning data. If you know the service/version, please
submit the following fingerprints at http://www.insecure.org/cgi-bin/servicefpsubmit.cgi :
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port80-TCP:V=6.40%I=7%D=8/18%Time=55D26703%P=x86_64-pc-linux-gnu%r(GetR
SF:equest,503,"HTTP/1\.1\x20403\x20Forbidden\r\nDate:\x20Mon,\x2017\x20Aug
SF:\x202015\x2022:58:11\x20GMT\r\nServer:\x20Soy_un_IIS_de_verdad_de_la_bu
```

```
SF:ena\r\nVary:\x20accept-language,accept-charset\r\nAccept-Ranges:\x20byt
SF:es\r\nConnection:\x20close\r\nContent-Type:\x20text/html;\x20charset=ut
SF:f-8\r\nContent-Language:\x20en\r\nExpires:\x20Mon,\x2017\x20Aug\x202015
SF:\x2022:58:11\x20GMT\r\n\r\n<\?xml\x20version=\"1\.0\"\x20encoding=\"UTF
SF:-8\"\?>\n<!DOCTYPE\x20html\x20PUBLIC\x20\"-//W3C//DTD\x20XHTML\x201\.0\
SF:x20Strict//EN\"\n\x20\x20\"http://www\.w3\.org/TR/xhtml1/DTD/xhtml1-str
SF:ict\.dtd\">\n<html\x20xmlns=\"http://www\.w3\.org/1999/xhtml\"\x20lang=
SF:\"en\"\x20xml:lang=\"en\">\n<head>\n<title>Access\x20forbidden!</title>
SF:\n<link\x20rev=\"made\"\x20href=\"mailto:webmaster@localhost\"\x20/>\n<
SF:style\x20type=\"text/css\"><!--/\*--><!\[CDATA\[/\*><!--\*/\x20\n\x20\x
SF:20\x20\x20body\x20{\x20color:\x20#000000;\x20background-color:\x20#FFFF
SF:FF;\x20}\n\x20\x20\x20\x20a:link\x20{\x20color:\x20#0000CC;\x20}\n\x20\
SF:x20\x20\x20p,\x20address\x20{margin-left:\x203em;}\n\x20\x20\x20\x20spa
SF:n\x20{font-size:\x20smaller;}\n/\*\]\]>\*/--></style>\n</head>\n\n<body
SF:>\n<h1>Access\x20forbidden!</h1>\n<p>\n\n\n\x20\x20\n\n\x20\x20\x20\x20
SF:You\x20don't\x20have\x20permi")%r(HTTPOptions,503,"HTTP/1\.1\x20403\x20
SF:Forbidden\r\nDate:\x20Mon,\x2017\x20Aug\x202015\x2022:58:11\x20GMT\r\nS
SF:erver:\x20Soy_un_IIS_de_verdad_de_la_buena\r\nVary:\x20accept-language,
SF:accept-charset\r\nAccept-Ranges:\x20bytes\r\nConnection:\x20close\r\nCo
SF:ntent-Type:\x20text/html;\x20charset=utf-8\r\nContent-Language:\x20en\r
SF:\nExpires:\x20Mon,\x2017\x20Aug\x202015\x2022:58:11\x20GMT\r\n\r\n<\?xm
SF:l\x20version=\"1\.0\"\x20encoding=\"UTF-8\"\?>\n<!DOCTYPE\x20html\x20PU  SF:BLIC\x20\"-
//W3C//DTD\x20XHTML\x201\.0\x20Strict//EN\"\n\x20\x20\"http:
SF://www\.w3\.org/TR/xhtml1/DTD/xhtml1-strict\.dtd\">\n<html\x20xmlns=\"ht
SF:tp://www\.w3\.org/1999/xhtml\"\x20lang=\"en\"\x20xml:lang=\"en\">\n<hea
SF:d>\n<title>Access\x20forbidden!</title>\n<link\x20rev=\"made\"\x20href=
SF:\"mailto:webmaster@localhost\"\x20/>\n<style\x20type=\"text/css\"><!--/
SF:\*--><!\[CDATA\[/\*><!--\*/\x20\n\x20\x20\x20\x20body\x20{\x20color:\x2
SF:0#000000;\x20background-color:\x20#FFFFFF;\x20}\n\x20\x20\x20\x20a:link
SF:\x20{\x20color:\x20#0000CC;\x20}\n\x20\x20\x20\x20p,\x20address\x20{mar
SF:gin-left:\x203em;}\n\x20\x20\x20\x20span\x20{font-size:\x20smaller;}\n/
SF:\*\]\]>\*/--></style>\n</head>\n\n<body>\n<h1>Access\x20forbidden!</h1>
SF:\n<p>\n\n\n\x20\x20\n\n\x20\x20\x20\x20You\x20don't\x20have\x20permi");
==============NEXT SERVICE FINGERPRINT (SUBMIT INDIVIDUALLY)==============
SF-Port443-TCP:V=6.40%T=SSL%I=7%D=8/18%Time=55D26709%P=x86_64-pc-linux-gnu
SF:%r(GetRequest,503,"HTTP/1\.1\x20403\x20Forbidden\r\nDate:\x20Mon,\x2017
SF:\x20Aug\x202015\x2022:58:17\x20GMT\r\nServer:\x20Soy_un_IIS_de_verdad_d
SF:e_la_buena\r\nVary:\x20accept-language,accept-charset\r\nAccept-Ranges:
SF:\x20bytes\r\nConnection:\x20close\r\nContent-Type:\x20text/html;\x20cha
SF:rset=utf-8\r\nContent-Language:\x20en\r\nExpires:\x20Mon,\x2017\x20Aug\
SF:x202015\x2022:58:17\x20GMT\r\n\r\n<\?xml\x20version=\"1\.0\"\x20encodin
SF:g=\"UTF-8\"\?>\n<!DOCTYPE\x20html\x20PUBLIC\x20\"-//W3C//DTD\x20XHTML\x
SF:201\.0\x20Strict//EN\"\n\x20\x20\"http://www\.w3\.org/TR/xhtml1/DTD/xht
SF:ml1-strict\.dtd\">\n<html\x20xmlns=\"http://www\.w3\.org/1999/xhtml\"\x
SF:20lang=\"en\"\x20xml:lang=\"en\">\n<head>\n<title>Access\x20forbidden!<
SF:/title>\n<link\x20rev=\"made\"\x20href=\"mailto:webmaster@localhost\"\x
SF:20/>\n<style\x20type=\"text/css\"><!--/\*--><!\[CDATA\[/\*><!--\*/\x20\
SF:n\x20\x20\x20\x20body\x20{\x20color:\x20#000000;\x20background-color:\x
SF:20#FFFFFF;\x20}\n\x20\x20\x20\x20a:link\x20{\x20color:\x20#0000CC;\x20}
SF:\n\x20\x20\x20\x20p,\x20address\x20{margin-left:\x203em;}\n\x20\x20\x20
SF:\x20span\x20{font-size:\x20smaller;}\n/\*\]\]>\*/--></style>\n</head>\n
SF:\n<body>\n<h1>Access\x20forbidden!</h1>\n<p>\n\n\n\x20\x20\n\n\x20\x20\
SF:x20\x20You\x20don't\x20have\x20permi")%r(HTTPOptions,503,"HTTP/1\.1\x20
SF:403\x20Forbidden\r\nDate:\x20Mon,\x2017\x20Aug\x202015\x2022:58:22\x20G
SF:MT\r\nServer:\x20Soy_un_IIS_de_verdad_de_la_buena\r\nVary:\x20accept-la
SF:nguage,accept-charset\r\nAccept-Ranges:\x20bytes\r\nConnection:\x20clos
SF:e\r\nContent-Type:\x20text/html;\x20charset=utf-8\r\nContent-Language:\
```

```
SF:x20en\r\nExpires:\x20Mon,\x2017\x20Aug\x202015\x2022:58:22\x20GMT\r\n\r
SF:\n<\?xml\x20version=\"1\.0\"\x20encoding=\"UTF-8\"\?>\n<!DOCTYPE\x20htm
SF:l\x20PUBLIC\x20\"-//W3C//DTD\x20XHTML\x201\.0\x20Strict//EN\"\n\x20\x20
SF:\"http://www\.w3\.org/TR/xhtml1/DTD/xhtml1-strict\.dtd\">\n<html\x20xml
SF:ns=\"http://www\.w3\.org/1999/xhtml\"\x20lang=\"en\"\x20xml:lang=\"en\"
SF:>\n<head>\n<title>Access\x20forbidden!</title>\n<link\x20rev=\"made\"\x
SF:20href=\"mailto:webmaster@localhost\"\x20/>\n<style\x20type=\"text/css\
SF:"><!--/\*--><!\[CDATA\[/\*><!--\*/\x20\n\x20\x20\x20\x20body\x20{\x20co
SF:lor:\x20#000000;\x20background-color:\x20#FFFFFF;\x20}\n\x20\x20\x20\x2
SF:0a:link\x20{\x20color:\x20#0000CC;\x20}\n\x20\x20\x20\x20p,\x20address\
SF:x20{margin-left:\x203em;}\n\x20\x20\x20\x20span\x20{font-size:\x20small
SF:er;}\n/\*\]\]>\*/--></style>\n</head>\n\n<body>\n<h1>Access\x20forbidde
SF:n!</h1>\n<p>\n\n\n\x20\x20\n\n\x20\x20\x20\x20You\x20don't\x20have\x20p SF:ermi");

Service detection performed. Please report any incorrect results at
http://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 207.29 seconds
```

Mod_security 3 y contenedores

Existe una tercera versión de *ModSecurity*, alojada en https://github.com/SpiderLabs/ModSecurity. *Libmodsecurity* es un componente del proyecto *ModSecurity v3*. El código base de la biblioteca es un interfaz para los **conectores** *ModSecurity* instalados en el servidor web, que absorben el tráfico web y aplican el procesamiento tradicional de *ModSecurity*. En general, proporciona la capacidad de cargar/interpretar reglas escritas en el formato *ModSecurity* y aplicarlas al contenido HTTP proporcionado por la aplicación a través de *Connectors*.

ModSecurity v2.x para *Apache 2* es un módulo de este y aún es mantenido. La versión 3 no es un módulo de Apache 2, proporciona una arquitectura nueva que **permite integrarlo con otros servidores web** (*Nginx*, *IIS*...) y aplicaciones que necesiten sus servicios de filtrado. Tiene las siguientes ventajas:

- Se han eliminado todas las dependencias de Apache 2
- Mayor rendimiento
- Nuevas características
- Nueva arquitectura

Para instalar el módulo es necesario compilarlo contra las fuentes del servidor correspondiente e instalar un conector. En estos enlaces se proporcionan instrucciones para ello:

- Instalar en ***Apache 2:*** https://kifarunix.com/install-modsecurity-with-apache-on-debian-12/
- Instalar en ***Nginx***: https://www.tecmint.com/install-modsecurity-nginx-debian-ubuntu/

Dado que este es un proceso laborioso, quizá sea más oportuno usar contenedores con él instalado, que solo tendremos que configurar para adaptarlos a nuestras necesidades:

- Existen **dos versiones** de *ModSecurity* 3 empaquetadas en contenedores *Docker, una para Nginx* y otra para *Apache 2*.
- La que se mantiene en la actualidad tiene **las reglas CRS integradas**: https://hub.docker.com/r/owasp/modsecurity-crs/ (*GitHub* correspondiente: https://github.com/coreruleset/modsecurity-crs-docker)

Medidas anti-DoS/DDoS: *mod_evasive*

Los **ataques de denegación de servicio** son uno de los mayores males a los que se enfrentan los servidores web hoy en día. No solo son malos porque **evitan poder dar servicio a los clientes legítimos** (o lo ralentizan hasta niveles inaceptables) sino que lo son porque su procesamiento consume recursos, y eso **puede tener efectos económicos**. Y los efectos económicos son aún más importantes en el caso de que **nuestro hosting esté en nube** y que nos cobre por uso de ancho de banda (un escenario muy común hoy en día).

Sí que es cierto que un buen *hosting* que se precie debería tener medidas implementadas contra este tipo de ataques, de manera que los evite o mitigue hasta el punto de no ser un problema grave. **Detectores de intrusiones** (*IDS*) o **herramientas de prevención de intrusiones** (IPS) como *Snort* (https://www.snort.org/), o herramientas de vigilancia de incidentes en tiempo real centralizadas, como *OSSIM AlienVault* (https://www.alienvault.com/products/ossim) u otro software SIEM equivalente, pueden evitar en parte la incidencia de estos ataques parando al origen antes de que sea un problema. También son servicios que proporcionan CDNs populares como ***Cloudflare***, incluso con planes gratuitos. Pero aun así, tenemos que pensar realmente en intentar implementar medidas contra los mismos por nuestra cuenta.

¿Por qué? Pues primero porque un buen administrador no debería tener fe ciega en una parte de la infraestructura que no controla directamente: no tiene por qué tener instalados buenos productos, todos los necesarios, actualizados, correctamente configurados, etc. También debemos pensar que **una capa más de seguridad** contra algo nunca hace daño, sobre todo si falla o consiguen traspasar la existente, y de esta manera nos la llevaremos con nosotros si nos movemos de *hosting* (***Defense in Depth***).

Muy frecuentemente vemos como grandes compañías que han invertido enormes cantidad de dinero en infraestructuras son aun así víctimas de ataques de denegación de servicio: ofrecer un poco más de resistencia no es en absoluto malo, pero debemos tener claro que si un atacante pone el empeño y los recursos en denegar el servicio de nuestra web muy probablemente lo conseguirá en mayor o menor medida.

Y es que una denegación de servicio real hace mucho que **no** se limita a "máquina aislada que emite un enorme nº de peticiones de diversos tipos que saturan al servidor de varias formas" (nº de conexiones abiertas máximo alcanzado, capacidad máxima de cómputo superada, capacidad de almacenamiento reducida a 0, provocar que algún parámetro del sistema supere el valor máximo permitido a base de peticiones,…), sino que estos ataques han evolucionado a una forma mucho más potente y destructiva: los *DDoS*. En un esquema clásico de *DoS,* un origen, compuesto por uno o unos pocos equipos "atacantes", envía un nº abismal de peticiones (o bien peticiones "pesadas" de procesar o corruptas) a un mismo objetivo, de manera que un *software* de vigilancia que detectase una subida súbita del tráfico "sospechoso" desde ese origen podría "cortar" la comunicación con los mismos. Un *DDoS* en cambio vence a esta forma de protección enviando ataques de forma programada y coordinada (única, por oleadas,…) desde un nº enorme de equipos situados en diferentes localizaciones del mundo potencialmente muy alejadas.

¿Y en qué cambia esto las cosas? El servidor podría bloquear a esos clientes, **pero el nº de posibles clientes puede ser altísimo** y comprobar si cada petición está bloqueada o no según su origen puede usar muchos recursos, perdiendo calidad de servicio y consiguiendo así parcialmente su objetivo. No solo eso, sino que el controlador del ataque **podría ir variando las máquinas que envían peticiones periódicamente** de manera dinámica, por lo que los ataques no se recibirían siempre desde los mismos clientes y por tanto la lista de clientes prohibidos no sería efectiva. Lo que es aún peor es que frecuentemente muchas de las máquinas que envían peticiones en un ataque *DDoS* son probablemente ordenadores de particulares que

han sido troyanizados por alguna clase de *malware* y que pueden estar haciendo esta clase de operaciones incluso sin saberlo. No obstante, también ha habido casos de ataques realizados por un *software* legítimo que sirva para lanzar tareas coordinadas, porque el dueño del equipo ha cedido voluntariamente el mismo para ello. A este conjunto de ordenadores que ejecutan un *DDoS* se les suele denominar *botnet*. La *botnet* también impide conocer la identidad real (IP) de quien realmente da orden de comenzar el ataque:

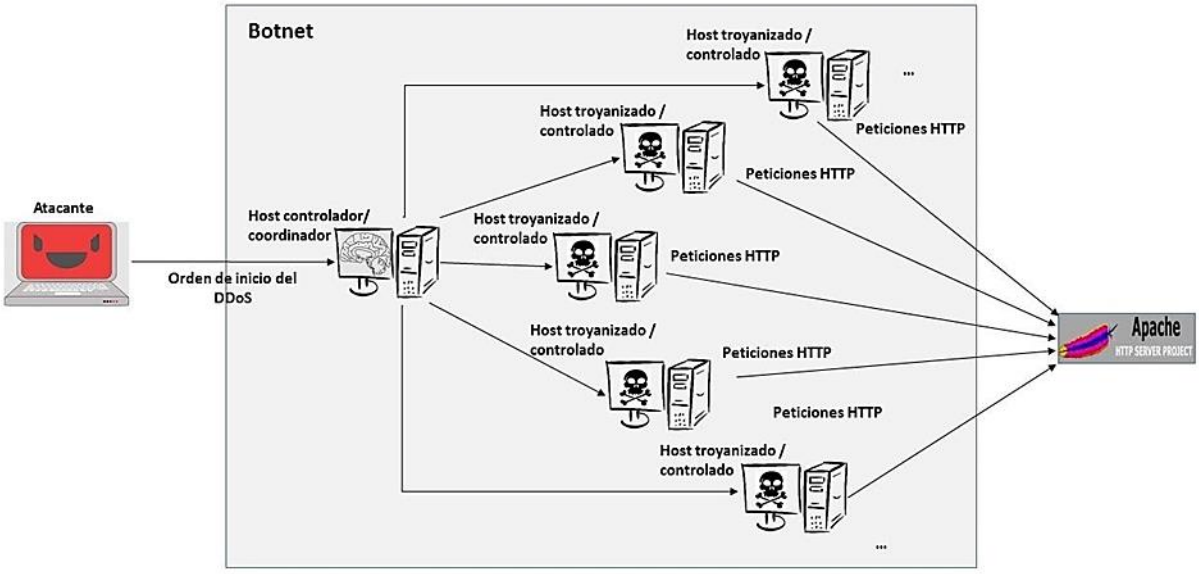


Figura 215. Esquema clásico de DDoS lanzado con una botnet

Dada la gravedad del problema lo único que podemos hacer realmente es, como decíamos, poner una capa de seguridad por nuestra cuenta y buscar un buen hosting que controle estos aspectos. Esta capa de seguridad podemos encontrarla en el módulo `mod_evasive`. Este módulo configura *Apache 2* para que sea algo más resistente a ataques *DoS* o *DDoS*, y proporciona medios de detección de ataques al poder notificar incidencias a cortafuegos, personas o *routers*. El `mod_security` de la sección anterior también implementa protección contra esta clase de ataques, pero no está de más tener dos capas de seguridad en funcionamiento contra este problema.

Para detectar peticiones potencialmente dañinas lo que hace el módulo es crear una tabla *hash* dinámica de *IPs* y *URIs*, denegando una petición (y los subsiguientes accesos desde la IP que la produce) si se cumple alguna de estas condiciones:

- Que se solicite la misma página más de un nº determinado de **veces por segundo**
- Que se hagan más de un nº concreto de **peticiones concurrentes por segundo**

Una IP bloqueada solo se mantiene así un determinado tiempo por defecto. Estas técnicas no son en absoluto definitivas, pero si pueden parar ataques poco elaborados o hechos por atacantes inexpertos o con pocos recursos. Obviamente, contra una *botnet* lo suficientemente grande y configurada adecuadamente este módulo **no puede hacer gran cosa**: el ancho de banda, la memoria y la capacidad de la CPU del servidor se acabarán agotando. No obstante, si se combina con `mod_security` (y un buen servicio de *hosting* o CDN que controle estos aspectos, como ya hemos dicho) el resultado será mucho mejor.

Puesta en marcha y configuración

La instalación y activación del módulo es muy similar a la de otros módulos que ya hemos visto:

```
apt-get install libapache2-mod-evasive
a2enmod evasive
```

```
operario@ubuntu:~$ sudo apt-get install libapache2-mod-evasive
[sudo] password for operario:
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias
Leyendo la información de estado... Hecho
Se instalarán los siguientes paquetes NUEVOS:
  libapache2-mod-evasive
0 actualizados, 1 se instalarán, 0 para eliminar y 3 no actualizados.
Necesito descargar 15,7 kB de archivos.
Se utilizarán 82,9 kB de espacio de disco adicional después de esta operación.
Des:1 http://es.archive.ubuntu.com/ubuntu/ trusty/universe libapache2-mod-evasiv
e amd64 1.10.1-2 [15,7 kB]
Descargados 15,7 kB en 0seg. (86,2 kB/s)
Seleccionando el paquete libapache2-mod-evasive previamente no seleccionado.
(Leyendo la base de datos ... 59457 ficheros o directorios instalados actualment
e.)
Preparing to unpack .../libapache2-mod-evasive_1.10.1-2_amd64.deb ...
Unpacking libapache2-mod-evasive (1.10.1-2) ...
Configurando libapache2-mod-evasive (1.10.1-2) ...
apache2_invoke: Enable module evasive
 * Restarting web server apache2
```

Figura 216. Instalando mod-evasive

Hecho esto, podemos configurar el módulo. Para ello, en el fichero de configuración asociado al módulo `evasive.conf` quitamos los comentarios a la configuración que viene por defecto de manera que se apliquen las siguientes directivas. El resto pueden quedar comentadas en nuestro sistema de pruebas, salvo quizá `DOSLogDir`, si queremos que el módulo guarde un fichero de log independiente:

```
<IfModule mod_evasive20.c>
    DOSHashTableSize 3097
    DOSPageCount 30
    DOSSiteCount 40
    DOSPageInterval 2
    DOSSiteInterval 1
    DOSBlockingPeriod 120
    #DOSEmailNotify una@cuentadecorreo.com
</IfModule>
```

```
 GNU nano 2.2.6          Archivo: evasive.conf                Modificado

<IfModule mod_evasive20.c>
    DOSHashTableSize    3097
    DOSPageCount        30
    DOSSiteCount        40
    DOSPageInterval     2
    DOSSiteInterval     1
    DOSBlockingPeriod   120

    #DOSEmailNotify      you@yourdomain.com
    #DOSSystemCommand    "su - someuser -c '/sbin/... %s ...'"
    #DOSLogDir           "/var/log/mod_evasive"
</IfModule>
```

Figura 217. Configuración de mod_evasive

El significado de estas es el siguiente, aunque los valores concretos deberán ser adaptados a cada caso/instalación en función del tipo de servicio que provean:

- `DOSHashTableSize 3097`: Incrementar el tamaño asignado en esta directiva mejorará el rendimiento del módulo a cambio de usar más memoria. No obstante, si disponemos de memoria suficiente, el valor por defecto debería ser mayor que el especificado por defecto.
- `DOSPageCount 30`: Añade la IP a la lista de bloqueadas si se pide la misma página más de 30 veces consecutivas.
- `DOSSiteCount 40`: Añade la IP a la lista de bloqueadas si se pide la misma URI más de 40 veces por segundo.
- `DOSBlockingPeriod 120`: Tiempo (en segundos) que una IP se bloquea si ha sido añadida a la lista de bloqueadas por algún motivo.
- `DOSPageInterval 2`: Controla el tiempo (en segundos) durante el que se cuenta el nº de peticiones consecutivas especificado en `DOSPageCount`. En el ejemplo de arriba, si alguien hace más de 30 peticiones en 2 segundos, se bloquea su IP.
- `DOSSiteInterval 1`: La Directiva `DOSSiteCount` controlaba el nº de peticiones máximas por segundo. Esta directiva cambia precisamente este aspecto: un valor de N indica que una IP se bloquea si se hacen `<valor de DOSSiteCount>` peticiones a la misma URI en N segundos.

El bloqueo de una IP por haber cumplido con alguna de estas condiciones implica que al cliente se le devolverá un error `403 Forbidden`. Adicionalmente, se suportan las siguientes características:

- **Listas de IPs permitidas**: IPs desde las cuales se acepta cualquier nº de peticiones a una misma página, útil para redes locales o equipos conocidos desde los que sepamos que va a generarse un gran volumen de tráfico legítimo. Esto se controla con la directiva `DOSWhitelist`.
- **Notificación por email de incidencias**: Se puede indicar un email al que enviar un mensaje si se detecta un ataque, para que así el administrador pueda reaccionar mientras se está produciendo. Obviamente, el servidor debe tener la capacidad de enviar mensajes de email, es decir, tiene que existir una configuración que indique el proveedor a través del cual se enviarán los mensajes por defecto. Esto se controla con la directiva `DOSEmailNotify`
- **Comandos externos**: La directiva `DOSSystemCommand` puede usarse para enviar comandos al sistema operativo. Ejemplos de comandos útiles podrían ser bloquear IPs detectadas automáticamente en el *firewall* el sistema (`iptables`).

- **Log**: El módulo posee su propio log a través de la directiva `DOSLogDir`. En esta directiva tenemos que indicar una ruta existente en la que el *mod* guardará los log. No obstante debe tenerse en cuenta que hacer log de un ataque *DDoS* puede someter al sistema a una carga de trabajo considerable (se guardará mucha información en el disco duro) y también podríamos quedarnos sin espacio si no lo configuramos adecuadamente.

Se pueden encontrar instrucciones sobre el resto de las opciones y cómo configurar en detalle las últimas que hemos visto en el fichero `/usr/share/doc/libapache2-mod-evasive/README.gz`, que se incluye en la instalación del *mod*.

Detección y bloqueo de intrusiones

Aunque `mod_evasive` es capaz de bloquear peticiones repetidas sobre un mismo recurso potencialmente peligrosas (*DoS*) de forma satisfactoria, veremos ahora otra aproximación para parar esta clase de ataques más flexible: bloquear temporalmente aquellos clientes que emitan una serie de peticiones de este tipo **analizando también su contenido**. Esto puede lograrse mediante el módulo `Fail2Ban`. `Fail2Ban` es un sistema de detección y prevención de intrusiones escrito en *Python*. Usa los datos de los logs de *Apache 2* para analizar el contenido de las peticiones, determinar si son accesos potencialmente maliciosos e interactúa con el *firewall* `Iptables` **para bloquear intentos de ataque de manera dinámica**: si un cliente se identifica, según los parámetros de configuración, como atacante, será bloqueado un tiempo determinado automáticamente.

Instalación y configuración

Para instalar `fail2ban` tenemos que hacer el típico procedimiento que hemos hecho con otros módulos: `sudo apt-get install fail2ban`. Una vez instalado, tenemos que configurarlo, pero copiando primero el fichero `jail.conf` que está en `/etc/fail2ban` a `jail.local` antes de hacer ningún cambio. Este fichero `.local` será el que cambiemos, dejando el original intacto por si acaso.

Una vez hecho esto, estableceremos los primeros parámetros de configuración. Estos son las IPs a ignorar, el tiempo de bloqueo (en segundos) y nº máximo de intentos/segundos a partir del cual el host es bloqueado: Estos tres parámetros identifican IPs de hosts que nunca se bloquearan (`ignoreip`, se pueden especificar varias separadas por un espacio en blanco), el tiempo de bloqueo de los clientes (`bantime`) y el nº máximo de incidentes por cada unidad de tiempo (`maxretry`/`findtime`) a partir del cual se bloqueará el cliente. Los valores por defecto (que dejaremos tal cual están) aparecen en la siguiente imagen:

```
 GNU nano 2.2.6          Archivo: jail.local

# in each jail afterwards.

[DEFAULT]

# "ignoreip" can be an IP address, a CIDR mask or a DNS host. Fail2ban will not
# ban a host which matches an address in this list. Several addresses can be
# defined using space separator.
ignoreip = 127.0.0.1/8

# "bantime" is the number of seconds that a host is banned.
bantime  = 600

# A host is banned if it has generated "maxretry" during the last "findtime"
# seconds.
findtime = 600
maxretry = 3
```

Figura 218. Valores por defecto de parámetros de fail2ban

Otra opción disponible es notificar vía email a un administrador responsable en caso de que haya alguna incidencia, al igual que lo que permitía `mod_evasive`. Al igual que con este módulo, para que esto sea posible necesitamos tener instalado `sendmail` o equivalente para que el servidor sea capaz de enviar

emails. Si nos interesase hacer esto por tener la infraestructura adecuada para ello, tendríamos que indicar una dirección de email con la directiva siguiente:

```
destemail = un_email@dominio.com
```

Y también cambiar la línea `action = %(action_)s` por `action = %(action_mw)s`. Esta última línea es la que indica que acción tomará el módulo cuando bloquee a un cliente. En la siguiente imagen se ven las tres que trae predefinidas. Nosotros dejaremos la existente por defecto, que básicamente bloquea al cliente.

```
  GNU nano 2.2.6           Archivo: jail.local

protocol = tcp

# Specify chain where jumps would need to be added in iptables-* actions
chain = INPUT

#
# Action shortcuts. To be used to define action parameter

# The simplest action to take: ban only
action_ = %(banaction)s[name=%(__name__)s, port="%(port)s", protocol="%(protoco$

# ban & send an e-mail with whois report to the destemail.
action_mw = %(banaction)s[name=%(__name__)s, port="%(port)s", protocol="%(proto$
            %(mta)s-whois[name=%(__name__)s, dest="%(destemail)s", protocol="$

# ban & send an e-mail with whois report and relevant log lines
# to the destemail.
action_mwl = %(banaction)s[name=%(__name__)s, port="%(port)s", protocol="%(prot$
             %(mta)s-whois-lines[name=%(__name__)s, dest="%(destemail)s", log$

# Choose default action.  To change, just override value of 'action' with the
# interpolation to the chosen action shortcut (e.g.  action_mw, action_mwl, etc$
# globally (section [DEFAULT]) or per specific section
action = %(action_)s
```

Figura 219. Acciones a tomar por fail2ban

Hecha la configuración inicial, podemos establecer ahora las **configuraciones de los posibles bloqueos** que puede llevar a cabo. Cada configuración tiene su propia sección que empieza con el siguiente formato `[<nombre de sección>]`. Cada una de estas secciones se denomina ***jail*** y para que estén activas debemos asegurarnos de que cada sección tiene la línea `enabled=true`. Si examinamos el resto del fichero vemos que existen multitud de *jails,* incluso para servicios que no son *Apache 2*, dado que el módulo puede trabajar con otros programas como SSH o similares. Nosotros activaremos sólo las *jail* relacionadas con *Apache 2*, que son las que nos interesan:

```
[apache]

enabled  = true
port     = http,https
filter   = apache-auth
logpath  = /var/log/apache*/*error.log
maxretry = 6
```

Figura 220. Jails de Apache 2 (I)

```
[apache-noscript]

enabled  = true
port     = http,https
filter   = apache-noscript
logpath  = /var/log/apache*/*error.log
maxretry = 6

[apache-overflows]

enabled  = true
port     = http,https
filter   = apache-overflows
logpath  = /var/log/apache*/*error.log
maxretry = 2
```

Figura 221. Jails de Apache 2 (II)

En general existen *jails* **para varios programas conocidos**, como el servidor *Nginx*, *asterisk*, etc. Solo debemos activar aquellas *jails* de productos que estén en el servidor para evitar posibles problemas. Existen también *jails* para PHP, que podremos considerar si nuestro servidor tiene instalada esa tecnología. En cualquier caso, una vez activadas las *jails* que queramos, tenemos que reiniciar `fail2ban` de la siguiente forma: `sudo /etc/init.d/fail2ban restart`.

En las imágenes vemos que cada *jail* tiene una línea llamada `filter`. Esta línea indica el nombre del filtro aplicado en cada *jail*. Los filtros son ficheros que contienen expresiones regulares que identifican posibles peticiones peligrosas, y que por tanto se usan para discernir cuando un cliente debe ser bloqueado.

Podemos ver todos los disponibles en la ruta mostrada en la siguiente imagen, teniendo en cuenta la modificación de estos filtros **exige un conocimiento en profundidad de las expresiones regulares** y de cómo se escribe el log de las posibles peticiones maliciosas que puedan afectarnos. En general es mejor confiar en el trabajo de los desarrolladores del módulo en cuanto a la capacidad de estos filtros para hacer su trabajo, ya que es extremadamente difícil que consigamos mejorar ese trabajo sin invertir una cantidad de tiempo considerable. No podemos descartar no obstante hacer nuestras propias *jails* con filtros personalizados. Como ayuda para esto podemos consultar el siguiente enlace: http://www.fail2ban.org/wiki/index.php/MANUAL_0_8

```
operario@ubuntu:/etc/fail2ban/filter.d$ l
3proxy.conf             exim-common.conf      qmail.conf
apache-auth.conf        exim.conf             recidive.conf
apache-badbots.conf     exim-spam.conf        roundcube-auth.conf
apache-common.conf      gssftpd.conf          selinux-common.conf
apache-nohome.conf      lighttpd-auth.conf    selinux-ssh.conf
apache-noscript.conf    mysqld-auth.conf      sieve.conf
apache-overflows.conf   named-refused.conf    sogo-auth.conf
assp.conf               nginx-http-auth.conf  sshd.conf
asterisk.conf           pam-generic.conf      sshd-ddos.conf
common.conf             perdition.conf        suhosin.conf
courierlogin.conf       php-url-fopen.conf    uwimap-auth.conf
couriersmtp.conf        postfix.conf          vsftpd.conf
cyrus-imap.conf         postfix-sasl.conf     webmin-auth.conf
dovecot.conf            proftpd.conf          wuftpd.conf
dropbear.conf           pure-ftpd.conf        xinetd-fail.conf
operario@ubuntu:/etc/fail2ban/filter.d$
```

Figura 222. Filtros disponibles en fail2ban

Hecho esto, podemos comprobar si `fail2Ban` está activo ejecutando `sudo iptables -L`. La salida debería ser como la siguiente, demostrando que el programa está monitorizando constantemente las peticiones y preparado para actuar sobre el *firewall* si lo considera necesario:

```
Chain INPUT (policy ACCEPT)
target     prot opt source               destination
fail2ban-Apache 2-overflows  tcp  --  anywhere             anywhere             multiport
dports http,https
fail2ban-Apache 2-noscript  tcp  --  anywhere             anywhere             multiport
dports http,https
fail2ban-Apache 2  tcp  --  anywhere              anywhere              multiport dports
http,https
fail2ban-ssh  tcp  --  anywhere             anywhere             multiport dports ssh

Chain FORWARD (policy ACCEPT)
target     prot opt source               destination
Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination
Chain fail2ban-Apache 2 (1 references)
target     prot opt source               destination         RETURN     all  --  anywhere
anywhere
Chain fail2ban-Apache 2-noscript (1 references)
target     prot opt source               destination         RETURN     all  --  anywhere
anywhere
Chain fail2ban-Apache 2-overflows (1 references)
target     prot opt source               destination         RETURN     all  --  anywhere
anywhere
Chain fail2ban-ssh (1 references)
target     prot opt source               destination
RETURN     all  --  anywhere             anywhere
```

Jails adicionales

Aunque `fail2Ban` viene de serie con un buen nº de *jails*, muy pocas están activas inicialmente y de las correspondientes a *Apache 2* algunas incluso no están mencionadas en el fichero `jail.local`. Concretamente, de este segundo caso podemos destacar dos que deberíamos considerar activar:

- `apache-auth`: Identifica intentos de autenticación fallidos (para parar posibles intentos de averiguación de claves por fuerza bruta).
- `apache-nohome`: Identifica accesos a ficheros inexistentes (para intentar parar a las herramientas que están escaneando la web en busca de ficheros concretos). En lugar de sobre el fichero de log de errores esta *jail* conviene activarla sobre el fichero de log de accesos.

```
[apache-auth]
enabled = true
port = http,https
filter = apache-auth
logpath = /var/log/apache*/*error.log
maxretry = 3

[apache-nohome]
enabled=true
port=http,https
filter=apache-nohome
logpath = /var/log/apache*/access.log
maxretry = 3
```

Figura 223. Jails adicionales de Fail2Ban para Apache 2

Por otro lado, en Internet podemos encontrar un gran nº de *jails* adaptadas a diferentes entornos y necesidades. Podemos listar un par de enlaces con ejemplos útiles en casos conocidos:

- *Jails* para `phpmyadmin` y para ataques `POST Flood`: http://linuxaria.com/howto/how-to-protecthttp://linuxaria.com/howto/how-to-protect-apache-with-fail2banApache 2-with-fail2ban
- Bloqueo de clientes que intenten acceder a lugares de la web prohibidos y también contra ataques típicos de *script kiddies*: https://github.com/miniwark/miniwark-howtos/wiki/Fail2Ban-setup-forhttps://github.com/miniwark/miniwark-howtos/wiki/Fail2Ban-setup-for-ApacheApache 2

Hechos estos cambios no debemos olvidarnos de reiniciar `fail2Ban` como hemos visto. Al hacerlo, si consultamos el *log* del módulo veremos las nuevas *jails* entrar en funcionamiento:

```
operario@ubuntu:/etc/fail2ban$ tail /var/log/fail2ban.log
2015-08-18 06:46:49,183 fail2ban.filter : INFO   Added logfile = /var/log/apache
2/error.log
2015-08-18 06:46:49,184 fail2ban.filter : INFO   Set maxRetry = 3
2015-08-18 06:46:49,185 fail2ban.filter : INFO   Set findtime = 600
2015-08-18 06:46:49,185 fail2ban.actions: INFO   Set banTime = 600
2015-08-18 06:46:49,191 fail2ban.jail   : INFO   Jail 'ssh' started
2015-08-18 06:46:49,194 fail2ban.jail   : INFO   Jail 'apache' started
2015-08-18 06:46:49,197 fail2ban.jail   : INFO   Jail 'apache-noscript' started
2015-08-18 06:46:49,199 fail2ban.jail   : INFO   Jail 'apache-overflows' started
2015-08-18 06:46:49,202 fail2ban.jail   : INFO   Jail 'apache-auth' started
2015-08-18 06:46:49,203 fail2ban.jail   : INFO   Jail 'apache-nohome' started
```

Figura 224. Reinicio de Fail2Ban en el log

Por último, para probar si `fail2Ban` está interceptando peticiones que nos hacen basadas en los *logs* del servidor, el *script* `fail2ban-regex`, distribuido con `fail2Ban` nos hace un informe de qué líneas del fichero de log especificado encajan con uno de los filtros. Opcionalmente puede imprimir todas las líneas que no encajan, aunque esa salida es muy grande y conviene volcarla a un fichero para su análisis. Esto se haría de la siguiente forma:

```
operario@ubuntu:~$ fail2ban-regex --print-all-missed /var/log/apache*/*error.log
 /etc/fail2ban/filter.d/apache-auth.conf>output_
```

Figura 225. Probando Fail2Ban

Bloque 4: Explotación de Servidores *Apache 2* bajo *Linux*

Configuración de múltiples host virtuales

Como ya hemos dicho, **cada *host* virtual de *Apache 2* representa un sitio *web* servido por el servidor**. De esta forma se permite a un sólo servidor alojar más de una *web* al mismo tiempo, y que parezca que todas ellas están siendo servidas por servidores diferentes a ojos de los usuarios. Así es posible aprovechar mejor los recursos del servidor, haciendo que este sirva tantas *web* como sus recursos permitan. Aunque a lo largo del libro hemos mencionado varias veces este concepto, no nos habíamos parado a describirlo en detalle. En esta sección repasaremos este concepto y profundizaremos más en sus posibilidades.

Los *host* virtuales se manejan de forma análoga a los módulos: cada uno tiene su propio archivo de configuración, con directivas que afectarán sólo a ese sitio *web*. Estos archivos deben estar en el directorio `sites-available` y **su nombre no puede empezar por los caracteres "." o "#"**. Es aconsejable que, para facilitar el manejo, el nombre de los *host* virtuales sea el mismo que los del dominio que representan, y que los archivos de configuración del *host* virtual se llamen también de esta forma.

Para activar y desactivar los *hosts* virtuales debemos usar los comandos `a2ensite` y `a2dissite`. Los *host* virtuales activos pueden verse en `/etc/apache2/sites-enabled/`. En este directorio debe existir al menos un *host* virtual: aunque solo tengamos una *web* en el servidor, *Apache 2* está configurado para tener como mínimo un *virtual host* (*Default Virtual Host,* en el fichero `000-default`), que será el que se asigne a esa única *web*. Todos los *host* virtuales disponibles aparecen listados en el directorio `/etc/apache2/sites-available`, tal y como se ve en las siguientes imágenes:

```
operario@ubuntu:/etc/apache2/sites-available$ ls
000-default.conf  default-ssl.conf
```

Figura 226. Host virtuales disponibles por defecto en Apache 2

```
operario@ubuntu:/etc/apache2/sites-enabled$ ls
000-default.conf  default-ssl.conf
```

Figura 227. Host virtuales activos en Apache 2

Hay dos tipos diferentes de *host* virtuales:

- **Basados en *IP***: donde una sola instancia del servidor tiene múltiples direcciones *IP* asignadas y cada nombre de dominio servido se asocia a cada una de esas *IPs*. Por tanto, cada web tiene una *IP* diferente, aunque están servidas por un único servidor.
- **Basados en el nombre (de dominio)**: donde el servidor tiene una única dirección *IP* asociada y varios nombres de dominio que se asocian con esa única dirección *IP*. Los diferentes sitios *web* se distinguen en este caso por ese nombre de dominio, dato que es el que usa *Apache 2* para servir las peticiones correctamente, escogiendo la web correcta entre todas las que sirve. El *virtual hosting* basado en nombre necesita pues que se pase un nombre de dominio en cada petición de la página. El servidor ve ese nombre y sirve la página que encaje con ese nombre de dominio. Para

que esto funcione, el nombre de dominio debe estar registrado con una entrada válida en un servidor *DNS*.

El siguiente ejemplo muestra dos nombres de dominio diferentes (`ejemplo.org` y `ejemplo.net`), que se alojarán en el mismo servidor. Ambos *hosts* están usando un `DocumentRoot` (directorio base donde se copian los contenidos de la *web*), *logs* de acceso y de error separados.

```
#Virtual Host 1 – ejemplo.org
<VirtualHost*:80>
ServerName www.ejemplo.org
ServerAdmin admin@ejemplo.org
DocumentRoot /var/www/html/www.example.org
CustomLog /var/log/apache2/www.example.org-access_log
ErrorLog/var/log/apache2/www.example.org-error_log
</VirtualHost>

#Virtual Host 2 – ejemplo.net
<VirtualHost *:80>
ServerName www.ejemplo.net
ServerAdmin admin@ejemplo.net
DocumentRoot /var/www/html/www.example.net
CustomLog /var/log/apache2/www.example.net-access_log
```

ErrorLog/var/log/apache2/www.example.net-error_log </VirtualHost>

La siguiente imagen muestra en cambio la configuración del *host* virtual por defecto que aparece en una instalación estándar de *Apache 2*:

```
 GNU nano 2.2.2              File: 000-default

<VirtualHost *:80>
        ServerAdmin webmaster@localhost

        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 /var/log/apache2/error.log
```

Figura 228. Host virtual por defecto en Apache 2

A modo de ejemplo básico complementario vamos a ver cómo configurar un *host* virtual, concretamente uno asociado a la *IP* `192.168.56.3`. Se asume que la máquina tiene una interfaz de red con esta *IP* asignada:

- Ir a `/etc/apache2/sites-available`
- Crear un archivo `site2.conf` con el siguiente contenido. En él vemos cómo **el *site* tendrá sus propios ficheros de *log* de acceso y errores** para no mezclar la información de ambos sitios (algo que se recomienda). Se supone además que previamente habremos creado el directorio `/var/www/html/site2` para que haga de `DocumentRoot` del nuevo sitio web y que se habrán movido los archivos de la *web* a servir al mismo).

```
 GNU nano 2.2.6            Archivo: site2.conf

<VirtualHost 192.168.56.3>
        ServerName Site2
        DocumentRoot /var/www/html/site2
        ErrorLog /var/log/apache2/site2-error.log
        TransferLog /var/log/apache2/site2-access.log
</VirtualHost>
```

Figura 229. Configuración del nuevo host virtual

- Ejecutar el comando `a2ensite site2` para habilitar el *host* virtual.
- Reiniciar *Apache 2* de la forma ya vista. Debería existir una entrada correspondiente a *site2* en el directorio `sites-enabled` de *Apache 2*, tal y como se muestra en la siguiente imagen:

```
operario@ubuntu:/etc/apache2/sites-enabled$ ls
000-default.conf  default-ssl.conf  site2.conf
```

Figura 230. Activación del nuevo host virtual

- Conviene comprobar además que los *logs* del nuevo *site* existen, para confirmar así que la nueva configuración se ha aplicado de forma correcta. Esto requiere comprobar el contenido de `/var/log/apache2`. Es importante tener presente que existe un máximo de descriptores de fichero manejable por *Apache 2* y que por tanto es posible que si tenemos muchos `VirtualHost` sobrepasemos este nº de ficheros si cada uno tiene un fichero de log. Se recomienda consultar el siguiente enlace para ver posibles soluciones a esta incidencia: http://httpd.apache.org/docs/current/vhosts/fd-limits.html

```
operario@ubuntu:/etc/apache2/sites-enabled$ ls -la /var/log/apache2/
total 596
drwxr-x---  2 root adm      4096 ago 20 09:45 .
drwxrwxr-x 10 root syslog   4096 ago 20 09:39 ..
-rw-r-----  1 root adm    154232 ago 20 09:45 access.log
-rw-r-----  1 root adm    204412 ago 20 09:45 error.log
-rw-r-----  1 root root   229316 ago 18 06:01 modsec_audit.log
-rw-r-----  1 root adm         0 ago  4 11:07 other_vhosts_access.log
-rw-r--r--  1 root root        0 ago 20 09:45 site2-access.log
-rw-r--r--  1 root root        0 ago 20 09:45 site2-error.log
operario@ubuntu:/etc/apache2/sites-enabled$
```

Figura 231. Logs separados para diferentes host virtuales de Apache 2

- Acceder al nuevo virtual *host* haciendo uso de su *IP* con `http://192.168.56.3` Ahora tenemos dos *webs* independientes servidas en la misma máquina.

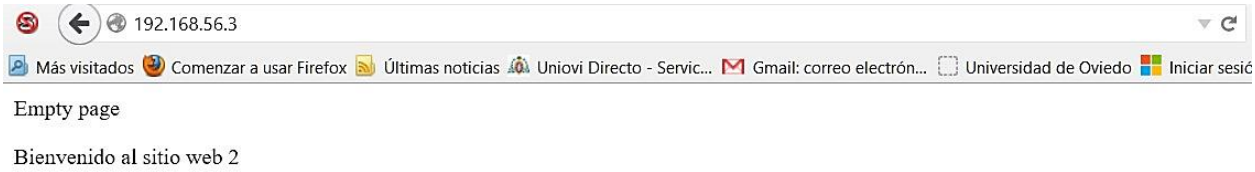


Figura 232. Nuevo host virtual

- Comprobamos finalmente que en el log correspondiente registra las peticiones al nuevo sitio web de manera independiente a las del resto.

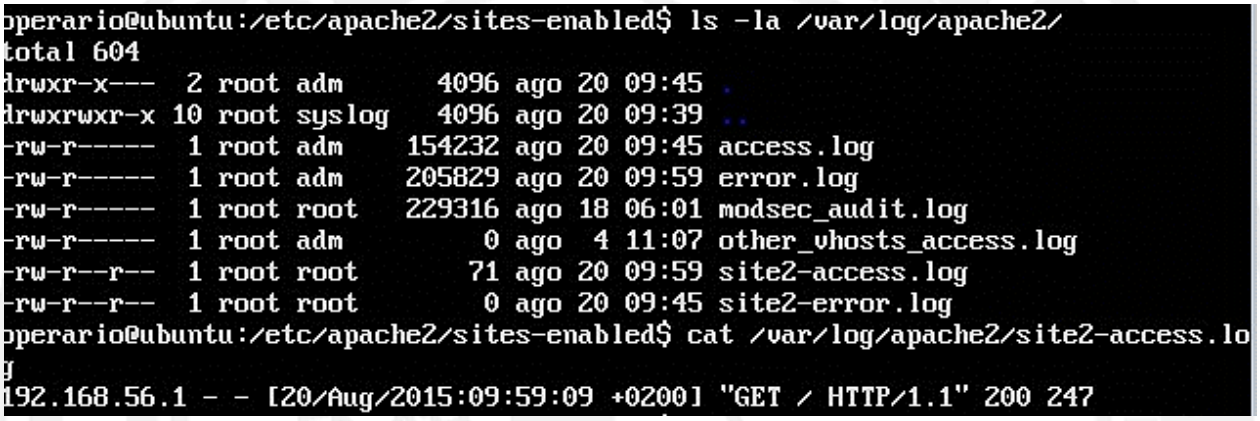


Figura 233. Log independiente del nuevo host virtual

A este nuevo *virtual host* podemos aplicarle configuraciones que ya hemos visto, como la compresión de archivos servidor. Es importante recordar que **podemos elegir entre hacer que todos los *host* virtuales tengan una configuración común** o bien hacer que **cada uno tenga la suya particular**, dependiendo de los contenedores de directivas que usemos a la hora de aplicar dichas configuraciones y el anidamiento que pongamos a los mismos (usando todos los contenedores `VirtualHost` que tengamos).

Un último detalle a clarificar es que podemos tener varias IPs asignadas a una misma máquina de varias formas, para que de esta forma se puedan tener varios host virtuales diferenciados por IP muy fácilmente. Fundamentalmente existen dos formas (**NOTA**: Es posible que el *Ubuntu* que estés usando **no use este gestor de red** (lo sabrás si los archivos que mencionamos están vacíos o no existen), sino otro llamado `netplan`, lo cual es típico en *Ubuntu 20.04* y superiores. En ese caso, sigue este manual: https://www.serverlab.ca/tutorials/linux/administration-linux/how-to-configure-networking-in-ubuntu-20-04-with-netplan/):

- **Teniendo varias interfaces de red físicas asignadas a un servidor** o bien un interfaz que admita la posibilidad de tener múltiples IPs asignadas. En caso de *VirtualBox*, podemos crear hasta 4 interfaces de red, algunos de los cuales pueden servir para este propósito.
- **Creando interfaces de red "falsos" que sean "alias" de interfaces de red reales**. Estos interfaces de red falsos funcionarán como los reales y les podremos asignar nuevas *IPs* que tengamos disponibles dentro de la red, sirviendo para este propósito. Existen dos formas de hacerlo:
    - De manera temporal, con un comando como el siguiente: `sudo ifconfig eth0:0 192.168.56.3 up`

- De manera permanente, modificando el fichero `/etc/network/interfaces`. El ejemplo presentado se ha realizado de esta manera con el siguiente fichero:

```
  GNU nano 2.2.2              File: interfaces                    Modified

# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth1
iface eth1 inet dhcp

auto eth2
iface eth2 inet static
address 192.168.56.2
netmask 255.255.255.0
network 192.168.56.0

#alias de eth2
auto eth2:0
iface eth2:0 inet static
name Ehternet alias LAN card
address 192.168.56.3
netmask 255.255.255.0
broadcast 192.168.56.255
network 192.168.56.0

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text^T To Spell
```

Figura 234. Múltiples IP asignadas a un interfaz de red del servidor

Las dos últimas líneas `broadcast` y `network` no son estrictamente necesarias.

Más ejemplos de configuración de *VirtualHost*

Visto el mecanismo para poner en marcha varias webs independientes en un mismo servidor, vamos a describir una serie de ejemplos útiles en situaciones comunes que se podrían dar en un servidor web.

Servir el mismo contenido en distintas IPs

Este ejemplo es útil en aquellos casos donde haya una LAN y una WAN y exista una web que pueda ser accedida desde ambos lugares, usando lógicamente dos IPs distintas correspondientes a estas redes. Así no tenemos que duplicar el contenedor `VirtualHost`. Se supone que ambas IPs se resuelven al nombre www.miempresa.com:

```
<VirtualHost 192.168.33.10 172.25.25.33>
    DocumentRoot "/var/www/html/webcorporativa"
    ServerName www.miempresa.com
    ServerAlias miservidor
</VirtualHost>
```

Este ejemplo tiene dos aspectos interesantes:

- En la red interna podríamos usar el nombre "`miservidor`" simplemente para acceder a la web.
- Si en lugar de la lista de IPs separadas por espacios usásemos *, todas las peticiones al servidor responderían con la misma página.

Webs en puertos distintos

Es frecuente tener **webs secundarias alojadas en puertos no estándar con diversos propósitos**, siendo el nombre del dominio de ambas el mismo o diferente. Esta es la forma de configurar el servidor para permitirlo:

- Lo primero es indicar a *Apache 2* que escuche en más puertos y no solo el 80. Para ello añadimos la siguiente configuración en `/etc/apache2/ports.conf`:

```
Listen 80
Listen 8080
```

- Hecho esto, ya podemos configurarlas:

```
<VirtualHost 172.25.25.33:80>
    ServerName www.miempresa.com
    DocumentRoot "/var/www/html/webcorporativa"
</VirtualHost>

<VirtualHost 172.25.25.33:8080>
    ServerName www.admin.miempresa.com
    DocumentRoot "/var/www/html/webcorporativa/admin"
</VirtualHost>

<VirtualHost 172.25.25.33:80>
    ServerName www.miempresa.es
    DocumentRoot "/var/www/html/webcorporativa/es"
</VirtualHost>

<VirtualHost 172.25.25.33:8080>
    ServerName www.admin.miempresa.es
    DocumentRoot "/var/www/html/webcorporativa/admin/es"
</VirtualHost>
```

Obviamente, **no debemos olvidar abrir los nuevos puertos** en el *firewall* del SO.

Host virtuales basados en IP

Si tuviéramos el caso de tener varias IPs asignadas a un mismo servidor y que dichas IPs estén asignadas a dominios diferentes, una configuración de host virtuales válida sería:

```
<VirtualHost 192.168.21.10>
    DocumentRoot /var/www/html/web1
    ServerName www.web1.com
</VirtualHost>

<VirtualHost 192.168.21.20>
    DocumentRoot /var/www/html/web2
```

```
    ServerName www.web2.com
</VirtualHost>
```

Host virtuales basados en IP y puertos

Imaginemos ahora el mismo caso anterior pero ahora por cada IP asignada al servidor queremos ejecutar webs en varios puertos distintos (en el ejemplo los puertos 80 y 5000). Para ello es asegurarnos de que *Apache 2* escucha en los puertos de cada IP que queramos usar en el fichero `ports.conf`:

```
Listen 192.168.21.10:80
Listen 192.168.21.10:5000 Listen 192.168.21.20:80
Listen 192.168.21.20:5000
```

Y luego ya podemos configurar los `VirtualHost`:

```
<VirtualHost 192.168.21.10:80>
    DocumentRoot /var/www/html/web1
    ServerName www.web1.com
</VirtualHost>

<VirtualHost 192.168.21.10:5000>
    DocumentRoot /var/www/html/web1/admin
    ServerName www.admin-web1.com
</VirtualHost>

<VirtualHost 192.168.21.20:80>
    DocumentRoot /var/www/html/web2
    ServerName www.web2.com
</VirtualHost>

<VirtualHost 192.168.21.20:5000>
    DocumentRoot /var/www/html/admin
    ServerName www.admin-web2.com
</VirtualHost>
```

Mezcla de host virtuales basados en nombre y basados en IP

Si una dirección IP aparece en una sola directiva `VirtualHost` y en ninguna más, entonces es un `VirtualHost` basado en IP. Eso no quiere decir, por supuesto, que no podamos tener una mezcla de ambos tipos de host virtuales:

```
Listen 80

# Estos host virtuales están basados en nombres de dominio
<VirtualHost 192.168.21.10>
    DocumentRoot /var/www/html/web1
    ServerName www.web1.com
</VirtualHost>

<VirtualHost 192.168.21.10>
    DocumentRoot /var/www/html/web2
    ServerName www.web2.com
</VirtualHost>
```

```
# Estos host virtuales están basados en IP
<VirtualHost 192.168.21.20>
    DocumentRoot /var/www/html/web3
    ServerName www.web3.com
</VirtualHost>

<VirtualHost 192.168.21.30>
    DocumentRoot /var/www/html/web4
    ServerName www.web4.com
</VirtualHost>
```

Usando _default_

`_default_` es un valor especial para la directiva `VirtualHost` que significa "*toda aquella IP/puerto del servidor que no esté asignada en ninguna otra directiva* `VirtualHost`". Se aplica también cuando el valor de la cabecera HTTP `Host` no corresponde a ningún host virtual registrado o directamente no existe esa cabecera.

El uso de este valor especial da bastante juego, y veremos algunos ejemplos de cómo usarlo para aplicaciones interesantes a continuación. Si no se usa esta palabra, toda petición a una IP del servidor sin `VirtualHost` asignado se servirá igual que una dirigida al host principal. El uso de `*` como valor de una directiva `VirtualHost` tiene más prioridad que `_default_`.

Para canalizar todas estas peticiones a una web por defecto podemos usar una configuración como la siguiente:

```
<VirtualHost _default_:*>
    DocumentRoot /var/www/html/webpordefecto/
</VirtualHost>
```

Una variante del ejemplo anterior es poder tener host virtuales por defecto en función del puerto en el que se haga la petición:

```
<VirtualHost _default_:80>
    DocumentRoot /var/www/html/webpordefecto/
    # ...
</VirtualHost>

<VirtualHost _default_:*>
    DocumentRoot /var/www/html/visitalawebprincipal/
    # ...
</VirtualHost>
```

El host virtual para el puerto 80 debe estar antes que el host virtual para el resto de los puertos. Finalmente, una variante de uso más sería el tener un host virtual por defecto solo para el puerto 80:

```
<VirtualHost _default_:80>
      DocumentRoot /var/www/html/webpordefecto/ ...
</VirtualHost>
```

Y con estos ejemplos cubrimos la mayor parte de casos típicos que se pueden dar en la puesta en marcha de una web concreta. No obstante, podemos ir más allá si lo necesitásemos, para lo cual se proporcionan los siguientes enlaces que describen necesidades especiales de puesta en marcha de host virtuales

- Múltiples host virtuales configurados de forma dinámica: http://httpd.apache.org/docs/current/vhosts/mass.html
- Múltiples host virtuales configurados de forma dinámica con `mod_rewrite`: http://httpd.apache.org/docs/current/rewrite/vhosts.html
- Más información general sobre host virtuales: https://serversforhackers.com/configuringhttps://serversforhackers.com/configuring-apache-virtual-hostsApache 2-virtual-hosts

Instalación de módulos típicos de lenguajes de servidor

Como hemos visto, los módulos de *Apache 2* **son una manera de agrupar y modularizar determinadas funcionalidades.** Cada módulo es el responsable de implementar y encapsular una determinada funcionalidad del servidor, de manera que una vez instalado, configurado correctamente y habilitado *Apache 2* adquiera dicha funcionalidad. Si posteriormente el módulo se deshabilita (o se desinstala), *Apache 2* automáticamente perderá dicha funcionalidad. Esto permite **una gran versatilidad de funciones al servidor** y también poder habilitar sólo aquellas funcionalidades que sean necesarias en un momento dado, evolucionando las funcionalidades del servidor según las necesidades que vayamos teniendo con el tiempo.

Esta máxima también se cumple con aquellos **módulos que habilitan la funcionalidad de soportar programas páginas dinámicas** en distintos lenguajes de servidor. Esta sección está pensada para describir los módulos más comunes hoy en día pertenecientes a esta categoría. `mod_python`, usado en secciones anteriores para ver ejemplos, pertenece a dicha categoría.

> ***Aunque ya lo hemos repetido hasta la saciedad anteriormente, en este caso es aún más importante incluso instalar sólo aquellos módulos que realmente vayamos a usar. Debe evitarse a toda costa soportar un lenguaje de programación del servidor que no use ninguna de las páginas alojadas, ya que hacer esto supone desventajas evidentes de seguridad y rendimiento.***

Los módulos que soportan lenguajes de creación de páginas dinámicas son módulos de terceros. A la hora de instalarlos existen principalmente hay dos formas:

- **La mayoría de ellos se pueden instalar en *Ubuntu* simplemente mediante** `apt-get`. Este comando lo descargará, instalará y configurará para poder trabajar directamente con el mismo, en algunos casos incluso activándolo directamente.
- Si el módulo no estuviese disponible mediante este comando, **se debe acudir a la página del fabricante de este**. Generalmente todo producto "serio" que pretende cooperar con *Apache 2* incluye las instrucciones detalladas para para instalarlo o un auto-instalador propio que nos facilite la labor, bien sea mediante un instalador automático o bien mediante su compilación y enlazado manual en nuestro servidor.

Estos módulos de terceros también pueden ser habilitarlos y deshabilitarlos individualmente con los comandos ya vistos `a2enmod` y `a2dismod`. A continuación describiremos la instalación de tres de ellos a modo de ejemplo, que son los correspondientes a lenguajes de desarrollo de páginas web dinámicas más usados de acuerdo con el **índice TIOBE**, usando datos de Agosto del año 2023: https://www.techrepublic.com/article/tiobe-index-language-rankings/#august

1. Python
2. C
3. C++
4. Java
5. C#
6. JavaScript
7. Visual Basic
8. SQL
9. Assembly language
10. PHP

Figura 235. Clasificación de uso de lenguajes según el índice TIOBE (Agosto 2023)

Lenguaje *Java*

El lenguaje Java para la programación de páginas web dinámicas está presente en varias tecnologías: *Servlets*, páginas *JSP* (*Java Server Pages*), Java *WebSockets*, etc. *Apache 2 Tomcat* (http://tomcat.apache.org/) es una implementación de código abierto de estas y otras tecnologías relacionadas con *Java*.

En esta guía vamos a explicar **cómo podemos hacer que *Tomcat 7* colabore con *Apache 2*** para poder servir aplicaciones basadas en tecnologías Java usando un módulo llamado `mod_jk`. En esta configuración el servidor web *Apache 2* es el encargado de recibir y devolver todas las peticiones HTTP de los usuarios. *Apache 2* sirve las web estáticas directamente y se encarga también de redirigir las peticiones relacionadas con tecnologías Java a *Tomcat*. Se trata de una configuración bastante sencilla y que presenta muchos beneficios, especialmente si se usa un esquema de balanceo de carga de la aplicación *Java*.

Instalación de *Tomcat*

El primer paso es instalar *Tomcat*, para lo cual seguiremos el procedimiento documentado para la versión estable más moderna existente. Ej.: https://www.digitalocean.com/community/tutorials/how-to-install-apache-tomcat-10-on-ubuntu-20-04

Para comprobar que *Tomcat* funciona correctamente, crearemos una aplicación de prueba sencilla consistente en un contador de visitas. Primero crearemos un directorio para la aplicación llamado `contadorVisitas` dentro de la ruta por defecto para alojar aplicaciones Java de *Tomcat* (https://tomcat.apache.org/tomcat-10.0-doc/manager-howto.html) y dentro del mismo una página JSP que tenga la implementación del contador de visitas y que se llame `index.jsp`:

Para verificar que *Tomcat* funciona correctamente tenemos que acceder a la IP del servidor usando el puerto `8080`, que es el puerto en el que *Tomcat* escucha peticiones por defecto. Si lo hacemos deberíamos ver la siguiente página:

```
<%@ page import="java.io.*, java.util.*" %>

<html>
	<head>
		<title>Contador de visitas en JSP</title>
	</head>
```

```
	<body>
	<%
		Integer visitas = (Integer)application.getAttribute("contador");
		if ((visitas == null) || (visitas == 0)) {
			out.println("Es tu primera visita a la página");
			visitas = 1;
		}
		else
		{
			visitas += 1;
			out.println("Es tu visita num. " + visitas);
		}
		application.setAttribute("contador", visitas);
	%>
	</body>
</html>
```

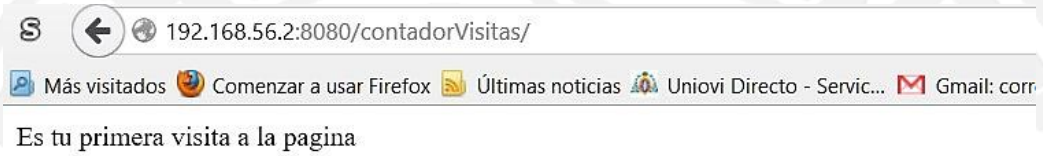


Figura 236. Primera visita a la página JSP de ejemplo

Los siguientes accesos a la página mostrarán el contador en funcionamiento:

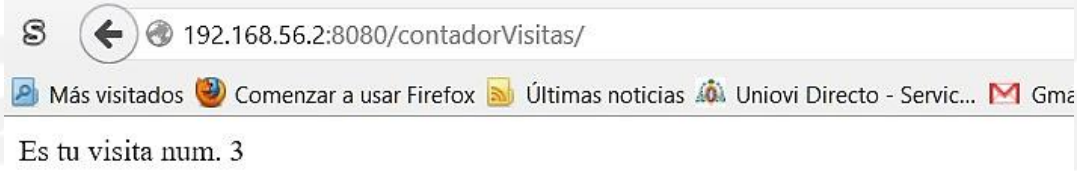


Figura 237. Accesos posteriores a la página JSP de ejemplo

Para que esto funcione, debemos asegurarnos previamente de que *Apache 2* no está funcionando en el puerto 8080 porque no puede haber dos aplicaciones diferentes (*Apache 2* y *Tomcat*) escuchando del mismo puerto.

Conectar Apache 2 y Tomcat con mod_jk

Aunque *Tomcat* está funcionando correctamente, no es muy usable obligar a nuestros usuarios a acceder a la web por el puerto 8080. Sería más interesante que solo con acceder al sitio web ya se nos redirigiera automáticamente a *Tomcat*. Esto es posible, como decíamos, gracias al módulo `mod_jk`. Este módulo se asegura de que un acceso a un host virtual que se haya configurado correctamente para ello en *Apache 2* sea redirigido a *Tomcat* si es oportuno, que procesará la petición y la devolverá a *Apache 2* el resultado sin que el usuario sepa quién de los dos programas ha servido realmente la petición. Para instalar este módulo seguimos el procedimiento acostumbrado:

```
operario@ubuntu:/etc/apache2/mods-enabled$ sudo apt-get install libapache2-mod-j
k_
```

Figura 238. Instalación de mod_jk

A continuación debemos asegurarnos de que *Tomcat* tiene activado el conector con el protocolo binario *AJP*, que es el usado por ambos programas para comunicarse. Para ello abrimos el archivo `server.xml` en la ruta donde *Tomcat* está instalado y verificamos la existencia de una línea similar a la siguiente, eliminando sus comentarios si los tuviera:

```
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

```
    <!--
    <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
               maxThreads="150" scheme="https" secure="true"
               clientAuth="false" sslProtocol="TLS" />
    -->

    <!-- Define an AJP 1.3 Connector on port 8009 -->

    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
  -
```

Figura 239. Conexión del conector AJP en Tomcat

El siguiente paso es crear un fichero `worker.properties` en `/etc/apache2` que indique a *Apache 2* donde está el conector con la instalación de *Tomcat* que tenemos en la propia máquina. El nombre del *worker* (`workertomcat`) es solo un ejemplo:

```
  GNU nano 2.2.6     Archivo: /etc/apache2/workers.properties       Modificado


#Definimos un worker que use el protocolo de comunicaciones AJP1.3
worker.list=workertomcat

#Definimos las propiedades del worker
worker.workertomcat.type=ajp13
worker.workertomcat.host=localhost
worker.workertomcat.port=8009
```

Figura 240. Fichero worker.properties

Y ahora le decimos a *Apache 2* que use el archivo `worker-properties` recién creado abriendo el fichero de configuración de `mod_jk` (`/etc/apache2/mods-available/jk.conf`) y modificando el valor de la directiva `JkWorkersFile` al siguiente:

```
JkWorkersFile /etc/apache2/workers.properties
```

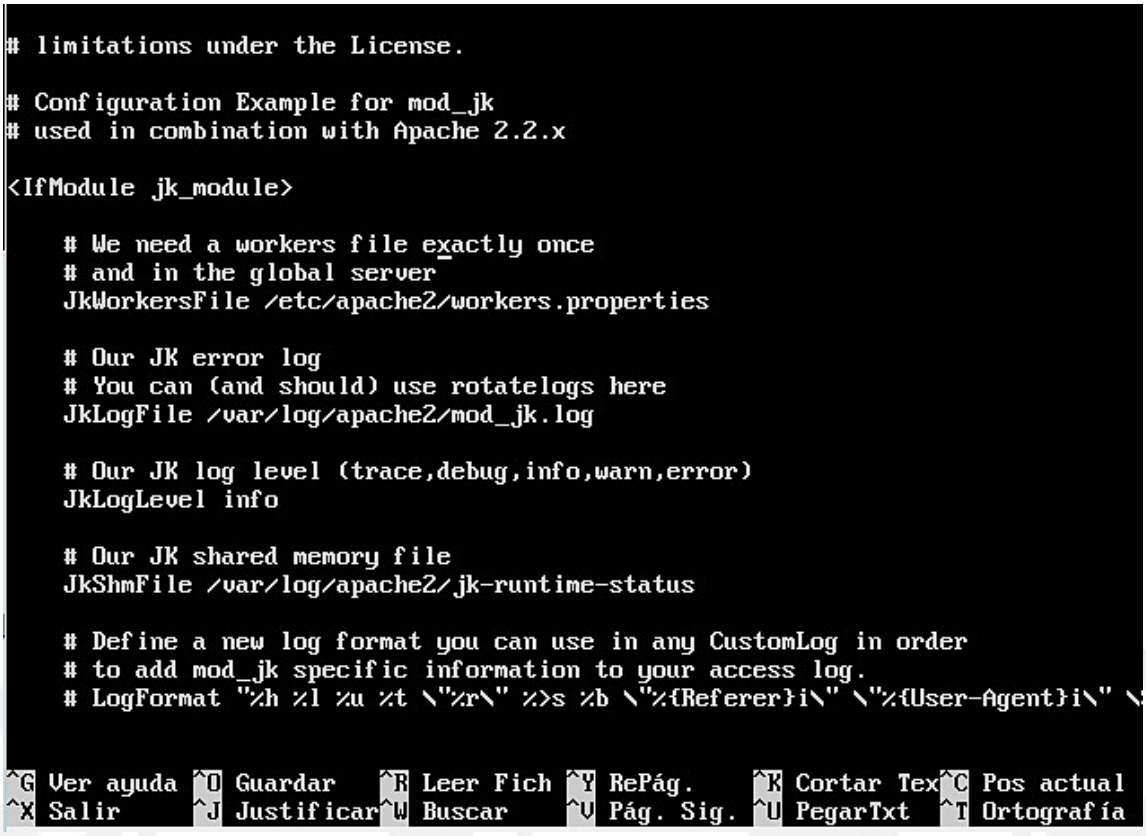


Figura 241. Modificando la configuración de mod_jk

Por último, para probar la página *JSP* vista, modificamos la configuración del `VirtualHost` predeterminado de *Apache 2*, para definir las URIs que debe pasar a través de *Tomcat*. En nuestro ejemplo dejaremos la configuración de este host virtual intacta, pero añadiendo una línea `JkMount` como la de la imagen, que indique la ruta de la URI donde está la aplicación y el *worker* que le dará servicio. Esto mismo podríamos haberlo hecho sobre cualquier otro host virtual, hacerlo en el de por defecto es sólo un ejemplo.

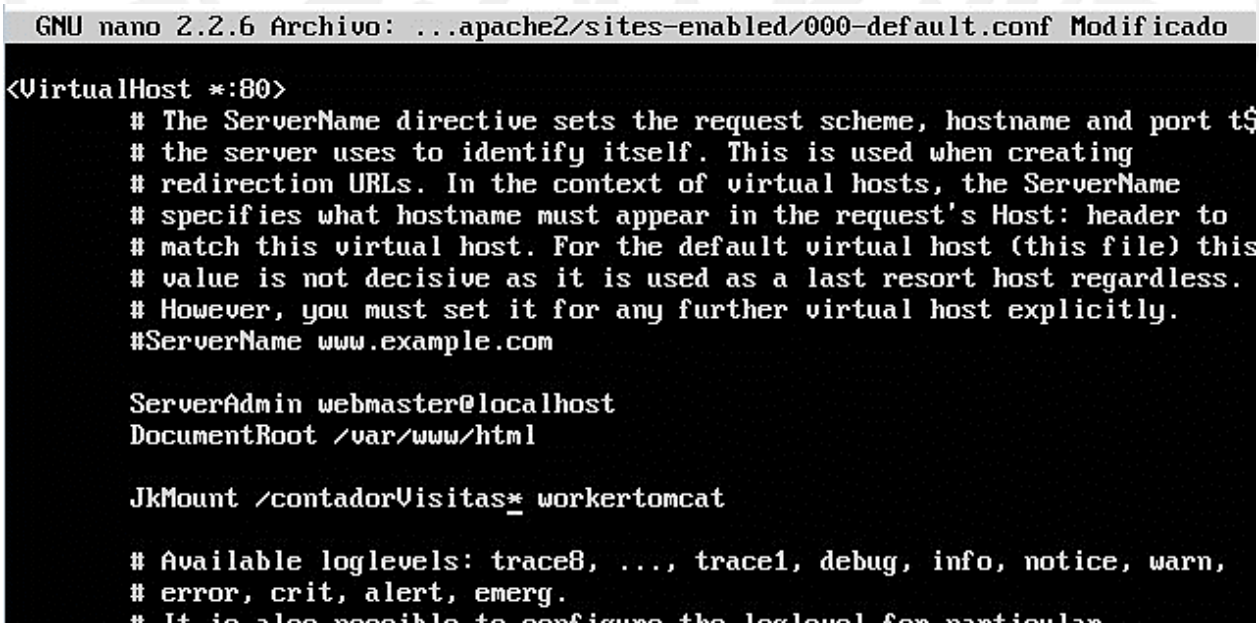


Figura 242. Añadiendo una ruta para Tomcat en Apache 2

Una vez completados estos pasos, ya estamos listos para dar el servicio mencionado. Solamente faltaría reiniciar *Apache 2* y *Tomcat* (en ese orden) y comprobar que no hay errores:

```
/etc/init.d/tomcat7 restart
/etc/init.d/Apache 2 restart
```

```
operario@ubuntu:/etc/apache2/mods-enabled$ sudo /etc/init.d/tomcat7 restart
 * Stopping Tomcat servlet engine tomcat7                              [ OK ]
 * Starting Tomcat servlet engine tomcat7                              [ OK ]
operario@ubuntu:/etc/apache2/mods-enabled$ sudo service apache2 restart
 * Restarting web server apache2
AH00558: apache2: Could not reliably determine the server's fully qualified doma
in name, using 127.0.1.1. Set the 'ServerName' directive globally to suppress th
is message
                                                                         [ OK ]
operario@ubuntu:/etc/apache2/mods-enabled$
```

Figura 243. Reinicio de Apache 2 y Tomcat

Y ahora ya podemos acceder a la ruta declarada sin usar un nº de puerto para comprobar que funciona correctamente: `http://192.168.56.2/contadorVisitas/`

Figura 244. Página JSP servida a través de Apache 2

Plataforma *.Net*

Es posible soportar una parte de las aplicaciones desarrolladas con el *.Net Framework* de *Microsoft* o con cualquier plataforma compatible gracias a `mod_mono`. Este módulo es la implementación de *Mono*, que es un proyecto de código abierto y multiplataforma que pretende crear una implementación compatible con el *CLR* de *Microsoft*, implementando el estándar *CLI* (http://www.ecma-international.org/publications/standards/Ecma-335.htm). Mediante este *software* podemos ejecutar las aplicaciones mencionadas. Podemos encontrar más información en:

- https://www.mono-project.com/docs/web/mod_mono/
- https://help.ubuntu.com/community/ModMono

Fundamentalmente `mod_mono` permite soportar aplicaciones construidas en el *framework* de desarrollo de aplicaciones web ASP .NET en *Linux* y *Windows*. El procedimiento de instalación básico es:

- Instalar el *mod* en sí, con el comando `sudo apt-get install libapache2-mod-mono`

```
operario@ubuntu:/etc/apache2/mods-enabled$ sudo apt-get install libapache2-mod-m
ono
```

Figura 245. Instalación de mod_mono

- `mod_mono` incluye una gran cantidad de librerías y paquetes en *Ubuntu*, por lo que la instalación y el espacio ocupado puede ser considerable. Tras la instalación, debemos activar el *mod* como lo hemos hecho con otros si la instalación no lo hiciera ya automáticamente: `sudo a2enmod mod_mono_auto mod_mono`.

- Como parte de los ficheros distribuidos con `mod_mono` se encuentra el servidor `mod-mono-server`, un programa distinto al módulo que procesa las peticiones de las aplicaciones *ASP .NET* que *Apache 2* le envía y que dentro tiene una máquina virtual Mono que es capaz de ejecutarlas. Este servidor y *Apache 2* están comunicados y colaboran para servir estas peticiones siguiendo un planteamiento similar al que usan *Apache 2* y *Tomcat*. No obstante, antes de seguir explicando cómo instalar este módulo es necesario encontrar donde se encuentra el binario de dicho servidor para luego comprobar la configuración del módulo. Esto podemos hacerlo mediante el siguiente comando:

```
operario@ubuntu:~$ sudo find / -name mod-mono-server4.exe
/usr/lib/mono/gac/mod-mono-server4/3.0.0.0__0738eb9f132ed756/mod-mono-server4.ex
e
/usr/lib/mono/4.5/mod-mono-server4.exe
operario@ubuntu:~$
```

Figura 246. Binario de mod-mono-server

- Localizado el servidor en el directorio que sea, tenemos que localizar también donde se ha instalado el ejecutable vía línea de comandos de dicho servidor de la siguiente forma:

```
operario@ubuntu:~$ whereis mod-mono-server4.exe
mod-mono-server4: /usr/bin/mod-mono-server4 /usr/share/man/man1/mod-mono-server4
.1.gz
operario@ubuntu:~$
```

Figura 247. Ejecutable de mod-mono-server4

- Localizado el ejecutable en línea de comandos, lo arrancamos para comprobar que funciona correctamente:

```
operario@ubuntu:~$ /usr/bin/mod-mono-server
mod-mono-server4
mod-mono-server4
Listening on: /tmp/mod_mono_server
Root directory: /home/operario
Hit Return to stop the server.

operario@ubuntu:~$
```

Figura 248. Prueba de funcionamiento de mod-mono-server

- Una vez que tenemos ambas rutas, podemos editar el fichero de configuración de `mod-monoserver` para asegurarnos que dicho fichero tiene las rutas correctas. Para ello editamos el fichero `/etc/mono-server4/mono-server4-hosts.conf` (o la ruta correspondiente a la versión que hayamos instalado) para comprobar que las directivas siguientes tienen los valores que aparecen en el cuadro:

```
MonoServerPath default /usr/bin/mod-mono-server4
MonoPath default /usr/lib/mono/4.5:/usr/lib
```

```
  GNU nano 2.2.6 Archivo: ...c/mono-server4/mono-server4-hosts.conf Modificado

# Default configuration, don't edit it!
<IfModule mod_mono.c>
  MonoUnixSocket default /tmp/.mod_mono_server4
  MonoServerPath default /usr/bin/mod-mono-server4
  AddType application/x-asp-net .aspx .ashx .asmx .ascx .asax .config .ascx
  MonoApplicationsConfigDir default /etc/mono-server4
  MonoPath default /usr/lib/mono/4.5:/usr/lib
</IfModule>
```

Figura 249. Edición de la configuración de mod-mono-server

- Para probar `mod_mono` vamos a crear una página de pruebas sencilla y tratar de ejecutarla. No obstante, a diferencia de lo que hicimos con *Tomcat*, ahora vamos a hacer que las aplicaciones .NET tengan su propio host virtual. Aunque podemos crear uno nuevo sin problemas de la forma vista, en este caso reutilizaremos el visto en el apartado de host virtuales llamado `site2`. Cambiar la configuración para que se soporte .NET en este host virtual es muy sencillo, puesto que solo tenemos que añadir el `Handler` de mono y luego opcionalmente hacemos que `index.aspx` sea la página por defecto cargada si se accede directamente al directorio:

```
  GNU nano 2.2.6  Archivo: /etc/apache2/sites-available/site2.conf  Modificado


<VirtualHost 192.168.56.3>
        ServerName Site2
        DocumentRoot /var/www/html/site2
        ErrorLog /var/log/apache2/site2-error.log
        TransferLog /var/log/apache2/site2-access.log

        SetHandler mono
        DirectoryIndex index.aspx_
</VirtualHost>
```

Figura 250. Modificando la configuración de site2 para soportar .NET

- El último paso antes de probar el módulo es crear un fichero de configuración `.webapp` para el servidor de `mono` que le indique como gestionar la aplicación ASP .NET que se va a crear en el host virtual `site2`. Este fichero se debe crear en el directorio `/etc/mono-server4` (o la carpeta correspondiente a la versión que hayamos instalado) y la llamaremos `asp.net.webapp`. El contenido es el siguiente, teniendo en cuenta que debemos especificar correctamente tanto el *path* como el nombre del host virtual para que funcione:

```
  GNU nano 2.2.6     Archivo: /etc/mono-server4/asp.net.webapp      Modificado


<apps>
        <web-application>
                <name>ejemplo.aspnet</name>
                <vpath>/</vpath>
                <path>/var/www/html/site2</path>
                <vhost>site2</vhost>
        </web-application>
</apps>
```

Figura 251. Configuración de mod-mono-server para el site2

- Ahora ya podemos probar la instalación del módulo con una página de prueba como ésta, a la que llamaremos `index.aspx`, situándola en el `DocumentRoot` de `site2`. Es muy importante no olvidar la primera línea `<%@Page Language="C#" %>` para evitar posibles problemas de compilación.

```
 <%@Page Language="C#" %>
<html>
	<head>
		<title>Ejemplo de ASP .NET</title>
	</head>
	<body>
		<form id="principal" runat="server">
			<asp:label id="cabecera" runat="server">
	Variables de entorno de ASP .NET:
			</asp:label>

			<%
			foreach (string v in Request.ServerVariables)
				Response.Write("<p><b>" + v + "</b> = " + Request[v] +
"</p>");
			%>
		</form>
	</body>
</html>
```

- Finalmente, podemos probar dicha página y ver que funciona:

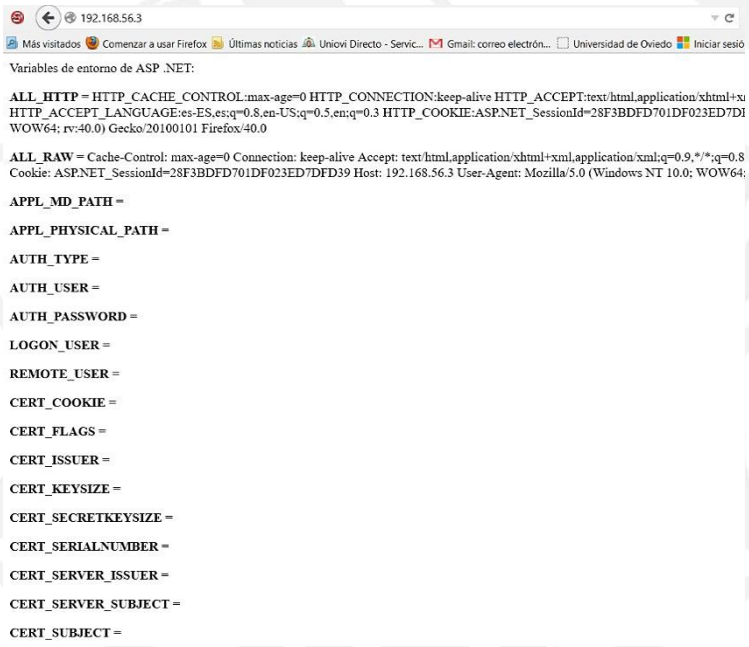


Figura 252. Página ASP .NET de prueba

Lenguaje *PHP*

Para instalar la última versión de PHP en Ubuntu podemos seguir este tutorial: https://linuxhint.com/install-latest-php-ubuntu22-04/. Sobre la instalación de PHP, cabe decir dos cosas adicionales:

- **Es muy frecuente instalar PHP junto con otros productos que se usarán para crear diferentes partes de la aplicación web**. Una de esas combinaciones más famosa es la llamada *LAMP Stack* (*Linux, Apache 2, MySQL y PHP*). Nosotros no entraremos en la instalación de MySQL (el único componente del *Stack* que falta por instalar), pero pueden encontrarse más instrucciones aquí: https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-on-ubuntu-22-04
- **Para probar si PHP está funcionando correctamente**, además de hacer una página sencilla de pruebas, es también muy frecuente crear una página como la siguiente, que muestra información detallada de la versión instalada de PHP y sus variables. Obviamente esta página da demasiada información, por lo que nunca debemos dejarla accesible a los usuarios del servidor.

```
 GNU nano 2.2.6          Archivo: info.php          Modificado

<?php
        phpinfo();
?>
```

Figura 253. Página que muestra el status de PHP

El resultado de acceder a esta página es el siguiente:

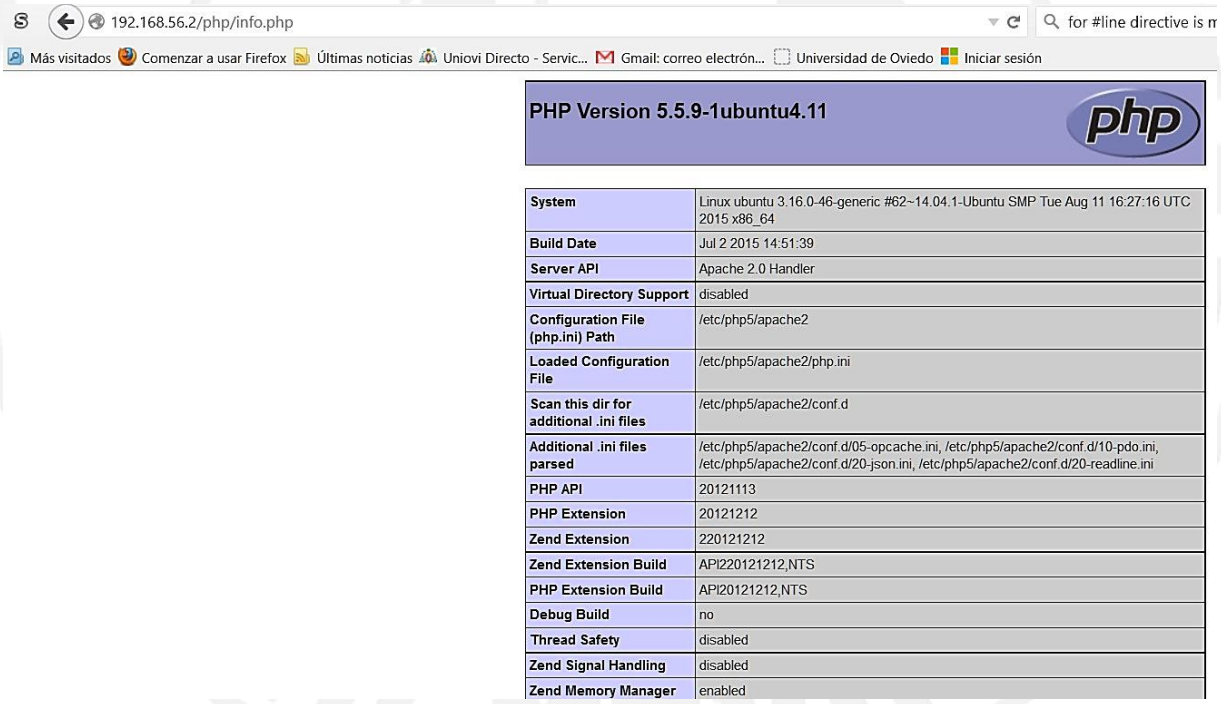


Figura 254. Status de PHP

Profundizar más en PHP está fuera del ámbito de este libro porque queremos limitarnos a temas de administración. No obstante, el rendimiento de PHP se puede mejorar si se habilita la compresión de las páginas que sirve como se indica en el siguiente enlace: http://bernaerts.dyndns.org/linux/75-debian/50-debian-compactflash-tune-apache

Configuración personalizada de webs particulares

Una opción interesante de *Apache 2* es **la posibilidad de que cada usuario registrado en el SO del servidor donde está instalado pueda tener una *web* personal independiente**. De esta forma cualquier usuario registrado que lo desee podrá servir páginas *web* directamente de los directorios de su cuenta personal de usuario, creando y gestionando dicha *web* de forma personal y privada a través de ficheros `.htaccess` suponiendo que no tiene acceso a la configuración del servidor.

Esto puede servir **para que los usuarios sean los principales responsables de su propia *web***, creándoles una cuenta de usuario y dándoles espacio de almacenamiento y libertad de actuación dentro del espacio asignado a su propia cuenta de usuario. Puede ser interesante como opción de *hosting web* gestionado por el propio cliente, si los usuarios representan a empresas o negocios, o bien para crear fácilmente *web* personales de perfiles dentro de una empresa.

Debe quedar claro que **todas esas *webs* se regirán por la configuración y módulos instalados en el *Apache 2* del servidor**. Por ejemplo, si únicamente hemos instalado `mod_php` en *Apache 2*, los usuarios que deseen crear una *web* personal dinámica sólo podrán usar *PHP*. Lo mismo ocurre con la configuración de seguridad y otros aspectos; si se ha prohibido o limitado el uso de archivos `.htaccess` con una directiva en `apache2.conf`, estas *web* personales tampoco podrán esta clase de archivos, y por tanto no podrán personalizar su configuración completamente. Se consigue así una solución intermedia entre libertad de actuación y administración para los usuarios y control centralizado de sus actividades por parte del administrador.

Toda esta funcionalidad la permite un módulo denominado `mod_userdir` (https://httpd.apache.org/docs/current/mod/mod_userdir.html). Aunque el *mod* tiene múltiples opciones que podemos consultar en el enlace anterior, básicamente lo único necesario para que un usuario pueda empezar a servir sus propias *web* inmediatamente es:

- **Introducir el contenido *web* en un directorio concreto dentro de su cuenta de usuario**. El nombre de este directorio se establece como parte de la configuración del propio módulo, pero por defecto es `public_html`.
- **Que el propio usuario dé permisos de ejecución a "otros"** como mínimo para que *Apache 2* pueda acceder al contenido de este directorio y servir sus páginas a usuarios anónimos (o bien elija el esquema de acceso a usuarios del SO que prefiera).
- Que, salvo que se desee lo contrario, **exista una página principal en el fichero** `index.html` (u otro nombre establecido por la configuración de *Apache 2 y* `mod_dir`) para que al acceder a la *web* personal se cargue su contenido automáticamente. En caso de no existir dicha *web* "por defecto", acceder a la *web* personal se comportará igual que lo que vimos en la sección que hablaba del listado de directorios.
- Acceder a la propia *web* personal, normalmente mediante una URI como la siguiente: www.miservidor.es/~nombredelusuario

Vamos pues a implementar este procedimiento para ver cómo funciona. Como primer paso para poder hacer que los usuarios del sistema operativo tengan su propia *web* personal lo primero es habilitar `mod_userdir` con el comando `a2enmod userdir` y reiniciar *Apache 2*. Una vez hecho esto, podemos modificar la configuración del *mod* cambiando los parámetros del fichero `userdir.conf` en el directorio `mods-enabled` de *Apache 2*. En nuestro caso nos sirve con la configuración por defecto, por lo que esto no será necesario. En la siguiente imagen se muestra el contenido por defecto de dicho fichero:

```
 GNU nano 2.2.6          Archivo: userdir.conf

<IfModule mod_userdir.c>
        UserDir public_html
        UserDir disabled root

        <Directory /home/*/public_html>
                AllowOverride FileInfo AuthConfig Limit Indexes
                Options MultiViews Indexes SymLinksIfOwnerMatch IncludesNoExec
                <Limit GET POST OPTIONS>
                        Require all granted
                </Limit>
                <LimitExcept GET POST OPTIONS>
                        Require all denied
                </LimitExcept>
        </Directory>
</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

Figura 255. Configuración por defecto de mod_userdir

A la hora de hacer una prueba de funcionalidad, debemos simplemente entrar en sesión como cualquier usuario y crear un directorio llamado `public_html` (salvo que cambiemos el nombre de este directorio en la configuración del *mod*). Hecho esto, basta simplemente con crear un fichero `index.html` con algún contenido de prueba en dicho directorio. En este caso lo haremos con el usuario operario que hemos usado en todos los ejemplos:

```
operario@ubuntu:~/public_html$ ls -la
total 12
drwxrwxr-x 2 operario operario 4096 ago 17 10:45 .
drwxr-xr-x 4 operario operario 4096 ago 17 10:45 ..
-rw-r--r-- 1 operario operario  116 ago 17 10:45 index.html
operario@ubuntu:~/public_html$ _
```

Figura 256. Página a servir desde un directorio de usuario

Con esto hemos dado todos los pasos de configuración. Para probar su funcionamiento, accedemos a la dirección del servidor usando el nombre de usuario para construir una URI como la siguiente (añadiendo el carácter ~): http://192.168.56.2/~operario. Comprobamos que la página pedida efectivamente se sirve:

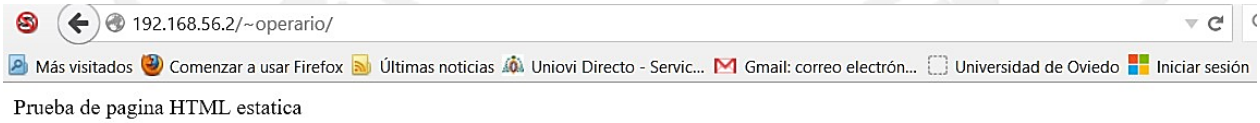


Figura 257, Página a servir desde un directorio de usuario

Antes de finalizar la sección tenemos que hablar de posibles problemas de incompatibilidad entre la configuración del servidor y este módulo. Si hemos aplicado la configuración anterior que prohíbe el listado de directorios mediante la técnica de no permitir servir ningún fichero que no esté expresamente autorizado en el contenedor `<Directory />`, comprobaremos que no es posible acceder a este directorio. *¿Por qué?* Pues el motivo es que realmente estamos sirviendo un fichero en una cuenta de usuario que no

ha sido específicamente autorizado. Por ello, no podremos usar este tipo de medidas si se usa `mod_userdir`, teniendo que limitar el acceso en localizaciones menos generales.

Por otro lado, hemos dicho que la directiva `AllowOverride` controla cómo se procesan los archivos `.htaccess`. Mediante estos archivos los usuarios del SO podrán hacer cambios sobre la configuración de su propio directorio, siendo un medio muy adecuado para llevar a cada directorio una configuración completamente personalizada que se adapte a su propósito concreto. De cara a su uso en directorios de usuario, debemos tener en cuenta los siguientes puntos:

- Las directivas especificadas en uno de estos archivos se aplican al directorio donde está el fichero `.htaccess` y a todos sus subdirectorios, salvo que alguno de esos subdirectorios tenga un `.htaccess` accesible (si la configuración lo permite) que tenga especificado unas opciones diferentes.
- Los archivos con este fin no tienen que llamarse obligatoriamente `.htaccess`, se pueden renombrar con la directiva `AccessFileName` (Ej.: `AccessFileName .config`), presente en `apache2.conf`.
- Como hemos enumerado anteriormente, `AllowOverride` tiene varios parámetros que permiten establecer limitaciones de los conjuntos de directivas que se pueden poner en un fichero
- `.htaccess`. Es típico con `mod_userdir` usar dos de ellas: `AllowOverride None` que básicamente hace que los ficheros `.htaccess` en directorios afectados por esta configuración no se procesen, (al no permitirse ninguna directiva en ellos) y `AllowOverride AuthConfig`, que permitiría establecer métodos y opciones de autenticación de usuarios como los que veremos en la próxima sección en un fichero `.htaccess` , y que por tanto afecten sólo a la carpeta donde está situado dicho fichero y a las subcarpetas que dependan de la misma.

Finalmente, permitir que estos ficheros sean accesibles por todo el mundo es un problema de seguridad al dejar visible públicamente datos de configuración del servidor. Para evitarlo podemos usar las siguientes opciones:

- **Renombrándolos** para que no sea obvio el acceso al os mismos(ya vimos cómo)
- **Prohibiendo su acceso**: *Apache 2* ya viene preparado para ello, pero no está de más comprobar que el fichero de configuración central del mismo cuenta con las siguientes líneas:

```
 GNU nano 2.2.6       Archivo: /etc/apache2/apache2.conf

#</Directory>



# AccessFileName: The name of the file to look for in each directory
# for additional configuration directives.  See also the AllowOverride
# directive.
#
AccessFileName .htaccess

#
# The following lines prevent .htaccess and .htpasswd files from being
# viewed by Web clients.
#
<FilesMatch "^\.ht">
        Require all denied
</FilesMatch>
```

Figura 258. Apache 2 prohíbe el acceso directo a ficheros .ht*

HACK FIRST
HACK HARD
NO MERCY
大和
Cobra kai

Reescritura de URIs: *Mod_rewrite*

`mod_rewrite` (http://httpd.apache.org/docs/current/mod/mod_rewrite.html) es un módulo de *Apache 2* enormemente versátil que **permite la manipulación de las URI de las peticiones** para convertirlas a una forma más afín a un determinado propósito. *Apache 2* puede, gracias a este módulo, manipular la URI de cada petición de manera que incluso lo que el cliente solicita en la petición **puede no parecerse en absoluto** a lo que se va a servir realmente.

Para ello, se definen una lista de **reglas basadas en expresiones regulares**. Si la URI de la petición encaja con alguna de esas expresiones regulares y se dan el resto de las **condiciones** establecidas junto con la regla (si las hubiera), entonces la URI de la petición se modifica según el patrón de sustitución establecido en dicha regla. Este proceso continua hasta que ya no hay más reglas aplicables sobre esa petición. Esta es solo la forma general de trabajar de este módulo, que iremos perfilando a continuación.

Uno de los usos más frecuentes de `mod_rewrite` es transformar URIs poco usables en otras mucho más sencillas de entender para un cliente. Por ejemplo, imaginemos que tenemos la siguiente URI: www.miweb.com/compras/procesar_compra.php?pedido=1236785;id=usuario1. Esta URI tiene detalles técnicos (como los parámetros con nombre) y es menos intuitiva que simplemente poner: www.miweb.com/compras/tramitarPedido/1236785/usuario1/cesta.htm.

La segunda URI es más fácil de entender, pero tiene otras ventajas:

- Ya **no revelamos que estamos usando PHP** para crear las páginas web de la aplicación. Ni tampoco el nombre real de la página usada (por lo que es mucho más difícil encontrarla para examinarla en busca de posibles vectores de ataque).
- Otra ventaja añadida es que **no indicamos la variable usada** para guardar el identificador de pedido, ni que estos están identificados por una cadena numérica (dato que podría usarse para intentar ver si podemos averiguar el contenido de otros pedidos) • Tampoco revelamos qué variable guarda el identificador de usuario.
- La segunda URI es **más fácilmente indexable** por buscadores.

En definitiva, no damos detalles técnicos de cómo están hechas las peticiones, lo que al final revierte en una mayor seguridad para la aplicación. Pasaremos ahora a ver cómo usar `mod_rewrite` para estas y otras funciones.

Uso básico del módulo

Lo primero que debemos averiguar es si el *mod* está habilitado en nuestra instalación de *Apache 2*. De no estarlo, habría que habilitarlo como de costumbre, con `a2enmod rewrite` y reinicio del servidor.

Prueba de instalación

Una vez habilitado, tendremos que comprobar que está funcionando correctamente. Además se mirar si aparece listado en `mods-enabled`, podemos hacer una sencilla prueba consistente en redirigir todas las peticiones a una sección concreta de una web (directorio `rewrite`) a una página por defecto que muestre que está en construcción que hayamos creado en esa localización. Esto se hace añadiendo en un fichero de configuración las siguientes directivas:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf          Modificado


<Directory "/var/www/html/rewrite">
        RewriteEngine on
        RewriteRule .* en_construccion.html
</Directory>
_
```

Figura 259. Prueba básica de mod_rewrite

Comprobamos como en dicha localización activamos `mod_rewrite` (`RewriteEngine on`) y luego hacemos que cualquier petición (expresión regular `.*`) se redirija a la página `en_construcción.html`. Probamos ahora que el comportamiento es el esperado, accediendo a dos páginas que no existen en dicho directorio:

Figura 260. Acceso a una página inexistente y mod_rewrite (I)

Figura 261. Acceso a una página inexistente y mod_rewrite (II)

En esta prueba podemos ver claramente cómo aunque la página servida en todos los casos será `en_construcción.html` no aparece referencia a la misma en la barra de direcciones ni hay indicios de que exista ninguna redirección. El servidor recibe una petición, el módulo la transforma antes de servirla y el servidor sirve la petición transformada de manera completamente transparente al cliente, sin que sea consciente de lo que está pasando.

Uso de expresiones regulares

`mod_rewrite` hace un uso intensivo de las ya mencionadas ***Perl Compatible Regular Expressions*** o *PCRE* (http://perldoc.perl.org/perlre.html). Esto significa que debemos tener un cierto dominio de estas para poder explotar realmente las ventajas de este *mod*. En los ejemplos que veremos usaremos expresiones regulares comunes y sencillas de entender para un gran nº de usos frecuentes, aprovechando los mismos para explicarlas sobre la marcha. Por el momento, para entender los ejemplos siguientes debemos entender dos cosas:

- `\w` se traduce como cualquier secuencia alfanumérica de nºs y caracteres (`[A-Za-z0-9_]`).
- `\d` se refiere a cualquier dígito `[0-9]`.

Si necesitásemos profundizar más en el uso de expresiones regulares, se recomienda consultar el siguiente enlace: http://www.regular-expressions.info/tutorialcnt.html. Otra opción es tener una tabla de referencia rápida como la disponible en: http://www.cheatography.com/davechild/cheat-sheets/regularhttp://www.cheatography.com/davechild/cheat-sheets/regular-expressions/expressions/

Ejemplo de uso

Como primer ejemplo del uso de `mod_rewrite` vamos a ver cómo hacer la transformación de la URI de la petición que mostramos al principio de la sección. Recordemos que se trata de que si recibimos la siguiente petición: www.miweb.com/compras/tramitarPedido/1236785/usuario1/cesta.htm debemos transformarla en la URI que de verdad haga la compra en nuestra aplicación, que en el ejemplo era: www.miweb.com/compras/procesar_compra.php?pedido=1236785;id=usuario1. En este caso colocaremos las directivas dentro de un contenedor del directorio `/compras`, que suponemos que existe en nuestro servidor:

```
<Directory /var/www/html/compras>
# Habilitar mod_rewrite en este directorio  RewriteEngine on

# Reescritura de la URI
# Nos llega: tramitarPedido/ID/USUARIO/cesta.htm
# Queremos conseguir:
#procesar_compra_y_enviar_pasarela.php?pedido=ID;id=USUARIO

RewriteRule ^tramitarPedido/(\d+)/(\w+)/cesta.htm$ procesar_compra.php?pedido=$1&id=$2
</Directory>
```

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf              Modificado


<Directory "/var/www/html/compras">
RewriteEngine on
RewriteRule ^tramitarPedido/(\d+)/(\w+)/cesta.htm$ procesar_compra.php?pedido=$$
</Directory>
```

Figura 262. Reescritura de una URI con mod_rewrite

Para probar que funciona correctamente, usaremos la siguiente página `procesar_compra.php` sencilla, suponiendo que hemos instalado previamente un módulo que nos permita procesar páginas en este lenguaje (visto en una sección anterior):

```
<?php
	// Obtenemos parametros de la URI
	$pedido = $_GET['pedido'];
	$id = $_GET['id'];
?>
<html>
<head>
	<title>Ejemplo de mod_rewrite</title>
</head>
<body>
	<h1>Datos de su pedido:</h1>
	<p>El usuario: <?php echo $id; ?> ha hecho el pedido <?php echo $pedido; ?> </p>
</body>
</html>
```

```
 GNU nano 2.2.6        Archivo: procesar_compra.php          Modificado


<?php
        $pedido = $_GET['pedido'];
        $id = $_GET['id'];
?>
<html>
        <head>
                <title>Ejemplo de mod_rewrite</title>
        </title>
        <body>
                <h1>Datos de su pedido:</h1>
                <p>El usuario: <?php echo $id; ?> ha hecho el pedido
                <?php echo $pedido; ?></p>
        </body>
</html>
```

Figura 263. Página web procesar_compra.php de ejemplo

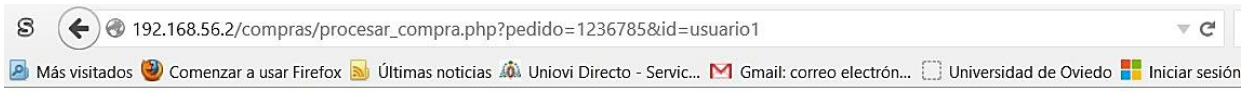
Sobre este ejemplo ya podemos explicar varias cosas:

- Conviene incluir la directiva `RewriteEngine On` en toda sección que use el *mod*, porque no sabemos si estará activado previamente o no.
- Hemos convertido una URI más sencilla de entender en otra que tiene el formato de las peticiones esperado por la aplicación, con los nombres de variables esperados por la misma, siendo esta la forma de trabajar habitual en estos casos. Para esta transformación, hemos hecho uso de los siguientes elementos:

`RewriteRule ^tramitarPedido/(\d+)/(\w+)/cesta.htm$ procesar_compra.php?pedido=$1;id=$2`	
Patrón	**Significado**
^	Comienzo de la URI
/(\d+)/	Parte de la URI que significa "/<cualquier nº>/". Si encaja con la petición, escriben los caracteres que encajan en el localizador $1
(\w+)/	Cualquier palabra seguida de "/". Si encaja con la petición, escriben los caracteres que encajan en el localizador $1
cesta.html	La cadena "cesta.html"
$	Fin de la entrada
Sustitución	
procesar_compra.php?pedido=	URI cambiada (literal)
$1	Primer parámetro capturado
$2	Segundo parámetro capturado

Veamos que hace el módulo ante diferentes entradas:

- `…procesar_compra.php?pedido=1236785&id=usuario1`: La petición no encaja con el patrón (es más, se está solicitando la página real directamente). Por tanto, no se realiza la sustitución y la petición se tramita correctamente puesto que la página realmente existe y los parámetros son correctos.

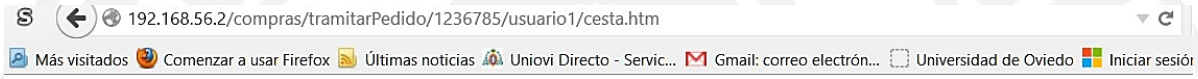


Datos de su pedido:

El usuario: usuario1 ha hecho el pedido 1236785

Figura 264. Acceso directo a una URI que es objetivo de la transformación de mod_rewrite

- `…/tramitarPedido/1236785/usuario1/cesta.htm`: La petición encaja perfectamente con el patrón por lo que la sustitución se realiza y se accede correctamente a la URI: `…procesar_compra.php?pedido=1236785;id=usuario1`

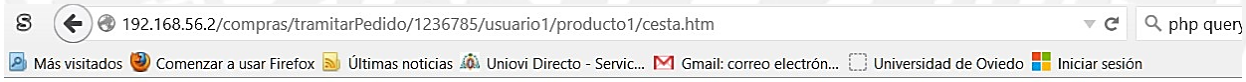


Datos de su pedido:

El usuario: usuario1 ha hecho el pedido 1236785

Figura 265. Transformación de una URI gracias a mod_rewrite

- `…/tramitarPedido/1236785/usuario1/producto1/cesta.htm`: La petición no encaja con el patrón (hay un directorio `/producto1` adicional que lo impide), por lo que no hay sustitución, se intenta acceder a la URI tal cual se muestra y se obtiene un error *404 Not Found*, ya que no existirá en el caso que hemos descrito.



Objecto no localizado!

El URL solicitado no ha sido localizado en este servidor. Si usted tecleó el URL manualmente, por favor revise su ortografía y vuélvalo a intentar.

Si usted cree que esto es un error del servidor, por favor comuníqueselo al administrador del portal.

Error 404

192.168.56.2
Apache/2.4.7 (Ubuntu) mod_jk/1.2.37 PHP/5.5.9-1ubuntu4.11 OpenSSL/1.0.1f

Figura 266. Patrón de transformación de mod_rewrite no válido

El primer ejemplo contempla un acceso directo a la página de destino real que sirve la petición, es decir, no usamos `mod_rewrite` porque la petición no encaja con el patrón y vamos directamente a la página sustituida. En este caso estamos permitiendo que los usuarios que conozcan esa página puedan acceder a ella revelando detalles técnicos de la web, pero posteriormente veremos cómo evitarlo. El segundo ejemplo es un ejemplo claro de cómo usar `mod_rewrite`, funcionando correctamente para ocultar la tecnología de desarrollo de dicha web y otros detalles técnicos. El tercero y último usa una URI muy similar a la esperada, pero que no encaja con el patrón. No se realiza una sustitución y se trata de resolver la

petición tal cual ha llegado, dando un error por no existir la página solicitada. Con esto queda clara la forma general de funcionar de `mod_rewrite`.

Cómo se ejecuta *mod_rewrite*

Aunque inicialmente puede pensarse que `mod_rewrite` usa toda la URI de la petición para buscar si un patrón encaja o no, en realidad no es así, puesto que usa una parte muy específica de la petición. No obstante eso no quiere decir que el resto de la URI no se pueda utilizar, ya que estas otras partes están disponibles en forma de variables utilizables a la hora de escribir reglas de `mod_rewrite`. No obstante, entender esto es algo complejo si no se enfoca correctamente, por lo que vamos a hacer un seguimiento paso a paso de cómo se descompone la petición:

- Se hace una petición a la URI www.miweb.com/verUsuario.aspx?userId=juan
- Esta petición se transforma en una petición HTTP como esta: `GET /verUsuario.aspx?userId=juan HTTP/1.1`. Esta cadena está disponible en la variable `%{THE_REQUEST}`
- `GET` es el método HTTP usado para la petición, y está disponible en la variable `%{REQUEST_METHOD}`
- El recurso pedido es la página `verUsuario.aspx`, disponible en la variable `%{REQUEST_URI}`
- Luego viene la parte de la *querystring* (los parámetros), disponible en la variable `%{QUERY_STRING}`
- El host al que se hizo la petición (`www.miweb.com`) se guarda en la variable `%{HTTP_HOST}`
- El protocolo (Ej.: HTTP/1.1) está disponible en `%{REQUEST_PROTOCOL}`
  Finalmente, el nombre del fichero pedido se guarda en la variable `%{REQUEST_FILENAME}`

Para entender la petición que veremos a continuación es importante recordar que estas variables, una vez fijadas, **no cambian de valor durante todo el proceso de examen de reglas de** `mod_rewrite`. También es importante tener presente que las variables listadas aquí **no son las únicas** que admite `mod_rewrite`, aunque si las más importantes. Realmente hay muchas más que podemos consultar aquí: http://www.askApache 2.com/htaccess/mod_rewrite-variables-cheatsheet.html#HTTP_HOST

Por tanto, ¿con qué parte de la URI trabaja `mod_rewrite` para hacer encajes de patrones? El *mod* trabaja con la parte referente a la URI (***URI part***), es decir, la parte que está entre el nombre del *host* y el comienzo de los parámetros (*querystring*). Por ejemplo:

- www.miweb.com/verUsuario/juan/ se transformaría en `GET /verUsuario/juan/ HTTP/1.1` y `mod_rewrite` encajaría los patrones de expresiones regulares con `/verUsuario/juan/`
- www.miweb.com/verUsuario.php?id=juan se transformaría en `GET /verUsuario.php?id=juan HTTP/1.1` y `mod_rewrite` encajaría los patrones de expresiones regulares con `/verUsuario.php`

No obstante, el significado de la parte de la URI que se encaja puede cambiar en función del contenedor de directivas que se esté usando para las reglas de `mod_rewrite`:

- Si es en un contenedor `<VirtualHost>`, el patrón a encajar se comprobará contra la URI, como hemos visto, incluyendo el primer "/" (ejemplo: `/miperfil/verPerfil.html`)
- Sin embargo, si el patrón se encuentra en un contenedor `<Directory>` o `.htaccess`, el patrón se encajará con una ruta del sistema de ficheros, eliminando el primer "/" (ejemplo: `miperfil/verPerfil.html` o sólo `verPerfil.html`, dependiendo de dónde se hayan definido las directivas). Esto es importante recordarlo porque puede ser que la misma expresión regular que funcione en un contenedor `<VirtualHost>` no lo hiciese en un `.htaccess`. Conviene por otro lado

recordar que el carácter `?` se puede usar para indicar que un carácter es opcional (en este caso sería `/?`).

- Si habilitado `mod_rewrite` en un contenedor `<Directory>` o `.htaccess` con la directiva `RewriteEngine on` no se estuviesen ejecutando las sustituciones, se puede probar a añadir `Options FollowSymLinks` para ver si así lo hace.
- Si se usa `mod_rewrite` en un fichero `.htaccess`, se puede usar la directiva `RewriteBase` para añadir un prefijo a la URI resultante de hacer la sustitución, si la sustitución resultante devolviese una ruta no válida debido al uso de un `Alias` o similar. Más información: http://httpd.apache.org/docs/current/mod/mod_rewrite.html#rewritebase
- El uso de `mod_rewrite` no está permitido en contenedores `<Location>`, `<Files>` o sus equivalentes con expresiones regulares.

Lo que nunca debemos olvidar a la hora de hacer uso de `mod_rewrite` en cualquier localización es poner la directiva `RewriteEngine On` en el contenedor donde lo estemos usando. Esto es muy importante porque de no hacerlo el módulo no funcionará y Apache 2 no dará ningún error visible, por lo que puede ser una circunstancia muy difícil de depurar.

Sintaxis de las principales directivas de mod_rewrite

Las principales directivas de `mod_rewrite` son `RewriteRule` y `RewriteCond`. Examinaremos en detalle la sintaxis de cada una de ellas antes de ver ejemplos que las usen, para así poder comprenderlos correctamente:

```
RewriteRule Patrón Sustitución [Flags]
```

- **Patrón**: Expresión regular que se comprobará contra la ***URI part***
- **Sustitución**: Puede ser una de las siguientes:
  - Cualquier clase de modificación de la *URI Part*
  - Ruta de un archivo
  - Una redirección a una URI alternativa
  - El carácter "-" (cuyo significado es no hacer nada)
- ***Flags***: Parte opcional. Modifican el significado de la regla de múltiples maneras. Los *flags* admitidos en `mod_rewrite` son muchos y podemos consultarlos en este enlace:
- http://httpd.apache.org/docs/current/rewrite/flags.html

```
RewriteCond Cadena Condición [Flags]
```

- **Cadena**: Normalmente cualquier variable del servidor de las vistas `%{<variable>}`
- **Condición**: Puede ser:
  - Una expresión regular ○ Una cadena con la que comparar ○ Una comprobación de ruta
- ***Flags***: Opcional. En este caso puede tomar estos valores:
  - `NC`: la comparación realizada en la comparación no es sensible a mayúsculas/minúsculas. ○ `OR`: o lógico, al resultado de evaluar esta condición se le hace un OR lógico con el del resto de condiciones a la hora de evaluar si una determinada regla se aplica o no (ver la siguiente sección para entender cómo se procesa una regla de `mod_rewrite`)
  - `NV`: Si se usa una cabecera HTTP en la condición de `RewriteCond`, impide que esta cabecera se añada a la cabecera HTTP Vary de la respuesta HTTP. La cabecera Vary es una cabecera

HTTP especial que indica qué cabeceras pueden influenciar en la respuesta de un determinado servidor: http://tools.ietf.org/html/rfc7231#section-7.1.4. Si se usa mal, puede dar lugar a problemas a la hora de interpretar los contenidos que envía el servidor.

- Para entender mejor este aspecto, se recomienda consultar: https://www.fastly.com/blog/best-practices-for-using-the-vary-header/

Se pueden usar varios flags separados por comas si es necesario para lo que queremos hacer, pero NUNCA SE PUEDE PONER UN ESPACIO EN BLANCO ENTRE FLAGS. Algo como [R, L] dará un error de configuración, sin embargo [R,L] no.

Evaluación de las directivas

En este apartado vamos a destacar cómo se gestiona una cadena de reglas de `mod_rewrite`. Las reglas se evalúan **en el orden en el que se especifican en el contenedor de recursos que requiere su uso**. Cuando una regla transforma la *URI part* de la petición, entonces la *URI part* transformada se pasa a la siguiente regla, y así sucesivamente. Dentro de lo que es la evaluación de cada regla, las condiciones de estas tienen un papel especialmente importante. Antes de explicar en detalle todo esto, conviene hacernos una idea del proceso general, ilustrado en la siguiente imagen:

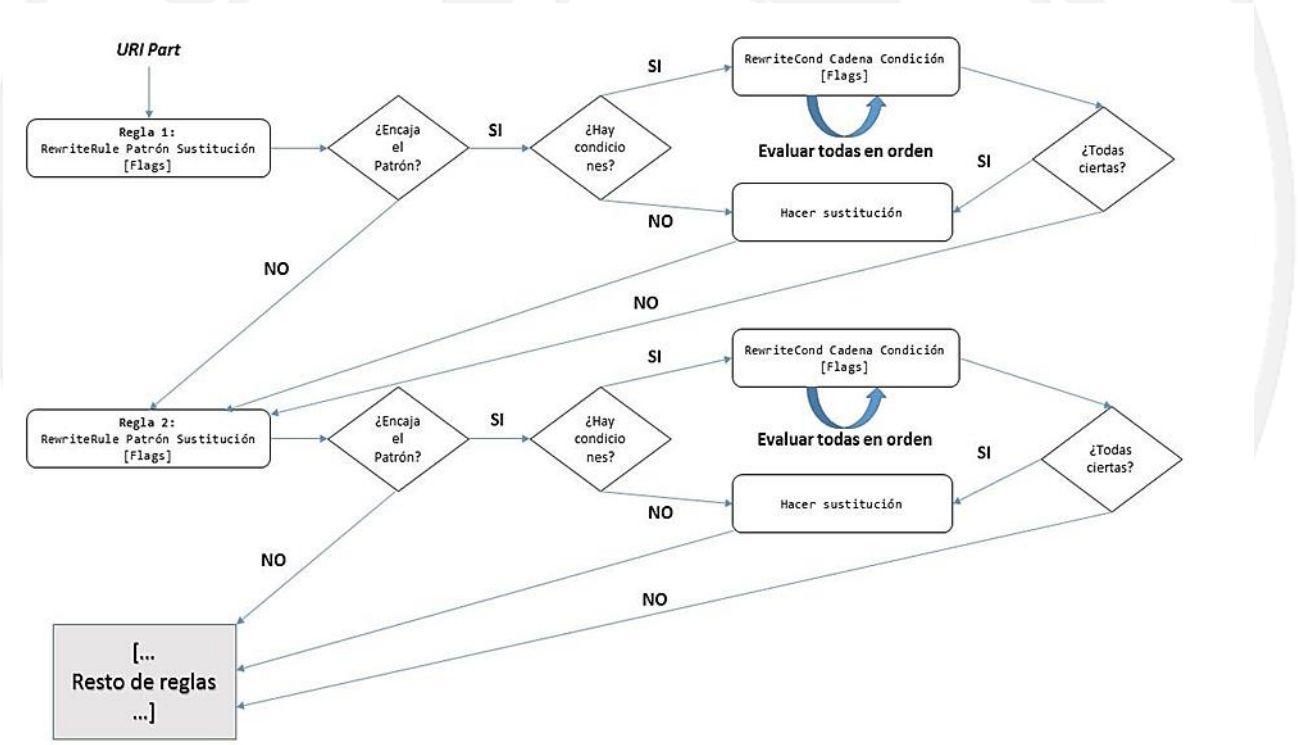


Figura 267. Procedimiento de evaluación de las reglas de mod_rewrite

Dentro del proceso descrito en el diagrama anterior cobra especial importancia la ya mencionada directiva `RewriteCond`. Cada `RewriteCond` se asocia con una sola `RewriteRule` y aparecen escritas en el fichero antes de la regla con la que está asociada (justo encima). A pesar de esto, y como se ve en el diagrama, solo se evalúa la condición si el patrón de la regla asociada encaja, que es lo primero que el *mod* examina. Las condiciones son opcionales. Como también puede apreciarse en el diagrama, si no hay ninguna y el patrón de la regla encaja, se hace la sustitución asociada. Veamos todo este proceso con el siguiente ejemplo:

```
...
```

```
RewriteEngine on
RewriteRule ^tramitarPedido/(\d+)/(\w+)/cesta.htm$ procesar_compra.php?pedido=$1;id=$2

RewriteCond %{THE_REQUEST} procesar_compra\.php
RewriteRule ^procesar_compra\.php - [F] …
```

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf          Modificado


<Directory "/var/www/html/compras">
RewriteEngine on
RewriteRule ^tramitarPedido/(\d+)/(\w+)/cesta.htm$ procesar_compra.php?pedido=$$

RewriteCond %{THE_REQUESTç procesar_compra\.php
RewriteRule ^procesar_compra\.php - [F]
</Directory>
```

Figura 268. Añadimos una condición a la regla anterior

La primera regla no tiene condición y se evalúa como la que hemos visto anteriormente. La segunda si la tiene, y, como veremos enseguida, impide el acceso directo a la página `.php`. *¿Por qué esta prohibición?* Para entenderla, debe tenerse en cuenta que hemos hecho los ejemplos anteriores con la intención de no dar a conocer detalles internos de la implementación de nuestra página (entre otros detalles, que está hecha en `.php`). Si dejamos acceder directamente a las páginas PHP, entonces a función de `mod_rewrite` se limitaría simplemente a tener una segunda forma de acceso a una funcionalidad. Y no queremos que haya una alternativa, queremos que esa sea **LA FORMA** de acceder a la funcionalidad. Veremos la ejecución de este proceso en detalle:

- Una petición como www.miweb.com/compras/tramitarPedido/1236785/usuario1/cesta.htm haría que la variable `THE_REQUEST` valiese `GET /compras/tramitarPedido/1236785/usuario1/cesta.htm HTTP/1.1`. Ante la **primera regla** que `mod_rewrite` se encuentra para evaluar vemos que:
  - La *URI part* encaja con el patrón
  - No tiene condiciones, por lo que se hace la sustitución y se avanza a la siguiente regla.
- Pasada la primera evaluación de reglas, vemos que la URI resultante ha cambiado a `/compras/procesar_compra.php?pedido=1236785&id=usuario`. Al ser esta la URI que llega a la siguiente regla, se calcula la nueva *URI Part*, donde se pierde la parte de los parámetros, siendo: `/compras/micestadelacompra.php`. Hecho eso, se evalúa la petición de nuevo con la **segunda regla**:
  - La *URI part* encaja con el patrón
  - Hay una condición establecida, así que debe evaluarse
  - En la condición se comprueba que la variable `THE_REQUEST` no contiene una petición a ninguna página `procesar_compra.php` (se debe recordar que las variables, una vez fijado su valor al comienzo de la evaluación de reglas de `mod_rewrite`, no cambian de valor, y la petición original (www.miweb.com/compras/tramitarPedido/1236785/usuario1/cesta.htm) no la tiene.
  - La condición falla, por lo que no se hace la sustitución especificada en esta segunda regla ni se aplica ningún *flag* que pudiese tener, resultando en que la URI no se modifica y la petición no se prohíbe. A efectos prácticos, se sirve la petición con la página `.php` y los parámetros dados, que fueron el resultado de la primera regla que si se ejecutó.

¿Qué pasaría ahora con las misma reglas anteriores si intentamos acceder directamente a la página `procesar_compra.php`*, incluso con parámetros con valores correctos?* (www.miweb.com/compras/procesar_compra.php?pedido=1236785;id=usuario1)

- **Primera regla**:
  - No encaja con el patrón, por lo que no se evalúa nada más ni se cambia nada en la petición
- **Segunda regla**:
  - Encaja con el patrón
  - Tiene una condición, que se evalúa
  - La condición se cumple
  - Al cumplirse se realiza la sustitución y se ejecutan los *flags*.
  - La sustitución "`-`" es especial y su significado es el de no cambiar nada, como vimos anteriormente.
  - En este caso el *flag* es `[F]`, cuyo significado es "`Forbidden`". La petición se prohíbe por tanto, y de esta manera también prohibimos cualquier acceso directo a la página `.php` en ese directorio. El resultado es una página de error `403 Forbidden`

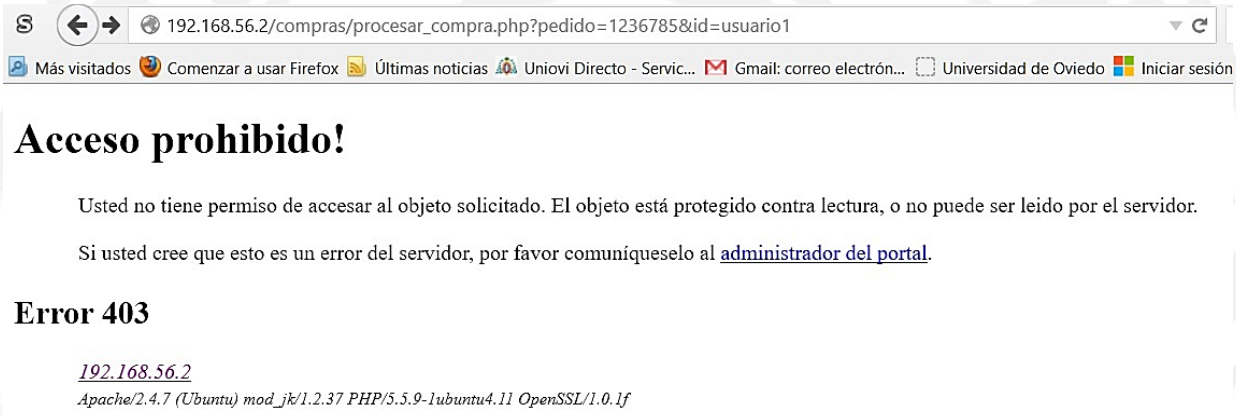


Figura 269. Impidiendo el acceso directo a la página php gracias a mod_rewrite

Finalmente, es importante recordar que para que una regla gestionada por `mod_rewrite` sea posible y se haga su sustitución, todas las condiciones aplicables sobre la misma deben cumplirse, salvo que se use el *flag* OR, en cuyo caso basta con que una de ellas lo haga.

Ejemplos comunes de uso

A continuación veremos **una serie de ejemplos de uso** de `mod_rewrite` para reforzar lo aprendido. Si bien se presentan varios casos típicos de uso, es probable que tengamos que desarrollar uno propio adaptado a un sitio web. Ante esa tesitura, se recomienda que se empiece con una regla simple que haga solo una parte de lo pretendido, se pruebe y, si es correcta, se continúe añadiéndole más funcionalidad. Esto es igualmente recomendable para `RewriteCond`. Nunca es recomendable, salvo que se cuente con una gran experiencia en el uso del *mod*, "lanzarse" realizar una regla compleja desde el principio. Esto es especialmente importante si se usan expresiones regulares, donde es fácil cometer algún error.

No obstante, tampoco tenemos que hacer todo "a mano". **Esta página**: https://regex101.com/#pcre **permite probar expresiones regulares contra posibles peticiones que recibiremos** y decidir si encajan o no, con lo que podemos hacer un buen *testing* de si la expresión regular filtra lo que buscamos. No obstante, conviene recordar que en esta página web el carácter `/` se considera de control, por lo que si

necesitamos evaluar literalmente la presencia de un `/` deberemos cambiarlo por `\/`, aunque luego en `mod_rewrite` se permita usar directamente `/`.

Eliminar www de la URI

Esta regla convierte cualquier petición a www.miweb.com a http://miweb.com, ejecutando una redirección en el navegador del cliente (con lo que este verá la URI final).

```
RewriteEngine on
RewriteCond %{HTTP_HOST} ^www\.miweb\.com$ [NC]
RewriteRule ^(.*)$ http://miweb.com/$1 [R=301,L]
```

La evaluación ante una petición como `www.miweb.com/usuarios/verPerfil.aspx` sería:

- `mod_rewrite` extrae la *URI part*, que sería `/usuarios/verPerfil.aspx`
- Se salva todo el contenido de la *URI part* en $1
- Se evalúa pues la condición que la regla tiene, comprobando que si la variable `HTTP_HOST` (el host al que se realizó la petición HTTP, en nuestro caso www.miweb.com) comienza efectivamente por www. La condición tiene como *flag* hacer comparaciones no sensibles a mayúsculas (NC)
- Como la condición encaja, se hace la sustitución y se evalúan los *flags*:
  - La sustitución antepone http://miweb.com a la *URI part*, quedando pues http://miweb.com/usuarios/verPerfil.aspx
  - El *flag* `[R=301]` hace una redirección permanente (código HTTP 301) a la URI resultante de la sustitución.
  - Finalmente, el *flag* `[L]` indica a `mod_rewrite` que esta es la última regla a examinar. Nótese que la condición no se evalúa como cierta si la petición ya fuse http://miweb.com, por lo que no se hace la sustitución de la regla

Evitar el *hotlinking*

El *hotlinking* es una técnica muy común en Internet, consistente **en que una web carga recursos de otra web** (imágenes, sonidos, etc.) usando su URI, de manera que se muestren en la misma. De esta manera www.printcasa.com puede ser una página que contenga una galería de imágenes pero que ninguna de ella esté alojada en www.printcasa.com, sino en otros dominios a los que se accede (`<img src=www.miweb.com/usuarios/usuario1/fotoPerfil.jpg>`).

Esto significa que www.miweb.com está siendo accedida para cargar la galería de imágenes, se consume ancho de banda asignado a la misma y para colmo realmente no se está accediendo a la página www.miweb.com, sino a parte de su contenido, desde otra web. Dicho de otro modo, **nos están consumiendo ancho de banda y no obtenemos a cambio ninguna visita efectiva a la página**. Dado que hoy en día, y más en escenarios de hosting en *cloud*, se cobra por uso y las visitas registradas pueden ser parte imprescindible para mantener la viabilidad económica de la página, este tipo de prácticas tan comunes deben intentar prohibirse.

La práctica más común para evitar el *hotlinking* es listar una serie de dominios desde los que efectivamente los permitimos y bloquear al resto (**una lista de permitidos**). Para saber desde que dominio ha llegado una petición, se puede comprobar el valor de la variable `HTTP_REFERER`. No obstante, esta es una cabecera cuyo valor se asigna por el navegador en el cliente, por lo que éste, con herramientas adecuadas como un *proxy HTTP*, podría modificarla a su antojo. No obstante, la técnica que vamos a ver

evitará la mayor parte del *hotlinking*, puesto que falsificar el `Referer` es demasiado trabajo solo para hacer *hotlinking* de unos recursos frente a directamente descargárselos y alojarlos en la web que los pide (usando entonces su propio ancho de banda). Para el siguiente ejemplo supondremos que solo la web www.miweb.org puede hacer *hotlinking* de imágenes, videos y otros recursos variados de la web www.miweb.com. En caso contrario, se mostrará un error `HTTP 403 Forbidden`. Las reglas que evitarían el *hotlinking* bajo estas condiciones son las siguientes:

```
RewriteEngine on

RewriteCond %{HTTP_REFERER} !^http://www.miweb\.org/?.*$ [NC]
#Si quisiésemos más dominios autorizados haríamos más RewriteCond como el anterior aquí

RewriteRule \.(ico|pdf|flv|jpe?g|mp3|mpg|mp4|mov|wav|wmv|png|gif|swf|css|js)$ - [F,NC]
```

Una petición de un recurso "sensible" desde www.miweb.org haría que la condición de la regla fallase (nótese que tiene un `!` al principio), por lo que no habría sustitución. En caso de que el dominio no fuese ese, y la petición fuese de uno de los tipos de ficheros listados, la sustitución (no hacer nada) se realizaría, pero el *flag* `[F]` devolvería el HTTP `403 Forbidden` que buscamos. A modo de ejemplo veremos qué ocurre con nuestra infraestructura de pruebas si se intenta cargar desde una fuente externa la imagen que aparece en la siguiente web de pruebas que hemos creado:

Figura 270. Web de pruebas con una imagen

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf

<Directory "/var/www/html/imgs">
        RewriteEngine on
        RewriteCond %{HTTP_REFERER} !^http://192.168.56.2/?.*$ [NC]

        RewriteRule \.(gif|jpg|png)$ - [F,NC]
</Directory>
```

Figura 271. Configuración anti-hotlinking de un directorio que guarda imágenes

Acceso prohibido!

Usted no tiene permiso de accesar al objeto solicitado. El objeto está protegido contra lectura, o no puede ser leido por el servidor.

Si usted cree que esto es un error del servidor, por favor comuníqueselo al administrador del portal.

Error 403

192.168.56.2
Apache/2.4.7 (Ubuntu) mod_jk/1.2.37 PHP/5.5.9-1ubuntu4.11 OpenSSL/1.0.1f

Figura 272. Acceso directo a una imagen sin usar una web que la carga prohibido

Una variante común para evitar el *hotlinking* es **mostrar una imagen alternativa** que normalmente avisa a la web de que el *hotlinking* de contenido no está admitido. Seguimos gastando nuestro propio ancho de banda, pero damos un aviso a quien nos haga *hotlinking* de que no debería seguir intentándolo. Las modificaciones a la regla anterior son fáciles:

```
RewriteEngine on

RewriteCond %{HTTP_REFERER} !^http://www.miweb\.org/?.*$ [NC]
#Si quisiésemos más dominios autorizados haríamos más RewriteCond como el anterior aquí
RewriteRule \.(gif|jpe?g|png|bmp|flv|swf)$ http://www.imagenes.com/nohotlinking.gif [R,NC]
```

Es importante comprobar como ahora en lugar de `[F]` (prohibido) hemos usado `[R]` (redirección a la URI sustituida, que contiene la imagen). También es extremadamente importante que la imagen a cargar en puesto de la que se solicita no esté en la misma localización que está protegida contra *hotlinking*, puesto que entonces al intentar cargar una imagen de ella se volverían a aplicar las reglas de `mod_rewrite`, **formándose un bucle infinito de carga** que el propio navegador detectará y nos avisará. En nuestro ejemplo la configuración quedaría de la siguiente forma, usando un host virtual distinto para cargar la imagen *anti-hotlinking*:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf


<Directory "/var/www/html/imgs">
        RewriteEngine on
        RewriteCond %{HTTP_REFERER} !^http://192.168.56.2/?.*$ [NC]

   RewriteRule \.(gif|jpg|png)$ http://192.168.56.3/nohotlinking.png [R,NC]
</Directory>
```

Figura 273. Carga de una imagen alternativa desde una localización distinta

Ahora una petición como esta:

Figura 274. Petición directa de una imagen

Haría una redirección a la imagen alternativa:

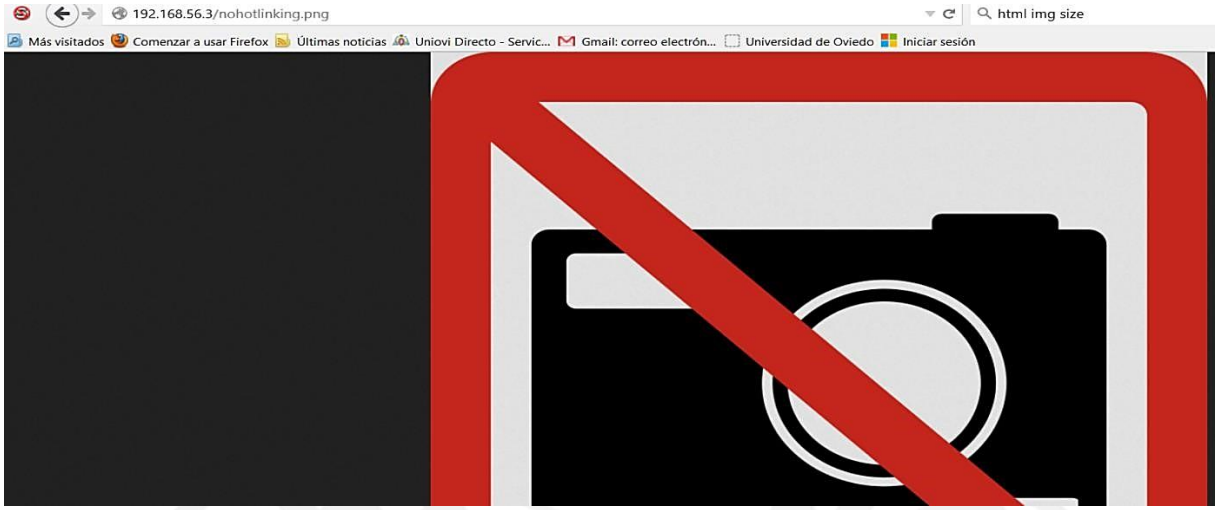


Figura 275. Imagen alternativa para hotlinking cuando se intenta acceder directamente a una imagen

Nótese que incluso si la página que referencia la imagen pertenece al mismo servidor también ocurre esto, solo se está mirando el `Referer` para determinar si hay o no *hotlinking*. Si por ejemplo creamos esta web en un host virtual distinto al que hemos protegido contra *hotlinking*:

```
 GNU nano 2.2.6          Archivo: index.html                Modificado

<html>
<head>
<title>Prueba de web 1</title>
</head>
<body>
<p> Prueba de pagina HTML estatica con imagen</p>
<img src="http://192.168.56.2/imgs/box.png" height="250" width="200">
</body>
</html>
```

Figura 276. Web que intenta hacer hotlinking

Al cargarla nos ocurriría esto:

Figura 277. Hotlinking de una página web evitado

Este mecanismo se puede hacer mucho más elaborado si en lugar de una imagen se usa un *script* en el servidor que genere una imagen personalizada para el cliente que intenta cargarlas u otro tipo de respuesta más compleja.

Errores personalizados

Como ya hemos dicho, los errores web son una potencial fuente de **problemas de seguridad cuando dan demasiada información acerca de la página**, tecnologías, productos, etc. Por ese motivo, es frecuente que se desarrollen páginas de error personalizadas, para estar seguros de que esos errores solo dan la información debida.

Además del propio mecanismo para crear páginas de error personalizadas que tiene el servidor (que ya hemos visto), como alternativa s**e puede usar `mod_rewrite` para conseguir mostrar la página que uno quiera** cuando se hace una petición sobre un fichero o directorio que no existe, aunque esto es solo uno de los posibles errores HTTP (el 404) de todos los que se podrían producir. Nótese que esto no sólo se puede usar para mostrar una página de error, a lo mejor interesa redirigirlo a la página principal o cualquier otra opción aceptable en la web concreta. Para hacer esto, las reglas serían algo así:

```
RewriteEngine on

RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule .* 404_divertido.html [L]
```

Las dos condiciones de la regla comprueban que la variable `REQUEST_FILENAME`, que como hemos visto guarda el nombre del fichero solicitado, no sea una fichero (`!-f`) o un directorio (`!-d`) conocidos. En ese caso, al cumplirse ambas condiciones, se ejecutará la sustitución de la regla que cambia cualquier petición por la página `404_divertido.html` que hemos creado al efecto. El *flag* `[L]`, como indicamos anteriormente, hace que no se procesen más reglas que pudieran ser aplicables (no tiene sentido ya que la petición es errónea).

También ha de tenerse en cuenta que en este caso no hay redirección como en el anterior, directamente se carga la página `404_divertido.html` que habremos creado previamente. En nuestra infraestructura de ejemplo podemos aplicar esta configuración sobre el directorio compras (**OJO**: Si ya tuviéramos directivas sobre ese directorio anteriores, conviene poner estas nuevas directivas dentro del contenedor para ese directorio que ya tuviéramos, o la configuración del contenedor que aparece más tarde en el fichero sobrescribirá a la del primero, aunque no tengan directivas iguales):

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf

<Directory "/var/www/html/compras">
        RewriteEngine On
        RewriteCond %{REQUEST_FILENAME} !-f
        RewriteCond %{REQUEST_FILENAME} !-d

        RewriteRule .* 404_divertido.html [L]
</Directory>
```

Figura 278. Error 404 personalizado con mod_rewrite

Y si accedemos a una URI inexistente ocurre esto:

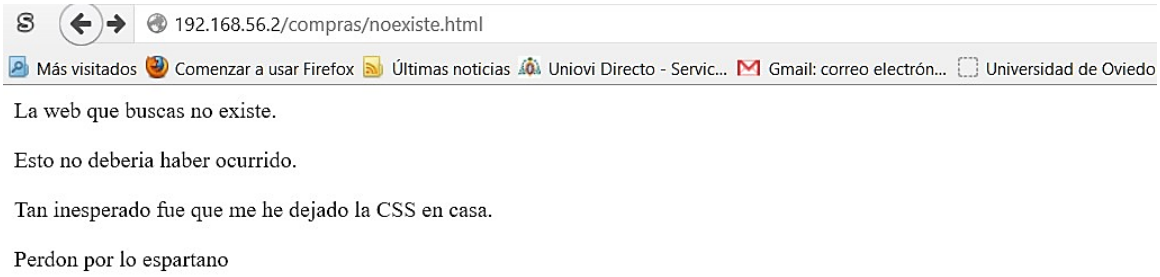


Figura 279. Error 404 creado con mod_rewrite

Mod_rewrite condicional

Es posible que si no tenemos control sobre todo el servidor donde hacemos *hosting* de una página no podamos saber si `mod_rewrite` está activo o no y, lo que es peor, no podamos solicitar su activación. Si introducimos reglas sin tener la completa seguridad de que el *mod* está activo entonces veremos un error `500 Internal Server Error` en caso de que no lo esté, ya que *Apache 2* no será capaz de saber qué módulo procesa las directivas que hemos puesto. Si podemos prescindir de las reglas de `mod_rewrite` para poder proporcionar la funcionalidad de la página, entonces podemos activarlas condicionalmente gracias a la directiva `IfModule`. El esquema a seguir sería algo como esto:

```
<IfModule mod_rewrite.c>
RewriteEngine on  #Reglas  ...
</IfModule>
```

Protección de un directorio

Evitar el acceso a determinados ficheros de un directorio dado con un `403 Forbidden` es muy sencillo de acuerdo con lo que ya hemos visto. El siguiente ejemplo impide el acceso directo a páginas HTML, ficheros de texto y ficheros PHP en un directorio de la web llamado `directorio_protegido`:

```
RewriteEngine on

RewriteCond %{THE_REQUEST} ^/directorio_protegido/.*$ [NC]
RewriteCond %{REQUEST_FILENAME} ^.+\.(php|html|htm|txt)$  RewriteRule .* - [F,NS,L]
```

Podemos implementar algo similar en nuestra infraestructura de pruebas de la siguiente forma:

```
<Location "/compras">
        RewriteEngine On

        RewriteCond %{THE_REQUEST} ^.*/directorio_protegido/?.*$ [NC]
        RewriteCond %{REQUEST_FILENAME} ^.+\.(php|txt)$
        RewriteRule .* - [F,NS,L]
</Location>
```

Figura 280. Restricción de acceso a tipos de fichero en una localización

Con esto, cualquier URI a una localización dependiente de compras será examinada en busca de si accede a `directorio_protegido` dentro de la misma. Si lo hace, se comprobará si el fichero pedido es un PHP o un `txt`. En ese caso, se prohibirá la petición:

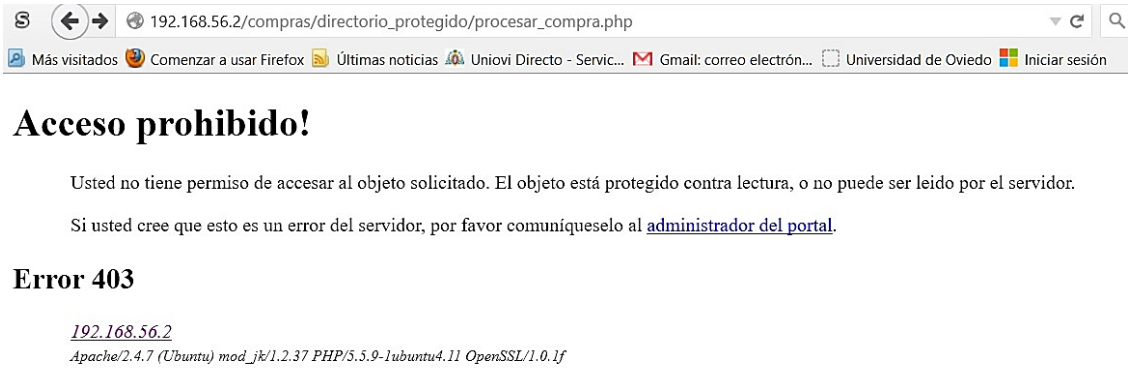


Figura 281. Acceso restringido

Protección contra *exploits* comunes

Los siguientes ejemplos están pasados para prohibir peticiones maliciosas conocidas:

```
RewriteEngine On
RewriteCond %{REQUEST_URI} !^/(login.php|admin/|debug/|private/).* [NC]
RewriteCond %{THE_REQUEST} ^[A-Z]{3,7}\ ///.*\ HTTP/ [NC,OR]
RewriteCond %{THE_REQUEST} ^[A-Z]{3,7}\ /.*\?\=?(http|ftp|ssl|https):/.*\ HTTP/ [NC,OR]
RewriteCond %{THE_REQUEST} ^[A-Z]{3,7}\ /.*\?\?.*\ HTTP/ [NC,OR]
RewriteCond %{THE_REQUEST} ^[A-Z]{3,7}\ /.*\.(aspx|ini|dll|exe|jar|pyc).*\ HTTP/ [NC,OR]
RewriteCond %{THE_REQUEST} ^[A-Z]{3,7}\ /.*\.(htpasswd|htaccess|aahtpasswd).*\ HTTP/ [NC]
RewriteRule .* - [F,NS,L]
```

Esto ejemplos suponen que:

- Las URI que contengan `login.php`, `admin`, `debug` o `private` no deben ser accedidas directamente. Nótese que estos nombres son sólo ejemplos, y deben adaptarse a cada caso.
- Como dijimos, la variable `THE_REQUEST` contiene la petición HTTP. La sintaxis de una petición HTTP es la siguiente, y debemos tenerla muy presente a la hora de entender por qué el patrón que trata con esta variable de entorno se crea de esa forma:

```
<método HTTP>[espacio en blanco]/<URI>[espacio en blanco]HTTP/<versión de HTTP>
Ejemplo: GET /privado HTTP/1.1
```

- Toda petición por tanto comenzará por el método `HTTP`. El método está compuesto por caracteres en mayúsculas, y ese es el motivo de la presencia del `[A-Z]` inicial en los patrones. Otro aspecto es entender por qué se pone un tamaño mínimo y uno máximo (`{3-7}`). El motivo es que cualquier método HTTP válido tendrá que estar formado por una palabra de entre 3 y 7 caracteres en mayúsculas (el método `GET`, `POST`, `OPTIONS`,…). No obstante, tenemos que recordar que **en ocasiones hay métodos que extienden el protocolo HTTP y que no son los habituales** (como `PROPFIND` o `BPROPFIND` del protocolo *WebDAV*), por lo que quizá tengamos que aumentar el límite superior algo más en caso de usarlos. En todo caso podemos suponer que entre 3 y 9 caracteres podrán encajar todos los posibles métodos HTTP que nos lleguen.
- El espacio en blanco se modela con `\<espacio en blanco>` (`\` es el carácter que indica que el siguiente carácter debe ser interpretado literalmente)
- Toda petición HTTP acaba en `HTTP/<versión>`. Aunque en estos ejemplos la versión no nos interesa.

- Debe tenerse en cuenta que si solo se admitiesen métodos `GET` podría limitarse la expresión regular a un máximo de 3 caracteres, `GET` y `POST` a 4, etc., para una mayor seguridad.
- También es cierto que si solo admitiésemos peticiones `HTTP/1.1` podríamos indicarlo al final de los patrones.

Dicho esto, ya podemos explicar los ejemplos de esta sección. Nótese que todos ellos están enlazados con un OR, por lo que basta con que uno de ellos se cumpla para que la petición se considere maliciosa y se prohíba:

- **El primer ejemplo** es sencillo y sólo prohíbe el acceso a las URI "privadas" que mencionamos.
- **En el segundo ejemplo** se evita que se permitan los caracteres `///` en la petición (`GET ///proveedor.html HTTP/1.1`), al ser una petición mal formada con posibles fines maliciosos.
- **El tercer ejemplo** evita peticiones como `GET /provider.html?http://www.evilweb.com HTTP/1.1` o `GET /provider.html?=http://www.evilweb.com HTTP/1.1`, que pueden dar lugar a comportamientos inesperados al introducir una URI extra en la petición que podría procesarse por la aplicación. La variante `RewriteCond %{THE_REQUEST} ^[A-Z]{3,7}\/.*\?.*\=(http|ftp|ssl|https):/.*\ HTTP/ [NC,OR]` obligaría a que las URI tuvieran un parámetro con nombre (`GET /provider.html?cualquiernombre=http://www.evilweb.com HTTP/`)
- **El cuarto ejemplo** prohíbe la presencia de dos `??` en la petición, al estar mal formada: `GET /provider.html??a=http://www.evilweb.com HTTP/1.1`
- **El quinto ejemplo** prohíbe el acceso a tipos de fichero cuyo acceso directo no es admisible y por alguna razón no se han podido eliminar del directorio donde se ha desplegado la web. Ejemplo: `GET /format.exe HTTP/1.1`
- **El sexto** y último ejemplo prohíbe el acceso a ficheros del sistema conocidos (como los `.htaccess`). Ejemplo: `GET /user/.htaccess HTTP/1.1`

Aunque esto son solo unos ejemplos, es fácil darse cuenta de que hacer una lista exhaustiva de todas las posibles peticiones maliciosas es una labor larga y complicada. En general, tiene más sentido instalar `mod_security`, que posee estos y muchos más filtros contra potenciales peticiones maliciosas, que intentar hacer nosotros algo así manualmente, ya que nunca conseguiremos igualar la potencia de dicho módulo. No obstante, está bien ver este tipo de acciones como un ejemplo adicional del módulo, y también por si por alguna razón tuviéramos que implementar alguna restricción concreta en alguno de nuestros directorios que `mod_security` no cubriera. Por ejemplo, podríamos optar por admitir solo `HTTP 1.1` y, sabiendo que nuestra aplicación solo admite `GET` y `POST`, limitar todas las peticiones a métodos HTTP de 3 o 4 caracteres, para complementar la labor de `mod_security`.

Admitir solo unos métodos concretos

Además de la aproximación vista anteriormente, podemos usar esta para admitir solo peticiones `GET` y `POST`, dando un error de `Forbidden` en caso contrario:

```
RewriteEngine On

RewriteCond %{REQUEST_METHOD} !^(GET|POST)$ [NC]  RewriteRule .* - [F,NS,L]
```

Prohibir el uso de proxys

Este ejemplo deniega toda petición `POST` que provenga de un proxy, lo que puede saberse por la presencia de determinadas cabeceras en la petición:

```
RewriteEngine On

RewriteCond %{REQUEST_METHOD} =POST

RewriteCond %{HTTP:VIA}%{HTTP:FORWARDED}%{HTTP:USERAGENT_VIA}%{HTTP:X_FORWARDED_FOR} %{HTT
P:PROXY_CONNECTION} !^$ [OR]

RewriteCond %{HTTP:XPROXY_CONNECTION}%{HTTP:HTTP_PC_REMOTE_ADDR} %{HTTP:HTTP_CLIENT_IP} !^
$
RewriteRule .* - [F,NS,L]
```

Prohibir versiones del protocolo HTTP

Con estas reglas solo se admiten peticiones de los protocolos `HTTP/1.0` y `HTTP/1.1`

```
RewriteEngine On

RewriteCond %{THE_REQUEST} !^[A-Z]{3,9}\ .+\ HTTP/(1\.0|1\.1) [NC] RewriteRule .* -
[F,NS,L]
```

Limitar los caracteres de las URI de las peticiones

Es posible que por la construcción de las peticiones de una web concreta las URIs válidas tengan un conjunto limitado de caracteres. Si somos capaces de delimitarlos, podemos usar `mod_rewrite` para filtrar URIs que sabemos que son erróneas por contener caracteres que no aparecerán en ninguna URI válida. La siguiente petición solo admite URIs que estén compuestas por letras, números y los caracteres `+`, `.`, `?`, `/`, `-`, `_`, `=` y `&`.

```
RewriteEngine On

RewriteCond %{THE_REQUEST} !^[A-Z]{3,9}\ [a-zA-Z0-9\.\+_/\-\?\=\&]+\ HTTP/ [NC]
RewriteRule .* - [F,NS,L]
```

Por ejemplo, esto admitiría peticiones HTTP como:

```
GET /user/perfil.php HTTP/1.1
GET /user/mi.perfil.aspx HTTP/1.1
GET /user/mi_perfil.aspx HTTP/1.1
GET /user/mi_perfil.aspx?id=123 HTTP/1.1  GET /user/mi-perfil.aspx?id=123+abc HTTP/1.1
```

No permitir peticiones erróneas

Es común que una web reciba peticiones mal formadas o corruptas con la intención de probar su seguridad. Existe software específico pensado para solventar esta clase de problemas (`mod_security`), pero eso no quiere decir que no podamos usar `mod_rewrite` para parar algunas de ellas. En definitiva, cuantas más capas de seguridad, mejor.

Peticiones sin cabeceras necesarias o con valores inválidos en las mismas

La siguiente petición muestra un error `403 Forbidden` en el caso de que una petición `POST` no tenga la cabecera `Content-Length` o bien su cabecera `Content-Type` no tenga alguno de estos valores: `application/x-www-form-urlencoded, multipart/form-data`

```
RewriteEngine On
```

```
RewriteCond %{REQUEST_METHOD} =POST
RewriteCond %{HTTP:Content-Length} ^$
RewriteCond %{HTTP:Content-Type} !^(application/x-www-form-
urlencoded|multipart/formdata.*)$ [NC]
RewriteRule .* - [F,NS,L]
```

Rechazar peticiones sin `http_host`:

La cabecera `HTTP_HOST` contiene el servidor al que se hace la petición. Si no existe, significa que la petición podría estar hecha por un *bot*, herramienta de escaneo o similar que no elabore peticiones con todas las cabeceras debidas. En todo caso, se deben prohibir por no provenir de un cliente legítimo:

```
RewriteEngine On

RewriteCond %{HTTP_HOST} ^$  RewriteRule .* - [F,NS,L]
```

Errores de cabeceras:

Entre las posibles peticiones corruptas puede haber otras incoherentes, como el caso de una petición a un fichero `.asp` que declare que su `content-type` es una imagen. El siguiente conjunto de reglas prohíbe este tipo de peticiones:

```
RewriteEngine On

RewriteCond %{HTTP:Content-Disposition} \.asp [NC]
RewriteCond %{HTTP:Content-Type} image/.+ [NC]  RewriteRule .* - [F,NS,L]
```

Obviamente esto es un ejemplo muy simple y pueden existir un gran nº de posibilidades en este sentido.

No hay presencia de `user-agent`:

Normalmente cualquier navegador envía un `User-Agent` con información del cliente, mientas que otras herramientas de escaneo de vulnerabilidades o similares pueden no enviar esta cabecera por error. Esto pararía estos clientes, aunque es una medida de seguridad fácil de saltar. Por otro lado, el no enviar `UserAgent` puede ser algo legítimo, ya que existirán clientes que tengan herramientas que incrementen su privacidad como *Privoxy* http://www.privoxy.org/.

```
RewriteEngine On

RewriteCond %{HTTP_USER_AGENT} ^-?$  RewriteRule .* - [F,NS,L]
```

Lo mismo se puede aplicar con la cabecera `Http-Referer`:

```
RewriteEngine On

RewriteCond %{HTTP_REFERER} ^-?$
RewriteRule .* - [F,NS,L]
```

Manipulación de la *query string* (parámetros de la URI):

Denegar la petición si la *querystring* contiene la palabra "admin"

Este es un ejemplo de un caso más general que podría describirse como "*denegar peticiones que contengan un determinado parámetro en su lista de parámetros*". Un ejemplo de cómo hacerlo sobre un directorio determinado sería este:

```
RewriteEngine On

RewriteCond %{QUERY_STRING} admin
RewriteRule ^/aqui_no_hay_admins - [F]
```

En nuestra infraestructura de pruebas quedaría así:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf

<Directory "/var/www/html/pruebas">
        RewriteEngine On

        RewriteCond %{QUERY_STRING} admin
        RewriteRule ^.* - [F]
</Directory>
```

Figura 282. Denegación de peticiones por tener un determinado parámetro

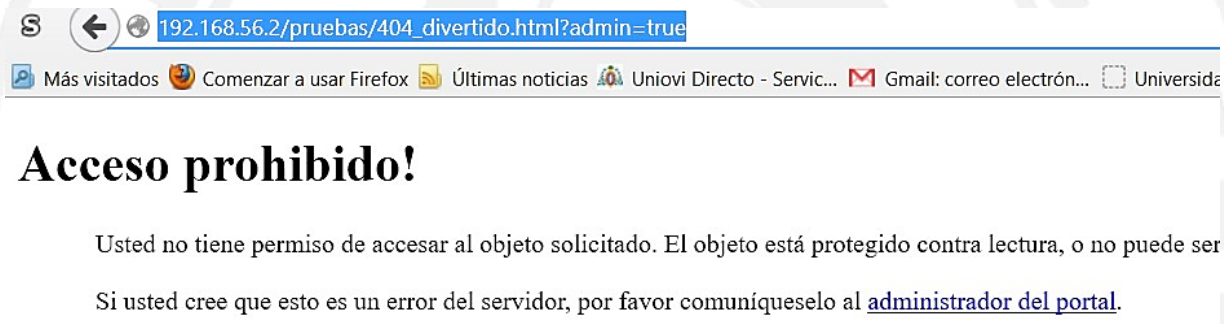


Figura 283. Petición denegada por contener un parámetro admin

Eliminar los parámetros de la uri

En el siguiente ejemplo vemos como eliminar completamente la *querystring* de todas las páginas que estén en una determinada localización, en nuestro caso `/pruebas`:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf

<Directory "/var/www/html/pruebas">
        RewriteEngine On

        RewriteCond %{QUERY_STRING} ^.+=.+$
        RewriteRule (.*)$ http://192.168.56.2/pruebas/$1? [R=301,L]
</Directory>
```

Figura 284. Quitando la querystring

Debemos tener en cuenta lo siguiente:

- La idea es convertir un `http://<mi servidor>/pruebas/<cualquier página>.html?<pares parámetro=valor>` en `http://<mi servidor>/pruebas/<cualquier página>.html`

- El `RewriteCond` es necesario porque detecta que hay *querystring*. Hay que tener en cuenta que si no lo pusiéramos, al hacer la redirección volvería a ejecutarse la regla (la página se recarga para que el cambio sea visible para el usuario), entrando en un bucle infinito de carga.
- Es importante ver que `(.*)$` hace que la página pedida se guarde en la variable `$1`, que luego usamos para hacer la sustitución. Usaremos más estos elementos en los ejemplos siguientes, incluso varios de ellos.
- El `?` final es el que deja la *querystring* vacía.

Añadir un par `parametro=valor` a la *querystring*:

Este ejemplo muestra como añadir un parámetro con un valor predeterminado si este parámetro no existiese en la petición realizada:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf

<Directory "/var/www/html/pruebas">
        RewriteEngine On
        RewriteBase /pruebas/

        RewriteCond %{QUERY_STRING} !parametro=
        RewriteRule ^pagina.html pagina.html?parametro=valor [R,L,QSA]
</Directory>
```

Figura 285. Añadiendo parámetros a la querystring

Cosas a tener en cuenta para entender el ejemplo:

- El `RewriteCond` es necesario por la misma razón mencionada en el ejemplo anterior: Puede dar lugar a recargas infinitas.
- `RewriteBase` nos sirve aquí para que la URI de la página recargada pertenezca al directorio correcto (en caso contrario la URI sería http://192.168.56.2/var/www/html/pruebas/pagina.html, que dará un error)
- `[QSA]` es el *flag* llamado *Query String Append*, que añadirá los valores indicados al principio de esta.

Figura 286. Resultado de añadir un parámetro a la querystring si no lo tenía ya

Cambiar la URI en función del valor de la *query string*:

Esto cambiaría peticiones como esta www.miweb.com/usuario.html?admin=true por www.miweb.com/admin.html?admin=true, pero solo si `admin` es `true`. En otras palabras, redirige la petición a una URI diferente en función de un valor de los parámetros pasados.

```
RewriteEngine On

RewriteCond %{QUERY_STRING} admin=true
RewriteRule ^/usuario.html /admin.html
```

En nuestra infraestructura de pruebas sería algo como lo siguiente:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf          Modificado

<Directory "/var/www/html/pruebas">
        RewriteEngine On

        RewriteCond %{QUERY_STRING} admin=true
        RewriteRule ^pagina.html admin.html
</Directory>
```

Figura 287. Cambio de URI en función de un valor de un parámetro

Figura 288. Resultado del cambio de la URI

Modificación de la *query string*

Este ejemplo cambiaría la primera aparición de "`public`" por "`private`" de la lista de parámetros solo cuando se accede a la página `admin.html`. Hay que tener en cuenta que se usan los indicadores `%1` y `%2`, que son referencias a las partes que encajan con la petición en `RewriteCond`:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf

<Directory "/var/www/html/pruebas">
        RewriteEngine On
        RewriteBase /pruebas

        RewriteCond %{QUERY_STRING} ^(.*)public(.*)$
        RewriteRule ^admin.html admin.html?%1private%2 [R]
</Directory>
```

Figura 289. Cambio de la querystring

Ahora al acceder a la siguiente URI: http://192.168.56.2/pruebas/admin.html?public=true veremos lo siguiente:

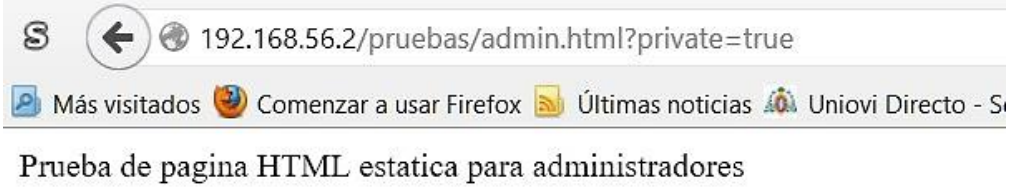


Figura 290. Cambio de la querystring por acceder a una página concreta

Bloqueo de clientes que se conectan a través de la red TOR

El software llamado *ToR* (***The Onion Routing project***, https://www.torproject.org/) es probablemente la forma más popular existente hoy día de **incrementar el anonimato** cuando se navega por Internet. Es la

implementación más conocida de una técnica llamada *Onion Routing* (http://www.onion-router.net/Publications/CACM-1999.pdf).

Gracias a esta técnica se hace **mucho más difícil saber el origen real de un cliente** que hace una petición a un servidor, dada su arquitectura y funcionamiento. Al servidor le llega una petición, pero es muy difícil que sea capaz de trazar quién fue el cliente que realmente la envió puesto que dicha petición, además de ir cifrada múltiples veces, pasa primero por una cadena de nodos aleatoria. A todo el conjunto de nodos de *ToR* le llamaremos a partir de ahora red *ToR*. *ToR* funciona gracias a que sus usuarios "donen" parte de su ancho de banda para permitir que su software funcione como *relay*. Esto significa que, además de aprovecharse de sus servicios de anonimato, también actuarán como nodos para proveer de anonimato a otros (pueden ser parte de una cadena de nodos en la conexión de otros usuarios). A algunos de esos nodos se les denomina "nodos de salida", al ser los que dan el último paso entre la cadena de nodos y el servidor al que se quiere realizar la conexión.

Si bien el anonimato en Internet es un derecho legítimo y no tiene en si nada malo, el problema es que **la red *ToR* suele ser usada para fines maliciosos**: Aprovechándose del anonimato los *crackers* intentan realizar ataques de diversa índole a servidores, siendo estos muy difíciles de parar y, sobre todo, de localizar al culpable. El problema es que la propia estructura de la red *ToR* hace que identificar conexiones realizadas a través de esta sea muy complejo. No obstante, algo sí que se puede hacer para por lo menos intentar cortar el mayor nº posible de conexiones que provengan de dicha red (a pesar de que si lo hacemos estaremos perdiendo visitas legítimas sin fines maliciosos a nuestra web).

Una medida típica es **bloquear el acceso a *IPs* que provengan de países o regiones que sean conocidos como orígenes frecuentes de ataques de seguridad**. Esto lo veremos posteriormente, cuando hablemos del *mod* `geo_ip`. Nótese que este tipo de bloqueos elimina todos los clientes del país o región, sin importar nada más, por lo que el impacto en nº visitas puede ser considerable y muy probablemente tendremos un porcentaje de clientes legítimos visiblemente molestos con nuestra web por ser discriminados solo por conectarse desde un lugar dado.

La segunda opción que nos queda (complementaria o no a la anterior) es **bloquear aquellas IPs que pertenezcan a nodos de la red *ToR*** (de salida, *relays*, todos…depende la lista de IPs que nos descarguemos). Esto no es un dato sencillo de obtener porque dichas *IPs* cambian frecuentemente, y mantener una lista actualizada para bloquear el acceso como hemos visto en el apartado de autenticación puede ser inviable por complejo (y por el coste de rendimiento de comprobar tantas *IPs*). Por suerte `mod_rewrite` nos ofrece una solución apropiada para ello, que implementaremos sobre el host virtual adicional que creamos anteriormente:

- Lo primero es obtener **una lista de IPs que pertenezcan a la red *ToR***. Esto lo podemos encontrar en este enlace, que contiene un fichero con las IPs de todos los nodos ToR que se actualiza cada media hora: https://www.dan.me.uk/torlist/ y que se puede descargar directamente con `wget`. Solo admite una descarga por IP y por hora.

```
operario@ubuntu:/var/www/html/tor$ sudo wget https://www.dan.me.uk/torlist/
```

Figura 291. Descarga de IPs de nodos de la red ToR

- Aunque en nuestro ejemplo nos lo descargaremos una vez, es sencillo **hacer una tarea programada que se descargue este fichero periódicamente** y vuelva a aplicar la configuración reiniciando el servidor, para tener estos datos actualizados frecuentemente.

- Mediante *scripting* (dado el gran nº de IPs del fichero descargado), añadir un valor *dummy* detrás de cada IP separando valor e IP con un espacio en blanco. Esto es necesario porque `RewriteMap` no lee listas, sino pares de valores. Si no ponemos un valor detrás, no será capaz de interpretar correctamente el fichero.

```
operario@ubuntu:/var/www/html/site2$ cat /etc/apache2/torips
192.168.56.1 -
192.168.23.2 -
```

Figura 292. Ejemplo de cómo cambiar una lista de IPs para que funcione con RewriteMap

- Finalmente, usamos la directiva `RewriteMap` para leer el fichero de IPs y `RewriteCond` para comparar la variable de entorno `REMOTE_ADDR` con esas IPs. Hecho esto, y si hay coincidencia, podemos elegir entre prohibir la petición (lo que se hace en la imagen) o redirigirla a una página inocua como vimos en ejemplos anteriores.

```
 GNU nano 2.2.6   Archivo: /etc/apache2/sites-enabled/site2.conf

<VirtualHost 192.168.56.3>
        ServerName Site2
        DocumentRoot /var/www/html/site2
        ErrorLog /var/log/apache2/site2-error.log
        TransferLog /var/log/apache2/site2-access.log

        RewriteEngine on
        RewriteMap tor-hosts txt:/etc/apache2/torips
        <Location "/notor">
                RewriteCond ${tor-hosts:%{REMOTE_ADDR}|NOT-FOUND} !=NOT-FOUND
                RewriteRule ^ - [F]
        </Location>_
```

Figura 293. Bloqueo de una localización de nuestra web a hosts de la red ToR

Además, hay que tener las siguientes precauciones:

- No se puede usar `RewriteMap` dentro de un contenedor `<Directory>` o fichero `.htaccess`. Se recomienda usarlo dentro de un contenedor `<VirtualHost>` como en la imagen, para así tener bien delimitada su aplicación y estar seguros de que no se sobrescribe con otras configuraciones.
- Es importante poner el `RewriteCond` y el `RewriteRule` que prohíben el acceso en una localización o directorio concreto (aunque haya que mover la web), porque si lo ponemos a nivel de servidor o host virtual completo entonces podríamos prohibir incluso el acceso a las páginas de error que muestra el propio servidor (téngase en cuenta que si se va a leer una página de error, el cliente que la solicita sigue siendo el mismo al que le prohibimos el acceso).
- En el caso de que usemos DNS también podemos comprobar nombres de dominios (variable de entorno `REMOTE_HOST`) si incluimos una doble condición como esta en lugar de la que vimos en la imagen anterior:

```
…
RewriteCond   ${tor-hosts:%{REMOTE_ADDR}|NOT-FOUND} !=NOT-FOUND [OR]
RewriteCond   ${tor-hosts:%{REMOTE_HOST}|NOT-FOUND} !=NOT-FOUND RewriteRule   …
```

En cualquier caso, el efecto de acceder desde una IP prohibida de nuestra lista sería este:

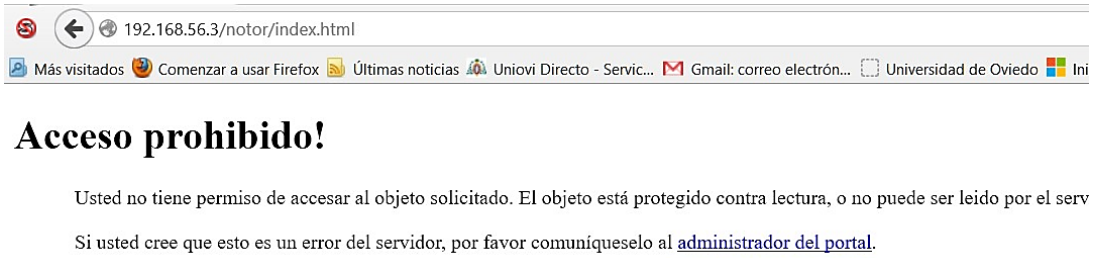


Figura 294. Acceso a una localización desde una IP prohibida

Obviamente, esta técnica no solo puede usarse para bloquear nodos de la red *ToR*, sino que también puede usarse para bloquear cualquier lista de IPs que tengamos en un fichero. Por ejemplo, conjuntos de IPs que se sabe que son orígenes de ataques, como las listas que se pueden encontrar en esta dirección: https://zeltser.com/malicious-ip-blocklists/. Hacer *whitelisting* de clientes (lista de IPs autorizadas a acceder, denegando el resto) también es muy sencillo a partir de este esquema.

Bloqueo de *User Agent Strings*

Un método de bloqueo selectivo de clientes diferente a los que hemos visto hasta el momento es el **bloqueo por *User Agent String***. Un *User Agent String* es **una cadena de texto que identifica al cliente que se conecta al nuestro servidor**. Básicamente es una forma que tiene el cliente de informar al servidor de lo que es (browser, programa de escaneo, gestor de descargas…) y que "marca" de cliente es (*Firefox*, *Chrome*, *Edge*…). Además, no solo pueden informar de esto, sino que a veces también dan más datos como tipo de SO, *plugins* instalados, capacidades, etc.

Se supone que con un *User Agent String* el servidor **podrá tomar determinadas decisiones** acerca de cómo servir el contenido pedido. Por ejemplo, si dice que el cliente es un teléfono *Android*, entonces el servidor puede optar por modificar la petición para servir una versión del sitio web pensada para móviles. No obstante, esto hay que tomarlo con muchas reservas, porque no hay que olvidar que el *User Agent String* es algo que el cliente manda **si quiere** y por tanto **podría decirnos que es algo distinto** a lo que realmente es. Muchos *crawlers* y herramientas de investigación dicen que son un navegador conocido para evitar ser tratados de forma diferente, y el servidor no puede hacer nada por evitarlo. Falsificar el *User Agent String* también es una forma muy sencilla de ver una web para móviles en un navegador de sobremesa, y de esta forma examinar la seguridad de la web móvil desde una plataforma con muchas más capacidades.

Dicho esto, queda claro que la técnica que vamos a ver ahora mismo no es en absoluto efectiva porque **se basa en examinar una información falsificable**. No obstante, no sabemos el nivel de conocimientos de los potenciales atacantes, por lo que bloquear el acceso a *User Agent String* que no son estrictamente el tipo de cliente que esperamos puede ser una medida disuasoria para cierta parte de potenciales usuarios maliciosos.

User Agent Strings hay muchos, casi tantos como productos para acceder a un sitio web. Y desgraciadamente de estos también hay muchos, por lo que hacer una lista negra de posibles clientes no admitidos es una labor costosa y no efectiva. Como siempre, lo mejor es llevar una aproximación basada en listas blancas. Los siguientes enlaces muestran un listado de *User Agent String* de los productos más conocidos:

- *Crawlers*, *bots* y herramientas similares: http://www.useragentstring.com/pages/Crawlerlist/
- Navegadores web: http://www.useragentstring.com/pages/Browserlist/

Para implementar un filtro por *User Agent String* tenemos que usar la directiva `BrowserMatchNoCase` del módulo `mod_setenvif` (http://httpd.apache.org/docs/current/mod/mod_setenvif.html) y también uno de los dos módulos de autorización que ya hemos visto. De esta manera se puede crear una variable de entorno en el caso de que se dé una condición y así establecer si se permite o no el acceso sobre una parte de la web en función de eso. Vamos a hacer una prueba sobre un directorio de nuestra web de pruebas llamado `chromeonly`. El primer paso es crear la configuración adecuada:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf


<Directory "/var/www/html/site2/chromeonly">
        BrowserMatchNoCase Chrome is_chrome
        Require env is_chrome_
</Directory>
```

Figura 295. Configuración para acceder a un sitio web solo si se usa el navegador Chrome

Aquí básicamente decimos que si el *User Agent String* del navegador tiene la palabra `Chrome` definimos la variable de entorno `is_chrome`. Luego usamos una directiva `Require` para que solo deje acceder a la localización si está presente dicha variable. Al probar esta configuración nos encontramos que, efectivamente, solo *Chrome* puede acceder a la página:

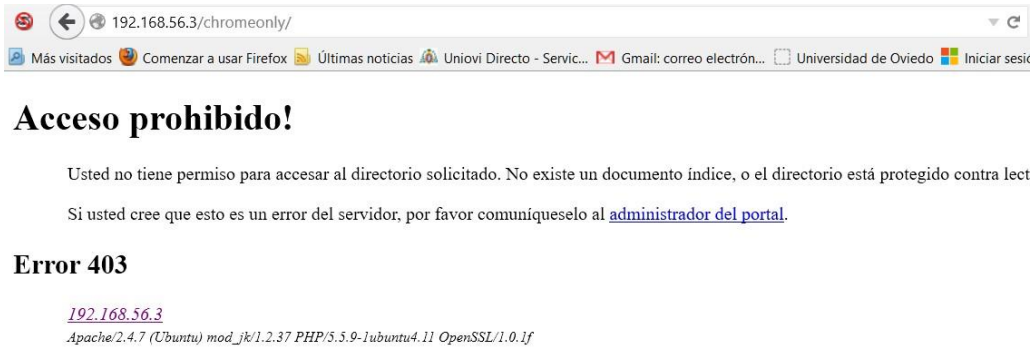


Figura 296. Acceso a la web protegida con Firefox

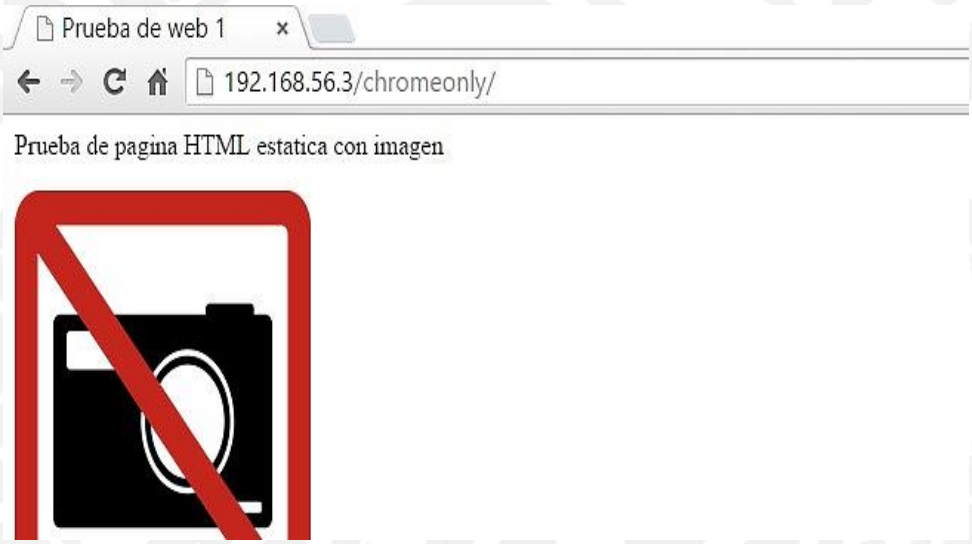


Figura 297. Acceso a la web protegida con Chrome

Si quisiéramos comprobar más de un *User Agent String* bastaría con escribir varias directivas `BrowserMatchNoCase` seguidas. Obviamente, echándole un ojo a los *User Agent Strings* de los dos enlaces anteriores se pueden hacer cosas más avanzadas, teniendo en cuenta siempre que la información que nos llega puede ser falsa y por tanto realmente no haya ninguna clase de protección. Otra cosa a tener en cuenta es que debemos tener mucho cuidado de no bloquear a clientes legítimos solo porque usen navegadores no muy conocidos.

Impedir el robo de ancho de banda: Otra alternativa

Cuando hablamos de las aplicaciones de `mod_rewrite`, vimos que una posible aplicación era precisamente prevenir el ***hotlinking* o robo de ancho de banda**. En esta sección simplemente vamos a hablar de una manera alternativa de implementar la misma funcionalidad. Esta forma de hacerlo necesita `mod_setenvif` (http://httpd.apache.org/docs/current/mod/mod_setenvif.html) como en el caso de la sección anterior, y puede ser útil no solo como alternativa sino también en el caso de que por la razón que sea no tengamos disponible `mod_rewrite`. Para implementar un mecanismo anti-*hotlinking* usando estas herramientas los conceptos son los mismos:

- Hay una serie de ficheros que nuestra web sirve a sus visitantes (imágenes, documentos, etc.)
- Queremos que para verlos se visite nuestra web y no se pueda enlazar desde otras.

Para hacer la prueba lo haremos sobre el mismo host virtual de la sección anterior, dentro del directorio `nohotlink`. La configuración de dicho directorio sería esta:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf


<Directory "/var/www/html/site2/nohotlink">
        #Dominios que pueden acceder al contenido de este enlace
        SetEnvIfNoCase Referer "^http://192.168.56.3/" admitido
        #Si queremos mas dominios basta con poner mas lineas SetEnvIfNoCase
        #a continuacion de la primera

        <FilesMatch "\.(png|gif|jpe?g)$">
                Require env admitido
        </FilesMatch>
</Directory>
```

Figura 298. Protección contra el hotlinking con setenvif

Vemos como hemos combinado la directiva `SetEnvIfNoCase`, que crea una variable de entorno si la cadena que aparece como segundo parámetro encaja con la variable que hemos puesto como primero, con un contenedor `FilesMatch` que engloba varios tipos de fichero distintos (podríamos poner todos los que quisiéramos). La directiva `Require` hace el control de acceso, al igual que en la sección anterior. Hecho esto, podemos ver cómo es posible cargar las imágenes desde las webs de nuestro propio dominio:

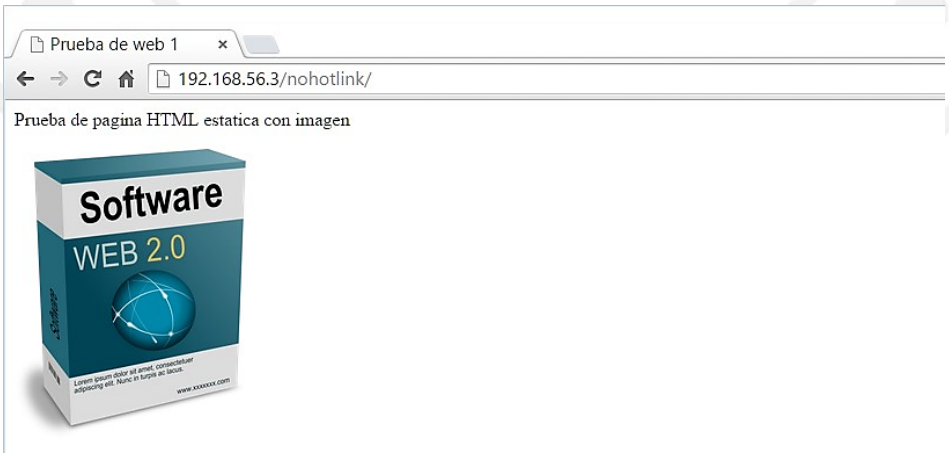


Figura 299. Acceso permitido a las imágenes

Pero no hacer enlaces directos a ellas desde otros sitios:

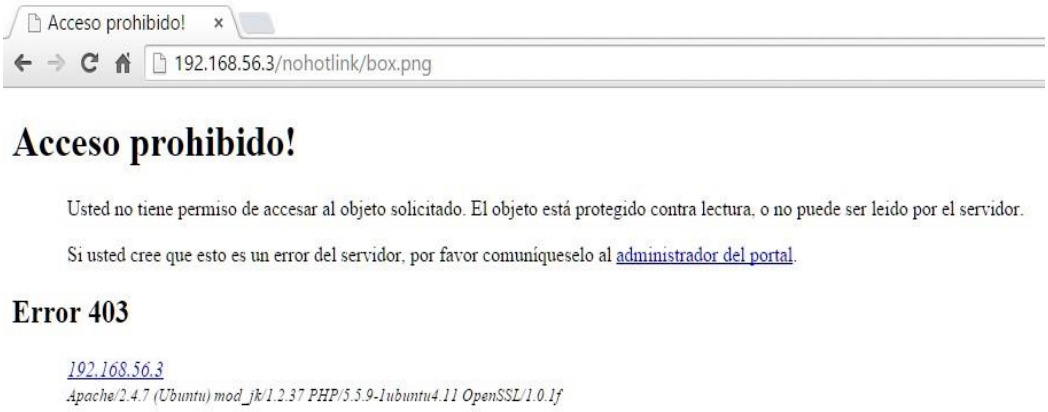


Figura 300. Hotlinking prohibido

Optimización automática del servicio de páginas web: Pagespeed

¿Qué es *Pagespeed*?

PageSpeed (llamado `mod_pagespeed` en *Apache 2*) es un módulo de *Apache 2* (que también está disponible para el servidor *Nginx*) de código abierto, y que contiene una gran cantidad de funcionalidades **que ayudan a optimizar el servicio de las páginas web de diversas formas**. Su fabricante es la conocida *Google*. El módulo contiene un gran nº de **filtros** asociados a diversos tipos de ficheros (`.css`, *JavaScript*, HTML e imágenes, entre otros), que permiten la optimización de sus contenidos de acuerdo con diversos criterios. Por tanto, el módulo mejora la capacidad de *Apache 2* para servir webs optimizando sus contenidos. Además, permite hacer esto sin que haya que hacer cambios en el código de las páginas web, ya que el módulo se integra dentro de la cadena de procesamiento de peticiones ya vista del servidor, haciendo todos estos procesos transparentes al programador.

En esta sección detallaremos los diversos usos que se le pueden dar, hablando de las funcionalidades que contiene y para qué podrían servirnos, su utilidad general y como activar cada una de ellas, pero no entraremos en el detalle de todos y cada uno de los más de 40 filtros que contiene ya que, como hemos dicho, no es el objetivo de este libro. Para ello, en el caso de que necesitemos controlar algún detalle técnico específico que no estuviera contemplado en esta sección, lo mejor es ir a consultar los aspectos concretos que necesitemos en su documentación oficial: https://developers.google.com/speed/pagespeed/module/configuration?hl=es

No obstante las funcionalidades y ventajas del uso de *Pagespeed* para cualquier página desarrollada en cualquier lenguaje son tantas que no podemos dejar sin tratar el mismo, por ser probablemente uno de los módulos **más complejos y útiles que podemos instalar**, junto con `mod_security`. Los contenidos que desarrollemos en esta sección son eso si suficientes para aprender a ponerlo en marcha y manejarlo en muchas situaciones típicas que nos podemos encontrar.

Instalación y puesta en marcha

Hay varias formas de instalar *PageSpeed*. Nosotros solo describiremos, por su sencillez, la instalación desde un paquete `.deb`, adecuado para el SO de pruebas que estamos usando. Para ello simplemente tenemos que:

- **Descargarlo** de https://developers.google.com/speed/pagespeed/module/download?hl=es (concretamente de la dirección https://dl-ssl.google.com/dl/linux/direct/mod-pagespeedhttps://dl-ssl.google.com/dl/linux/direct/mod-pagespeed-beta_current_amd64.debbeta_current_amd64.deb)
  Instalarlo con: `sudo dpkg -i mod-pagespeed-*.deb`

Al hacer esto se añadirá automáticamente el repositorio de paquetes de *Google* al sistema para que así se pueda actualizar automáticamente cuando hagamos una actualización del sistema concreto, algo que se recomienda por temas de funcionalidad y seguridad.

Habilitar *Pagespeed*

PageSpeed está formado por un filtro de salida y varios manejadores de contenidos, encajándose en la cadena de servicio de peticiones de *Apache 2* ya vista. Al igual que otros módulos de *Apache 2*, la configuración de este se encuentra en el fichero `pagespeed.conf` de `/etc/apache2/mods-available/`. La instalación de este módulo también habilita `mod_deflate` por defecto, para que los contenidos vayan comprimidos y mejorar así la optimización.

El **filtro de salida** es el componente que procesa, optimiza y reescribe el contenido HTML de las respuestas que va a enviar el servidor. El filtro de salida hará más o menos cosas según los filtros de *Pagespeed* que habilitemos para que se ejecuten cuando se aplique el mismo. Eso sí, el filtro de salida no reescribe cualquier petición siguiendo las directrices de esos filtros, tenemos que indicarle primero qué tipo de ficheros debe interceptar y reescribir. Por tanto, para que funcione tenemos que asegurarnos que en el fichero de configuración del módulo tenemos esta línea (que ya debería aparecer):

```
# Apache 2 envía toda respuesta HTML a mod_pagespeed para que la procese según su configuración
AddOutputFilterByType MOD_PAGESPEED_OUTPUT_FILTER text/html
```

Si por alguna razón no funcionase correctamente o quisiésemos habilitarlo/deshabilitarlo por algún otro motivo, tenemos que actuar sobre la directiva `ModPagespeed`, jugando con los siguientes valores:

- `On`: Habilitar. Se puede usar en contenedores `<Directory>`, ficheros `.htaccess`, cabeceras HTTP o incluso parámetros de la URI (`?ModPagespeed=on`)
- `Off`: Deshabilita la reescritura de las respuestas que se envían al cliente, pero sigue procesando otras opciones. Se puede usar en los mismos sitios que el anterior.
- `Unplugged`: Deshabilita completamente el módulo. Solo puede usarse en el nivel de configuración global de *Apache 2* y en contenedores `<VirtualHost>`. Nótese que un valor de `on` en un `<VirtualHost>` sobrescribe un `Unplugged` global

Configurar los filtros de *Pagespeed*

Como hemos dicho, **la potencia de este *mod* reside en los filtros que posee** y en cuántos apliquemos antes de enviar la respuesta al cliente, siempre que los contenidos de la respuesta sean de un tipo que hemos declarado como reescribible por este módulo, como vimos en la sección anterior. **Cada filtro de *Pagespeed* tiene un propósito determinado** y están pensados para cumplir con una función muy específica, como veremos un poco más adelante en la tabla que los describe. Al ser tan concretos, uno por sí solo no hace gran cosa, pero mediante la unión de varios es donde realmente se obtiene la potencia del módulo y sus utilidades más interesantes. En esta sección por tanto lo que estamos aprendiendo es a añadir funcionalidades al filtro de salida que el módulo implementa, seleccionando filtros de *Pagespeed* que actuarán precisamente en este punto del procesamiento de la cadena de peticiones, ya visto en una sección anterior.

Dado que existen más de 40 filtros, enumerar todos los que queremos usar en una página concreta puede ser especialmente tedioso. Por ese motivo, el *mod* ofrece tres **perfiles de uso de filtros** (llamados ***rewrite levels***) que, de especificarlos, nos incluyen múltiples filtros de una sola vez. Otra ventaja es que los perfiles contemplan aquellos filtros que son más adecuados para un determinado propósito, simplificando la configuración enormemente. Esos niveles son:

- `PassThrough`: No aplica ningún filtro (pero luego se podrán habilitar los que queramos individualmente).
- `CoreFilters`: Activa aquellos filtros que se consideran más útiles para la mayoría de los sitios web, y que además se cree que no plantearán ninguna incompatibilidad. Activar este perfil hará que la web vaya probablemente más rápido. Es el nivel por defecto.
- `OptimizeForBandwidth`: Selecciona un conjunto de filtros que mejoran el nivel de compatibilidad con cualquier sitio web sobre los seleccionados en `CoreFilters`, incluso ante sitios web que no hayan sido creados con *PageSpeed* en mente.

Para usar un perfil tenemos que usar la directiva `ModPagespeedRewriteLevel`. Por ejemplo:

```
ModPagespeedRewriteLevel PassThrough
```

Si quisiésemos **habilitar filtros individualmente**, tendríamos que usar la directiva `ModPagespeedEnableFilters`, poniendo cada filtro separado por una coma. Pueden usarse tantos `ModPagespeedEnableFilters` como se estime oportuno. En este ejemplo no usamos un perfil pero luego activamos manualmente 5 filtros.

```
ModPagespeedRewriteLevel PassThrough
ModPagespeedEnableFilters combine_css,extend_cache,rewrite_images
ModPagespeedEnableFilters rewrite_css,rewrite_javascript
```

Para **deshabilitarlos** hay una directiva `ModPagespeedDisableFilters` equivalente, pudiendo usar también cualquier nº de ellas:

```
ModPagespeedDisableFilters rewrite_images,combine_css
```

Finalmente, hay una forma de **prohibir totalmente el uso de determinados filtros**, evitando así que se activen en elementos de la configuración que se procesen más tarde. Esto se permite con la directiva `ModPagespeedForbidFilters`. Al igual que con las directivas anteriores, pueden usarse tantas directivas `ModPagespeedForbidFilters` como se estime oportuno:

```
ModPagespeedForbidFilters rewrite_css,rewrite_javascript
```

El orden en el que se pongan los filtros en estas directivas es irrelevante, puesto que se evalúan siempre en el orden en el que aparecen en **la siguiente tabla que los lista todos**, así como una descripción breve de su función. Para una descripción en detalle de cómo funciona cada uno de ellos, se recomienda consultar el siguiente enlace: https://developers.google.com/speed/pagespeed/module/filter-headhttps://developers.google.com/speed/pagespeed/module/filter-head-add?hl=esadd?hl=es

Nombre del filtro de Pagespeed	Incluido en `CoreFilters`	Incluido en `OptimizeForBandwidth`	Utilidad
add_head	**Sí**	No	Añade un elemento `<head>` al documento si no tiene uno ya

add_instrumentation	No	No	Añade *JavaScript* a las páginas para medir la latencia y enviarla al servidor para tener estadísticos de tiempos de servicio
canonicalize_javascript_libraries	No	No	Si la página usa librerías de *Javascript* conocidas, se cambian por otras situadas en un servicio de hosting de librerías *Javascript* conocidas, permitiendo un uso más eficiente de las mismas
collapse_whitespace	No	No	Quita el exceso de espacios en blanco en ficheros HTML, "minificándolos". Evita modificar el contenido de etiquetas `<pre>`, `<script>`, `<style>` y `<textarea>`
combine_css	**Sí**	No	Si la página tuviera múltiples CSS, las combina en una sola
combine_heads	No	No	Si el documento tuviese varios `<head>`, los combina en uno solo
combine_javascript	**Sí**	No	Combina múltiples elementos `<script>` en uno sólo
convert_gif_to_png	**Sí**	**Sí**	Usado por `recompress_images`, convierte imágenes *.gif* en *.png*
convert_jpeg_to_progressive	**Sí**	**Sí**	Usado por el módulo anterior, convierte *jpegs* grandes al formato de carga progresiva
convert_jpeg_to_webp	**Sí**	**Sí**	Usado por el módulo anterior, convierte *jpegs* al formato *webp* si el navegador lo soporta
convert_meta_tags	**Sí**	No	Añade una cabecera de respuesta por cada etiqueta meta existente, con un atributo `httpequiv`
convert_png_to_jpeg	**Sí**	**Sí**	Usado por el módulo anterior, convierte *gif* y *png* a imágenes *jpegs,* sí parecen comprimir mejor y no usan transparencias
convert_to_webp_lossless	No	No	Convierte *gif* y *png* al formato *webp* si el navegador lo soporta
dedup_inlined_images	No	No	Si la página tuviese imágenes repetidas, reemplaza las mismas por *JavaScript* que las carga a partir de la primera aparición de esta
defer_javascript	No	No	No ejecuta el *JavaScript* contenido en la página hasta que se haya cargado completamente
elide_attributes	No	No	Elimina atributos HTML que no son significativos de acuerdo con la especificación HTML

extend_cache	**Sí**	No	Incrementa el tiempo que pasa en la caché los ficheros CSS, *JavaScript* y de imágenes que no han sido optimizados por otros filtros
extend_cache_css	**Sí**	No	Usado por el módulo `extend_cache`, extiende el tiempo de vida de la caché de las CSS
extend_cache_images	**Sí**	No	Usado por el módulo `extend_cache`, extiende el tiempo de vida de la caché de las imágenes
extend_cache_pdfs	No	No	Usado por el módulo `extend_cache`, extiende el tiempo de vida de la caché de los ficheros PDF
extend_cache_scripts	**Sí**	No	Usado por el módulo `extend_cache`, extiende el tiempo de vida de la caché de los scripts
fallback_rewrite_css_urls	**Sí**	No	Reescribe los recursos referenciados por las CSS que no pueden ser procesados o transformados para que ocupen menos espacio
flatten_css_imports	**Sí**	No	Hace *inlining* de CSS, reescribiendo todas las reglas `@import` en una línea
in_place_optimize_for_browser	No	**Sí**	Hace diversas optimizaciones a los recursos dependientes del navegador usado.
include_js_source_maps	No	No	Incluye *source maps* en los ficheros *JavaScript* reescritos. Estos elementos ayudan al navegador a obtener el fichero *JavaScript* original a partir del transformado. Esto permite depurar *Javascript* aunque estemos ejecutando la versión "minificada".
inline_css	**Sí**	No	Convierte pequeñas porciones de código CSS dentro del documento HTML en una sola línea
inline_google_font_css	No	No	Convierte pequeñas porciones de código CSS usado por `fonts.googleapis.com` dentro del documento HTML en una sola línea
inline_images	**Sí**	No	Usado por el módulo `rewrite_images`, reemplaza las imágenes pequeñas por *Data* URIs (`data:`).
inline_import_to_link	**Sí**	No	Convierte etiquetas `<style>` que solo contengan `@imports` de CSS en etiquetas `<link>` equivalentes, haciendo por tanto *inlining* de CSS
inline_javascript	**Sí**	No	Convierte pequeñas porciones de código *JavaScript* dentro del documento HTML en una sola línea

inline_preview_images	No	No	Usa imágenes pequeñas de baja calidad creadas "en línea" mientras no se carguen las imágenes reales del servidor
insert_dns_prefetch	No	No	Añade etiquetas `<link rel="dns-prefetch" href="//www.mipagina.com">` para disminuir el tiempo que tarda en resolverse el nombre DNS
insert_ga	No	No	Añade *Google Analytics* a cada página HTML.
insert_image_dimensions	No	No	Añade los atributos `width` y `height` a aquellas imágenes que no los declaren
jpeg_sampling	**Sí**	**Sí**	Usado por `recompress_images`, cambia el perfil de color de los *jpegs* por otro más óptimo
lazyload_images	No	No	Carga las imágenes solo en el momento en el que el cliente puede verlas (por ejemplo, al hacer *scroll* en la página)
local_storage_cache	No	No	Cachea recursos usados en el documento en la *local storage* de HTML5
make_google_analytics_async	No	No	Si la página usa el API de *Google Analytics* (https://developers.google.com/analytics/) de forma síncrona, convierte este uso en asíncrono para mejorar el rendimiento de la página
move_css_above_scripts	No	No	Traslada los estilos de la página a una localización por encima de las etiquetas `<script>`
move_css_to_head	No	No	Traslada los estilos de la página a la cabecera (`<head>`)
outline_css	No	No	Convierte bloques de CSS embebidos en la página en ficheros externos equivalentes que contengan esas mismas instrucciones (pero que se pueden cachear para mejorar el rendimiento)
outline_javascript	No	No	Convierte bloques de *Javascript* embebidos en la página en ficheros externos equivalentes que contengan esas mismas instrucciones (pero que se pueden cachear para mejorar el rendimiento)
pedantic	No	No	Añade tipos por defecto a tags `<script>` y `<style>` si no están ya especificados y la página no es HTML5. Esto se hace para poder pasar los validadores de HTML4

prioritize_critical_css	No	No	Reemplaza las CSS con versiones en una sola línea que solo incluyen los elementos que se emplean realmente en la página
recompress_images	**Sí**	**Sí**	Usado por `rewrite_images`, recomprime imágenes quitando además los metadatos y transformando imágenes *.gif* en *.png*, para transmitir menos datos al cliente
recompress_jpeg	**Sí**	**Sí**	Usado por `recompress_images`, recomprime imágenes *.jpeg* y les quita los metadatos que considere innecesarios.
recompress_png	**Sí**	**Sí**	Usado por `recompress_images`, recomprime imágenes *.png* y les quita los metadatos que considere innecesarios.
recompress_webp	**Sí**	**Sí**	Usado por `recompress_images`, recomprime imágenes *.webp* y les quita los metadatos que considere innecesarios.
remove_comments	No	No	Quita todos los comentarios a los ficheros HTML (pero no de *JavaScript* o CSS que esté dentro de dichos ficheros)
remove_quotes	No	No	Quita las comillas innecesarias en los valores de atributos HTML
resize_images	**Sí**	No	Usado por `rewrite_images`, redimensiona las imágenes cuando su etiqueta `<img>` especifica un ancho y alto menor que el de la imagen a mostrar
resize_mobile_images	No	No	Idéntico al anterior, pero carga imágenes aún más pequeñas que solo se sirven a dispositivos móviles
resize_rendered_image_dimensions	**Sí**	No	Usado por `rewrite_images`, redimensiona las imágenes cuando su tamaño a la hora de mostrarla resulta ser menor que el de la imagen en sí
rewrite_css	**Sí**	**Sí**	Transforma los ficheros CSS para quitarles el exceso de espacios en blanco y comentarios. A esto se le suele llamar **"minificar" una CSS**. Opcionalmente, si se habilita dicha opción, reescribe y cachea las imágenes referenciadas por dicha CSS. El objetivo es minimizar el espacio ocupado por estos elementos

rewrite_domains	No	No	Reescribe el dominio usado para acceder a los recursos que no han sido modificados por *Pagespeed* en función del valor actual de las directivas del *mod* `MapRewriteDomain` y `ShardDomain` (https://developers.google.com/speed/pagespee d/module/domains)
rewrite_images	**Sí**	**Sí**	Optimiza imágenes, recodificándolas y recortando las partes sobrantes. En el modo `OptimizeForBandwidth` no cambia las URIs de los mismos.
rewrite_javascript	**Sí**	**Sí**	Transforma el código *JavaScript* para quitarle el exceso de espacios en blanco y comentarios. A esto se le suele llamar **"minificar" *JavaScript***. El objetivo es minimizar el espacio ocupado por estos elementos
rewrite_javascript_external	**Sí**	**Sí**	Hace lo mismo que el módulo anterior pero con ficheros *JavaScript* externos. En el modo `OptimizeForBandwith` no cambia las URIs de los mismos.
rewrite_javascript_inline	**Sí**	**Sí**	Hace lo mismo que el módulo anterior pero con bloques de código *JavaScript* en la página
rewrite_style_attributes	No	No	Aplica la configuración del filtro `rewrite_css` visto antes a los atributos `style`
rewrite_style_attributes_with_url	**Sí**	No	Aplica la configuración del filtro `rewrite_css` visto antes a atributos `style` que contengan el texto "`url(`"
sprite_images	No	No	Combina las imágenes de fondo de las CSS en una sola imagen
strip_image_color_profile	**Sí**	**Sí**	Usado por `recompress_images`, le quita el perfil de color a las imágenes
strip_image_meta_data	**Sí**	**Sí**	Usado por `recompress_images`, **le quita los metadatos EXIF a las imágenes**, lo que impide que las imágenes den demasiada información sobre su origen, lo que podría ser un problema de seguridad
trim_urls	No	No	Acorta las URIs haciéndolas relativas a la URI base

Por tanto, gracias a esta tabla podemos ver claramente como el conjunto de funcionalidades ofrecido por *Pagespeed* es muy amplio y de aplicación en diversos escenarios o con distintos tipos de web. Una forma adecuada de proceder es **activar el perfil que mejor se adapte a nuestras necesidades** y luego ir activando manualmente los filtros no incluidos en el perfil que nos interesen (o desactivándolos, si fuera necesario).

Es importante también ir probando si la activación de un filtro rompe la ejecución correcta de la página o no cada vez que lo hagamos. Esto es un proceso lento y delicado, pero una vez realizado ya no tendremos que volver a configurarlo (salvo que la página se actualice con cambios significativos). Antes de terminar con esta sección es importante destacar varias cosas:

- **"Minificar" cualquier contenido de la página** (HTML, CSS, *JavaScript*) es bueno desde el punto de vista del rendimiento y de la seguridad: un código más ofuscado es más difícil de interpretar y por tanto dificultará saber detalles técnicos de nuestra página que pueden abrir la puerta a ataques de seguridad. Un fichero "minificado" ocupa menos y por tanto la ganancia de rendimiento, al transmitirse menos información vía red, será más evidente cuantas más peticiones se realicen. Se recomienda activar todo filtro que "minifique" recursos si es posible.
- **Quitar los comentarios** es imprescindible en una web en producción, porque un comentario puede dar información que comprometa la seguridad de nuestra plataforma. Una página sin comentarios ocupa menos, por lo que supone también una mejora de rendimiento. Este *mod* hace este trabajo por nosotros, si activamos el filtro correspondiente. Se recomienda por tanto también activar este tipo de filtros si es posible.
- **Quitar los metadatos** de los recursos también es conveniente por temas de seguridad, en especial de documentos e imágenes. Los metadatos de una imagen alojada en nuestro servidor pueden ser muy "golosos" para un atacante, ya que podría permitir averiguar datos sobre nuestra empresa, empleados, dispositivos etc. Por tanto, se recomienda tener activos este tipo de módulos por seguridad.

Una vez conseguimos tener estas tres clases de módulos operativos y comprobado que nuestra web funciona correctamente, activar otros módulos es cuestión de gusto personal, del tipo de recursos que usemos y de si de verdad son efectivos en nuestro caso, algo que tenemos que comprobar manualmente. De nada serviría por ejemplo activar el filtro de comprimir imágenes si las nuestras ya están comprimidas al máximo cuando las subimos al servidor: el tiempo perdido en esta operación no conllevaría ganancia alguna.

Potenciales problemas con los contenidos/recursos

En esta sección describiremos posibles inconvenientes que *Pagespeed* puede tener en función del contenido de las peticiones HTTP que maneje el servidor y que tenemos que conocer para estar seguros de que todo se ejecutará de forma correcta.

Cabeceras cache-control

Es importante describir qué ocurre cuando este *mod* se usa en peticiones que tienen incluida la cabecera HTTP `Cache-Control:no-transform`. Esta cabecera es parte del estándar del protocolo HTTP y se creó para **indicar que el valor de determinadas cabeceras de la petición HTTP nunca debe ser transformado** por ninguna entidad que procese la petición entre cliente y servidor.

Está pensada para aquellas aplicaciones que **NECESITAN** que el contenido de esas cabeceras sea idéntico al que se emitió y no haya sido modificado por otras entidades para ahorrar ancho de banda (compresión, cambio de formato) u otros motivos. En concreto, la definición de la cabecera puede encontrarse aquí: http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.5, mientras que las cabeceras cuyo valor no debería modificarse si se usan se listan aquí:

http://www.w3.org/Protocols/rfc2616/rfc2616http://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html - sec13.5.2sec13.html#sec13.5.2.

Honrando esta especificación, y dado que *Pagespeed* puede ejecutar un gran nº de transformaciones sobre diferentes elementos de las respuestas (*JavaScript*, imágenes, *CSS*,…), si *Pagespeed* se encontrase una petición con esta cabecera **no modificaría los contenidos** para cumplir con el protocolo. No obstante, si aun así deseamos hacerlo, ateniéndonos a las consecuencias, podemos decir que lo haga igualmente añadiendo esto `ModPagespeedDisableRewriteOnNoTransform off` a la configuración del *mod*. *Pagespeed* nunca modificará ficheros HTML si hay una cabecera `Cache-Control:notransform`, esta directiva para saltarse el protocolo solo se aplica sobre otros tipos de fichero.

Conversión a minúsculas de contenidos

Otro problema ocurre con el **uso de mayúsculas/minúsculas en el cuerpo de las respuestas**. Los *tags* HTML y sus atributos no son sensibles a mayúsculas, mientras que en XML y XHTML sí que lo son. *Pagespeed* usa la cabecera `Content-Type` para saber si puede convertir el contenido a minúsculas (algo que intenta hacer por defecto), pero es posible encontrar casos en los que aparece la cabecera `Content-Type: text/html` y los recursos son realmente XML o XHTML.

Si *Pagespeed*, fiándose de que el `Content-Type` tiene un valor correcto, convierte todo a minúsculas, entonces **los clientes que lean el cuerpo de la respuesta podrían fallar**. Para evitar incompatibilidades difíciles de detectar por este motivo, el *mod* no convierte por defecto a mayúsculas los elementos HTML ni los nombres de los atributos. No obstante, si estamos seguros de que todos los `Content-Type` son correctos y por tanto nunca alteraremos un fichero XML o XHTML con el riesgo de romper la ejecución de la página, podemos decirle al *mod* que sí convierta estos elementos a minúsculas con `ModPagespeedLowercaseHtmlNames on`.

Como el contenido se manda comprimido gracias al uso por defecto de `mod_deflate`, hacer esto es probable que nos dé una mejora en el rendimiento al transmitir peticiones más pequeñas, puesto que alcanzaremos un ratio de compresión mayor.

Control de la caché

Todas las páginas procesadas por *Pagespeed* se envían con las cabeceras que controlan la caché fijadas al siguiente valor: `no-cache, max-age=0`. De esta forma, según el protocolo HTTP, los cambios hechos a las páginas no se cachearán durante mucho tiempo (ver http://www.w3.org/Protocols/rfc2616/rfc2616http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html - sec14.9.1sec14.html#sec14.9.1).

Sin embargo, algunas aplicaciones necesitan un control particular de esta cabecera por alguna razón, y si *Pagespeed* cambia el valor entonces no funcionarán como deben. Si queremos dejar el valor de esta cabecera tal cual estaba antes de que *Pagespeed* procese la petición, basta con incluir la siguiente línea en el fichero de configuración: `ModPagespeedModifyCachingHeaders off`. Salvo que la página use realmente esta directiva por algún motivo, se recomienda dejar la opción a `on` para que se maximice la posibilidad de servir HTML optimizado por *Pagespeed*.

Modificación de las cabeceras que dan información

Si usamos *Pagespeed*, el *mod* incluirá la cabecera `X-Mod-Pagespeed` en la respuesta HTTP. Esta cabecera muestra el nº de versión completa del *mod* que estamos usando. Por lo dicho anteriormente acerca de

dar información al cliente, **se recomienda omitir el valor de esta cabecera** para no dar pistas sobre la versión del *mod* que estamos usando y de esta forma no ser vulnerables a potenciales errores de seguridad que se detectasen en el mismo. Para ello, basta con incluir la cabecera `ModPagespeedXHeaderValue "<texto que queremos mostrar>"`.

Conservar las URIs de la página

La directiva `PreserveUrlRelativity on` hace que las URI se mantengan tal cual llegan al *mod* en lugar de convertirlas a URI absolutas, algo que podría dar problemas con el código de las aplicaciones o con el uso de HTTPS. Se recomienda activar esta opción.

Ruta donde guardar recursos externos

Algunos de los filtros de *Pagespeed* vistos en la tabla anterior, como `defer_javascript` y `lazyload_images`, entre otros, necesitan usar recursos externos que deben ser cargados de alguna localización del disco duro. Por defecto la ruta donde se guardan estos ficheros es `/pagespeed_static` pero existe una directiva que permite cambiar esta ruta. Por ejemplo `ModPagespeedStaticAssetPrefix /miramdisk/temporales/`.

Uso de *Pagespeed* en diferentes localizaciones

A la hora de usar `mod_pagespeed` tenemos que elegir en qué contenedor especificamos sus directivas. El *mod* puede usarse en:

- **Ficheros** `.htaccess`: En este caso se recomienda que las referencias a ficheros concretos se hagan usando *paths* completos a los ficheros. Se permite el uso de comodines típicos para referirse a múltiples ficheros.
- **Configuración central del servidor**: Por defecto la configuración de *PageSpeed* al instalarse se aplica a nivel de todo el servidor.
- **Host virtual**: Un host virtual puede sobrescribir parte de la configuración del *mod* si así lo necesitase. En caso de que el host virtual no tenga ninguna directiva del *mod*, se usará tal cual la configuración global. Si la tuviese, todo depende del valor de la directiva `InheritVHostConfig`. Si está a `off`, cada host virtual que use *PageSpeed* debe repetir las opciones de configuración que necesitásemos, aunque estén presentes ya en el servidor central u otros host virtuales.

  Si está a `on`, como su nombre indica, el host virtual **heredará** los valores de las directivas en la configuración global. La forma más conveniente de configurar el *mod* es poner todas las directivas comunes al principio y luego especificar las variaciones host a host, como se ve en el siguiente ejemplo de archivo de configuración:

```
#Activamos el mod
ModPagespeed On

#Configuración central y común para todos los host, ya que la heredan
ModPagespeedInheritVHostConfig on

#Donde guarda el mod los ficheros que va procesando
ModPagespeedFileCachePath "/var/cache/mod_pagespeed/"
#Filtros de Pagespeed que queremos activos (ver después)
ModPagespeedEnableFilters combine_css,combine_javascript
```

```
# Todas las respuestas con contenido HTML se envían a Pagespeed.  AddOutputFilterByType
MOD_PAGESPEED_OUTPUT_FILTER text/html


NameVirtualHost *:80
<VirtualHost *:80>
DocumentRoot /var/www/html/web1
ServerName www.miempresa1.com
# En este host virtual no queremos precisamente este filtro de Pagespeed
ModPagespeedDisableFilters combine_css
</VirtualHost>

<VirtualHost *:80>
DocumentRoot /var/www/html/web1
ServerName www.miempresa2.es
# Este host virtual no usará Pagespeed
ModPagespeed Off
</VirtualHost>
```

Prohibir filtros deshabilitados

Hemos visto al principio de esta sección como habilitar filtros, bien **de forma individual** (`EnableFilters`) o bien **a través de un perfil**. También es posible habilítalo incluyendo mención a los mismos en la *Querystring* o bien en las cabeceras HTTP. Esto puede plantear el problema de que se habilite un filtro que se ha deshabilitado previamente (en lugar de prohibirlo), y que al hacer esto tengamos problemas de compatibilidad en algunas páginas.

Podemos evitarlo prohibiendo todos los filtros deshabilitados con la directiva `ModPagespeedForbidAllDisabledFilters true`, pero hemos de tener en cuenta que los filtros que no están activos, bien por el perfil o bien manualmente, se consideran deshabilitados, y por tanto se prohibirían. La configuración por tanto quedaría mucho más rígida y de esta forma no se permitirá usar ningún medio para variar lo que hayamos especificado en los ficheros de configuración de antemano.

Uso en varios servidores

Pagespeed está preparado **para que se ejecute en varios servidores**, teniendo incluso opciones de monitorización globales cuando varios servidores están usándolo, como veremos en la sección de administración. No obstante, si se ejecutase este módulo en varios servidores es muy importante que **todos los servidores usen la misma configuración** (bien copiándola o bien leyéndola de alguna localización remota común). De no hacerlo, el comportamiento podrá ser errático, especialmente si los servidores forman partes de un clúster donde no se sabe a priori que servidor va a atender una petición y este detalle debería ser transparente al usuario. Diferentes configuraciones del módulo entre servidores pueden acabar con esta transparencia.

En este sentido puede ser de ayuda usar `memcached` (http://memcached.org/, https://developers.google.com/speed/pagespeed/module/system#memcached). Este software permite a varios servidores **compartir una misma caché distribuida de objetos** que asegurarán que todos sirven exactamente el mismo contenido.

Por otro lado, si por alguna razón no fuese posible mantener las mismas URIs en todos los servidores, y por tanto no fuese tampoco posible tener la misma configuración en todos los servidores debido a ello, la directiva `ModPagespeedAddOptionsToUrls on` codifica las opciones de configuración aplicables a los

recursos servidos en la *querystring* de la URI, permitiendo así tener la misma configuración a pesar de esto. No obstante, debe tenerse en cuenta que esto hace las URIs especialmente largas y complejas y además delatamos el uso de este módulo a nuestros clientes.

Páginas de administración

Este módulo también permite habilitar **una web especial de administración** para controlar todos sus aspectos. Permite por ejemplo saber en todo momento qué filtros están habilitados. La mayor parte de estas páginas están orientadas a la consulta de datos y no a la modificación de parámetros, aunque permiten purgar la caché de contenidos y partes de estas si permiten modificar valores. Podemos ver toda la información que permite ver la web de administración en el siguiente enlace: https://developers.google.com/speed/pagespeed/module/admin?hl=es. Un resumen de la funcionalidad aparece en la siguiente tabla:

Página	Funcionalidad
Caches	Configuración de la cache e inspección de sus contenidos, así como opción de "purgarla"
Configuration	Permite ver la configuración general, como los filtros activos, opciones activadas, etc.
Console	Muestra unos gráficos que detallan la optimización del servidor a medida que sirve peticiones
Histograms	Datos de latencia, tiempo de carga de las páginas, tiempo de optimización y reescritura de las páginas. Etc.
Message History	Log de *Pagespeed*, mensajes, etc.
Statistics	Permite ver las estadísticas del módulo desde que se ha arrancado, con aspectos como cuantos recursos se han optimizado por los diversos filtros activos, latencias y efectividad de la caché

Activación de las páginas de administración

Las páginas de administración están divididas en *handlers*. En función de cuántos activamos para una localización dada podremos ver más o menos información cuando accedamos a esta localización, correspondiendo cada *handler* a un tipo de esta. La siguiente tabla muestra qué clase de información podemos obtener con cada *handler* activado:

Handler	Descripción de lo que permite ver en la web de administración
`mod_pagespeed_global_statistics`	Idéntico al anterior pero en configuración multi-servidor
`mod_pagespeed_message`	Habilita la página de mensajes de log

`mod_pagespeed_statistics`	Incluye estadísticas, histogramas y un subconjunto de opciones para ver la caché
`pagespeed_admin`	Habilita todas las páginas de manera que se incluyan todas las funciones de administración del servidor, incluyendo todas las demás, por lo que no haría falta habilitarlos
`pagespeed_console`	Habilita la página de gráficos de optimización
`pagespeed_global_admin`	Hace la misma función que el anterior pero cuando hay varios servidores involucrados

Todos los *handlers* son opcionales, y podemos añadir individualmente los que queramos en función de la funcionalidad de administración que queramos mostrar en la web. Hay que tener en cuenta que los *handlers* `pagespeed_admin` y `pagespeed_global_admin` permiten acceso de lectura y escritura al estado del servidor, mientras que los otros son de solo lectura. Debido a que estas páginas muestran información extremadamente sensible, la configuración de acceso a estas páginas debe **ser LO MÁS RESTRICTIVA POSIBLE**. Por ejemplo, podría ser algo como esto:

```
<Location /pagespeed_admin>
      Require local
      SetHandler <nombre del handler tal y como aparece en la tabla anterior>
</Location>
```

Debido a lo sensible de la información, tampoco estaría de más **que se requiriese autenticación** por parte del usuario que accede a estas páginas. La configuración de las páginas de administración viene no obstante lista para ser editada en el propio fichero de configuración del módulo, por lo que en el caso de querer dejar a más IPs acceder a estas páginas tendríamos que añadirlas aquí:

```
GNU nano 2.2.6  Archivo: /etc/apache2/mods-enabled/pagespeed.conf

  # By default, pagespeed_admin and pagespeed_global_admin present
  # the same data, and differ only when
  # ModPagespeedUsePerVHostStatistics is enabled.  In that case,
  # /pagespeed_global_admin sees aggregated data across all vhosts,
  # and the /pagespeed_admin sees data only for a particular vhost.
  #
  # You may insert other "Allow from" lines to add hosts you want to
  # allow to look at generated statistics.  Another possibility is
  # to comment out the "Order" and "Allow" options from the config
  # file, to allow any client that can reach your server to access
  # and change server state, such as statistics, caches, and
  # messages.  This might be appropriate in an experimental setup.
  <Location /pagespeed_admin>_
      Order allow,deny
      Allow from localhost
      Allow from 127.0.0.1
      SetHandler pagespeed_admin
  </Location>
  <Location /pagespeed_global_admin>
      Order allow,deny
      Allow from localhost
      Allow from 127.0.0.1
      SetHandler pagespeed_global_admin
  </Location>
```

Figura 301. Configuración de las páginas de administración de Pagespeed

Finalmente, si quisiésemos estadísticas de cada host virtual por separado en lugar de que se sumen las de todos los host activos, podemos usar la directiva `ModPagespeedUsePerVhostStatistics on`.

Optimización de recursos de otros dominios

Hemos visto que este módulo reescribe ficheros HTML y otros recursos que usen como imágenes, etc. En función de los filtros que se activen en el mismo. *¿Qué ocurre si estos recursos no están alojados en el mismo dominio de la página sino que residen en otros?* (*hotlinking*). Lo que ocurre es que el módulo no los modificará salvo que estos otros dominios aparezcan como un valor en la propiedad `ModPagespeedDomain`. Por ejemplo:

```
ModPagespeedDomain http://www.imagenes.hostingexterno.com
ModPagespeedDomain http://www.estilosgratis.org/css
ModPagespeedDomain *.miempresa.org
```

Nótese que podemos especificar comodines para referirnos a los dominios y también podemos especificar directorios concretos de los mismos si necesitamos más precisión. Los recursos optimizados mantienen su dominio y ruta para evitar problemas en el código de las páginas.

No obstante, este es un uso básico de cómo gestionar los dominios con este módulo, para más información, incluidas operaciones más complejas a realizar con los dominios y recursos reescritos se recomienda consultar: https://developers.google.com/speed/pagespeed/module/domains?hl=es.

Pagespeed y HTTPs

Debido a que hoy en día el uso del protocolo HTTPS es imprescindible para asegurar la confidencialidad de los datos transmitidos entre cliente y servidor no tendría sentido que *Pagespeed* no pudiese trabajar con páginas servidas bajo ese protocolo. Para permitirlo, el módulo cuenta con varios mecanismos, algunos de los cuales veremos a continuación.

No obstante, aunque no hagamos uso de ninguna de las formas de trabajar con HTTPs que vamos a describir, el módulo procesa igualmente los documentos HTML que se soliciten vía HTTPs, ya que está incluido en la cadena de proceso de las peticiones que hemos visto y por tanto tiene acceso a los contenidos. No ocurre lo mismo con cualquier otro recurso que no sean páginas HTML. Salvo que hagamos uso de alguna de las formas de gestionar las conexiones HTTPs que veremos a continuación, el módulo solo procesará recursos externos a las páginas (imágenes, CSS, *Javascript*, etc.) que se soliciten vía HTTP. Pero esto es en sí **un potencial riesgo de seguridad**: partes de una página segura están sirviéndose de forma insegura, por lo que o bien nos aseguramos de que la conexión entre el servidor que tiene las páginas y el que tiene otros recursos es privada y segura o bien activamos alguna de las formas que *Pagespeed* tiene para hacer peticiones de recursos *HTTPs*. Estos mecanismos están controlados por las siguientes directivas del módulo:

- Directiva `MapOriginDomain`
- Directiva `LoadFromFile`
- Directiva `FetchHttps`
- Directiva `ModPagespeedFetchFromModSpdy`: Que se puede usar para obtener recursos vía HTTPS que residan en el mismo dominio que las páginas mediante el módulo `mod_spdy`. No veremos este modo, pero puede consultarse toda la información sobre el mismo en: https://developers.google.com/speed/pagespeed/module/https_support?hl=es#fetch_from_mod_spdy

Directiva MapOriginDomain

Esta directiva permite hacer una correspondencia entre un dominio https y el mismo dominio http. De esta manera una petición `https://` a un nombre de dominio concreto será servida realmente por otro dominio http que es un "espejo" del mismo, permitiendo optimizar los recursos obtenidos mediante http según los filtros que haya instalados. Por ejemplo:

```
ModPagespeedMapOriginDomain "http://<dominio que sirve los recursos>" "https://<dominio al que se hace la petición>"
```

El dominio `https` debe ser el segundo parámetro de la directiva

Directiva LoadFromFile

Esta directiva sirve para especificar de qué ficheros (o rutas) deben cargarse los recursos pedidos vía `https`.

```
ModPagespeedLoadFromFile "https://www.miweb.es" "/var/www/html/ficheros/";
```

Si este mecanismo se usa en conjunto con el anterior en el mismo servidor, debemos asegurarnos de que ambos afectan a dominios diferentes y no al mismo.

Obtener los recursos directamente vía HTTPs

En este módulo es posible obtener recursos a optimizar directamente vía HTTPS, sin necesidad de indicar al módulo un lugar aparte de donde obtenerlos, aunque viene desactivada por defecto. Para activarla, simplemente tenemos que incluir la siguiente directiva en el fichero de configuración: `ModPagespeedFetchHttps enable`

Además, esta directiva admite parámetros separados por comas. Estos parámetros son:

- `enable`: Habilita esta forma de obtener recursos vía HTTPS
- `disable`: Deshabilita esta forma de obtener recursos vía HTTPS
- `allow_self_signed`: Permite que se usen certificados auto-firmados, si la web usa un certificado por el que no se ha pagado a una autorizada certificadora de confianza.
- `allow_unknown_certificate_authority`: Permite certificados con autoridades certificadoras no conocidas. Es algo peligroso porque entonces cualquier autoridad podría firmar el certificado, pero útil si hemos instalado una autorizad certificadora propia que emite los certificados para nuestra empresa.
- `allow_certificate_not_yet_valid`: Admite certificados inválidos. No se recomienda su uso.

Configuración de certificados SSL/TLS

Para que la obtención de recursos vía HTTPS funcione, el módulo **debe conocer el directorio y fichero donde están listadas las autoridades certificadoras de confianza y los certificados**. La instalación de *Pagespeed*, si el servidor ya estuviese configurado para tratar peticiones HTTPS como es nuestro caso, se encarga de hacer gran parte de esta configuración. Podemos no obstante indicar el directorio donde se encuentra la información con la siguiente directiva:

```
ModPagespeedSslCertDirectory /etc/ssl/certs
```

En el ejemplo se muestra el directorio por defecto para guardar esta información en sistemas basados en *Debian*. Otros SO pueden tener distintas localizaciones.

Finalmente, hay que comentar que esto es sólo parte de la configuración del módulo para trabajar con HTTPS, la que se ha considerado más útil para el caso general. Si se quisiese profundizar más en estos aspectos, o adaptar la configuración de acuerdo a algunas necesidades especiales se recomienda consultar el siguiente enlace: https://developers.google.com/speed/pagespeed/module/https_support?hl=es

Probando *Pagespeed*

A continuación vamos a hacer sobre nuestra infraestructura de pruebas una serie de acciones para probar los distintos conceptos de *Pagespeed* que hemos explicado y ver qué resultado producen. Ante todo, ha de tenerse en cuenta que estamos probando el módulo con páginas muy simples, por lo que el efecto del módulo no será muy visible. Con páginas más complejas sí que veremos transformaciones de código y ganancias de rendimiento mucho más relevantes.

Instalación y activación de un perfil de filtros

Como hemos dicho anteriormente, el módulo hay que descargarlo en forma de paquete `.deb` de la web de su fabricante, teniendo cuidado de descargar la versión de 32 o de 64 bits en función del SO que tengamos instalado:

```
operario@ubuntu:~$ wget https://dl-ssl.google.com/dl/linux/direct/mod-pagespeed-
beta_current_amd64.deb
```

Figura 302. Descargando Pagespeed de la página del fabricante

Hecho esto, podemos instalarlo con `dpkg`

```
operario@ubuntu:~$ sudo dpkg -i mod-pagespeed-*.deb
```

Figura 303. Instalando Pagespeed

El módulo recién instalado realmente no tiene ningún efecto en las páginas porque no tiene activo ningún filtro, a pesar de que sí que tiene la asociación con ficheros HTML ya activa como filtro de salida:

```
 GNU nano 2.2.6  Archivo: /etc/apache2/mods-enabled/pagespeed.conf

<IfModule pagespeed_module>
    # Turn on mod_pagespeed. To completely disable mod_pagespeed, you
    # can set this to "off".
    ModPagespeed on

    # We want VHosts to inherit global configuration.
    # If this is not included, they'll be independent (except for inherently
    # global options), at least for backwards compatibility.
    ModPagespeedInheritVHostConfig on

    # Direct Apache to send all HTML output to the mod_pagespeed
    # output handler.
    AddOutputFilterByType MOD_PAGESPEED_OUTPUT_FILTER text/html

    # If you want mod_pagespeed process XHTML as well, please uncomment this
    # line.
    # AddOutputFilterByType MOD_PAGESPEED_OUTPUT_FILTER application/xhtml+xml

    # The ModPagespeedFileCachePath directory must exist and be writable
    # by the apache user (as specified by the User directive).
    ModPagespeedFileCachePath           "/var/cache/mod_pagespeed/"

    # LogDir is needed to store various logs, including the statistics log
    # required for the console.
    ModPagespeedLogDir "/var/log/pagespeed"
                          [ 351 líneas leídas ]
```

Figura 304. Pagespeed es un filtro de salida para ficheros HTML

Para probar que funciona probaremos a activar el perfil de filtros `CoreFilters`:

```
GNU nano 2.2.6  Archivo: /etc/apache2/mods-enabled/pagespeed.conf Modificado

  # "CoreFilters" uses a set of rewrite filters that are generally
  # safe for most web pages. Most sites should not need to change
  # this value and can instead fine-tune the configuration using the
  # ModPagespeedDisableFilters and ModPagespeedEnableFilters
  # directives, below. Valid values for ModPagespeedRewriteLevel are
  # PassThrough, CoreFilters and TestingCoreFilters.
  #
  # ModPagespeedRewriteLevel PassThrough
  ModPagespeedRewriteLevel CoreFilters
```

Figura 305. Activando el perfil de filtros CoreFilters

Finalmente, reiniciamos *Apache 2* y probamos que su funcionamiento es correcto. Por ejemplo, esta es una página de nuestra infraestructura de pruebas antes de instalar el módulo:

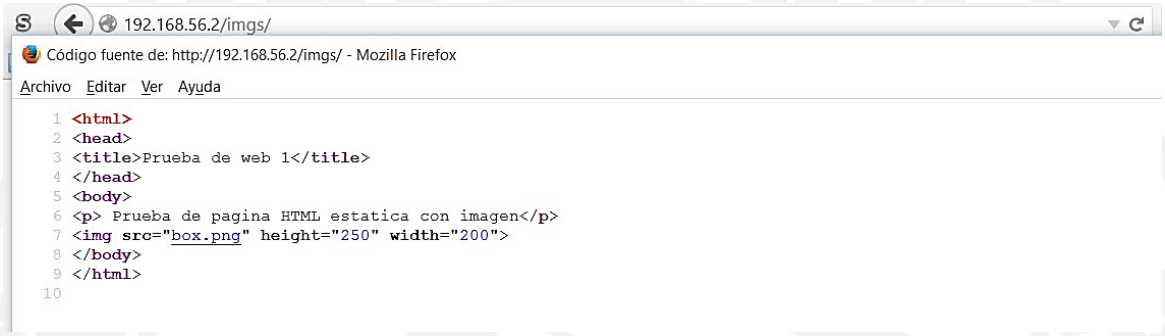


Figura 306. Página antes de ser procesada por Pagespeed

Y este es su código fuente después, que evidentemente ha sido modificado por *Pagespeed* según los filtros que hemos instalado:

Figura 307. Página después de ser procesada por Pagespeed

Habilitando filtros individualmente

Aunque ya tenemos activos un buen conjunto de filtros por el perfil que hemos habilitado, podemos complementarlos con filtros adicionales en función de si nos son útiles en nuestras webs. En la siguiente imagen vemos como activamos dos que previamente no lo están por el perfil, como son `collapse_whitespace` e `inline_preview_images`:

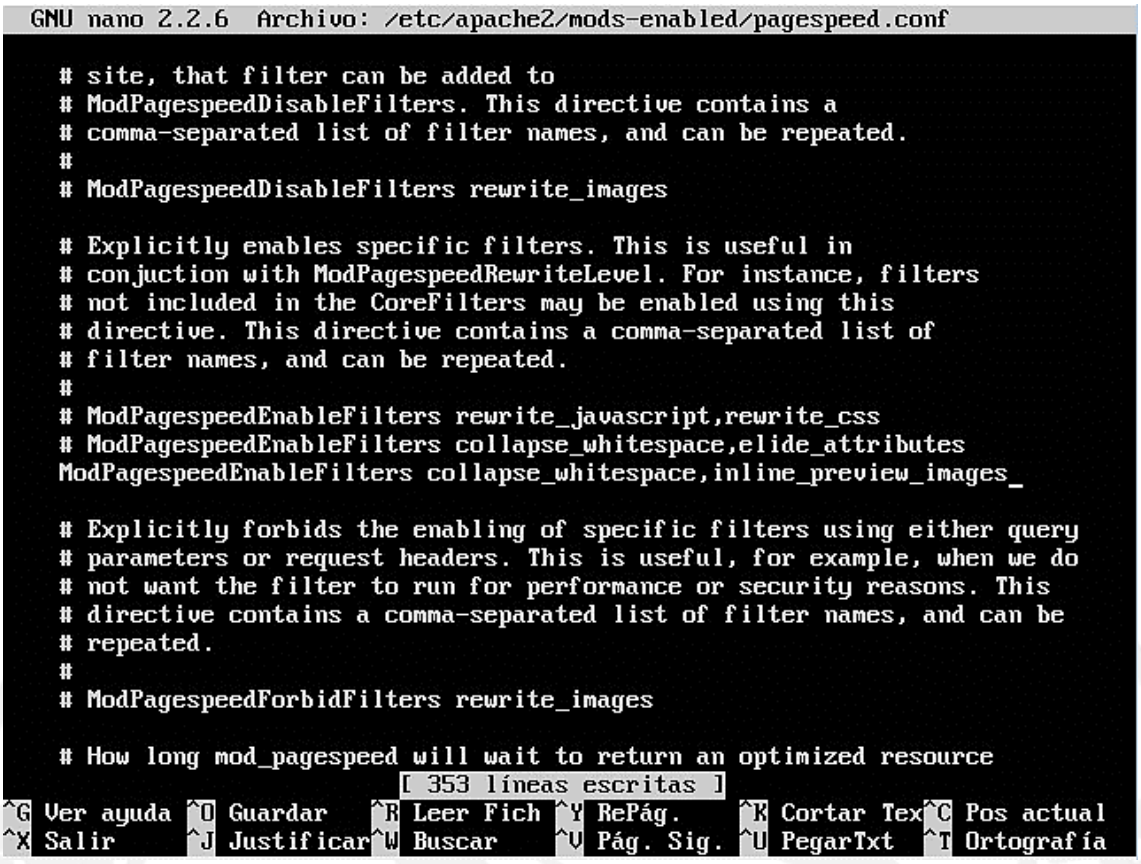


Figura 308. Habilitando filtros individualmente

Al reiniciar *Apache 2* vemos que hay más modificaciones a la página anterior hechas por el módulo debido a los nuevos filtros:

Figura 309. Página modificada por Pagespeed con más filtros activos

Habilitando las páginas de administración

Para habilitar las páginas de administración descritas anteriormente simplemente debemos ir al fichero `pagespeed.conf` y añadir IPs autorizadas a verlas (o bien cambiar la cantidad de información que se da a través de esas páginas modificando el valor del `Handler`) si el acceso sólo desde `localhost` no es posible, como es nuestro caso. En la siguiente imagen vemos como añadimos a nuestra máquina cliente como autorizada:

```
GNU nano 2.2.6  Archivo: /etc/apache2/mods-enabled/pagespeed.conf Modificado

  # By default, pagespeed_admin and pagespeed_global_admin present
  # the same data, and differ only when
  # ModPagespeedUsePerVHostStatistics is enabled.  In that case,
  # /pagespeed_global_admin sees aggregated data across all vhosts,
  # and the /pagespeed_admin sees data only for a particular vhost.
  #
  # You may insert other "Allow from" lines to add hosts you want to
  # allow to look at generated statistics.  Another possibility is
  # to comment out the "Order" and "Allow" options from the config
  # file, to allow any client that can reach your server to access
  # and change server state, such as statistics, caches, and
  # messages.  This might be appropriate in an experimental setup.
  <Location /pagespeed_admin>
      Order allow,deny
      Allow from localhost
      Allow from 127.0.0.1
      Allow from 192.168.56.1
      SetHandler pagespeed_admin
  </Location>
  <Location /pagespeed_global_admin>
      Order allow,deny
      Allow from localhost
      Allow from 127.0.0.1
      SetHandler pagespeed_global_admin
  </Location>
```

Figura 310. Control de acceso a las páginas de administración de Pagespeed

Y gracias a eso ahora tenemos disponible mucha información acerca del módulo, su funcionamiento y posibles incidencias, como se ve en las siguientes imágenes:

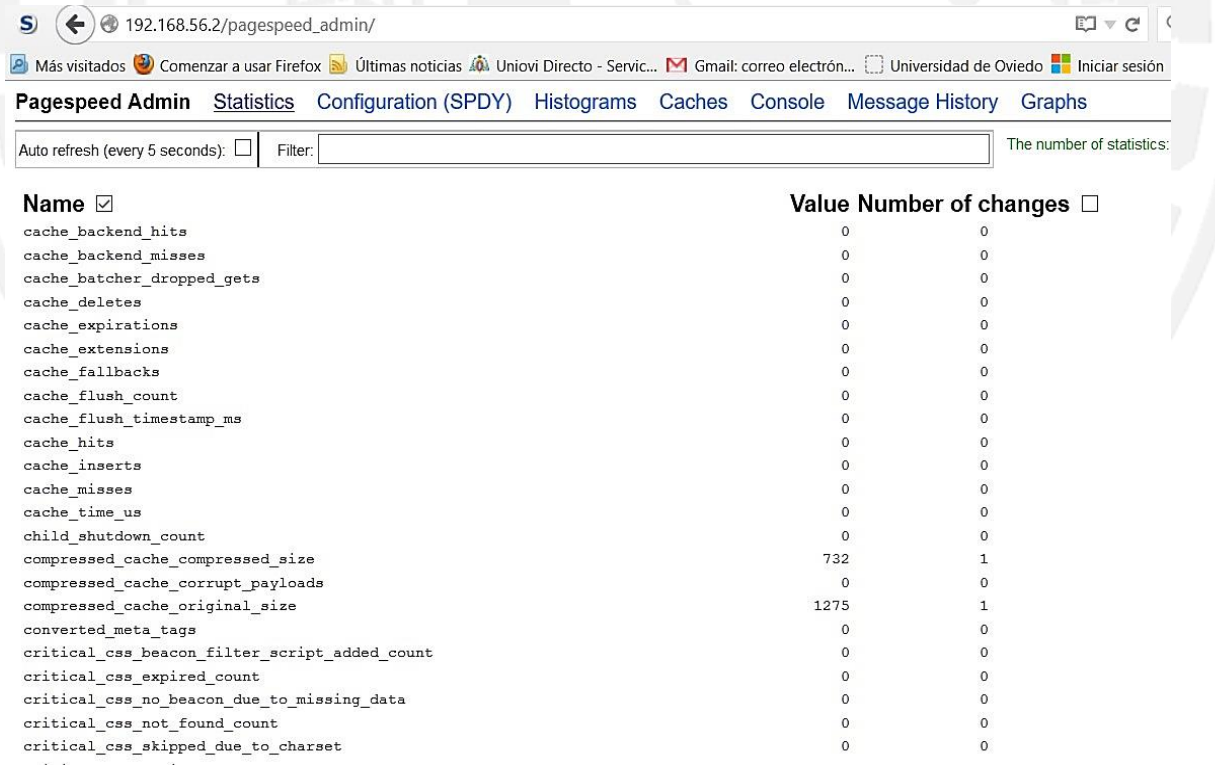


Figura 311. Estadísticas de uso de Pagespeed

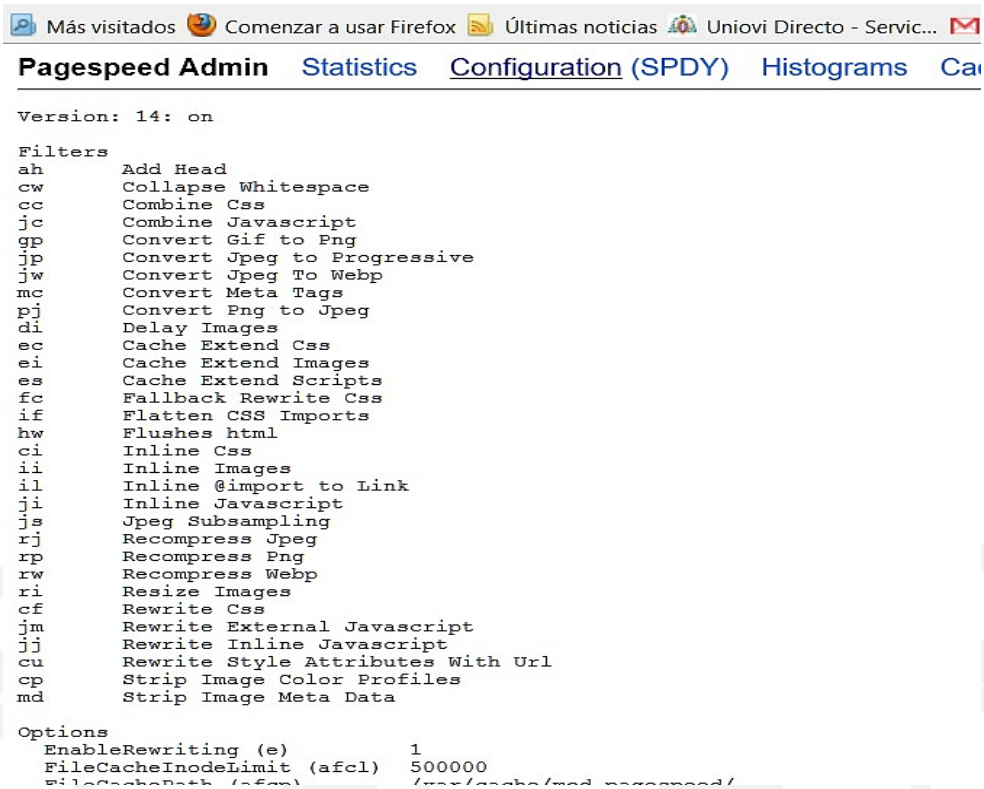


Figura 312. Configuración de Pagespeed

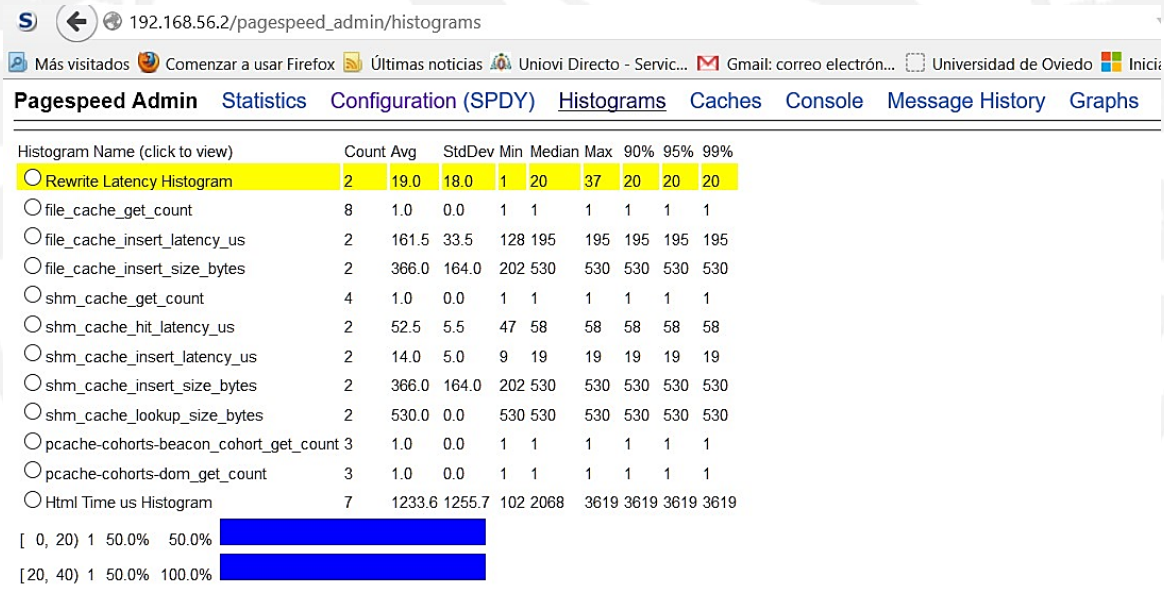


Figura 313. Histogramas de uso de Pagespeed

Figura 314. Pantalla de control de la caché de Pagespeed

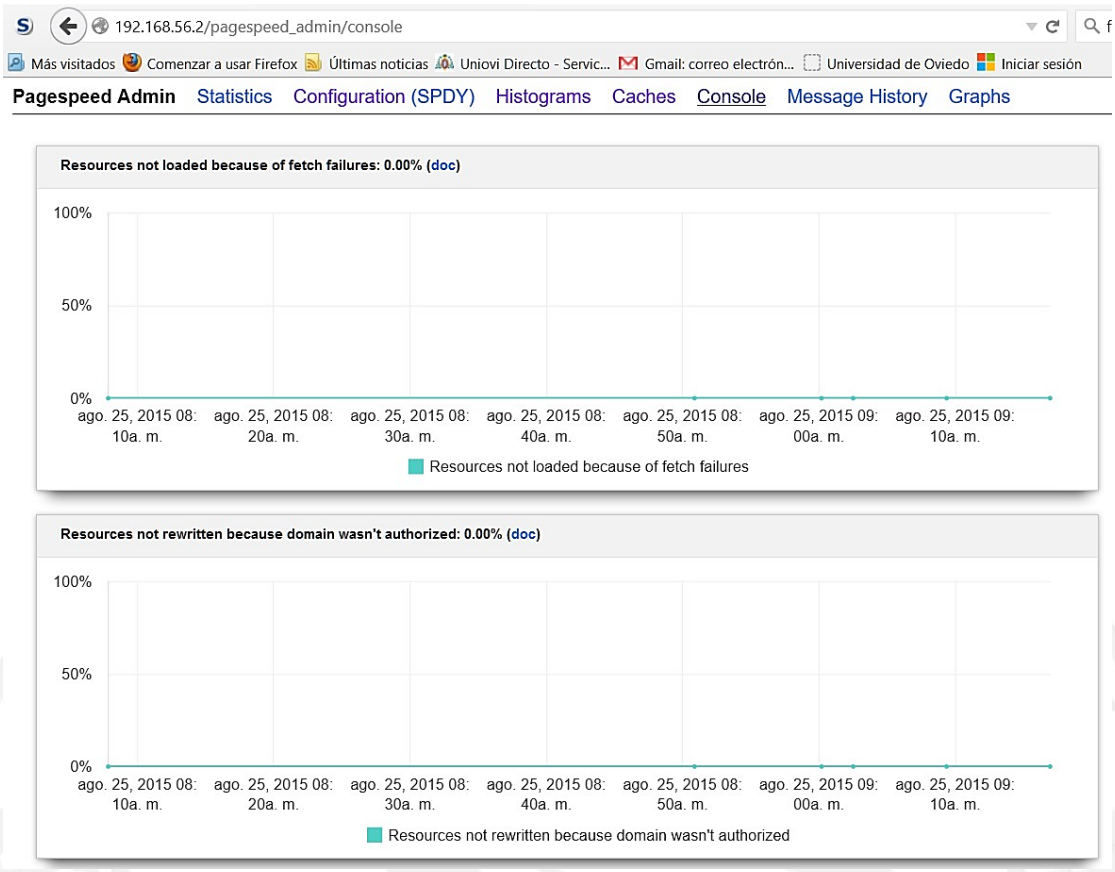


Figura 315. Consola de fallos de Pagespeed

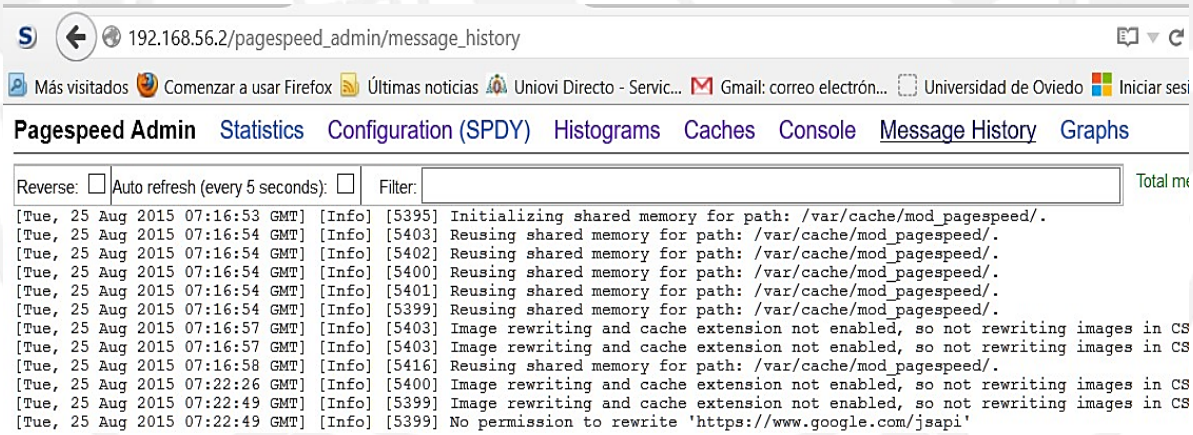


Figura 316. Log de Pagespeed

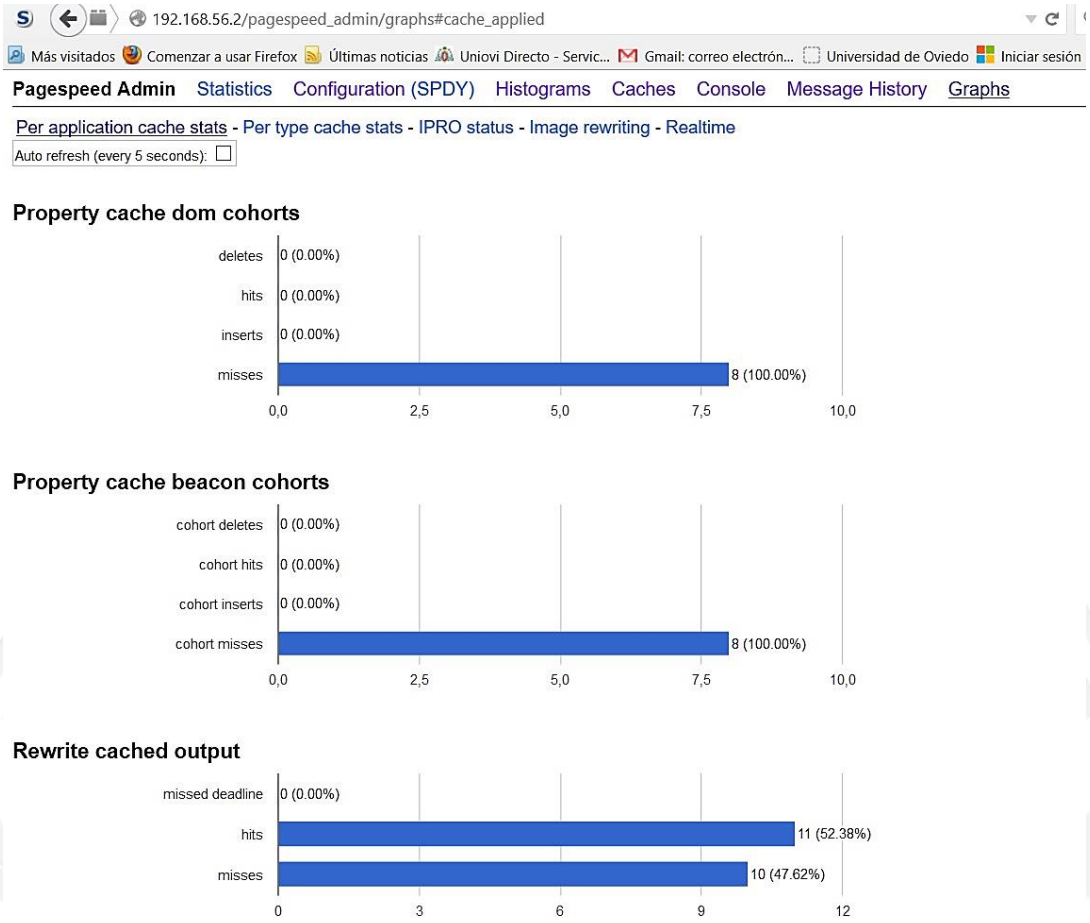


Figura 317. Gráficos de uso de Pagespeed

Habilitando TLS

Para finalizar configuraremos *Pagespeed* para que pueda trabajar con contenidos que se transmiten vía HTTPs. Para ello simplemente añadimos la siguiente directiva al final del fichero de configuración del módulo, puesto que no viene especificada en la instalación por defecto:

```
 GNU nano 2.2.6  Archivo: /etc/apache2/mods-enabled/pagespeed.conf

        Allow from localhost
        Allow from 127.0.0.1
        SetHandler pagespeed_global_admin
    </Location>

    # Enable logging of mod_pagespeed statistics, needed for the console.
    ModPagespeedStatisticsLogging on

    # Page /mod_pagespeed_message lets you view the latest messages from
    # mod_pagespeed, regardless of log-level in your httpd.conf
    # ModPagespeedMessageBufferSize is the maximum number of bytes you would
    # like to dump to your /mod_pagespeed_message page at one time,
    # its default value is 100k bytes.
    # Set it to 0 if you want to disable this feature.
    ModPagespeedMessageBufferSize 100000

    ModPagespeedFetchHttps enable,allow_self_signed
</IfModule>
```

Figura 318. Permitiendo FetchHttps en Pagespeed

Hecho esto, tenemos que comprobar que los recursos accedidos vía HTTPs siguen siendo modificados por *Pagespeed* de la misma forma que si lo fueran vía HTTP:

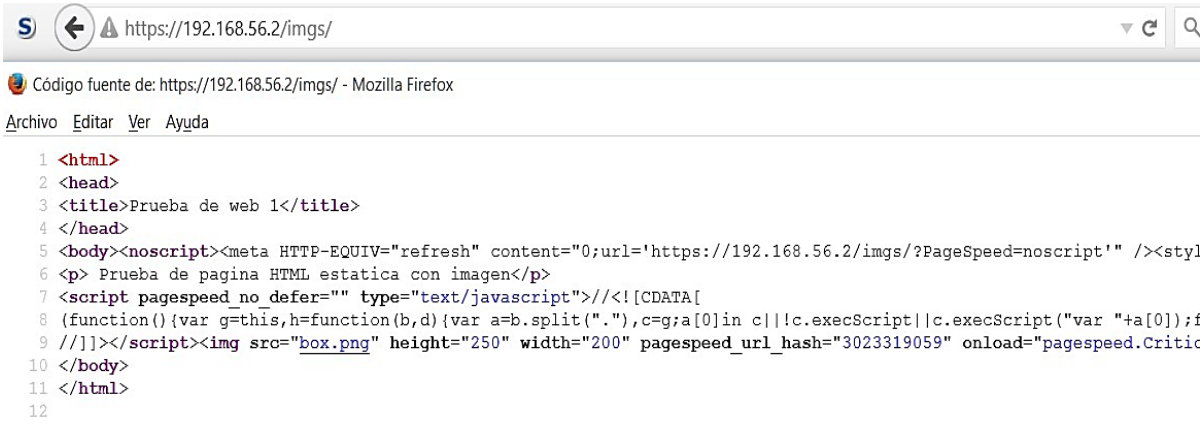


Figura 319. Pagespeed bajo HTTPs

Sin embargo, al cargar esta página que tiene una imagen protegida contra *hotlinking* vemos un problema de incompatibilidad entre dicha protección y un filtro de *Pagespeed*. Para optimizar imágenes *Pagespeed* inserta código *JavaScript* que las obtiene cuando la página se carga, por lo que la petición de dicha imagen tendrá como `Referer` el propio cliente (es decir, es como si el cliente estuviese solicitándola directamente). Por ese motivo, habría que deshabilitar el filtro que gestiona las imágenes afectadas si queremos conservar la protección anti-*hotlinking* vista.

WebDAV

¿Qué es *WebDAV*?

Es un método tradicional para subir archivos a un servidor web es usar FTP. No obstante, esto tiene las siguientes desventajas:

- **Requiere abrir puertos adicionales en el *firewall***, lo que puede dar pie a ataques. Abrir puertos incrementa la susceptibilidad de un servidor de ser atacado, por lo que debe ser evitado siempre que sea posible.
- **FTP no tiene un mecanismo de bloqueo de archivos**, lo cual permite que dos usuarios puedan subir versiones distintas del mismo archivo al mismo tiempo y que una de ellas sea sobrescrita. Esto podría causar problemas graves de coherencia en los archivos de una web.
- Usar FTP **obliga a editar el contenido de forma local**, en la máquina cliente, y luego hacer la transferencia de los contenidos modificados al servidor. Si quisiésemos editar un archivo del servidor tendríamos que descargarlo del mismo, editarlo en nuestra máquina local y subir posteriormente el archivo modificado de nuevo, lo cual es un proceso lento.

  No obstante, existen editores que se integran con un servidor FTP de manera que permiten abrir ficheros de un servidor FTP y subir los cambios a medida que los vamos haciendo en el cliente, haciendo la descarga y subida de los archivos editados transparente al usuario y creando la ilusión de estar editando directamente un archivo situado en un FTP (*WinScp*, http://winscp.net/eng/docs/lang:es).

Una solución a estos inconvenientes es usar ***Web-based Distributed Authoring and Versioning*** (*WebDAV*), un protocolo usado para **publicación y gestión de contenido en servidores remotos**, y también servidores web. *WebDAV* es una extensión del protocolo HTTP/1.1 y está descrito en los RFC 2518 (http://tools.ietf.org/html/rfc2518) y 3253 (https://www.ietf.org/rfc/rfc3253.txt). Esta extensión permite a los usuarios editar contenidos de forma colaborativa y administrar archivos en servidores web remotos, de manera que podamos crear, modificar o eliminar documentos en un servidor remoto o en una "carpeta compartida web".

El uso principal que se le puede dar a *WebDAV* aplicado a los servidores web es **permitir editar los contenidos de una web desde un cliente remoto**, sin necesidad de acceder directamente al servidor, permitiendo así la actualización sencilla de los contenidos de una web en tiempo real desde cualquier sitio por parte de usuarios autorizados para ello. A efectos de uso, *WebDAV* funciona **como un almacenamiento remoto de archivos accesible vía web** desde cualquier parte. Además, *WebDAV* usa el puerto 80 (el mismo puerto que usa HTTP para las peticiones web) por lo que normalmente no necesita modificar la configuración del firewall o abrir puertos adicionales.

La mayoría de los sistemas operativos modernos proporcionan soporte nativo para *WebDAV*. Con un cliente y red adecuados, el manejo puede ser tan sencillo como manejar archivos en directorios locales. *Apache 2* proporciona soporte para *WebDAV* y, como ya hemos dicho, podemos usarlo para editar el contenido de nuestras web de forma remota. Concretamente, cuando se integra con *Apache 2*, *WebDAV* permite a los clientes hacer las siguientes operaciones:

- **Manipular recursos en un directorio especial *WebDAV*** del servidor, destinado a publicar contenidos. Por ejemplo, usuarios con los permisos correctos podrán copiar o mover archivos

dentro de dicho directorio *WebDAV*. La gran ventaja sobre FTP es que las modificaciones y otras operaciones se pueden hacer directamente sobre el propio servidor.

- **Modificar propiedades** asociadas a diversos recursos. Por ejemplo, un usuario puede leer y cambiar las propiedades de un archivo fácilmente.
- **Bloquear y desbloquear recursos**, de forma que múltiples usuarios puedan leer un fichero de forma concurrente pero sólo uno pueda modificarlo al mismo tiempo.
- **Buscar entre el contenido** y las propiedades de los archivos en un directorio WebDAV.

Habilitar un directorio *WebDAV* para **publicar contenidos en un servidor es muy sencillo**, como veremos en breve. Una vez creado este directorio, los usuarios que tengan los permisos adecuados podrán publicar documentos en el servidor y manipular archivos en el mismo. No obstante, para ello necesitarán contar con herramientas adecuadas, como las que describiremos en la siguiente sección.

Clientes *WebDAV*

A la hora de acceder y publicar contenidos en un directorio o sitio web que admita *WebDAV* hay varias categorías de productos que permitirán hacer un nº mayor o menor de operaciones. Como veremos más adelante, no todas ellas soportan todas las operaciones o configuraciones de acceso al sitio:

- **Clientes *Windows* (Windows 200X y XP)**: Se puede establecer una conexión a un directorio *WebDAV* desde el propio explorador de archivos de *Windows*, mostrando los contenidos como si fuese parte del sistema de archivos del ordenador local. Una vez conectados al directorio *WebDAV*, **los clientes *Windows* pueden hacer *Drag and Drop* de archivos**, obtenerlos, modificar sus propiedades y contenidos y otras operaciones típicas. Este tipo de cliente permite que aplicaciones existentes usen recursos de un directorio *WebDAV* y compartir archivos incluso a través de cortafuegos y servidores proxy, siempre que su configuración lo permita (no bloquee el puerto 80).
- **Navegadores web**: Este cliente permite conectarse a un directorio *WebDAV*, abriendo dicho directorio como una carpeta web. Funcionalmente permite hacer las mismas operaciones que el cliente anterior, con las extensiones del navegador adecuadas instaladas.
- **Productos de la familia *Microsoft Office* (*Office 2000* o superior)**: Estos programas permiten crear, publicar y editar documentos directamente de un directorio *WebDAV*. Para un servidor web esto podría ser una característica muy útil si está usando documentos generados por estas aplicaciones, ya que podríamos lograr la actualización inmediata de los mismos según los estamos editando.
- **Herramientas específicas**: Como por ejemplo `cadaver` (http://www.WebDAV.org/cadaver/) en modo texto o el soporte que trae de serie para ello los distintos entornos gráficos disponibles. En este enlace podemos encontrar un listado de clientes que admiten el protocolo *WebDAV*: https://en.wikipedia.org/wiki/Comparison_of_WebDAV_software

Instalación de soporte *WebDAV* en *Apache 2*

Lo primero que debe hacerse es habilitar el módulo que da soporte a este protocolo de esta forma, reiniciando *Apache 2* como siempre después:

```
sudo a2enmod dav
sudo a2enmod dav_fs
```

Posteriormente necesitamos crear o especificar un directorio donde vayamos a permitir la edición de archivos con este protocolo. Podría ser incluso toda la web, pero las implicaciones de seguridad que esto tiene son muy graves si alguien ajeno consigue conectarse. En nuestro ejemplo creamos el directorio `/var/www/html/site2/WebDAV`, porque vamos a permitir *WebDAV* solo en uno de nuestros host virtuales.

También tenemos que asegurarnos de que dicho directorio tiene permisos adecuados, y por ello lo mejor es asignarle como propietario al usuario anónimo de Internet de *Apache 2* (`www-data`) con `chown`:

```
sudo chown www-data /var/www/html/site2/WebDAV
```

Además, dejar el contenido editable vía *WebDAV* sin protección alguna es un disparate desde el punto de vista de la seguridad. A la hora de proteger los contenidos podemos usar los medios de control de acceso que hemos visto. No obstante, los medios que usemos influirán en los posibles clientes que podamos usar para editar contenidos, ya que no todos admiten ambos.

Protección de los contenidos mediante usuario/password

Hacer esto no es complejo puesto que se trata de usar autenticación *Basic* o *Digest* como ya hemos visto. Si quisiéramos hacerlo mediante autenticación *Basic* simplemente tendríamos que ejecutar lo siguiente (en nuestro ejemplo, creamos un usuario llamado `despliegue`):

```
operario@ubuntu:/var/www/html$ sudo htpasswd -c /etc/apache2/webdav.pwd desplieg
ue
New password:
Re-type new password:
Adding password for user despliegue
```

Figura 320. Creación de un usuario para realizar operaciones con WebDAV

Conviene dejar el fichero con las claves fuera de las rutas donde están las webs navegables para que no pueda accederse al mismo. Por otro lado, tampoco nos interesa que los usuarios de la máquina puedan leerlo, por lo que conviene cambiarle los permisos y el propietario para que solo `root` y `www-data` (como único miembro del grupo `www-data`) puedan acceder a su contenido, eliminando los del resto de usuarios. Esto puede hacerse de la siguiente forma:

```
sudo chown root:www-data /etc/apache2/WebDAV.password
sudo chmod 640 /etc/apache2/WebDAV.password
```

Esta opción tiene sentido si el acceso vía *WebDAV* se permite desde muchas localizaciones y entonces necesitamos un usuario/*password* para impedir que quien conozca la ruta donde se ha activado *WebDAV* pueda editar los contenidos. No obstante, estamos usando autenticación *Basic*, por lo que usuario y *password* (y también todos los contenidos que editemos) **se transmitirán en texto plano**. Esto no es admisible salvo que se trate de una red local protegida por otros medios o se use TLS (opción de la que hablaremos después).

Protección de los contenidos mediante restricción de IPs

Con esta opción **solo unas determinadas máquinas autorizadas** a ello podrán editar contenidos vía *WebDAV*. Esto tiene sentido si estamos en un entorno donde las máquinas de desarrollo y de producción están separadas y solo unas pocas máquinas de desarrollo están autorizadas a subir/editar contenidos a producción. La forma de hacerlo ya la hemos visto, pues consiste en usar las restricciones de IPs de alguno de los módulos de autenticación de los que ya hemos hablado. Nuevamente, el problema es que

los contenidos editados viajarán en texto plano, lo que puede ser un problema de seguridad. Se necesita también en este caso TLS para lograr un adecuado nivel de seguridad, salvo que la red donde los contenidos se transmiten esté protegida por otros medios.

Configuración de *Apache 2* para el uso de *WebDAV*

Si ya se ha instalado *WebDAV* y se ha creado un directorio donde guardar los contenidos que se editen, solo queda modificar la configuración del servidor para habilitar *WebDAV* en alguna parte de las webs que estamos alojando. A modo de ejemplo, lo haremos sobre un host virtual alternativo al de por defecto, en el directorio que hemos mencionado antes. La configuración que permitiría esto es la siguiente, donde hemos elegido proteger los contenidos mediante usuario/password:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/sites-enabled/site2.conf   Modificado

<VirtualHost 192.168.56.3>
        ServerName Site2
        DocumentRoot /var/www/html/site2
        ErrorLog /var/log/apache2/site2-error.log
        TransferLog /var/log/apache2/site2-access.log

        Options +Indexes +FollowSymLinks +MultiViews


        <Directory /var/www/html/site2/webdav>
                Options +Indexes
                DAV on

                AuthType Basic
                AuthName "Usuario WebDAV"
                AuthUserFile /etc/apache2/webdav.pwd
                Require valid-user
        </Directory>
```

Figura 321. WebDAV activo sobre un directorio

Como vemos, la activación de *WebDAV* simplemente consiste en añadir la directiva `DAV on` dentro del contenedor adecuado. Una vez activa esta configuración, para probar si está funcionando correctamente podemos usar cualquier navegador web y conectarnos a la ruta donde vamos a permitir usar el protocolo *WebDAV*, como aparece en la siguiente imagen:

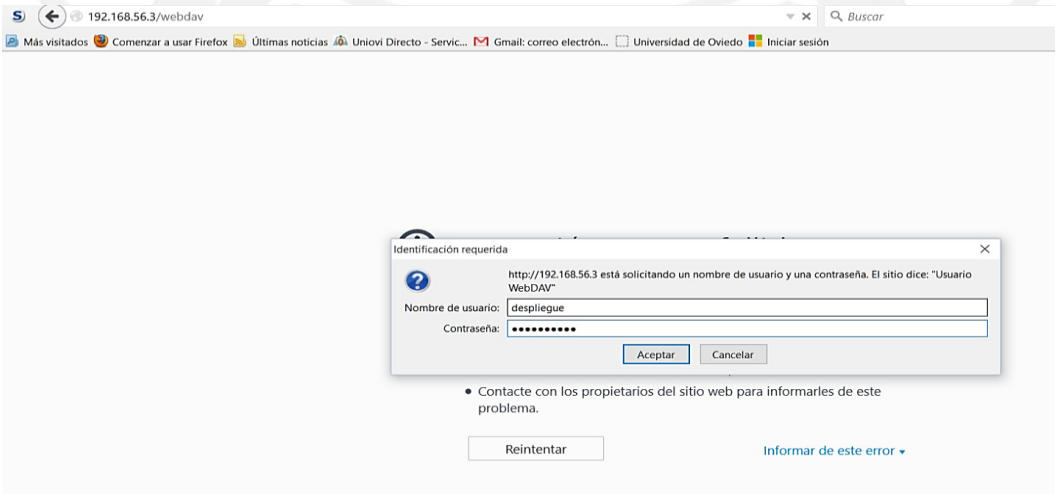


Figura 322. Conexión a un directorio WebDAV desde un navegador

Si introducimos el usuario/password visto anteriormente veremos lo siguiente, confirmando así que la instalación realizada de *WebDAV* funciona:

Figura 323. Directorio con WebDAV habilitado desde un navegador

No obstante, para ver la pantalla anterior se ha activado `mod_autoindex`. En caso de que no lo esté, lo que el navegador vaya a mostrarnos es lo mismo que haría con un acceso normal, siguiendo las reglas que hemos visto sobre el listado de directorios. Obviamente desde un navegador no podemos editar ficheros, para eso es necesario usar otro tipo de clientes, que veremos en la próxima sección. Antes de eso, en la siguiente imagen vemos la variante de la configuración anterior pero restringiendo por máquinas en lugar de por usuarios:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/sites-enabled/site2.conf


<VirtualHost 192.168.56.3>
        ServerName Site2
        DocumentRoot /var/www/html/site2
        ErrorLog /var/log/apache2/site2-error.log
        TransferLog /var/log/apache2/site2-access.log

        Options +Indexes +FollowSymLinks +MultiViews

        <Directory /var/www/html/site2/webdav>
                Options +Indexes
                DAV on
                Require ip 192.168.56.1
                Require ip 192.168.56.2
        </Directory>
```

Figura 324. WebDAV activo restringido solo a determinadas máquinas

Publicar contenido mediante *WebDAV*

En la sección anterior hemos visto que un navegador permite acceder al contenido de un sitio web que admite el protocolo *WebDAV*, pero solo permite leerlo si no cuenta con otras herramientas o complementos adicionales. Para poder editar contenidos en un sitio web que admite este protocolo hace falta una de las herramientas perteneciente a alguna de las categorías mencionadas anteriormente. En esta sección veremos dos.

Cliente *WebDAV cadaver*

`cadaver` (http://www.WebDAV.org/cadaver/) es un cliente *WebDAV* en línea de comandos que permite hacer todo tipo de operaciones sobre sitios web que tengan este protocolo activado. Su instalación en *Ubuntu* es sencilla: `sudo apt-get install cadaver`. Una vez instalado, podemos conectarnos a nuestro directorio *WebDAV* como se ve en la siguiente imagen, donde se supone que no hay usuario/*password* que controle la edición y que la restricción de acceso es por IP de la máquina:

```
operario@ubuntu:~$ cadaver http://192.168.56.3/webdav
dav:/webdav/> ls
Listando colección `/webdav/': exitoso.
        pagina.html                          126  ago 25 13:01
dav:/webdav/>
```

Figura 325. cadaver conectándose a una localización con WebDAV activo

Una vez dentro de la línea de comandos del programa podemos ejecutar determinadas operaciones que nos permitirán manipular / editar los ficheros de la localización. Por ejemplo, si en el directorio donde hemos ejecutado `cadaver` hubiera un fichero llamado `fichero.txt` que queremos subir al directorio *WebDAV* ejecutaríamos lo siguiente:

```
put fichero.txt
```

Apareciendo dicho fichero en el directorio remoto:

```
dav:/webdav/> ls
Listando colección `/webdav/': exitoso.
 Col:   css                                0  ago 25 13:16
        fichero.txt                       37  ago 25 13:15
        pagina.html                      126  ago 25 13:01
dav:/webdav/>
```

Figura 326. Fichero subido por cadaver

Los siguientes comandos crean y acceden a un directorio:

```
mkdir css cd css
```

Y podemos también editar ficheros directamente dentro del directorio activo con:

```
edit plantilla.css
```

Se abrirá un editor tipo `vi` que permitirá editar los ficheros o crearlos nuevos si no existen. Una vez terminada la edición, al cerrar el editor se guardará el fichero automáticamente:

```
"/tmp/cadaver-edit-CMcvDf.css" 1L, 1C escritos
Se realizarion cambios.
Transfiriendo cambios a '/webdav/css/plantilla.css'
 Progreso: [                              ]   0,0% of 1 bytes Progreso: [=======
======================>] 100,0% of 1 bytes exitoso.
Unlocking `plantilla.css': exitoso.
dav:/webdav/css/> ls
Listando colección `/webdav/css/': exitoso.
        plantilla.css                        1  ago 25 13:16
dav:/webdav/css/>
```

Figura 327. Edición de contenidos con cadaver

Al final de todas las modificaciones planteadas, el directorio *WebDAV* tendrá este aspecto:

```
dav:/webdav/css/> cd ..
dav:/webdav/> ls
Listando colección `/webdav/': exitoso.
 Col:   css                                0  ago 25 13:16
        fichero.txt                       37  ago 25 13:15
        pagina.html                      126  ago 25 13:01
dav:/webdav/>
```

Figura 328. Listado de directorio WebDAV con cadaver

Finalmente, si en lugar de por IP el directorio estuviese protegido por usuario/*password*, al arrancar cadáver se nos preguntaría por las credenciales de un usuario autorizado:

```
operario@ubuntu:~$ cadaver http://192.168.56.3/webdav
Autenticación requerida para Usuario WebDAV en el servidor '192.168.56.3':
Nombre de usuario: despliegue
Contraseña:
dav:/webdav/>
```

Figura 329. WebDAV protegido por usuario/password con cadaver

Obviamente, todas las modificaciones que hagamos son visibles por otros clientes y, por supuesto, también por el navegador:

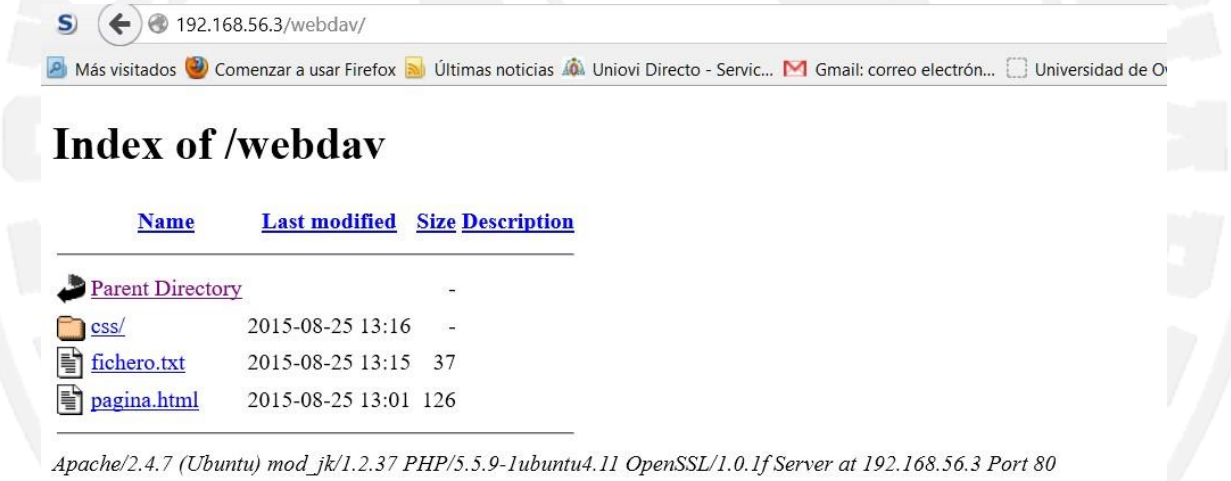


Figura 330. Carpeta editada por WebDAV desde otro cliente

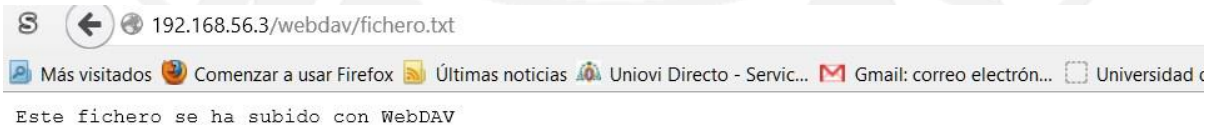


Figura 331. Nuevo contenido publicado por WebDAV accesible por el navegador

Explorador de *Windows*

El explorador de *Windows* también es un cliente *WebDAV* que permite una edición de contenidos muy simple e intuitiva, porque es muy similar a la edición de contenido local. Para ello, simplemente tenemos que introducir la URI de nuestro directorio que admite *WebDAV* (`http://192.168.56.3/WebDAV`) en el explorador, como se ve en la siguiente imagen:

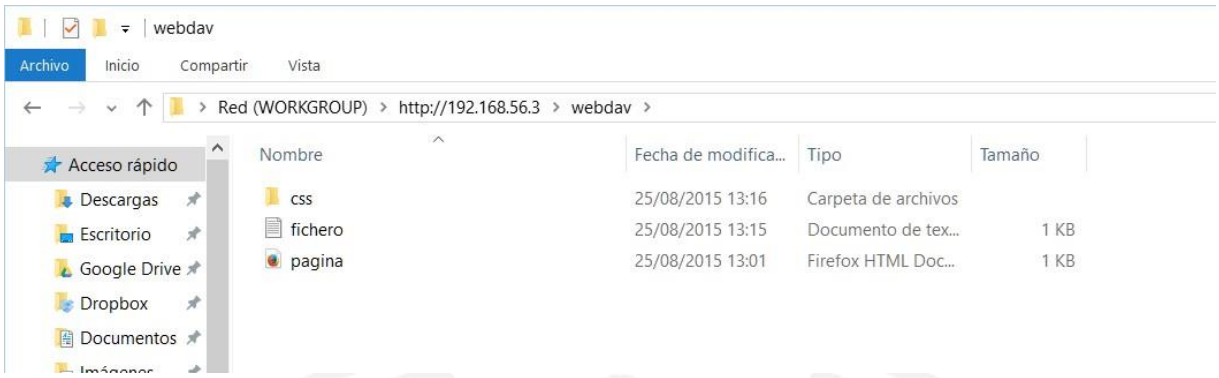


Figura 332. Windows 10 como cliente WebDAV

Como hemos dicho, la edición de contenidos es extremadamente simple, igual que la de un fichero local:

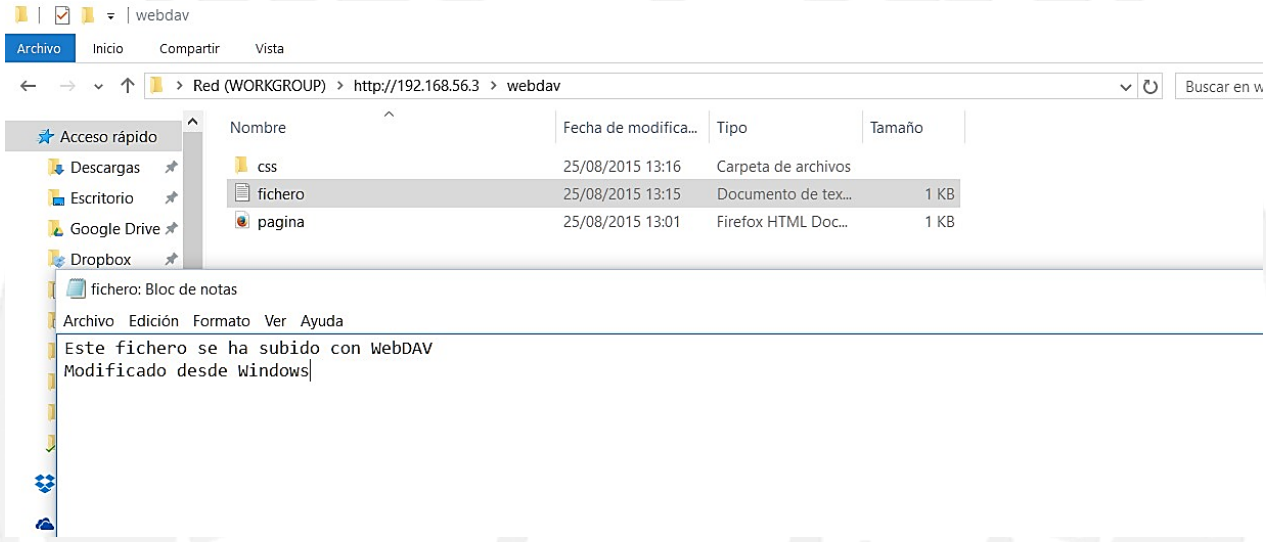


Figura 333. Edición de contenidos desde Windows con WebDAV

Y, por supuesto, la actualización de los contenidos es inmediata y visible al resto de clientes, como en el caso de **cadaver**:

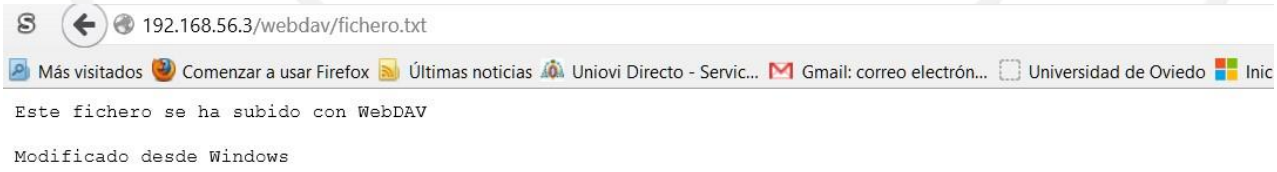


Figura 334. Contenido editado desde Windows

Desgraciadamente, en el momento de escribir este libro no parece que las últimas versiones de Windows sean capaces de gestionar la petición de usuario y clave para un directorio *WebDAV* que los requiera, por lo que este cliente por el momento queda restringido al caso visto.

Protección de contenidos con TLS

Como ya hemos dicho, con las soluciones anteriores estamos transmitiendo tanto los contenidos como usuarios/*passwords* en texto plano, lo que no es en absoluto admisible. La forma de activar TLS sobre un

directorio *WebDAV* no difiere de la ya vista en la sección anterior que hablaba de este tema, así que simplemente describiremos el proceso. Lo primero de todo es crear un certificado válido para el host virtual sobre el que estamos haciendo las pruebas (192.168.56.3):

```
operario@ubuntu:~$ sudo /usr/sbin/make-ssl-cert /usr/share/ssl-cert/ssleay.cnf /
etc/apache2/ssl/apache_site2.pem_
```

Figura 335. Generando un certificado para un host virtual

Debemos recordar especificar el host virtual para el que creamos el certificado a la hora de crearlo:

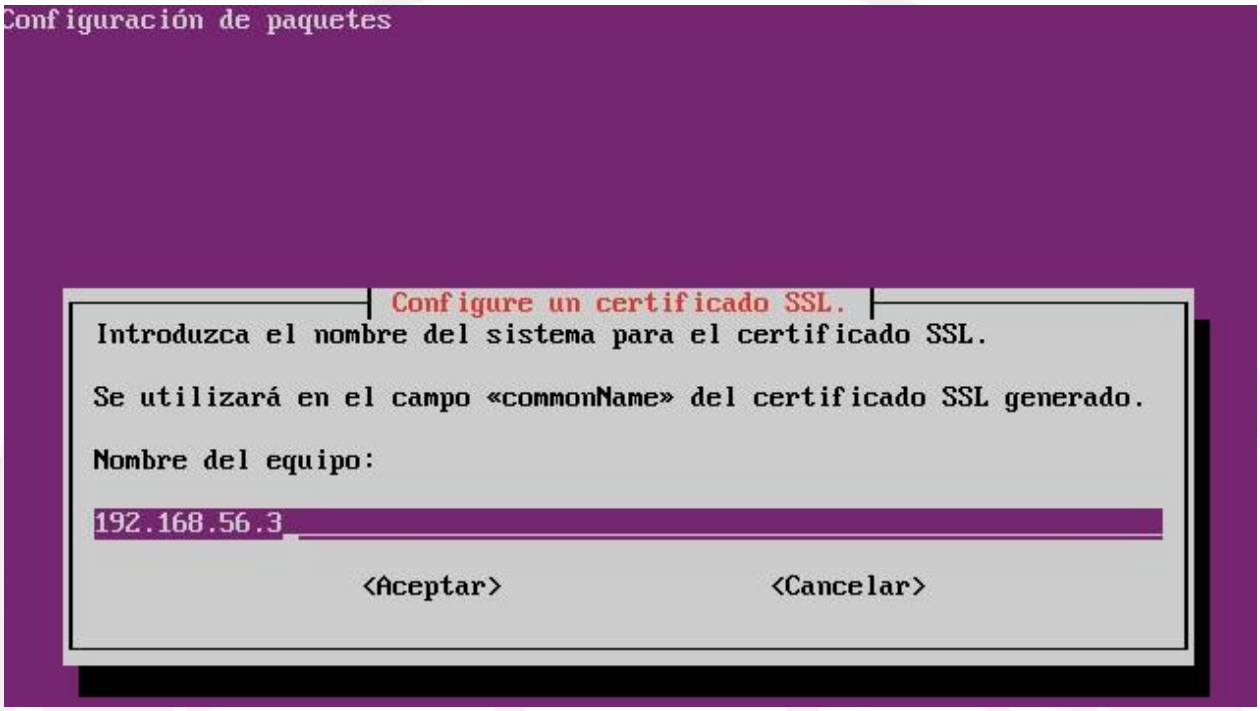


Figura 336. Poniendo el nombre de host adecuado al certificado

Hecho esto, simplemente tenemos que activar el uso de TLS en el host virtual como ya hemos hecho anteriormente:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/sites-enabled/site2.conf   Modificado

<VirtualHost 192.168.56.3>
        ServerName Site2
        DocumentRoot /var/www/html/site2
        ErrorLog /var/log/apache2/site2-error.log
        TransferLog /var/log/apache2/site2-access.log

        Options +Indexes +FollowSymLinks +MultiViews
        SSLEngine on
        SSLCertificateFile /etc/apache2/ssl/apache_site2.pem

        <Directory /var/www/html/site2/webdav>
                Options +Indexes
                DAV on
                Require ip 192.168.56.1
                Require ip 192.168.56.2
        </Directory>
```

Figura 337. Activando TLS con WebDAV

Y finalmente, solo quedaría hacer la pertinente prueba de que puede accederse al contenido mediante HTTPS:

Figura 338. WebDAV bajo TLS

Desgraciadamente, ni el explorador de Windows ni la versión probada de **cadaver** son capaces de gestionar conexiones *WebDAV* bajo la última versión de TLS, por lo que tendremos que buscar un cliente adecuado de entre el listado mencionado anteriormente.

Mejores prácticas en webs de subida de archivos

Hoy en día no es descartable que tengamos que alojar webs que necesiten un espacio para alojar archivos que los propios usuarios suben, ya que es una funcionalidad cada vez más frecuente. No obstante, esto es una operación muy delicada, puesto que una mala programación de esta funcionalidad puede comprometer la seguridad de todo el servidor.

Aunque hablar de buenas prácticas de programación de este tipo de funcionalidades se sale del ámbito de este libro, sí que podemos establecer unas pautas desde el punto de vista de la administración. Por este motivo, es necesario dedicar una sección a describir estos casos, ver los posibles problemas y también soluciones.

Técnicas de ataque

Se conocen varios fallos por los cuales un atacante puede aprovecharse de estas funcionalidades para hacer alguna clase de daño a una web. A continuación enumeraremos los más importantes.

Deficiencias en la programación de la página que permite la subida de archivos

Un programador negligente, o ignorante de los riesgos que corre a la hora de programar estas funcionalidades, puede desarrollar un formulario de subida de archivos que **no compruebe o no lo haga adecuadamente el tipo de archivo que el usuario sube**. Si los archivos subidos luego son accesibles por otros usuarios (algo muy común) y un usuario malicioso sube una web dinámica en un lenguaje admitido por el servidor (`.php`, `.aspx`), cualquier usuario podría visitar dicha web dinámica subida, que se ejecutará con el mismo nivel de privilegio que el resto de las páginas. Como esta web la sube un usuario, puede contener una gran cantidad de ataques / técnicas de robo de información de los usuarios / cualquier otra acción maliciosa.

Básicamente, estamos dejando a cualquiera que suba código que el servidor ejecutará (casi es comparable a dejar al servidor como público para que cualquier suba sus contenidos web), y eso puede tener consecuencias catastróficas, ya que el atacante tiene carta blanca para alojar en nuestro servidor cualquier cosa que desee.

Deficiencias en la validación de tipos MIME

Una de las técnicas más usadas para validar su un usuario está subiendo un tipo de archivo admitido es **comprobar el tipo MIME declarado en los archivos que se suben**. No obstante, no es conveniente confiar únicamente en este dato puesto que, al igual que cualquier otra información que venga del cliente, sea cual sea, el tipo MIME de un archivo puede falsificarse fácilmente por parte del cliente, siendo pues un dato no fiable. Confiar únicamente en él para admitir o no un archivo es una temeridad desde el punto de vista de la seguridad. Esto no quiere decir que no deba comprobarse, sino que más bien no debe ser lo único que se compruebe en los ficheros que se suben.

Deficiencias al gestionar extensiones de archivo

Bloqueo de extensiones

Algunos programadores sin conocimientos de seguridad **hacen una lista de bloqueo de extensiones de archivos prohibidas** para que, si se intenta subir un archivo de alguno de estos tipos al servidor, automáticamente se impida la operación. Como siempre que se usen listas negras, tenemos el problema subyacente de que es posible que la lista sea incompleta y no se bloqueen todos los posibles ficheros peligrosos. Por otro lado, existe un ataque llamado "***.htaccess file upload***" donde el atacante primero sube un fichero `.htaccess` que ha creado con directivas como la siguiente, que le dice a *Apache 2* que interprete los ficheros con extensión `.gif` como si fuese código *JavaScript*:

```
AddType application/javascript .gif
```

Por culpa de este archivo (que ahora estará en el directorio donde se guardan los archivos subidos) se podrá, una vez subido el fichero `.htaccess`, subir posteriormente un fichero con extensión `.gif` pero que realmente sea un fichero de texto con código *JavaScript*, que cualquier usuario que acceda al mismo ejecutará. Por lo tanto, confiar sólo en la extensión de un archivo tampoco es bueno desde el punto de vista de la seguridad.

Por otro lado, este problema de seguridad también hace que tener un fichero `.htaccess` dentro del directorio donde se guardan los archivos subidos con directivas para proteger esta localización de posibles usos maliciosos sea desaconsejable. El comportamiento por defecto de muchos lenguajes de programación que tienen funcionalidades para facilitar la subida de archivos es sobrescribir los ficheros que ya existen. Por tanto, basta con que nos olvidemos de hacer la comprobación de si un fichero existe para que alguien sobrescriba nuestro fichero `.htaccess` por uno propio que elimine estas protecciones.

Múltiples extensiones

A veces la forma de comprobar la extensión de un archivo es **partir el nombre de este por la última aparición del carácter '.'** y quedarse con la segunda parte de la cadena. Esto tampoco es efectivo, puesto que los ficheros pueden tener más de una extensión y el orden en el que se declaran normalmente no es relevante. Por la forma en la que está programado *Apache 2*, es posible que un fichero subido llamado `mifichero.aspx.l33` **se interprete realmente como código ejecutable en ASP .NET**, dado que la última extensión es de un tipo MIME desconocido y *Apache 2* tomaría la primera como referencia para calcular el tipo MIME del fichero subido. Sin embargo si la programación del formulario de subida está hecha como hemos descrito la extensión del fichero sería `.l33`, que no está en la lista negra y, aunque no tiene sentido porque es una extensión inventada, la subida de este fichero se saltará el filtro impuesto por la programación.

En cambio, una aproximación basada en listas de archivos permitidos es mucho más segura, pero puede también plantear problemas. Si en los ficheros de configuración de *Apache 2* existe esta línea en un contenedor que incluye también al directorio que contiene los archivos subidos:

```
AddHandler php-script .php
```

Apache 2 interpretará cualquier fichero que contenga una extensión `.php` como un fichero ejecutable `.php`. Por tanto, sería posible subir un archivo `.php.gif` que pase un filtrado de listas blancas y que aun así se ejecute como script, con las mismas consecuencias que vimos relativas a dejar que un usuario suba una página dinámica al servidor.

Validación únicamente en el cliente

Validar si un archivo a subir es correcto en el cliente no es algo intrínsecamente malo, puesto que **mejora mucho la usabilidad** de esta parte de la aplicación y evita que nos lleguen errores por descuidos que tengamos que procesar en el servidor. **Lo que es un error enorme es validar solamente en el cliente**. **NUNCA podemos suponer en el mundo de la web que con validar en el lado del cliente es suficiente**, puesto que todo lo que reside en la parte del cliente puede verse alterado, corrompido o borrado por el mismo, y por tanto **NO ES FIABLE**. Por ese motivo, si una aplicación tiene un formulario de subida de ficheros que valida dichos ficheros con *JavaScript* y en el servidor no los vuelve a comprobar porque supone que ya vienen validados está cometiendo un **GRAVÍSIMO ERROR DE SEGURIDAD**.

Esto obviamente no solo se aplica a formularios de subida de ficheros, sino que se aplica a cualquier clase de información que nos llega del cliente, sea cual sea su procedencia (parámetros de formularios, cookies, *user agent strings*, etc.). El cliente es propietario de la web que ve, y puede editar tanto la web como las peticiones (gracias a proxies web como *Burp* (https://portswigger.net/burp/proxy.html)) para mandarlos lo que se le antoje, de ahí que tengamos que validar en el servidor todo lo que nos llega, donde realmente si tenemos control de dicha información.

Técnicas de defensa

Dicho todo lo anterior, se enumeran a continuación una serie de técnicas **para protegernos de estos ataques desde el punto de vista de un administrador**. Lo ideal es intentar aplicarlas todas:

- Mediante directivas, definir para la localización donde se guardan los ficheros subidos qué archivos son accesibles y qué archivos no. Por ejemplo, la siguiente configuración solo permite al acceso a unos pocos tipos de imágenes dentro del directorio `/uploads`:

```
 GNU nano 2.2.6        Archivo: /etc/apache2/web1.conf

<Directory "/var/www/html/uploads">
        Require all denied
        <FilesMatch "\.(gif|jpe?g|png)$">
                Require all granted
        </FilesMatch>
</Directory>
```

Figura 339. Protección típica para un directorio donde se suben archivos

- Si se define un fichero `.htaccess` con directivas de protección como las vistas, **nunca ponerlo en el mismo directorio donde se suben los archivos**, debe ponerse en el directorio padre.
- Si es posible, **hacer que el directorio donde se suban los archivos esté aislado** del resto de contenidos web
- **No permitir que se sobrescriban archivos** con una adecuada asignación de permisos.
- Crear una **lista de tipos MIME permitidos** y derivar de ella posibles extensiones admitidas, como comprobación extra a todas las demás.
- **Dar un nombre aleatorio a los ficheros subidos**, y asignarle manualmente la extensión correspondiente a su tipo MIME declarado.
- Incluir validación en el cliente de los ficheros y **TAMBIÉN EN EL SERVIDOR**.

Geolocalización de clientes con *mod_geoip*

Este módulo permite, a partir de la IP de cada cliente que hace una petición, **identificar el país y región de la que proviene este cliente** a partir de una base de datos que el propio módulo posee. La principal ventaja del módulo, además de poder tener una serie de valores en variables de entorno que den datos de la ubicación del cliente a la hora de programar páginas dinámicas, es la de poder hacer bloqueos por región (o admisiones por región) a la hora de determinar quién puede y quién no puede acceder a una determinada localización de la web. Puede ser por tanto, una capa más de seguridad en nuestra instalación. La instalación de este *mod* no difiere de las del resto que hemos visto:

```
sudo apt-get install libapache2-mod-geoip
```

Una vez instalado, es necesario ir al fichero de configuración del módulo (`/etc/apache2/modsavailable/geoip.conf`) y asegurarnos de que la directiva `GeoIPDBFile` está activa, quedándonos algo como esto:

```
<IfModule mod_geoip.c>
  GeoIPEnable On
  GeoIPDBFile /usr/share/GeoIP/GeoIP.dat
</IfModule>
```

Una vez reiniciemos *Apache 2* como de costumbre, debemos probar si el módulo está operativo. Una primera prueba es intentar usar las variables de entorno del módulo desde una página dinámica creada al efecto a modo de prueba. Una posible opción sería esta:

Fichero `/var/www/html/geoip/geoip.php`**:**

```
<html>
  <body>
    <?php
      $pais = Apache 2_note("GEOIP_COUNTRY_NAME"); print "Pais: " . $pais;
    ?>
  </body>
</html>
```

Una vez accedamos a esta página deberíamos ver el país del cliente que la visita. Pero sólo con esto no podemos hacer gran cosa, por lo que ahora veremos algunos aspectos más del módulo que nos ayudarán a usarlo para operaciones más complejas. Ha de tenerse en cuenta que en nuestra infraestructura de pruebas **no podemos hacer pruebas con** `mod_geoip` **porque estamos usando direcciones de red local** (`192.168.X.X`), que obviamente no van a pertenecer a ningún país. Lo que si podemos hacer es comprobar que el módulo funciona adecuadamente con una web como esta:

```
 GNU nano 2.2.6                Archivo: geo.php

<html>
        <head>
                <title>Geolocalizacion</title>
        </head>
        <body>
        <?php
                $pais = apache_note("GEOIP_ADDR");
                print "<p>Te has conectado desde: $pais</p>";
        ?>
        </body>
</html>
```

Figura 340. Prueba de mod_geoip

Que, en caso de que el módulo esté funcionando correctamente, nos devolverá esto:

Figura 341. Variable de entorno de mod_geoip

Por tanto, para poder experimentar con este módulo necesitamos que nuestra máquina virtual tenga una IP pública, ya que en caso contrario, aunque `mod_geoip` esté correctamente instalado, solo obtendremos variables de entorno vacías.

Variables de entorno

Además de la variable `GEOIP_COUNTRY_NAME` del ejemplo anterior, el uso del módulo permite el uso de las siguientes variables de entorno. Las variables de entorno se organizan en grupos y, si tienen un valor correcto, permiten saber muchas cosas acerca del cliente que nos visita:

Variables con información del país

Variable	Información que contiene
`GEOIP_ADDR`	La dirección IP del cliente, que se usará para calcular el resto de información
`GEOIP_CONTINENT_CODE`	Código de dos caracteres que identifica el continente asociado con la IP: • **AF**: África • **AS**: Asia • **EU**: Europa • **NA**: América del norte • **OC**: Oceanía • **SA**: América del sur

GEOIP_COUNTRY_CODE	Código de dos caracteres según el estándar ISO 3166-1 para identificar el país al que pertenece la IP. Se puede consultar una tabla de países en la siguiente dirección: https://en.wikipedia.org/wiki/ISO_3166-1. Además, podría devolvernos alguno de estos códigos adicionales: • **A1:** La IP pertenece a un proxy anónimo • **A2**: La IP es de un proveedor por satélite • **EU:** La IP pertenece a una IP usada por varios países europeos distintos • **AP:** La IP pertenece a una IP usada por varios países asiáticos/del pacífico distinto**s** • US: La IP está asociada a una base militar estadounidense fuera del territorio estadounidense.
GEOIP_COUNTRY_NAME	Nombre del país asociado con la IP

Variables con información de la región

Variable	Información que contiene
GEOIP_REGION	Código de dos caracteres, siguiendo el estándar ISO-3166-2 o FIPS 10-4, y que identifica al estado o región del que proviene la IP. Los códigos para el primer estándar se listan aquí: https://en.wikipedia.org/wiki/ISO_3166-2. El segundo estándar fue retirado hace pocos años, aunque tiene todavía uso. Además, la variable podría tener tres valores especiales adicionales: • **AA**: Fuerzas armadas americanas • **AE**: Fuerzas armadas europeas • **AP**: Fuerzas armadas de países del pacífico El siguiente enlace proporciona una CSV con un listado de códigos de ambos estándares para automatizar su procesamiento: http://www.maxmind.com/download/geoip/misc/region_codes.csv
GEOIP_REGION_NAME	Nombre de la región asociado con la IP

Variables con información de la ciudad

Variable	Información que contiene
GEOIP_CITY	Nombre de la cuidad asociada con la IP. El siguiente enlace tiene una lista de cuidades: http://www.maxmind.com/GeoIPCity-534-Location.csv
GEOIP_METRO_CODE	El "*metro code*" asociado con la IP. Este valor solo se usa si la IP pertenece a una cuidad de los estados unidos
GEOIP_AREA_CODE	El prefijo telefónico del área asociada con la IP. Este valor solo se usa si la IP pertenece a una cuidad de los estados unidos

GEOIP_LATITUDE	Latitud geográfica del origen de la IP
GEOIP_LONGITUDE	Longitud geográfica del origen de la IP
GEOIP_POSTAL_CODE	Código postal asociado con el origen de la IP, aunque este dato solo está disponible para IPs de ciertos países

Otras variables

Variable	Información que contiene
GEOIP_ISP	El nombre del proveedor de Internet que dio al cliente su dirección IP
GEOIP_ORGANIZATION	Nombre de la organización asociada con la IP
GEOIP_NETSPEED	Velocidad de red asociada con la dirección IP. Los posibles valores son: • Dialup • Cable/DSL • Corporate • Cellular

Ejemplos de uso

A continuación veremos varios ejemplos de uso del módulo con diferentes fines útiles. Las directivas de este módulo pueden habilitarse a nivel de servidor completo, directorio o localización. La activación condicional (para que se procesen las directivas en función de si está instalado) se realizaría con `<IfModule mod_geoip.c>`

Redirección de clientes según su país de origen

```
GeoIPEnable On
GeoIPDBFile /path/to/GeoIP.dat

# Redigir un pais (CA)
RewriteEngine on
RewriteCond %{ENV:GEOIP_COUNTRY_CODE} ^CA$
RewriteRule ^(.*)$ http://www.canada.com$1 [R,L]

# Redirigir varios paises (CA, US, MX)
RewriteEngine on
RewriteCond %{ENV:GEOIP_COUNTRY_CODE} ^(CA|US|MX)$
RewriteRule ^(.*)$ http://www.northamerica.com$1 [R,L]
```

Bloqueo de países basándose en su país de origen (lista de bloqueo)

En este caso se han elegido Rusia y China por el alto porcentaje de peticiones maliciosas que provienen de esas regiones:

```
GeoIPEnable On
```

```
GeoIPDBFile /path/to/GeoIP.dat

SetEnvIf GEOIP_COUNTRY_CODE CN BlockCountry
SetEnvIf GEOIP_COUNTRY_CODE RU BlockCountry

Deny from env=BlockCountry
```

Permitir clientes solo de ciertos países (lista de permitidos)

```
GeoIPEnable On
GeoIPDBFile /path/to/GeoIP.dat

SetEnvIf GEOIP_COUNTRY_CODE US AllowCountry
SetEnvIf GEOIP_COUNTRY_CODE CA AllowCountry
SetEnvIf GEOIP_COUNTRY_CODE MX AllowCountry

Deny from all
Allow from env=AllowCountry
```

Finalmente, existen bases de datos actualizadas y detalladas de países y ciudades en las siguientes direcciones. No obstante, **estas bases de datos se han vuelto de pago** y es posible que no se puedan encontrar equivalentes gratuitas usables por *Apache 2* en estos momentos. No obstante, se puede probar a buscar los archivos en la *Wayback Machine* por si la versión más actual del mod siguiera admitiéndolos. En caso de que no, conviene mirar la documentación del mod más actual para saber qué orígenes de datos de países y cuidades admite en estos momentos.

```
http://geolite.maxmind.com/download/geoip/database/GeoLiteCountry/GeoIP.dat.gz
http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
```

En el siguiente enlace podemos encontrar un tutorial de cómo instalarlas y usarlas, así como una forma de acceder al API de `mod_geoip` desde PHP: https://wonderproxy.com/guides/geoip-nuts-and-bolts

Bloque 5: Servidores *Apache 2* en Arquitecturas Multiservidor

¿Qué vamos a ver?

Una vez vistas una gran cantidad de opciones de administración y securización sobre un solo servidor web y su SO de base, es hora de pasar a ver posibles opciones **cuando se combinan varios servidores para cumplir una determinada función**. De esta forma se pueden construir distintas arquitecturas que tendrán propósitos diversos para lograr una infraestructura de servidores más compleja y que de un mejor servicio adaptado a diferentes necesidades.

Las arquitecturas multiservidor que veremos están pensadas para **mejorar distintos aspectos** sobre las posibilidades que nos da un solo servidor: seguridad, alta disponibilidad y balanceo de carga principalmente:

- **Balanceo de carga**: Poder contar con varias máquinas hace que podamos dar un mejor servicio si distribuimos las peticiones entre múltiples servidores.
- **Redundancia**: Por otro lado, al tener varios servidores, si alguno deja de estar operativo los otros puedan ocupar su lugar y continuar dando el servicio.
- **Disponibilidad**: Obviamente, tener N servidores gestionando una web también hace que el servidor pueda soportar más peticiones y que sea más difícil deshabilitar el servicio con ataques tipo DoS.

Con estas ventajas en mente, a lo largo de las secciones siguientes describiremos **distintas arquitecturas de servidores**, mostraremos su propósito y, por supuesto, veremos cómo se realizaría su instalación. Para hacer las tareas de instalación necesitamos previamente contar con tres máquinas *Ubuntu Linux*, todas ellas con *Apache 2* instalado de partida. Obviamente, estas infraestructuras se suelen construir con muchas más máquinas, pero uno de nuestros objetivos principales es mantener un mínimo consumo de recursos. Estas máquinas se dividirán además en 2 grupos, con el siguiente cometido:

- **Máquina *frontend*** (`frontend1`): Es un *Ubuntu* con *Apache 2* instalado y actualizado, e inicialmente sin ningún módulo adicional. Podemos decir que esta máquina es la resultante de haber realizado una instalación de un SO junto con una instalación básica de *Apache 2*. Podríamos implementarle diversas opciones de seguridad (anonimización, permisos, optimización de módulos instalados...) pero no lo haremos para centrarnos en el tema de esta sección.
- **Máquinas *backend*** (`backend1` **y** `backend2`): Son máquinas que, partiendo del estado de las máquinas `frontend`, tendrán además **un servidor *Tomcat 7*** conectado con el *Apache 2* instalado, puesto que en parte de las tareas que vamos a realizar vamos a usar la tecnología de desarrollo JSP para mostrar una serie de funcionalidades. Ambas máquinas servirán páginas extremadamente sencillas para estos ejemplos.

Cada máquina tiene de partida las **conexiones de red**: Una NAT para acceder a Internet (DHCP), otra *Host-Only* para que el SO anfitrión pueda ejercer de cliente de cualquiera de ellas (red `192.168.56.X`) y, por supuesto, una conexión de red a una red interna de *VirtualBox* (`intnet`) para que todas puedan ver al resto (red `192.168.23.X`). Obviamente, todas las máquinas **tendrán una IP fija única en dicha red interna**. Al no

contar en la mayoría de los ejercicios con un servidor DNS, podríamos independizar las explicaciones de las IPs concretas usadas si todas las máquinas tuvieran estas líneas al final de su fichero `/etc/hosts`, de manera que podamos direccionarlas por un nombre:

```
192.168.23.50 frontend1
192.168.23.2 backend1
192.168.23.20  backend2
```

Figura 342. Fichero /etc/hosts propuesto para cada máquina

El nombre de las máquinas debe coincidir con lo especificado en estas líneas, por lo que no debemos olvidar cambiarlo si no coincidiese con lo que aparece en un `/etc/hosts`.

NOTA: A lo largo de todas las secciones que vamos a ver, vamos a usar hasta 3 máquinas Linux para diferentes propósitos. Es **MUY IMPORTANTE** que, si se van a reutilizar las mismas máquinas para hacer actividades distintas, cuando acabemos con una la desactivemos, de manera que las máquinas queden "limpias" para empezar la siguiente actividad. De no hacerlo, es muy probable que las acciones realizadas para poner en marcha una determinada infraestructura impidan que la siguiente funcione correctamente y, lo que es peor, haya errores muy difíciles de tratar que simplemente se deban a "restos" de una actividad anterior.

Proxies web con *mod_proxy*

En líneas generales, un servidor proxy es **una máquina a la que un cliente se conecta pidiéndole un determinado servicio** (un archivo o web de otro servidor, etc.). El servidor proxy evalúa la petición de acuerdo con unas reglas que posee y **decide si procesarla y cómo procesarla**. La funcionalidad básica de un proxy es pues la de procesar una petición en nombre de otra entidad. Hay dos clases de proxys, que veremos a continuación.

Proxy forward

Permite el acceso a Internet (u otra red) a un conjunto de clientes, haciendo que todas las peticiones de los clientes a esa red pasen por él. Es, por tanto, una especie de "puesto fronterizo" que controla todo el tráfico que proviene de una red y que va hacia otra (y viceversa), estando el proxy en medio de ambas como "centinela" del tráfico. Un proxy de este tipo permite lograr una serie de ventajas:

- **Seguridad**: La capacidad de un proxy de redirigir peticiones permite usarlo como **puerta de enlace a Internet**. La máquina proxy puede ser la única autorizada a acceder a Internet, de forma que se prohíbe a los clientes hacerlo directamente. Por ello, la máquina proxy es la única expuesta al exterior y se protegen de esta forma los clientes haciéndolos "anónimos".

  La máquina proxy también **puede tener instaladas herramientas de seguridad** como las vistas, para monitorizar el tráfico entrante y saliente (en este caso podría tener sentido, por ejemplo, que `mod_security` monitorizase el tráfico web saliente también, para asegurarse de que ningún servidor de la red interna está enviando peticiones maliciosas, señal de que puede tener alguna clase de malware).
- **Caché**: Los proxys pueden **guardar temporalmente las páginas que sirven** y sus elementos asociados. Gracias a eso la siguiente vez que se haga una petición de la misma página, se enviará una copia de la guardada localmente. Esto mejora el rendimiento de la red (el proxy suele estar en la red local, mucho más rápida que el acceso al servidor externo) y permite sobrecargar menos la conexión a Internet contratada. No obstante, necesita configurarse adecuadamente para que los clientes no empiecen a obtener información muy desactualizada.
- **Filtrado**: El proxy puede tomar decisiones sobre las peticiones que pasan a través de él: Puede **prohibir el acceso** a determinadas webs o contenidos, puede bloquear anuncios o pop-ups bloqueando las URIs de los anunciantes, …

La siguiente imagen muestra la idea básica de cómo se estructura un *proxy forward* típico:

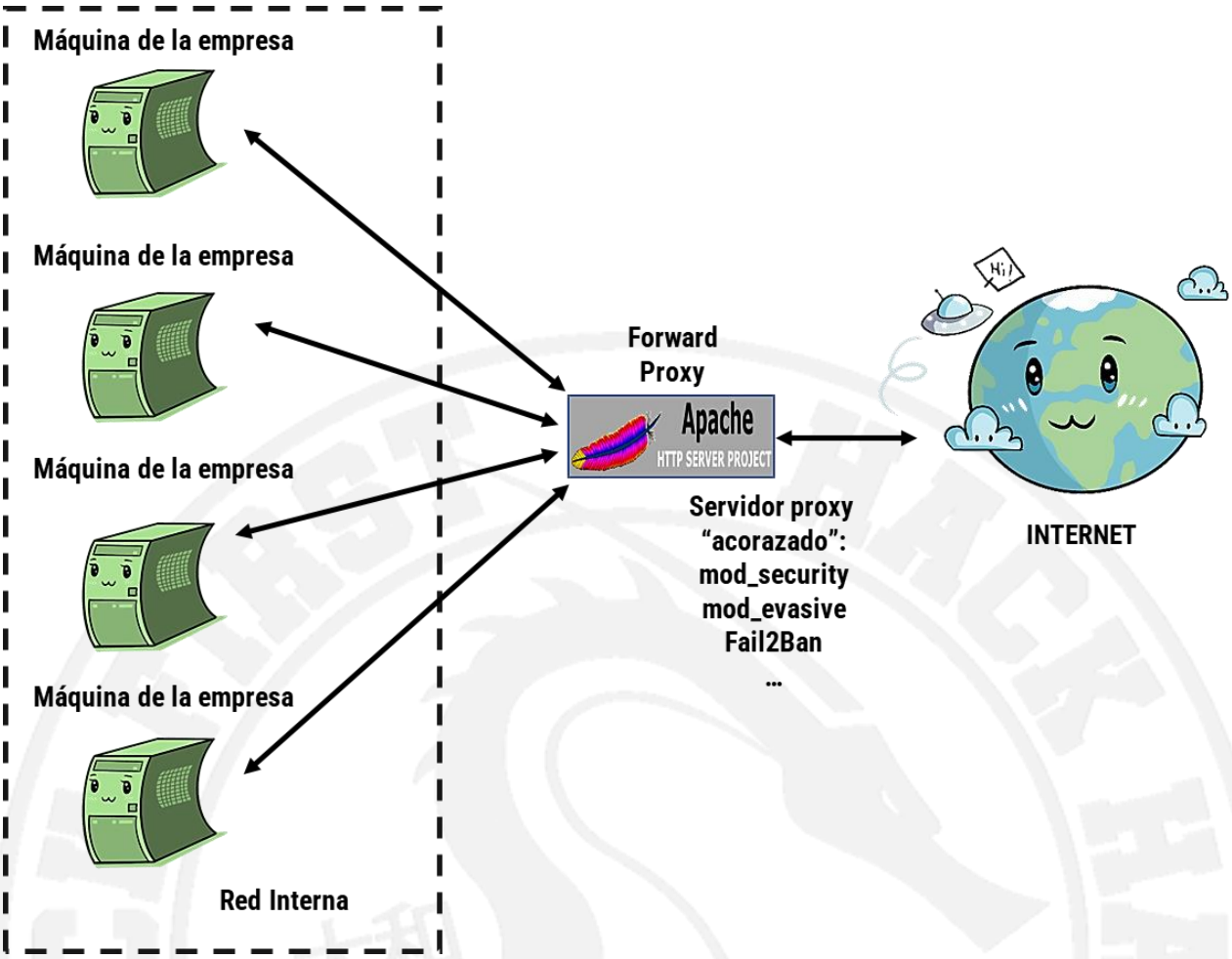


Figura 343. Forward proxy

Reverse Proxy

Este tipo de proxy es similar al anterior, pero **situado del lado de los servidores** más que del lado del cliente. Sigue habiendo **dos redes**: la de los clientes que se conectan (por ejemplo, Internet) y la interna de las máquinas de la empresa. Los clientes siguen haciéndole peticiones al *proxy*, que es la única máquina "expuesta" de nuevo. Este las examina y, **en caso de estar autorizadas**, retorna solo un conjunto de recursos determinado y preconfigurado, obtenido de alguna de las máquinas de la red interna. Por tanto, conectan normalmente con servicios situados en servidores en una red local de una empresa y **ocultan la existencia de esta**, de sus servidores y la arquitectura de la red a los clientes externos.

Gracias a un esquema de este tipo, un cliente puede conectarse por ejemplo al *proxy* pidiendo una aplicación *Tomcat* que realmente está sirviendo un servidor en la red local, pero el cliente y el servidor con el que realmente trabaja **nunca estarán en contacto directo**, es el *reverse proxy* el que hace de "intermediario" (y lo mismo con todo el resto de los servicios que haya en la red interna). De alguna manera puede considerarse al *proxy reverse* como un "espejo" de todos los servicios de la red interna que controla, dejando accesibles la suma de todos ellos (por supuesto, de todos los que la configuración deje como accesibles a la otra red, obviamente).

Por otro lado, otra gran ventaja es que permite la existencia de **varios servidores de la red interna que usen una misma IP pública** para dar servicio sin el uso de *host headers*. También debe tenerse en cuenta que pueden servir también como *firewall* de aplicaciones para protegerse contra ataques web comunes. Estos servidores tienen además las siguientes ventajas:

- **El cifrado TLS** a veces no lo hace el servidor de destino en sí, sino el *reverse proxy* en el que delega estas funciones, que puede tener soporte hardware para ello y por tanto ganar rendimiento. Obviamente, el tráfico entre el *proxy reverse* y los servidores de la red interna **estará sin cifrar**, asumiendo que la red interna es segura.
- **Puede distribuir la carga de las peticiones a varios servidores**. Esto incluye reescribir las URIs de las peticiones para que cada una se vaya a un servidor distinto, en caso de que una misma aplicación esté distribuida en partes en varios servidores. Dicho de otra forma, permite la creación de una ***web farm*** con balanceo de carga y alta disponibilidad, como veremos en una sección posterior.
- **Puede mejorar el rendimiento**, cacheando contenidos estáticos y dinámicos y/o comprimiendo el mismo sin necesidad de que los servidores que realmente sirvan las peticiones se ocupen de esa tarea.
- Se puede usar para **instalar herramientas de seguridad en él** solamente, descargando de esa responsabilidad a cada servidor. Dado que todas las peticiones van a pasar por el *proxy*, si la petición supera las reglas de estas herramientas y se le puede dejar pasar, los servidores de destino tendrán el mismo nivel de seguridad que tendrían si tuvieran instaladas físicamente esas herramientas. También es frecuente que tengan instalado un *firewall* especialmente fortificado. Obviamente, para que esto sea verdad los servidores de la red interna no deben ser visibles desde el exterior.
- **No es necesario pedir una IP pública** para cada servidor de la empresa.

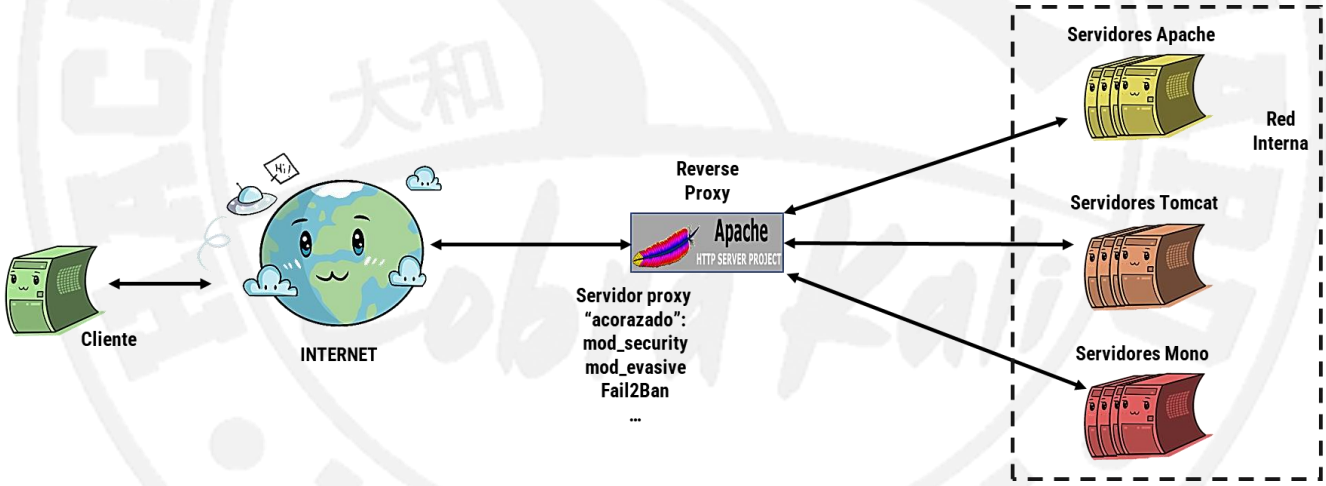


Figura 344. Reverse proxy

Arquitectura de un *proxy*

Un proxy normalmente **se encuentra dentro de la misma red que las máquinas a las que presta servicio**. Para usarlo, es necesario configurar cada cliente de manera que los programas que necesiten el uso de Internet **dirijan sus peticiones al mismo**. Los proxys normalmente se encuentran escuchando en un nº puerto fijo de la máquina que los ejecuta (`8080`, `8181`, `8888`…) y los navegadores tienen un apartado de configuración específico para ello. Para otras aplicaciones, es necesario usar opciones de configuración propias o bien variables de entorno como esta: `export http_proxy="http://proxy.miempresa.es:8080"`

Una **secuencia típica de funcionamiento** de un *proxy forward* es la siguiente:

- Un cliente envía la petición HTTP a la web www.ejemplo.es al servidor que tiene configurado como proxy.
- El proxy recibe la petición y, si está autorizado tanto el cliente como la petición, la envía a www.ejemplo.es.
- www.ejemplo.es responde al proxy. Por tanto, todos los servidores a los que cualquiera de nuestras máquinas de la red interna (que usan un proxy forward) dirijan una petición verán como cliente la IP del proxy.
- El proxy transmite la respuesta al cliente de la red interna que la pidió.

Un ***reverse proxy*** seguirá en cambio la siguiente secuencia:

- La web de una compañía www.muyseguros.com tiene una IP y DNS públicos a los que se puede acceder desde cualquier parte.
- La compañía tiene además una serie de servidores de aplicaciones con una IP y DNS privados detrás de un cortafuegos.
- Estos servidores sólo son visibles desde dentro de la red interna de la compañía, pero no desde el exterior.
- Para acceder a dichos servidores desde el exterior, el reverse proxy se encarga de mapear solicitudes a una dirección concreta a uno de esos servidores internos.
- Por ejemplo, www.muyseguros.com/aplicacióndecontabilidad se mapea a http://192.168.34.2/contabilidad
- La máquina de la red interna sirve la petición y contesta al reverse proxy.
- El *reverse proxy* envía la respuesta al cliente que hizo la petición.

Existen diferentes programas que permiten darnos la funcionalidad de un proxy, pero uno de los más sencillos de manejar es el servidor web *Apache 2*. ***Apache 2* puede hacer de proxy FTP o HTTP**, pudiendo añadir otros protocolos mediante módulos. Todo esto es posible gracias al módulo `mod_proxy` (http://httpd.apache.org/docs/current/mod/mod_proxy.html).

Instalación de un *proxy forward*

Para instalar un *proxy forward* necesitamos un *Apache 2* (preferiblemente con una instalación básica por defecto, para no mezclar múltiples servicios) al que se le haya activado previamente `mod_proxy` de la misma forma que lo hemos hecho con otros módulos (`sudo a2enmod proxy`). Hecho esto, hay algunos parámetros que debemos configurar en el fichero de configuración del módulo para la puesta en marcha definitiva del proxy: `ProxyRequest "On"` configura *Apache 2* como un *forward proxy*.

- **Control de acceso**: Los clientes autorizados a usar el *proxy* se especifican con el esquema `allow,deny` típico de *Apache 2* o bien con la directiva `Require`, en función de qué módulo de autenticación de los dos vistos usemos.
- **Debemos ser muy estrictos con la restricción de clientes**, ya que, si dejamos demasiado abierto el proxy, podría usarse para enviar spam u otros usos no lícitos. Por tanto, es extremadamente importante no dejar nunca un proxy abierto.

Un ejemplo de una posible configuración sería:

```
ProxyRequests On
<Proxy *>
```

```
    Order deny,allow
    Deny from all
    Allow from internal.example.com
</Proxy>
```

Veamos el siguiente ejemplo: supongamos que la máquina backend1 quiere conectarse a Internet a través de `frontend1` en lugar de directamente, tal y como hace ahora, gracias a su interfaz de red `eth0`. Para conseguirlo, lo primero es activar `mod_proxy` en `frontend1` y reiniciar *Apache 2*:

```
operario@frontend1:~$ sudo a2enmod proxy
```

Figura 345. Instalando mod_proxy

Ahora podemos ir al fichero de configuración del módulo y **habilitar este *Apache 2* como *forward proxy*** de la manera que hemos visto antes. Nuevamente, hay que insistir en que es **MUY IMPORTANTE** no olvidarse de restringir las IPs que pueden conectarse a este proxy para que les haga de *forward proxy*. Los *forward proxy* abiertos (sin restricciones) son un riesgo de seguridad, como el propio fichero de configuración ya avisa. En el siguiente ejemplo admitimos que cualquier máquina de la red `192.168.23.X` pueda usar el proxy, pero ninguna más:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/mods-enabled/proxy.conf    Modificado

<IfModule mod_proxy.c>

        # If you want to use apache2 as a forward proxy, uncomment the
        # 'ProxyRequests On' line and the <Proxy *> block below.
        # WARNING: Be careful to restrict access inside the <Proxy *> block.
        # Open proxy servers are dangerous both to your network and to the
        # Internet at large.
        #
        # If you only want to use apache2 as a reverse proxy/gateway in
        # front of some web application server, you DON'T need
        # 'ProxyRequests On'.

        ProxyRequests On
        <Proxy *>
           AddDefaultCharset off
           Require all denied
           Require ip 192.168.23

           #Require local
        </Proxy>
```

Figura 346. Ejemplo de configuración de un forward proxy

Ahora procederemos a indicar, en la máquina `backend1`, dónde está el *proxy* que queremos usar, dándole un valor a la variable del sistema `http_proxy`. Nótese que esto solo durará una sesión, tendríamos que añadirlo a las variables del sistema que se asignan en el arranque si queremos hacerlo permanente (se recomienda consultar https://help.ubuntu.com/community/EnvironmentVariables para ello).

Hay que tener en cuenta por otro lado que esta variable de entorno `http_proxy` es la que usan un conjunto de herramientas para decidir si intentan conectarse vía proxy o no, pero no todas las posibles aplicaciones tienen que hacerlo. Es posible que una aplicación concreta tenga su propio sitio en su configuración particular para indicar si se usa o no un proxy y dónde está, en cuyo caso deberemos asegurarnos de que cada aplicación que usemos tenga la configuración adecuada.

```
operario@backend1:~$ export http_proxy=frontend1
```

Figura 347. Dándole un valor a la variable de entorno http_proxy

Y ahora intentamos hacer una petición desde `backend1` a una dirección de Internet:

```
operario@backend1:~$ wget www.google.es
--2015-08-27 21:17:49--  http://www.google.es/
Resolviendo frontend1 (frontend1)... 192.168.23.50
Conectando con frontend1 (frontend1)[192.168.23.50]:80... conectado.
Petición Proxy enviada, esperando respuesta... 500 Internal Server Error
2015-08-27 21:17:49 ERROR 500: Internal Server Error.
```

Figura 348. Intento de conexión vía proxy

Observamos que, aunque `backend1` ha contactado con `frontend1`, el *proxy* nos devuelve un error 500, indicando que algo ha ido mal en la instalación del *proxy*, como así ha sido a juzgar por lo que vemos en el log de error de frontend1:

```
[Thu Aug 27 21:17:47.865914 2015] [proxy:warn] [pid 1468:tid 140117661894400] [c
lient 192.168.23.2:46587] AH01144: No protocol handler was valid for the URL htt
p://www.google.es/. If you are using a DSO version of mod_proxy, make sure the p
roxy submodules are included in the configuration using LoadModule.
```

Figura 349. Error de mod_proxy en el log de frontend1

Como se indica en el fichero de log, aunque tenemos `mod_proxy` activo, no hemos hecho lo propio **con otro módulo que permite a este entender el tráfico del protocolo HTTP**, por lo que tenemos que activarlo también para que todo funcione. De hecho, existen varios módulos disponibles para habilitar la funcionalidad de *proxy forward* con varios tipos (protocolos) de tráfico, pero obviamente solo debemos activar aquellos protocolos que realmente usemos. En nuestro caso solo necesitamos tráfico HTTP, pero están disponibles los siguientes módulos para el siguiente tipo de tráfico/aplicaciones:

Protocolo	Módulo que lo soporta
AJP13 (*Apache JServe Protocol*, versión 1.3)	mod_proxy_ajp
CONNECT (para SSL)	mod_proxy_connect
FastCGI	mod_proxy_fcgi
FTP	mod_proxy_ftp
HTTP/0.9, HTTP/1.0 y HTTP/1.1	mod_proxy_http
SCGI	mod_proxy_scgi
WS y WSS (*Web-sockets*)	mod_proxy_wstunnel

Por tanto, activamos pues en `frontend1` el módulo `mod_proxy_http`:

```
operario@frontend1:~$ sudo a2enmod proxy_http
Considering dependency proxy for proxy_http:
Module proxy already enabled
Enabling module proxy_http.
To activate the new configuration, you need to run:
  service apache2 restart
```

Figura 350. Activando la capacidad de hacer de proxy para el protocolo HTTP

Y volvemos a probar a hacer la petición, esta vez con éxito:

```
operario@backend1:~$ wget www.google.es
--2015-08-27 21:26:21--  http://www.google.es/
Resolviendo frontend1 (frontend1)... 192.168.23.50
Conectando con frontend1 (frontend1)[192.168.23.50]:80... conectado.
Petición Proxy enviada, esperando respuesta... 200 OK
Longitud: no especificado [text/html]
Grabando a: “index.html.1”

    [ <=>                                   ] 19.583      --.-K/s   en 0,01s

2015-08-27 21:26:21 (1,30 MB/s) - “index.html.1” guardado [19583]
```

Figura 351. Forward proxy en funcionamiento

Y, como es lógico, todo el tráfico de `backend1` a Internet queda registrado en frontend1 para su análisis posterior si fuese necesario (o para que herramientas como *Fail2Ban* puedan hacer su labor):

```
operario@frontend1:~$ tail /var/log/apache2/access.log
192.168.56.1 - - [26/Aug/2015:07:15:15 +0200] "GET / HTTP/1.1" 200 3594 "-" "Moz
illa/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0"
192.168.56.1 - - [26/Aug/2015:07:15:16 +0200] "GET /icons/ubuntu-logo.png HTTP/1
.1" 200 3688 "http://192.168.56.50/" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40
.0) Gecko/20100101 Firefox/40.0"
192.168.56.1 - - [26/Aug/2015:07:15:16 +0200] "GET /favicon.ico HTTP/1.1" 404 50
2 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
"
192.168.56.1 - - [26/Aug/2015:07:15:16 +0200] "GET /favicon.ico HTTP/1.1" 404 50
2 "-" "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
"
192.168.23.2 - - [27/Aug/2015:21:17:47 +0200] "GET http://www.google.es/ HTTP/1.
1" 500 801 "-" "Wget/1.15 (linux-gnu)"
192.168.23.2 - - [27/Aug/2015:21:26:03 +0200] "GET http://www.google.es/ HTTP/1.
1" 500 801 "-" "Wget/1.15 (linux-gnu)"
192.168.23.2 - - [27/Aug/2015:21:26:19 +0200] "GET http://www.google.es/ HTTP/1.
1" 200 20501 "-" "Wget/1.15 (linux-gnu)"
```

Figura 352. Log del proxy forward

En estas circunstancias, la máquina interna `backend1` podría ya prescindir de su interfaz `eth0`, porque no debería ser accesible públicamente en el escenario que hemos planteado.

Control de tráfico

Además del control de las peticiones que pueda hacer cualquier módulo de seguridad de los vistos que instalemos en `frontend1`, una cosa que se puede hacer con la propia funcionalidad de *proxy* es bloquear determinados sitios web por política de empresa o por cualquier otro motivo que queramos. Es típico establecer filtros a nivel de proxy para no dejar a los clientes acceder a determinados sitios web, y ahora vamos a ver cómo hacerlo. **Una nota de advertencia antes de continuar**: no conviene bloquear los servidores de actualizaciones de Ubuntu o podemos estar comprometiendo la seguridad de los *hosts* de la red interna. En general, hay que ser cuidadoso con lo que se bloquea y sobre todo con las molestias que esto pudiera causar. Formas de bloquear hay varias:

- Mediante la directiva `ProxyBlock`: Esta directiva admite una serie de palabras, hosts o dominios separados por espacios. Cualquier petición HTTP, HTTPS o FTP que se haga a sitios cuyos nombres contengan las palabras, hosts o dominios especificados es bloqueada por el *proxy*.

Además, el propio módulo se encarga de obtener las IPs correspondientes a los dominios especificados y cachearlas para hacer un bloqueo más efectivo. El siguiente ejemplo bloquea peticiones de la red interna a las conocidas páginas de *Facebook* y *Twitter*:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/mods-enabled/proxy.conf

<IfModule mod_proxy.c>

        # If you want to use apache2 as a forward proxy, uncomment the
        # 'ProxyRequests On' line and the <Proxy *> block below.
        # WARNING: Be careful to restrict access inside the <Proxy *> block.
        # Open proxy servers are dangerous both to your network and to the
        # Internet at large.
        #
        # If you only want to use apache2 as a reverse proxy/gateway in
        # front of some web application server, you DON'T need
        # 'ProxyRequests On'.

        ProxyRequests On
        <Proxy *>
           AddDefaultCharset off
           Require all denied
           Require ip 192.168.23

           #Require local
        </Proxy>

        ProxyBlock www.facebook.com www.twitter.com
```

Figura 353. Uso de ProxyBlock (I)

Si probamos a acceder a alguno de esos dominios vemos que el *proxy* nos devuelve un error impidiéndonos el acceso:

```
operario@backend1:~$ wget www.facebook.com
--2015-08-27 21:37:47--  http://www.facebook.com/
Resolviendo frontend1 (frontend1)... 192.168.23.50
Conectando con frontend1 (frontend1)[192.168.23.50]:80... conectado.
Petición Proxy enviada, esperando respuesta... 403 Proxy Error
2015-08-27 21:37:47 ERROR 403: Proxy Error.
```

Figura 354. Uso de ProxyBlock (II)

- Mediante contenedores `Proxy` adicionales: Además del contenedor `<Proxy *>` visto, podemos especificar contenedores `Proxy <nombre de dominio o IP>` que establezcan permisos concretos sobre dominios particulares. Por ejemplo, este ejemplo permite el uso del *proxy* por la red de máquinas ya mencionada, pero bloquea el acceso al buscador *Bing*:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/mods-enabled/proxy.conf

        <Proxy *>
           AddDefaultCharset off
           Require all denied
           Require ip 192.168.23

           #Require local
        </Proxy>

        ProxyBlock www.facebook.com hwww.twitter.com

        <Proxy http://www.bing.es>
                Require all denied
        </Proxy>
```

Figura 355. Uso de contenedores Proxy para bloqueo de direcciones

Lo que obviamente pasa si tratamos de hacerlo:

```
operario@backend1:~$ wget www.bing.es
--2015-08-27 21:48:08--  http://www.bing.es/
Resolviendo frontend1 (frontend1)... 192.168.23.50
Conectando con frontend1 (frontend1)[192.168.23.50]:80... conectado.
Petición Proxy enviada, esperando respuesta... 403 Forbidden
2015-08-27 21:48:08 ERROR 403: Forbidden.
```

Figura 356. Bloqueo de webs con un contenedor proxy

Es importante no olvidar el prefijo `http://` (o `https://`) con los contenedores `Proxy`. Y, finalmente, tenemos un contenedor mucho más flexible `ProxyMatch`, que es idéntico al anterior pero que admite expresiones regulares. Por ejemplo, esta directiva prohíbe conectarse a cualquier dominio que lleve la palabra *Microsoft* por HTTP:

```
        <Proxy *>
           AddDefaultCharset off
           Require all denied
           Require ip 192.168.23

           #Require local
        </Proxy>

        ProxyBlock www.facebook.com www.twitter.com

        <Proxy http://www.bing.es>
                Require all denied
        </Proxy>

        <ProxyMatch http://.*microsoft.*\..*>
                Require all denied
        </ProxyMatch>
```

Figura 357. Uso de ProxyMatch para bloqueo de webs

```
operario@backend1:~$ wget www.microfost.com
--2015-08-27 21:54:09--  http://www.microfost.com/
Resolviendo frontend1 (frontend1)... 192.168.23.50
Conectando con frontend1 (frontend1)[192.168.23.50]:80... conectado.
Petición Proxy enviada, esperando respuesta... 302 Found
Ubicación: http://www.microsoft.com [siguiente]
--2015-08-27 21:54:10--  http://www.microsoft.com/
Conectando con frontend1 (frontend1)[192.168.23.50]:80... conectado.
Petición Proxy enviada, esperando respuesta... 403 Forbidden
2015-08-27 21:54:10 ERROR 403: Forbidden.
```

Figura 358. Web bloqueada por ProxyMatch

Una ventaja adicional es que todos los intentos de conexión bloqueados quedan registrados en el log de `frontend1`, para poder "cazar" a los infractores:

```
[Thu Aug 27 21:54:08.016022 2015] [authz_core:error] [pid 2366:tid 1404378449815
04] [client 192.168.23.2:46613] AH01630: client denied by server configuration:
proxy:http://www.microsoft.com/
```

Figura 359. Log de peticiones bloqueadas

Finalmente, si quisiéramos aprovecharnos de las ventajas que los proxys aportan mediante el uso de cachés, deberíamos cargar y consultar la documentación del módulo `mod_cache`: http://httpd.apache.org/docs/current/mod/mod_cache.html.

Instalación de un *reverse proxy*

Como hemos visto, un proxy de estas características permite a un servidor servir contenido que realmente se encuentra en otro de forma transparente. No obstante, el uso más común es proporcionar acceso controlado a servidores web que estén tras un cortafuegos. **Es una «puerta de acceso» a recursos** de una compañía y un método muy común para desplegar aplicaciones web y hacerlas accesibles públicamente sin que las máquinas que realmente dan el servicio sean conocidas.

Como hemos dicho, esta arquitectura propuesta tendrá una máquina que recibirá las peticiones desde una red externa (en nuestro caso nuestro host *Windows*) y la redirigirá a la máquina de la red interna adecuada en función de qué recurso se pida, recogiendo la respuesta y enviándola al cliente. Por tanto, usará `frontend1` y los dos *backends*. Esto supone lo siguiente:

- **No hay comunicación directa entre los posibles clientes y las máquinas de la red interna**, el *proxy* es el único "embudo" por el que necesariamente pasan todas las peticiones. En nuestra arquitectura de ejemplo las máquinas `backend1` y `backend2` podrían deshabilitar la tarjeta de red con IPs de la subred `56` para que no haya comunicación directa con ellas y así que estén realmente aisladas.
- **El proxy `frontend1` debería estar adecuadamente securizado para prevenir intrusiones** (si un usuario malicioso consigue una cuenta de usuario privilegiada en el *proxy* podrá atacar a las máquinas de la red interna). En este sentido las medidas de seguridad serían las mismas que en el caso de un *forward proxy*, ya que en ambos casos hablamos de puntos de contacto únicos con la otra red.

- **Las máquinas de la red interna** podrían implementar todas las medidas de seguridad tomadas en lo relativo a minimización de módulos instalados, permisos y limitación de las peticiones que aceptan solamente, dejando el resto en manos del *proxy*.
- **Se logra la especialización de máquinas**: El *proxy* se encarga de todos los temas relativos a la seguridad más complejos, mientras que el resto puede especializarse en gran medida (dentro de unos límites, puesto que no se trata de dejarlas configuradas de cualquier manera, como dijimos en el punto anterior) en únicamente dar servicio a las peticiones.

Procederemos ahora a hacer la prueba de implementación. Para ello, es conveniente que si `frontend1` es un *forward proxy* desactivemos esa funcionalidad (`ProxyRequests off`), puesto que normalmente no conviene mezclar ambos tipos de *proxy* en la misma máquina. Hecho esto, cambiamos ligeramente las páginas de ejemplo de ambos *backends* para que se distinga correctamente cuándo se está sirviendo contenido de uno o del otro a la hora de hacer las pruebas. En este caso solo vamos a hacerlo con páginas HTML estáticas, pero debe tenerse en cuenta que cualquier servicio que los *backends* estuvieran dando podría ejecutarse de la misma forma, ya que en ese sentido el proxy hace de "espejo" de todos los servidores de la red interna. Las páginas de ambos *backends* son estas:

Figura 360. Página de backend1

Figura 361. Página de backend2

Ahora, en `frontend1`, vamos a hacer un reverse proxy para que cuando se acceda a la ruta `/servicio1` en realidad se acceda a `backend1`, mientras que la ruta `/servicio2` hará lo propio con `backend2`. Esto puede hacerse gracias a la directiva `ProxyPass` de la siguiente forma:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/mods-enabled/proxy.conf

<IfModule mod_proxy.c>

        # If you want to use apache2 as a forward proxy, uncomment the
        # 'ProxyRequests On' line and the <Proxy *> block below.
        # WARNING: Be careful to restrict access inside the <Proxy *> block.
        # Open proxy servers are dangerous both to your network and to the
        # Internet at large.
        #
        # If you only want to use apache2 as a reverse proxy/gateway in
        # front of some web application server, you DON'T need
        # 'ProxyRequests On'.

#       ProxyRequests On
#       <Proxy *>
#          AddDefaultCharset off
#          Require all denied
#          Require ip 192.168.23
#       </Proxy>

        ProxyPass /servicio1 http://192.168.23.2
        ProxyPass /servicio2 http://192.168.23.20
```

Figura 362. Uso de ProxyPass

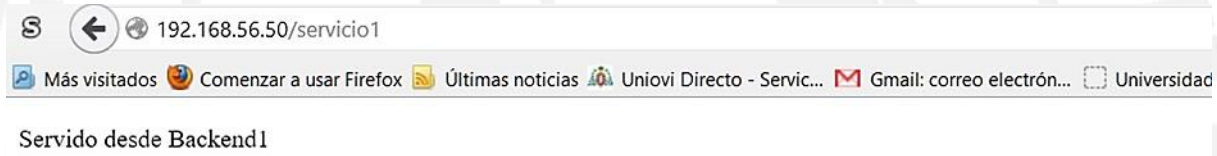


Figura 363. Backend1 tras un reverse proxy

Comprobamos que las rutas establecidas funcionan correctamente:

Figura 364. Backend2 tras un reverse proxy

Nótese que en ningún momento vemos la dirección de la máquina real que sirve las peticiones, sino que es la dirección del proxy lo único que aparece. Por otro lado, si la petición encaja con lo especificado en la directiva `ProxyPass`, el resto de la petición se mapea contra la ruta de la máquina interna. Por ejemplo, solicitar http://192.168.56.50/servicio1/imgs realmente sirve http://192.168.56.2/imgs. `ProxyPass` se puede usar en un contenedor `<Location>` (no en `<Directory>` ni en ficheros `.htaccess`). En ese caso, el primer argumento de `ProxyPass` no hace falta (se toma el de `Location`).

Figura 365. Acceso a contenidos de backend1 a través del reverse proxy

Una variante de la directiva anterior es `ProxyPassReverse`. Esta directiva indica a *Apache 2* que cambie las cabeceras HTTP `Location`, `Content-Location` y URI para que reflejen la URI del proxy en lugar de la de la máquina de la red interna. Esto es de utilidad si se hacen redirecciones en la página servida. Si no se cambian las URIs de estos elementos, entonces la página servida accederá directamente a las máquinas *backend* saltándose el *proxy* (suponiendo, claro está, que sean accesibles "desde fuera". Esta directiva tiene que usarse en conjunto con una directiva `ProxyPass` equivalente para que funcione correctamente:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/mods-enabled/proxy.conf    Modificado

<IfModule mod_proxy.c>

        # If you want to use apache2 as a forward proxy, uncomment the
        # 'ProxyRequests On' line and the <Proxy *> block below.
        # WARNING: Be careful to restrict access inside the <Proxy *> block.
        # Open proxy servers are dangerous both to your network and to the
        # Internet at large.
        #
        # If you only want to use apache2 as a reverse proxy/gateway in
        # front of some web application server, you DON'T need
        # 'ProxyRequests On'.

#       ProxyRequests On
#       <Proxy *>
#          AddDefaultCharset off
#          Require all denied
#          Require ip 192.168.23
#       </Proxy>

        ProxyPass /servicio1 http://192.168.23.2
        ProxyPassReverse /servicio1 http://192.168.23.2
        ProxyPass_/servicio2 http://192.168.23.20
        ProxyPassReverse /servicio2 http://192.168.23.20
```

Figura 366. Uso de ProxyPassReverse

No obstante, esto solo cambiará el valor de las cabeceras HTTP mencionadas. Si lo que queremos es **que nos cambie las URIs presentes dentro del contenido de las páginas web de respuesta**, entonces tendremos que usar el *mod* `mod_proxy_html` (http://httpd.apache.org/docs/current/mod/mod_proxy_html.html). De no hacerlo, cualquier URI absoluta a una máquina *backend* se saltará el *proxy*. El módulo básicamente es un filtro de salida que reescribe enlaces HTML cuando las peticiones se sirven desde detrás de un *proxy*, de manera que nos aseguremos que estos enlaces puedan funcionar desde lugares fuera del *proxy*. Es por tanto un complemento a la funcionalidad de `ProxyPassReverse`, afectando a aquellos elementos que esta directiva no manipula.

A modo de ejemplo, si tenemos un servidor `tomcat.miempresa.es` que solo es accesible desde la red interna de la compañía, y el servidor público (*reverse proxy*) www.miempresa.es, que se usa para acceder a los servicios del primero en la ruta www.miempresa.com/tomcatapp, cualquier enlace que la página tenga a sí misma del estilo `<a href="http://tomcat.miempresa.es/app/index.jsp">Inicio</a>` se cambiará a `<a href="http://www.miempresa.es/tomcatapp/app/index.jsp">Inicio</a>` pudiendo usarse por parte de clientes externos.

Finalmente, las directivas `ProxyPass` también se admiten dentro de contenedores `<VirtualHost>`, siendo además una forma sencilla de habilitar HTTPS con el uso de *proxies*, como en este ejemplo:

```
<VirtualHost *:443>

    SSLEngine On
    SSLCertificateFile /etc/apache2/ssl/file.pem

    ProxyPass / http://192.168.10.10/
    ProxyPassReverse / http://192.168.10.10/

</VirtualHost>
```

Y con esto hemos visto las formas más comunes de usar un proxy en *Apache 2*. Obviamente, se puede profundizar más en su estudio en caso de que tengamos alguna necesidad especial. Si es el caso, se recomienda consultar la documentación oficial para ver el resto de las funcionalidades y directivas: http://httpd.apache.org/docs/current/mod/mod_proxy.html.

Montar una infraestructura *Round-Robin DNS*

Esta arquitectura está pensada como una versión "pobre" de un clúster web o granja de servidores, donde nos aprovechamos de una característica del servicio de DNS para dirigir las peticiones sucesivas que se hagan a una determinada dirección **a un conjunto de máquinas, pero de manera secuencial**. De esta forma, una petición de un determinado recurso puede ser servida por `backend1`, la siguiente por `backend2` y así sucesivamente, aunque el recurso sea el mismo en cada petición.

El servidor DNS se encargará de ir devolviendo las diferentes IPs correspondientes a un mismo nombre en forma de lista pero que varía entre peticiones **con una política *Round-Robin***, creando la ilusión de tener un clúster de máquinas simplemente recorriendo las máquinas disponibles. La principal ventaja de esta arquitectura es que permite gestionar más peticiones por segundo sin necesidad de instalar ningún *software* específico ni máquina adicional si ya tenemos un servidor DNS operativo. Lógicamente, se supone que **todos los *backends* sirven las mismas páginas**.

Sin embargo, no debemos quedarnos con una idea equivocada de esta infraestructura, puesto que si se usa en combinación con balanceadores de carga o granjas web (ver la próxima sección) puede ser una forma muy potente y "barata" de aumentar la capacidad de nuestra infraestructura. El funcionamiento de esta infraestructura se puede resumir en varios pasos:

- Un DNS puede tener **múltiples registros (IPs) para un mismo nombre de dominio**. Por tanto, un DNS puede devolver una lista de IPs para un dominio dado.
- Cuando un navegador pide una dirección para la que le han devuelto una lista de IPs, el navegador **las probará una a una** hasta que la primera le responda.
- Las IPs devueltas pueden corresponder a servidores reales **o bien a balanceadores de carga**. Como veremos en la próxima sección, un balanceador de carga reparte las peticiones que le llegan entre varios servidores idénticos para lograr un mejor servicio.

En caso de que sean IPs de balanceadores de carga, tenemos más ventajas adicionales:

- Si un balanceador es el que le "toca" servir la petición y pierde parte de sus servidores, el resto de los que le queden operativos podrá dar servicio **sin que la infraestructura se vea afectada**. Si está sobrecargado de peticiones y da *Timeout*, entonces, ante la falta de respuesta el navegador se lo solicitará al resto.
- Si de nuevo la petición debe ser atendida por un balanceador y este se encuentra fuera de servicio (anulando así la posibilidad de que todos los servidores que tiene asociados puedan dar servicios), el navegador **lo intentará con el siguiente**.
- Como podemos ver, el hecho de que un balanceador "caiga" es bastante raro, pero el hecho de que lo hagan 2, 3 o más ya es todavía más complicado. Por tanto, Round-Robin DNS + balanceadores de carga (que construiremos uno en la próxima sección) es una infraestructura que asegura fácilmente **una disponibilidad de los servicios muy alta**.

La infraestructura a montar tendría el siguiente aspecto:

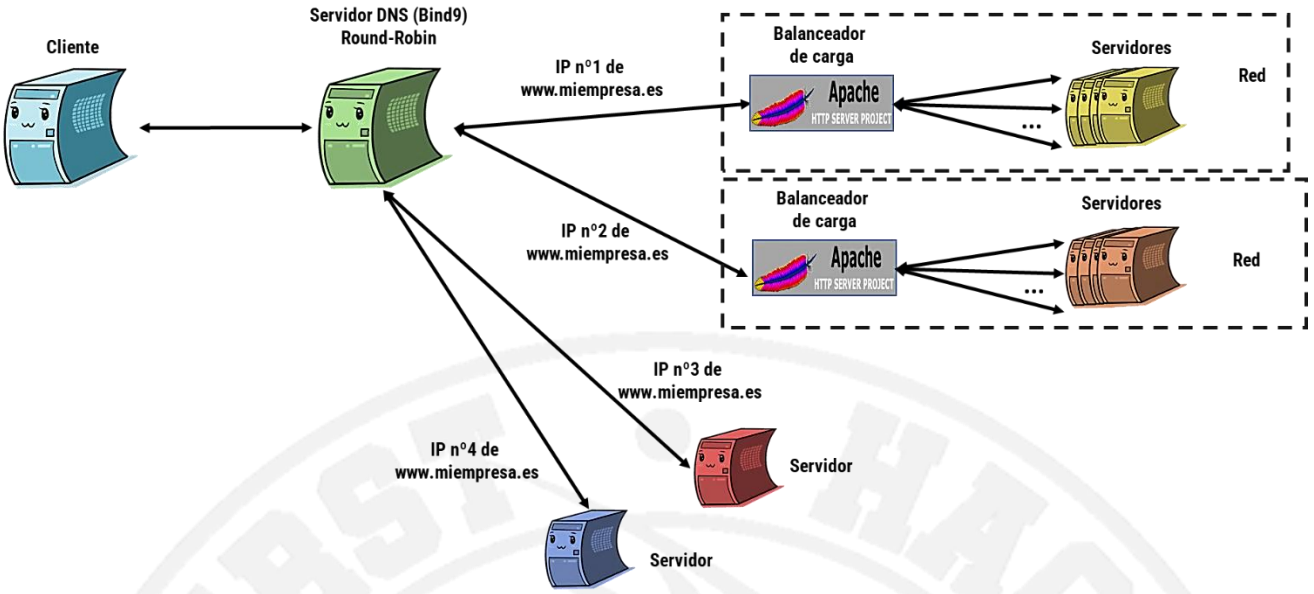


Figura 367. Esquema de una arquitectura Round-Robin DNS

Para hacer la prueba de una parte de esta infraestructura vamos a usar la máquina `frontend1` y los dos *backends* (servidores aislados) para simular que se alterna entre un servicio y otro. Ambos *backends* servirán una página distinta que les identifique, aunque, como hemos dicho, en un caso real las webs servidas deberían ser las mismas.

El primer paso es instalar un servidor DNS en `frontend1`, ya que para que esta infraestructura funcione necesitamos poder manipular el comportamiento de un DNS, algo que no podemos hacer con un DNS externo por no tener acceso a su configuración. De esta manera `frontend1` será capaz de asociar nombres de dominio a IPs a partir de ahora, aunque para nuestras pruebas solo usaremos un par de asociaciones. Implementaciones de DNS hay muchas, pero nosotros usaremos una de las más conocidas, *Bind9* (http://www.bind9.net/), que se instala con: `sudo apt-get install bind9`. No vamos a explicar el funcionamiento de un DNS porque se sale del ámbito de este libro, solo vamos a mencionar la configuración mínima para que funcione para nuestros propósitos:

```
operario@frontend1:~$ sudo apt-get install bind9
```

Figura 368. Instalación de Bind9

Una vez instalado, configuraremos el DNS para que asocie un nombre de dominio a las IPs de los *backends*. En el ejemplo usaremos el nombre `www.rrdnsredondo.org`, y el comportamiento esperado es que vaya alternando entre las IPs de los *backends* en peticiones sucesivas. Para ello a continuación creamos un fichero con la configuración de ese nombre DNS (fichero de zona) en una localización accesible. En nuestro caso lo haremos en `/etc/bind/zones` (la última carpeta debemos crearla). Su nombre debe ser idéntico al del nombre DNS que hayamos puesto y la extensión del fichero será `.db`. A modo de ejemplo, se muestra el siguiente, llamado `rrdnsredondo.org.db`, donde se supone que el servidor DNS está en la IP `192.168.56.50` y los servidores web en la `56.2` y la `56.20`. Los valores numéricos de los parámetros son de ejemplo, aunque válidos. Con este fichero es posible asociar el nombre DNS `www.rrdnsredondo.org` a las IPs mencionadas:

```
$TTL 604800
@     IN SOA ns.rrdnsredondo.org. root.rrdnsredondo.org. (
                  201508288     ; Serial
                    10800             ; Refresh
```

```
                        3600            ; Retry
                       86400            ; Expire
                       86400 ) ; Negative Cache TTL
;
@       IN NS  ns.rrdnsredondo.org.
ns      IN A   192.168.56.50
box     IN A   192.168.56.50
www     IN A   192.168.56.2
www     IN A   192.168.56.20
```

```
 GNU nano 2.2.6    Archivo: /etc/bind/zones/rrdnsredondo.org.db

$TTL 604800
@       IN      SOA     ns.rrdnsredondo.org.    root.rrdnsredondo.org. (
                                201508288       ; Serial
                                10800           ; Refresh
                                3600            ; Retry
                                86400           ; Expire
                                86400)          ; Negative cache TTL
;
@       IN      NS      ns.rrdnsredondo.org.

ns      IN      A       192.168.56.50
box     IN      A       192.168.56.50
www     IN      A       192.168.56.2
www     IN      A       192.168.56.20
```

Figura 369. Fichero de configuración de Bind9

A continuación, crearemos un fichero de búsqueda DNS inversa en la misma localización anterior para nuestra nueva zona que sea como el mostrado, donde se usa el nombre DNS del ejemplo anterior. El nombre del fichero será "`rev`" seguido de la subred con la que estamos trabajando, acabando en in-"`addr.arpa`" (en nuestro caso, `rev.56.168.192.inaddr.arpa`). Este fichero permite asociar las IPs mencionadas al nombre DNS que estamos usando:

```
$TTL 604800
@       IN SOA ns.rrdnsredondo.org. root.rrdnsredondo.org. (
                      201508288     ; Serial
                        604800            ; Refresh
                         86400            ; Retry
                       2419200            ; Expire
                        604800 ) ; Negative Cache TTL
;
@       IN NS ns.
50 IN PTR ns.rrdnsredondo.org
```

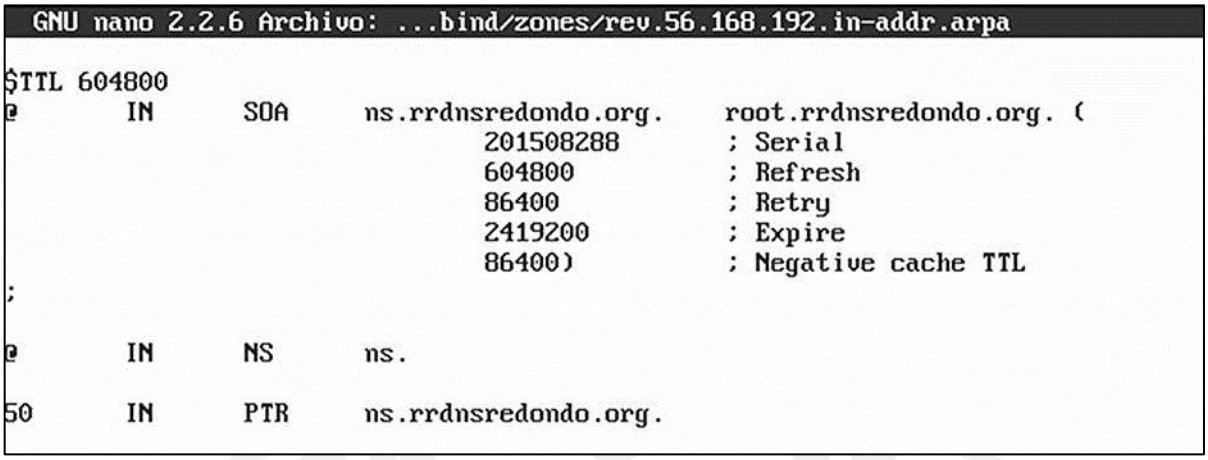


Figura 370. Configuración de un fichero de búsqueda inversa para Bind9

Finalmente, tenemos que modificar el fichero `named.conf.local` del directorio `/etc/bind9` para crear un vínculo a los ficheros anteriores añadiendo unas líneas como estas:

```
//
// Do any local configuration here
//

// Consider adding the 1918 zones here, if they are not used in your
// organization
//include "/etc/bind/zones.rfc1918";

zone "rrdnsredondo.org" {
    type master;
    file "/etc/bind/zones/rrdnsredondo.org.db";
};
zone "56.168.192.in-addr.arpa" {
    type master;
    file "/etc/bind/zones/rev.50.168.192.in-addr.arpa";
};
```

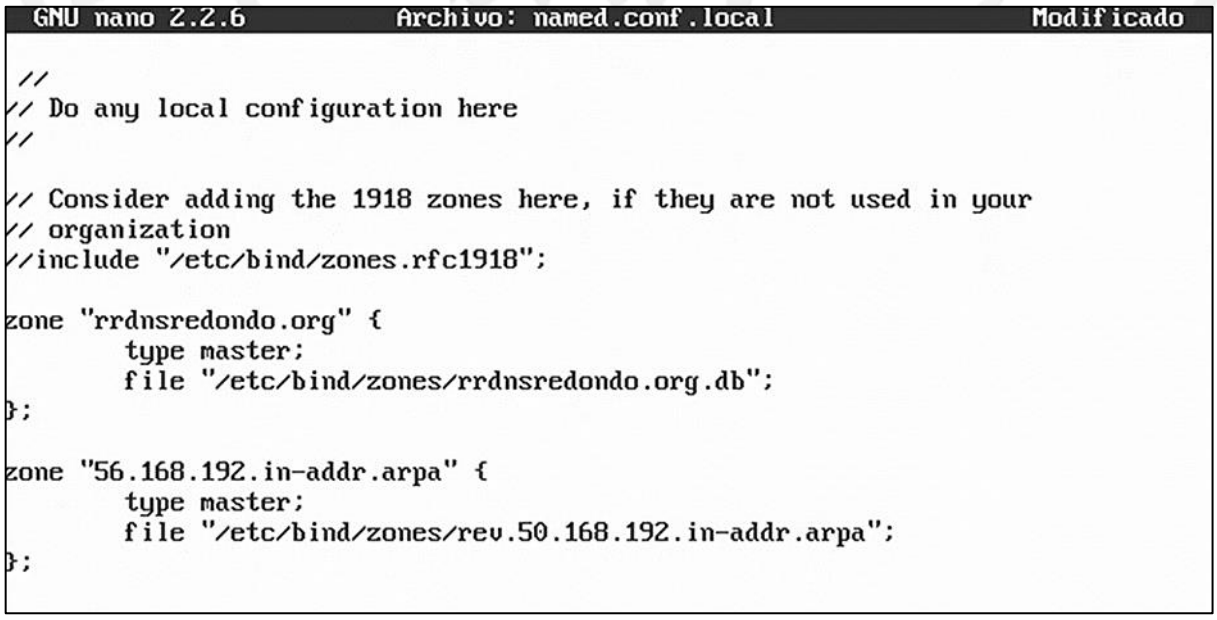


Figura 371. Finalizando la configuración de Bind9 con una nueva zona

Para acabar con la configuración de *Bind9* se debe reiniciar el servidor DNS para que adquiera la nueva configuración: `sudo /etc/init.d/bind9 restart`

Pruebas de funcionamiento

Para ahorrar recursos, las pruebas de funcionamiento pueden hacerse desde la misma máquina que instalo el servidor DNS. Para ello debemos configurar el servidor para que use el servidor DNS recién instalado modificando el archivo `/etc/resolv.conf` de forma que haya una entrada `nameserver` que tenga la IP de nuestro servidor (en el ejemplo sería `192.168.56.50`, la suya propia). Como estamos actuando sobre la red `56` (NAT, con salida a Internet), habrá otros servidores DNS ya asignados. Vamos a comentarlos temporalmente para hacer la prueba de funcionamiento:

```
 GNU nano 2.2.6          Archivo: /etc/resolv.conf                Modificado

# Dynamic resolv.conf(5) file for glibc resolver(3) generated by resolvconf(8)
#     DO NOT EDIT THIS FILE BY HAND -- YOUR CHANGES WILL BE OVERWRITTEN
#nameserver 212.89.28.19
#nameserver 212.89.0.77

nameserver 192.168.56.50
```

Figura 372. Asignando un nuevo DNS

El fichero `.conf` mencionado se modifica cada vez que la máquina se arranca o bien cada vez que se reinicia el sistema de red, ya que el servicio de obtención de direcciones por DHCP guarda ahí su configuración sobrescribiendo lo que hubiese anteriormente. Por tanto, esta forma de hacerlo **es solo temporal** y debe repetirse si se hace alguna de las operaciones mencionadas que sobrescriben este fichero. Se recomienda no obstante guardar copia para restaurarlo en caso de necesidad. Si quisiéramos asignar un DNS permanentemente deberíamos modificar `/etc/network/interfaces` como aparece en el siguiente cuadro, separando las IPs de los DNS por espacios en blanco si tuviéramos varias. En nuestro caso dejamos sin modificar este fichero porque es para una prueba puntual de la infraestructura.

```
auto eth0
iface eth0 inet static
address 192.168.X.X
netmask 255.255.255.0
gateway 192.168.X.X
dns-nameservers X.X.X.X Y.Y.Y.Y
```

Ahora ya estamos preparados para la prueba final, para lo cual, desde la máquina servidora DNS, ejecutamos `nslookup <nombre del domino>` (en el ejemplo www.rrdnsredondo.org). Vemos que las IPs devueltas en peticiones sucesivas cambian, permitiendo alternar entre servidores.

```
operario@frontend1:/etc/bind$ nslookup www.rrdnsredondo.org
Server:         192.168.56.50
Address:        192.168.56.50#53

Name:   www.rrdnsredondo.org
Address: 192.168.56.20
Name:   www.rrdnsredondo.org
Address: 192.168.56.2

operario@frontend1:/etc/bind$ nslookup www.rrdnsredondo.org
Server:         192.168.56.50
Address:        192.168.56.50#53

Name:   www.rrdnsredondo.org
Address: 192.168.56.2
Name:   www.rrdnsredondo.org
Address: 192.168.56.20

operario@frontend1:/etc/bind$ nslookup www.rrdnsredondo.org
Server:         192.168.56.50
Address:        192.168.56.50#53

Name:   www.rrdnsredondo.org
Address: 192.168.56.20
Name:   www.rrdnsredondo.org
Address: 192.168.56.2
```

Figura 373. Prueba de una infraestructura Round-Robin DNS

Obviamente, aquí lo único que hemos hecho es usar la funcionalidad básica de *BIND9*. Para ampliar los conocimientos de cómo manejar un servidor DNS, se recomienda consultar:

- https://www.isc.org/wp-content/uploads/2014/01/B99ARM.pdf
- https://help.ubuntu.com/community/BIND9ServerHowto#Primary%20Master%20Server
- http://www.linuxhomenetworking.com/wiki/index.php/Quick_HOWTO_:_Ch18_:_Configuring_DNS

Clúster *Load-Balanced* con *Apache 2* bajo *Linux*

La arquitectura que vamos a presentar se conoce como clúster de servidores web `loadbalanced` (LB) o granja de servidores web, y consiste en que una máquina con *Apache 2* (`frontend`) hará de balanceador de carga de trabajo de N máquinas (`backends`) que proveen un determinado servicio web (también con *Apache 2* en nuestro caso, aunque no tiene por qué), de manera que las peticiones que vayan llegando se distribuyan a una máquina o a otra en función del algoritmo de reparto de carga de trabajo que el balanceador use.

Para que esta arquitectura tenga sentido, **todos los nodos *Apache 2 backend* deberían servir la misma página web**, para que el cliente no sepa quién es el que realmente atiende las peticiones. No tendría sentido que una misma web fuese diferente según el servidor que la sirva en este escenario. El siguiente esquema muestra la arquitectura a montar:

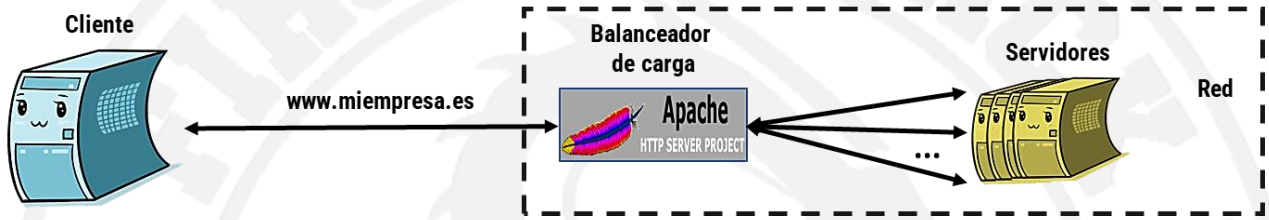


Figura 374. Esquema básico de un balanceador de carga

Las principales ventajas de esta arquitectura son aumentar la capacidad de procesar peticiones de toda la arquitectura (especialmente si añadimos más nodos servidores a la arquitectura) y el aumento de disponibilidad (si un *backend* cae los otros podrán seguir dando servicio). Para implementar esta arquitectura sobre nuestra infraestructura de pruebas, vamos a usar la máquina `frontend1` como balanceador de carga, que distribuirá las peticiones que le lleguen a los servidores *Apache 2* de `backend1` y `backend2`.

La forma de hacerlo es gracias al módulo de *Apache 2* `mod_proxy_balancer` (http://httpd.apache.org/docs/current/mod/ mod_proxy_balancer.html). Este módulo **convierte a un servidor *Apache 2* en un balanceador de carga con todas sus funcionalidades**, permitiendo servir las páginas desde cualquiera de los N servidores web del *backend* que tiene asociados y sirviéndolas al cliente sin que este sea consciente de que está tratando con una granja de servidores. El módulo `mod_proxy_balancer` necesita el módulo `mod_proxy` ya visto. Es más, cuando hablábamos de los reverse proxy, comentábamos que una de sus ventajas es que permitía si era necesario repartir la carga de trabajo entre varios servidores de la red interna. Esta es la implementación concreta de dicha ventaja. Además de HTTP, soporta balanceo de carga para los protocolos FTP y AJP13.

No obstante, `mod_proxy` y `mod_proxy_balancer` no pueden funcionar sin que se añada un tercer tipo de módulo: aquél que especifica el algoritmo que se debe usar para decidir qué servidor del *backend* es el que se encarga de atender cada petición que llega al balanceador.

Algoritmos de balanceo de carga

Actualmente existen tres algoritmos diferentes, implementados por módulos también, que permiten configurar cómo se hace el balanceo de carga. Estos son:

- ***Request Counting***: `mod_lbmethod_byrequests`
- ***Weighted Traffic Counting***: `mod_lbmethod_bytraffic`
- ***Pending Request Counting***: `mod_lbmethod_bybusyness`

El algoritmo a usar se controla con la directiva `lbmethod`, que se usará cuando se defina el clúster, como veremos al final de esta sección.

Algoritmo *Request Counting*

Se habilita mediante `lbmethod=byrequests`, siempre que el módulo que la define esté habilitado. Este algoritmo funciona distribuyendo las peticiones entre los servidores del balanceador **asegurándose de que cada uno obtiene la parte de las peticiones totales para la cual se ha configurado previamente**. A este parámetro se le llama `lbfactor`, siendo un valor normalizado. Existe otro parámetro, llamado `lbstatus`, que mide como de "urgente" es la necesidad de admitir trabajos de alguno de los nodos, indicando cuál es aquel que lleva menos tiempo sin admitir trabajos. La petición se asigna distribuyéndola según el `lbfactor` de cada servidor, disminuyendo el valor de `lbstatus` a los servidores que admiten peticiones.

Más información: http://httpd.apache.org/docs/current/mod/mod_lbmethod_byrequests.html

Algoritmo *Weighted Traffic Counting*

Se habilita mediante `lbmethod=bytraffic`, siempre que el módulo que la define esté habilitado. Es similar al anterior, pero `lbfactor` significa **cuánto tráfico, en bytes, está dispuesto a admitir un servidor**, siendo un valor normalizado. La principal diferencia con el método anterior es que no cuenta nº de peticiones, sino la cantidad de tráfico recibida por cada servidor.

Más información: http://httpd.apache.org/docs/current/mod/mod_lbmethod_bytraffic.html

Algoritmo *Pending Request Counting*

Se habilita mediante `lbmethod=bybusyness`, siempre que el módulo que la define esté habilitado. Este algoritmo cuenta cuántas peticiones tiene asignadas y pendientes de servir cada servidor del clúster. Las nuevas peticiones se asignan automáticamente a aquel servidor que tenga menos peticiones pendientes. En caso de empate, se resuelve a qué servidor se asigna la petición usando el primer algoritmo visto.

Más información: http://httpd.apache.org/docs/current/mod/mod_lbmethod_bybusyness.html

Creación de la infraestructura

Para poder usar un balanceador de carga, lo primero es asegurarse de que hemos activado los módulos `mod_proxy` y `mod_proxy_http`, como ya hemos visto. Hecho esto, activaremos `mod_proxy_balancer`:

```
operario@frontend1:~$ sudo a2enmod proxy_balancer
```

Figura 375. Activando mod_proxy_balancer

Y posteriormente uno de los tres algoritmos de balanceo vistos. Para nuestros ejemplos usaremos el método implementado por `mod_lbmethod_bybusyness`:

```
perario@frontend1:~$ sudo a2enmod lbmethod_bybusyness
```

Figura 376. Activando el método de lbmethod_bybusyness

Ahora ya podemos ir al fichero `proxy.conf` de configuración del módulo y configurar la ruta donde se accederá al clúster, qué nodos del *backend* forman parte de este, los permisos (quien puede acceder al clúster de nodos) y el método de balanceo instalado, tal y como se ve en la siguiente imagen:

```
 GNU nano 2.2.6   Archivo: /etc/apache2/mods-enabled/proxy.conf    Modificado

<IfModule mod_proxy.c>

        # If you want to use apache2 as a forward proxy, uncomment the
        # 'ProxyRequests On' line and the <Proxy *> block below.
        # WARNING: Be careful to restrict access inside the <Proxy *> block.
        # Open proxy servers are dangerous both to your network and to the
        # Internet at large.
        #
        # If you only want to use apache2 as a reverse proxy/gateway in
        # front of some web application server, you DON'T need
        # 'ProxyRequests On'.


        <Proxy "balancer://cluster_pruebas">
                BalancerMember "http://192.168.23.2" route=1
                BalancerMember "http://192.168.23.20" route=2
                Require ip 192.168.56.1
                ProxySet lbmethod=bybusyness
        </Proxy>
        ProxyPass "/cluster" "balancer://cluster_pruebas"
        ProxyPassReverse "/cluster" "balancer://cluster_pruebas"
```

Figura 377. Configuración completa del clúster con balanceo de carga

Opcionalmente, podemos habilitar una herramienta que el propio módulo trae para configurar y monitorizar el mod a través del propio navegador. Esta herramienta está accesible a través de la ruta `/balancer-manager` de nuestro servidor y los cambios que se hagan en la misma desaparecen al reiniciar *Apache 2*. Si quisiésemos habilitarla, tendríamos que añadir al fichero de configuración lo siguiente, restringiendo quién puede acceder a dicha ruta como hemos hecho con otros módulos de control:

```
 GNU nano 2.2.6        Archivo: proxy_balancer.conf              Modificado

<IfModule mod_proxy_balancer.c>

        # Balancer manager enables dynamic update of balancer members
        # (needs mod_status). Uncomment to enable.
        #
        <IfModule mod_status.c>
           <Location /balancer-manager>
                   SetHandler balancer-manager
                   Require ip 192.168.56.1_
           </Location>
        #</IfModule>

</IfModule>
```

Figura 378. Habilitando el balancer-manager

Reiniciado *Apache 2*, ya podemos hacer las pruebas pertinentes accediendo a la ruta configurada.

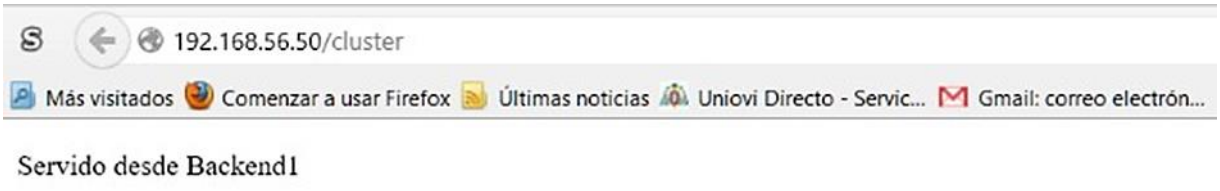


Figura 379. Primera petición al clúster

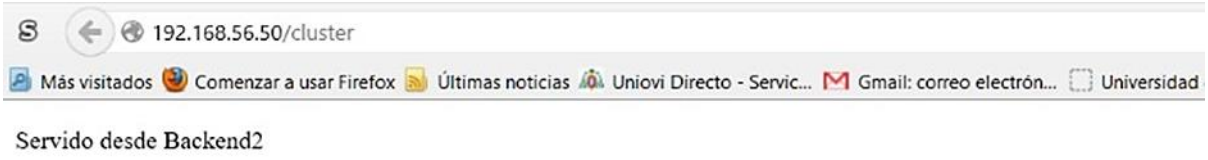


Figura 380. Segunda petición al clúster

Si ahora paramos un nodo de la siguiente forma:

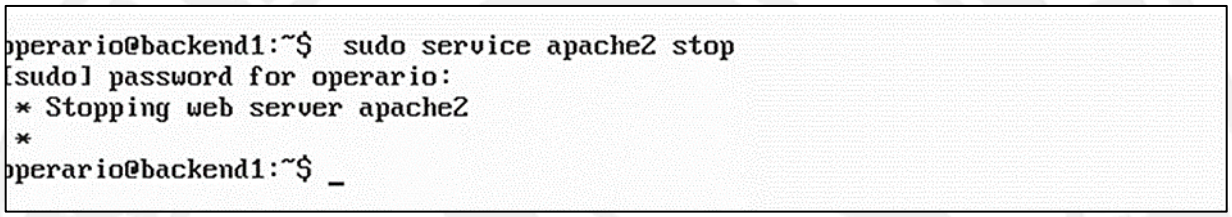


Figura 381. Parando un nodo del clúster

Comprobamos que el clúster sigue funcionando de manera transparente al usuario.

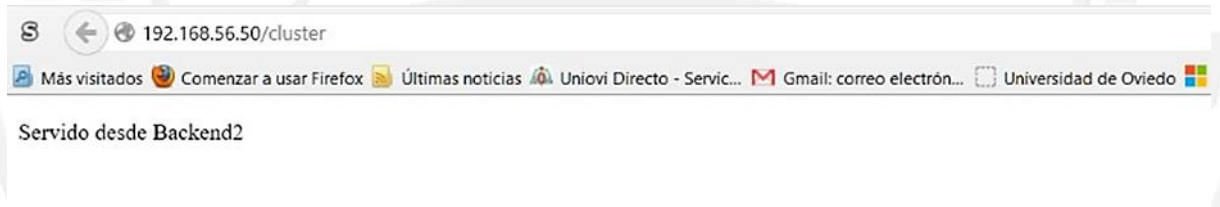


Figura 382. Clúster en funcionamiento con un nodo caído

Finalmente, hay que decir que esta configuración puede habilitarse también a nivel de host virtual, por si nos interesase que el clúster estuviese separado de dicha forma.

Uso de *rsync* para sincronizar contenidos web

Algunas de las arquitecturas multi-máquina que hemos visto hasta ahora hacen una suposición: todos los nodos *backend* alojan las mismas páginas. Esto es algo lógico, pero puede ser difícil de gestionar cuando tenemos muchos servidores *backend* y se produce una actualización del código de alguna de las webs servidas. Obviamente en ese caso habría que **replicar en cada servidor *backend* la nueva versión de la web**, algo que puede ser muy laborioso y propenso a fallos si el nº de nodos es grande, **salvo que usemos herramientas de automatización de gestión como *Ansible*** o ***Puppet***, por ejemplo.

Si no queremos usar estas herramientas, entra en juego la herramienta `rsync` que, bien configurada, permite distribuir a una serie de servidores (a los que llamaremos "*workers*") los contenidos de una determinada localización de origen (a la que llamaremos "*master*") de manera automática y periódica, sin más intervención del usuario que copiar los nuevos contenidos a la localización de origen mencionada. Esto ahorra mucho trabajo a la hora de desplegar contenidos dentro de un conjunto de máquinas que lo sirven. Además, la transmisión puede ser cifrada para preservar la confidencialidad de los contenidos distribuidos. Para ello, usaremos `rsync` junto con `openssh`, para que la transferencia sea segura, y el *daemon* `cron` de *Linux* para que además sea periódica y automática. La infraestructura que vamos a montar se puede representar así:

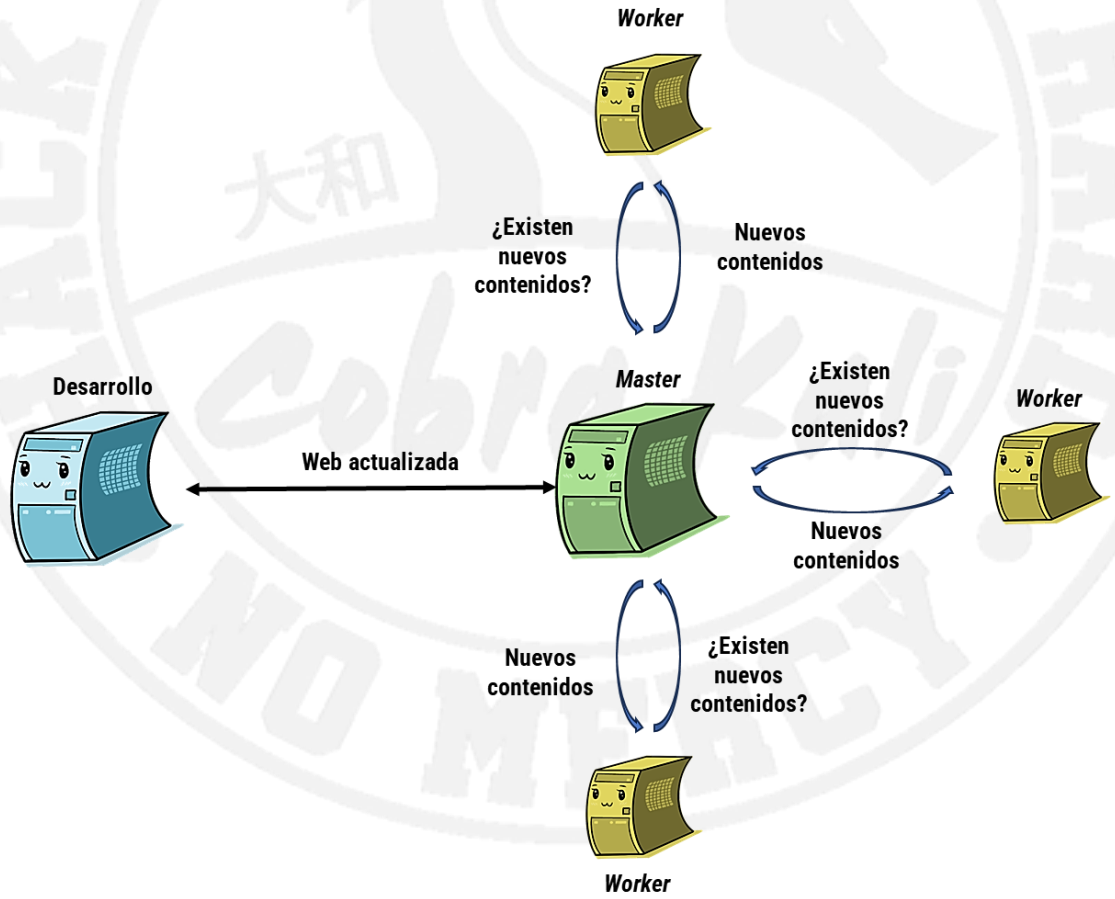


Figura 383. Esquema de funcionamiento de rsync

Para el ejemplo que vamos a hacer vamos a usar un par de máquinas de las mencionadas, instaladas con la infraestructura inicial. Se usarán las máquinas `backend1` (*master*) y `backend2` (**worker**). No obstante, antes de empezar, es mejor conocer un poco más la herramienta que vamos a usar, *rsync*.

El programa *rsync*

`rsnyc` es un programa para *Windows* y *Linux* GNU GPL que **sincroniza ficheros y directorios de una localización origen a otra de destino**. Su gran virtud es que su protocolo de comunicaciones minimiza el tamaño de los datos a transmitir, siendo por tanto muy eficiente. Dicho protocolo parte cada fichero a transmitir en trozos (*chunks*) y compara uno a uno con la copia del fichero existente en destino con técnicas de *hashing* (MD5). Una vez hecho esto, sólo envía al destino aquellos bloques de cada fichero que no coincidan, por lo que se minimiza la información a transmitir.

Otras ventajas reseñables son que permite enviar la información comprimida (`zlib`) y hacer *mirroring* de directorios y subdirectorios completos. Como hemos dicho en la introducción, también es capaz de comportarse como un *daemon* y también permite la transmisión encriptada mediante `ssh`. Por otro lado, dado que es capaz de limitar el ancho de banda consumido en las transferencias, minimiza su efecto sobre el rendimiento de un sistema.

Instalación y replicación manual

El primer paso para instalar una infraestructura `rsync` es intentar que funcione de manera manual, es decir, que cada vez que invoquemos a `rsync` desde un *worker* éste sincronice sus contenidos con el *master*. No es lo que buscamos y realmente es algo más difícil de manejar en un sistema real, pero son los cimientos para luego hacer la infraestructura completa. Lo que si haremos desde ya es cifrar las transferencias, y para ello es necesario instalar `openssh` en ambas máquinas (si no lo estuviera ya) con la siguiente orden: `sudo apt-get install openssh-server openssh-client`:

```
operario@backend1:~$ sudo apt-get install openssh-server openssh-client
```

Figura 384. Instalación de OpenSSH

El siguiente paso será instalar la propia herramienta `rsync` también en ambas máquinas (si no lo estuviera ya): `sudo apt-get install rsync`:

```
operario@backend1:~$ sudo apt-get install rsync
```

Figura 385. Instalando rsync

Aunque podríamos usar un usuario existente, es buena práctica crear un usuario "normal" (sin privilegios de `root`) que se ocupe de las tareas de sincronización, es decir, la identidad de usuario bajo la cual se harán todas las replicaciones. Para ello, en la máquina `backend1` (*master*), ejecutaremos el siguiente comando para crear un usuario que en este caso le llamaremos "`redondo`": `sudo useradd -d /home/redondo -m -s /bin/bash redondo`:

```
operario@backend1:~$ sudo useradd -d /home/redondo -m -s /bin/bash redondo
```

Figura 386. Creación de un nuevo usuario para replicación

Esto creará un usuario redondo con el directorio home en `/home/redondo` y *bash* como *shell*. Además, necesitamos darle una clave válida con `sudo passwd redondo`. Finalmente, antes de probar manualmente `rsync`, es necesario, si no estuviera hecho ya, abrir el puerto `22` de la máquina `backend1` para permitir conexiones vía `ssh` con `rsync`, lo que puede hacerse con `sudo ufw allow 22`.

Ahora ya estamos en condiciones de probar `rsync` desde la máquina `backend2` (*worker*). Dado que estamos tratando con contenidos web, vamos a ejecutar lo siguiente para que `backend2` sirva las mismas webs exactamente que `backend1`, haciendo que el contenido de su directorio `/var/www/html` sea el mismo:

```
sudo rsync -avz -e ssh redondo@192.168.23.2:/var/www/html /var/www/
```

No obstante, previamente conviene crear un fichero cualquiera solo en `/var/www/html` para comprobar que efectivamente se está copiando algo. Nótese que en el destino no incluirnos el directorio `html`, porque al hacer el `rsync` sobre dicho directorio en la máquina "*master*" se copiara también en la máquina destino (y entonces los contenidos aparecerán en `/var/www/html/html`).

```
operario@backend2:~$ sudo rsync -avz -e ssh redondo@192.168.23.2:/var/www/html /
var/www
```

Figura 387. Invocación manual de rsync

Al hacer esto, el programa nos preguntará lo siguiente:

```
The authenticity of host '192.168.23.2 (192.168.23.2)' can't be established.
ECDSA key fingerprint is e5:6c:3e:cb:42:3d:b3:9e:51:45:8e:98:04:af:24:82.
Are you sure you want to continue connecting (yes/no)?
```

Figura 388. Confirmación de conexión de rsync

A lo que debemos decir que sí y dar la clave del usuario `redondo` creado anteriormente. Si todo va bien, podremos ver en consola la lista de ficheros que se están transmitiendo y al final ambos servidores deberían tener el mismo contenido en `/var/www/html`, incluyendo subdirectorios. No obstante, tal y como hemos usado `rsync` cualquier contenido que estuviese en `backend2` que no estuviese en `backend1` no será borrado, es decir, el *worker* conserva sus propios archivos que no tienen los demás. Este comportamiento puede cambiarse, como veremos en el siguiente ejemplo, para que `backend1` y `backend2` sean copias exactas.

Replicación automática

Una vez que las copias manuales funcionan, ahora daremos los pasos necesarios **para hacer la replicación automática**. Para ello lo que haremos será generar un par de claves pública/privada que permitan hacer una conexión `ssh` autenticada entre *worker* y *master* sin necesidad de intervención humana, puesto que no se requerirá clave alguna al usar certificados para la identificación de máquinas. El primer paso para ello es ejecutar las siguientes órdenes desde la máquina *worker* (`backend2`), que crean un directorio para guardar el certificado para `rsync` en la carpeta del usuario `root`:

```
sudo mkdir /root/rsync  sudo ssh-keygen -t dsa -b 1024 -f /root/rsync/mirror-rsync-key
```

Al hacer esto, veremos algo como lo siguiente:

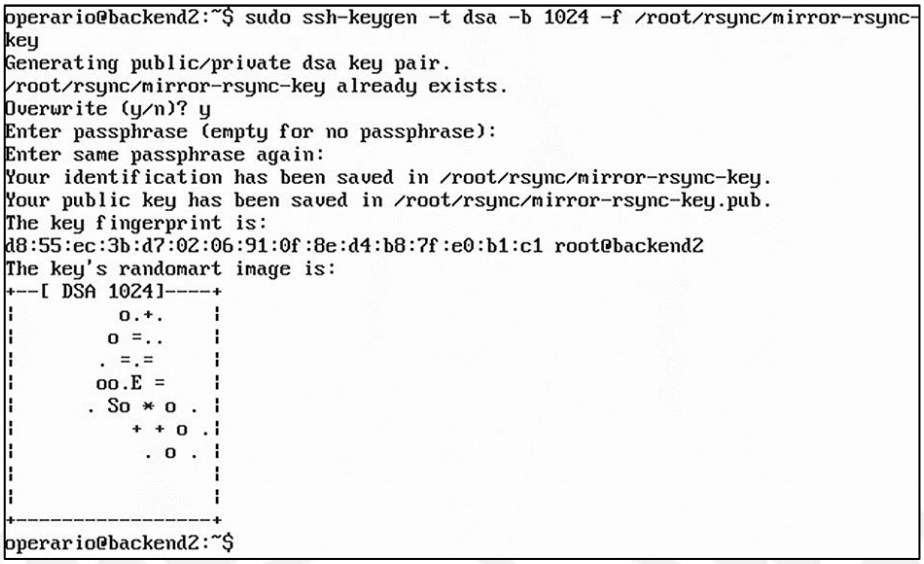


Figura 389. Certificado para rsync generado

Durante este proceso es muy importante **no poner ninguna clave** cuando el programa nos pregunte por una, o sino la copia no será automática, que es lo que queremos. Hecho esto, copiamos el certificado generado a la máquina `backend1` (*master*, origen de los contenidos) ejecutando esta orden desde `backend2` (la máquina `backend1` contendrá la clave generada en `backend2` en el directorio que se indica en la misma):

```
sudo scp /root/rsync/mirror-rsync-key.pub redondo@192.168.23.2:/home/redondo/
```

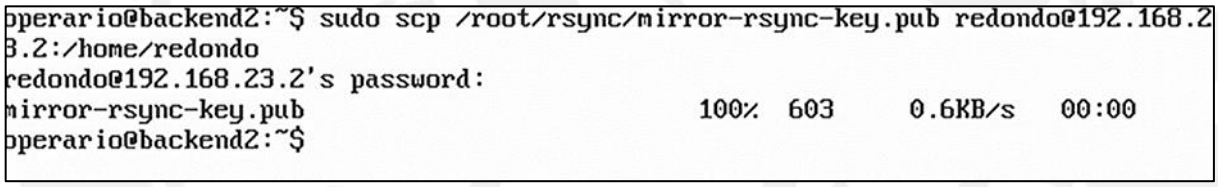


Figura 390. Copiando el certificado de la máquina worker a la máquina master

Con esta clave `rsync` podrá hacer *login* con los permisos adecuados para transferir los contenidos y sin necesidad de introducir una clave por la entrada estándar, permitiendo así la copia automática sin intervención humana. Conviene comprobar que se ha hecho bien la copia de la clave antes de continuar.

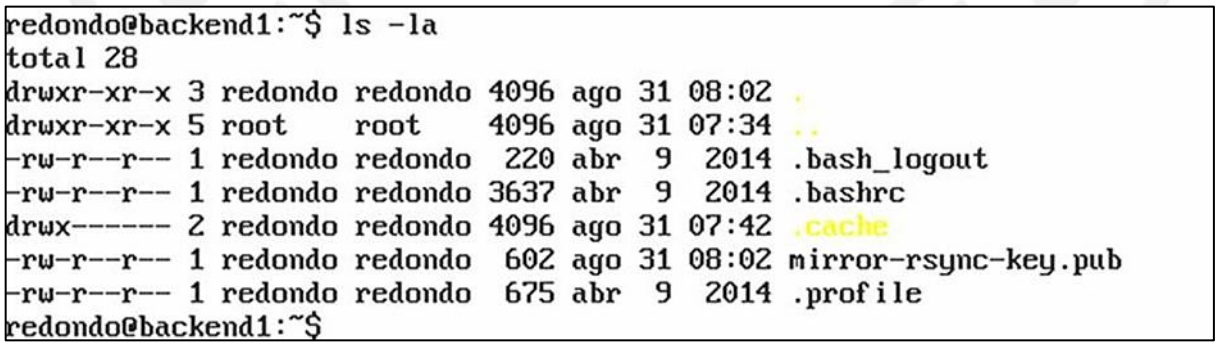


Figura 391. Certificado de rsync copiado

Ahora, entrando en sesión en `backend1` como el usuario creado para `rsync` (`redondo` en los ejemplos), hacemos el proceso indicado en la imagen y cuadro siguientes. Este proceso sirve para copiar la clave de la máquina `backend2` recién transferida a un fichero de claves de máquinas autorizadas, para que `rsync` haga *login* con `ssh` en `backend1` para posteriormente hacer la transferencia. En este fichero se guardarán

por tanto todas las claves de todas las máquinas *worker* en las que `rsync` hará la transferencia de ficheros automática cuando se ejecute, por lo que si se añaden nuevas máquinas básicamente hay que repetir este proceso, ya que el operador `>>` añade y no borra los contenidos de `authorized_keys`:

```
mkdir ~/.ssh  chmod 700 ~/.ssh
mv ~/mirror-rsync-key.pub ~/.ssh/  cd ~/.ssh
touch authorized_keys  chmod 600 authorized_keys
cat mirror-rsync-key.pub >> authorized_keys
```

```
redondo@backend1:~$ mkdir ~/.ssh
redondo@backend1:~$ chmod 700 ~/.ssh
mredondo@backend1:~$ mv ~/mirror-rsync-key.pub ~/.ssh/
redondo@backend1:~$ cd ~/.ssh/
redondo@backend1:~/.ssh$ touch authorized_keys
redondo@backend1:~/.ssh$ chmod 600 authorized_keys
redondo@backend1:~/.ssh$ cat mirror-rsync-key.pub >> authorized_keys
redondo@backend1:~/.ssh$
```

Figura 392. Proceso de registro de la clave de backend2 como máquina autorizada en backend1

Con esto añadimos la clave anterior como clave autorizada para hacer *login* por `ssh` (la estructura de directorios creada es la que permite automáticamente a `ssh` leer dicha clave al hacer *login* con esa identidad de usuario) en la máquina `backend1`. Para saber si este proceso ha transcurrido correctamente, hacemos desde el directorio `.ssh` creado: `cat authorized_keys` y comprobamos que tiene contenido, entre el cual debería aparecer una clave perteneciente a la máquina recién autorizada para hacer login (ver el final de cada clave), tal y como se ve en la siguiente imagen:

```
redondo@backend1:~/.ssh$ cat authorized_keys
ssh-dss AAAAB3NzaC1kc3MAAACBAKUQZVhR5Um1Oh+DI0P8HTuWEmuzQZAbaJg+T0QE0pUIuCVf7ujN
R0E76V+Mvb+raAIvuVt2LUVSzmTKMEcvI5frX8MeOYx667hTNk5TE4+QM0f7PzbvJN1cizWdhLSKQ1O2
YXGSWtU7/yv42mZW0tCcwW3d8rhj9IbvnvOHRlTJAAAAFQDEevWNANslv1sB8QwtHFuQwL5evwAAAIBE
rGpHzgSW4f9tx52Pb0rOwL/t3uFtskTmwZRPJEW0c4I/oGyOQhoR61CfKFqbuf9uHq4Q/Cc1vv/6nBv4
uuLpmwrltD9Dz3jSzZ2+Bx3sN4WH4UY56b9UEdUpSjR39ZeuJkW4ZVn+vouV1IXKFUW6Y2mqr1LLyetZ
zguVn9MKCwAAAIAog0ffD/25OYuVXD5aCFGWsmDxwEbpQAX/EFfg7GxaITPmVk7Skfowp/yErwkBqqGC
68fSdhk/JOZUmiPUB6PWNOJjfmgV0sVqbv8z+FfC/5Ge6IGeQvoQ0TW5BGJGzkZHZMb2XHLSOdxB7yhf
zgZmx1amyMoidebptWHb4orqCg== root@backend2
redondo@backend1:~/.ssh$ _
```

Figura 393. Clave de la máquina backend2 añadida al conjunto de claves autorizadas

La forma más segura de configurar este servicio es permitir conexiones sólo desde las máquinas destino de las copias (en nuestro caso, sólo `backend2`) y únicamente con el usuario creado al efecto (`redondo`). Para ello añadimos esto justo al principio del fichero `/home/redondo/.ssh/authorized_keys` creado antes, donde `192.168.23.20` es la IP de la máquina *worker* (`backend2`):

```
command="/home/redondo/rsync/checkrsync",from="192.168.23.20",no-portforwarding,no-X11-
forwarding,no-pty
```

NOTA: El nuevo contenido debe ir en la primera línea y el existente debe ir **EN LA LÍNEA SIGUIENTE**, evitando dejar espacios en blanco al final de la primera línea.

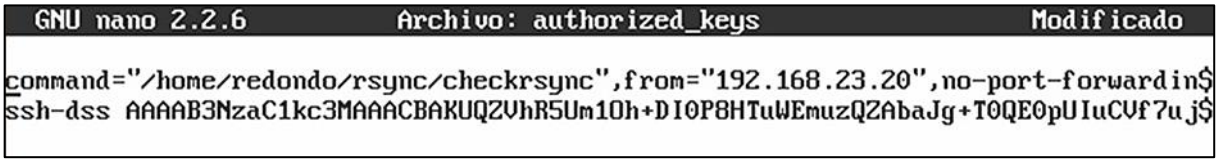


Figura 394. Contenido del fichero authorized_keys

Esta línea adicional indica que antes de admitir la conexión desde la IP dada, se ejecutará un *script* (`checkrsync`) que hará unas comprobaciones adicionales. A continuación, se muestra este *script*, que es una medida de seguridad adicional que rechaza todos los comandos ejecutados bajo esa identidad de usuario en `ssh`, salvo el propio `rsync`. Todo esto debemos crearlo como el usuario `redondo` en el propio `home` de este:

```
mkdir ~/rsync  nano ~/rsync/checkrsync
```

Y dentro de este fichero debemos incluir este contenido:

```
#!/bin/sh

case "$SSH_ORIGINAL_COMMAND" in
        *\&*)
                echo "Rejected"
                ;;          *\(*)
                echo "Rejected"
                ;;          *\{*)
                echo "Rejected"
                ;;          *\;*)
                echo "Rejected"
                ;;          *\<*)
                echo "Rejected"
                ;;          *\`*)
                echo "Rejected"
                ;;          rsync\ --server*)
                $SSH_ORIGINAL_COMMAND
                ;;
        *)
                echo "Rejected"
                ;;
Esac
```

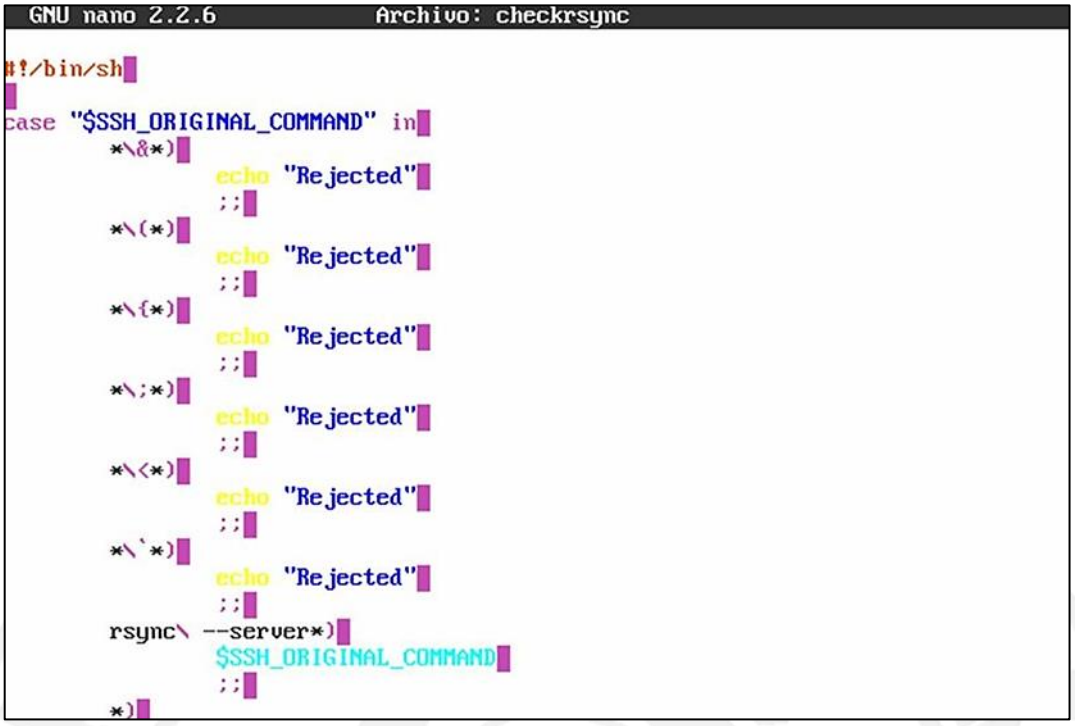


Figura 395. Script para controlar que a través de SSH el usuario redondo solo puede hacer rsync

Finalmente, cambiarnos los permisos al *script* para que sea ejecutable: `chmod 700 ~/rsync/checkrsync`. Por tanto, el objetivo se ha cumplido: tenemos un sistema de conexión automática desde máquinas autorizadas, con un usuario particular para un servicio con el que sólo se puede ejecutar dicho servicio en caso de conexión remota con el mismo.

Pruebas de funcionamiento

Ahoya ya estaríamos en condiciones de testear si `rsync` funciona sin clave desde `backend2`. Para ello ejecutamos la siguiente orden:

```
sudo rsync -avz --delete --exclude=**/stats --exclude=**/error -exclude=**/files/pictures
-e "ssh -i /root/rsync/mirror-rsync-key" redondo@192.168.23.2:/var/www/html /var/www/
```

Si todo va bien, debería poder hacerse la copia sin pedir una clave para el usuario redondo. Los `--exclude` que hemos puesto son opcionales, se muestran solo a modo de ejemplo para saber cómo indicar a la herramienta qué cosas no se quieren copiar. Nosotros no los usaremos. Esta forma de usar `rsync` también elimina el contenido previo del directorio (`-delete`), transfiriendo los archivos del *master* sin que quede ninguno en la localización de destino que no estuviese en el mismo (se trata por tanto de una copia exacta, siendo esta la otra forma de usar `rsync` que mencionamos anteriormente).

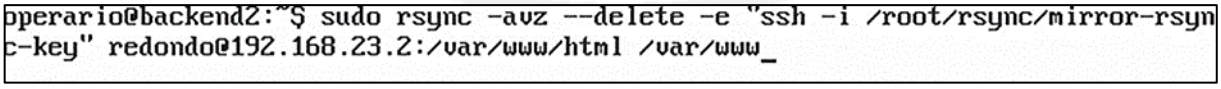


Figura 396. Prueba de rsync sin claves de usuario

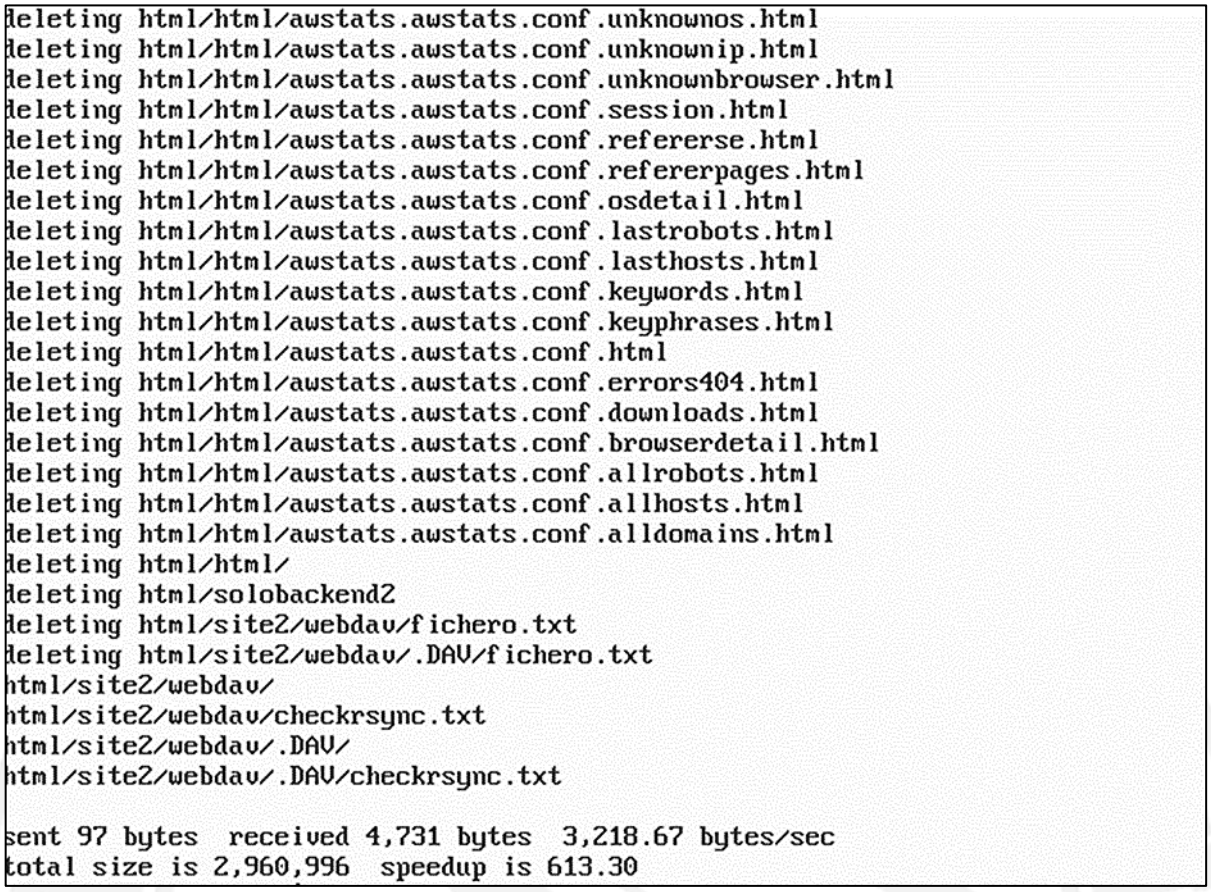


Figura 397. Ejecución de rsync y log de los archivos copiados

Si esta replicación ha tenido éxito, entonces ya podemos finalmente crear un trabajo cron en `backend2` para ejecutar este comando periódicamente, ejecutando `sudo crontab -e` y escribiendo en el fichero del *daemon* `cron` lo siguiente para crear la orden periódica:

```
*/5 * * * * /usr/bin/rsync -azq --delete --exclude=**/stats -exclude=**/error --
exclude=**/files/pictures -e "ssh -i /root/rsync/mirrorrsync-key"
redondo@192.168.0.3:/var/www/html /var/www/
```

No obstante, nosotros lo haremos sin usar `exclude` para nuestras pruebas, la orden a escribir es idéntica a la que ejecutamos antes:

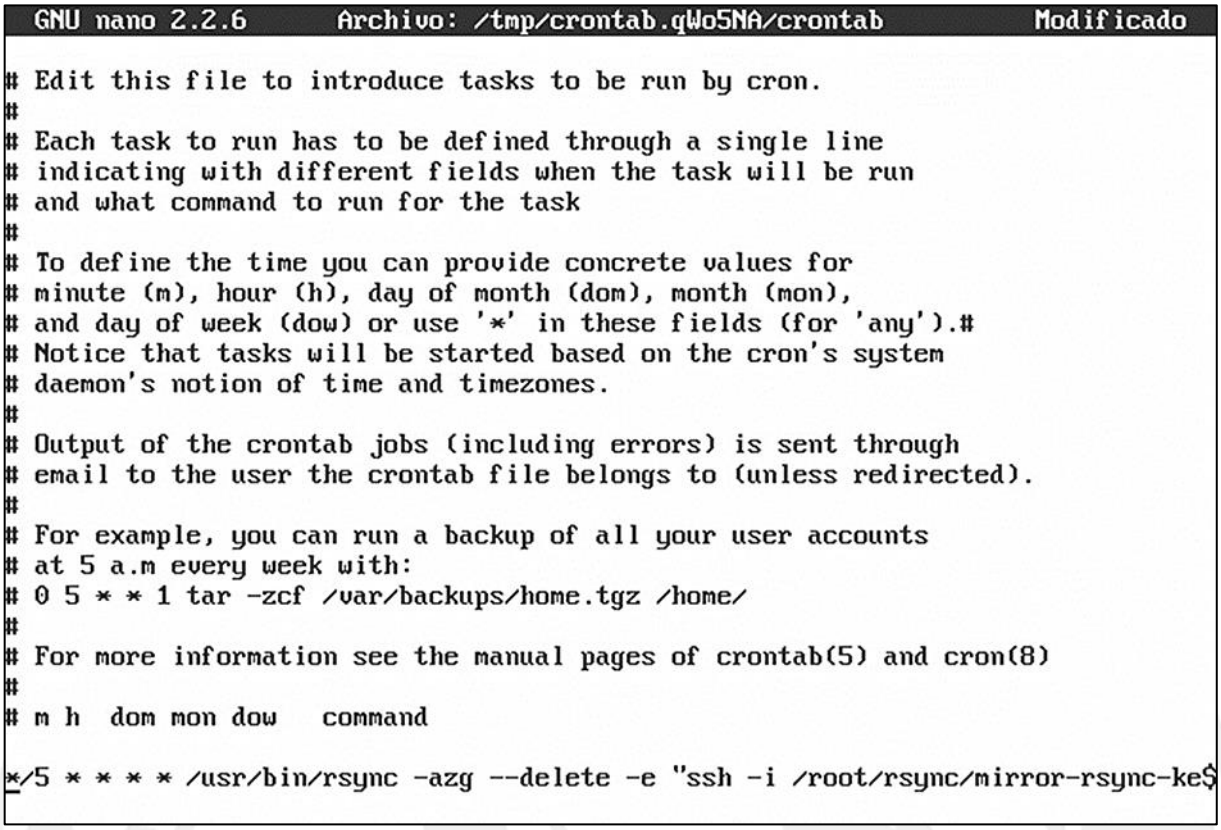


Figura 398. rsync periódico gracias a cron

Finalmente, para probar si funciona correctamente, se puede modificar o añadir algún contenido en `backend1` dentro del directorio replicado (`/var/www/html`) y esperar a ver si se replica en `backend2`. La temporización del `cron` puesta en el ejemplo son 5 minutos, así que la copia no será inmediata salvo que se cambie dicho valor.

NOTAS FINALES: Si en algún momento del proceso el sistema devuelve un error referente a saltos de línea o caracteres extraños, significa que el fichero no está en formato *Unix*. Podemos arreglarlo fácilmente con el comando `dos2unix`. Si se desea consultar el estado del *daemon* `cron`, éste tiene el *log* en `/var/log/syslog`. Si falla la primera prueba con el `rsync`, podemos ver su log en `/var/log/auth.log`

Gestión de sesiones en arquitecturas multi-servidor

¿De qué trata este bloque?

Todos los ejemplos de arquitecturas multi-máquinas que hemos visto hasta ahora **adolecen de un problema intrínseco**: si las aplicaciones que sirven no guardan sus sesiones en un lugar accesible por todas las posibles réplicas existentes de la misma (téngase en cuenta que habrá una copia de la misma web por cada servidor desplegado), entonces **en cuanto un cliente sea atendido por un servidor distinto del que creó su sesión ésta se perderá**, y con ella todos los datos que tuviera (con lo que si la web gestiona usuarios el usuario logeado se perderá, junto con su configuración, datos, preferencias… de la sesión en marcha). Como esto no es admisible, hay que buscar alternativas si, como hemos dicho, la sesión no se guarda en una localización compartida por todos los *backends* (en cuyo caso el problema no existiría). Básicamente hay dos aproximaciones para enfrentarse a este problema:

- **Usar *sticky sessions***: Este mecanismo hace que, en el momento en que un servidor crea la sesión de un usuario, el *framework* o tecnología de desarrollo **se asegura que sea precisamente ese servidor el que siempre atienda todas las peticiones subsiguientes de ese usuario**. Obviamente el problema aparece cuando ese servidor deja de estar operativo, ya que toda la información de sesión de todos los usuarios que estuviese atendiendo se perdería. Aunque lo cierto es que no debería ser algo que pasase frecuentemente (los servidores bien administrados no están continuamente quedándose inoperativos), sin duda puede ser una molestia puntual para los usuarios. La gran ventaja de esta solución es que **es técnicamente más sencilla de implementar** y consume menos recursos que la siguiente. Nosotros veremos ejemplos de tecnologías que soportan este tipo de sesiones.
- **Usar replicación de sesiones**: Muchos *frameworks* y tecnologías de desarrollo vienen con **soporte para replicar automáticamente las sesiones de los usuarios entre múltiples servidores**, de manera que sea indistinto que una petición la atienda uno u otro, dado que todos tienen acceso a los mismos datos (todas las sesiones de todos los usuarios). Como hemos dicho, esto es algo que depende de la tecnología de desarrollo, y a modo de ejemplo nosotros veremos cómo hacerlo con *Apache Tomcat* (páginas JSP) y también con páginas programadas en PHP, gracias al software `memcached`. Existe *software* que de terceros que permite hacer esto, como *Pacemaker*, pero no lo veremos en esta sección.

Sesiones *Sticky*

Un ejemplo muy claro de uso de *sticky sessions* podemos verlo en un producto que ya hemos descrito: `mod_proxy_balancer`. Si bien en el ejemplo que hemos visto no las usa, es muy fácil cambiarlo para que lo haga y entonces se soporte una infraestructura que impida que un usuario pierda su sesión en una web que está alojada en un clúster. Cuando `mod_proxy_balancer` usa *sticky sessions*, **hay que configurar el nombre de la *cookie*** que contiene la información de qué servidor del clúster se va a usar. Para ello, hay que usar el atributo `stickysession`, que debe añadirse a la directiva `ProxyPass` ya vista. Hay que tener especial cuidado porque este valor es sensible a mayúsculas. La cookie debe estar presente, no importa si la establece el servidor *backend* o *Apache 2*. El valor de esta *cookie* se comparará con el valor de `route` de cada miembro del clúster para determinar qué servidor se usa.

Otra forma de implementar *sticky sessions* es mediante codificación en la URI. Usando este método el servidor busca un parámetro en la URI de la petición, especificándose el nombre de dicho parámetro mediante `stickysession` también. Nuevamente, el valor del parámetro se compara con el valor de *route* para determinar el servidor usado.

Pueden soportarse ambos métodos al mismo tiempo, configurando el nombre de la cookie y el nombre del parámetro de la URI separado por una barra vertical:

```
ProxyPass    "/micluster"       "balancer://cluster_pruebas"
stickysession=JSESSIONID|jsessionid

<Proxy "balancer://cluster_pruebas">
    BalancerMember "http://192.168.23.2:80" route=node1
    BalancerMember "http://192.168.23.20:80" route=node2
</Proxy>
```

En este ejemplo vemos que el nombre de la *cookie* que determina qué nodo se va a usar por cada cliente es `JSESSIONID`, y el nombre del parámetro de la URI a usar sería `jsessionid`. Ambos valores se usan a la hora de determinar qué servidor es el que va a atender la petición en un clúster, asumiendo que al menos uno estará presente. Si estuvieran presentes ambos, tiene prioridad el parámetro de la URI. Finalmente, si ni el servidor ni *Apache 2* crearan la cookie adecuada ni el parámetro de la URI, se puede usar `mod_headers` para crear una *cookie* adecuada que permita el uso de *sticky sessions* de la forma que se ve en el siguiente ejemplo:

```
Header       add    Set-Cookie    "ROUTEID=.%{BALANCER_WORKER_ROUTE}e;
env=BALANCER_ROUTE_CHANGED
<Proxy "balancer://cluster_pruebas">
    BalancerMember "http://192.168.1.50:80" route=1
    BalancerMember "http://192.168.1.51:80" route=2
    ProxySet stickysession=ROUTEID
</Proxy>
ProxyPass "/micluster " "balancer://cluster_pruebas "
ProxyPassReverse "/micluster" "balancer://cluster_pruebas"      path=/"
```

Clúster PHP con replicación de sesión usando *memcached*

Una aplicación desarrollada en PHP que esté alojada en un clúster y que use sesiones tendrá que enfrentarse al problema descrito de qué ocurre si peticiones sucesivas de un mismo cliente son atendidas por distintos servidores. Si bien un balanceador de carga *Apache 2* con *sticky sessions* como el descrito puede resolver este inconveniente, vamos a ver la otra solución, implementable gracias a un programa llamado `memcached` (http://memcached.org/).

`memcached` **es un sistema de caché de objetos distribuido que guarda información en memoria RAM** en lugar de en disco para mejorar el rendimiento. PHP (y otros productos) tienen un módulo que permite usar `memcached` para guardar sesiones. Esto tiene la principal ventaja de que, al ser un sistema para guardar información distribuido, las sesiones de aplicaciones PHP estarán automáticamente replicadas entre múltiples servidores de la infraestructura donde se esté ejecutando `memcached`. De esta manera, las sesiones de las aplicaciones serán redundantes y si un servidor deja de estar operativo el resto de los servidores que tengan `memcached` instalado impedirán que el usuario pierda su sesión, al tener toda la información. Un esquema general de cómo es esta infraestructura es el siguiente, aunque en nuestros

ejemplos prescindiremos de instalar el balanceador de carga puesto que ya lo hemos hecho en un ejemplo anterior.

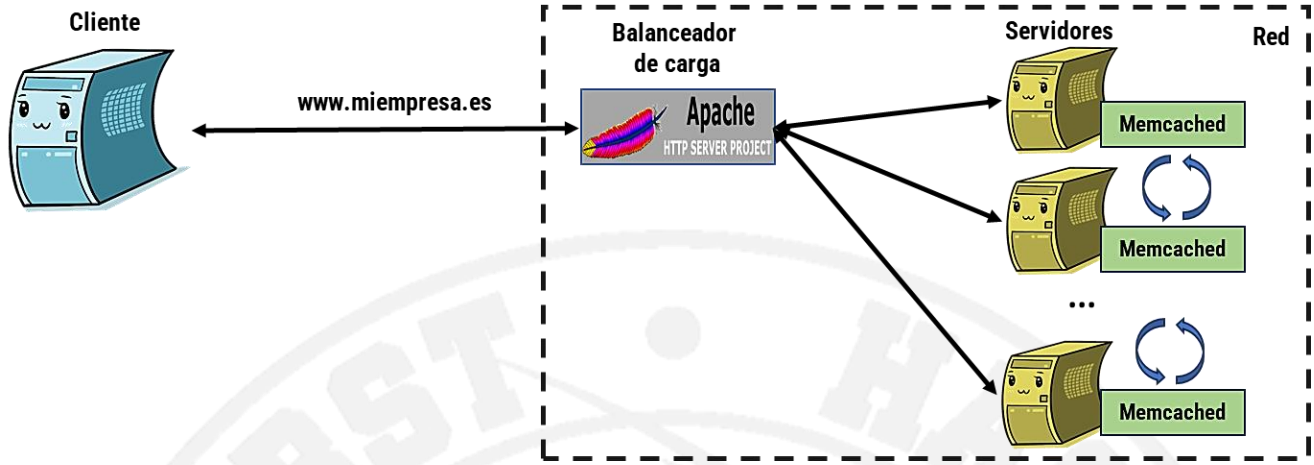


Figura 399. Esquema de uso de memcached para replicación de sesiones con PHP

Para ver cómo implementar una infraestructura de esta clase vamos a usar dos máquinas: `backend1` y `backend2`, con PHP instalado (ver actividades anteriores) y una aplicación sencilla que use sesiones para implementar un contador de visitas. Nuestro host *Windows* hará de cliente de dicha aplicación. El primer paso es instalar el programa `memcached` y la librería PHP que lo usa en ambas máquinas *backend*, lo que puede hacerse así: `sudo apt-get install php5-memcache memcached` (en distribuciones de *Ubuntu* modernas PHP 5 ya no está soportado, por lo que habrá que buscar el nombre del paquete concreto, probablemente `php-memcache`, https://ubuntu.com/server/docs/programming-php)

Dado que por defecto `memcached` solo sirve peticiones a *localhost*, tenemos que cambiarlo para que escuche peticiones en la red en la que están los otros servidores, de manera que acepte peticiones de los otros servidores que van a usar `memcached`, y así que los datos de todos los nodos se distribuyan por la red y sean los mismos. Para `backend1`, el cambio sería el siguiente, sobre el fichero `/etc/memcached.conf`:

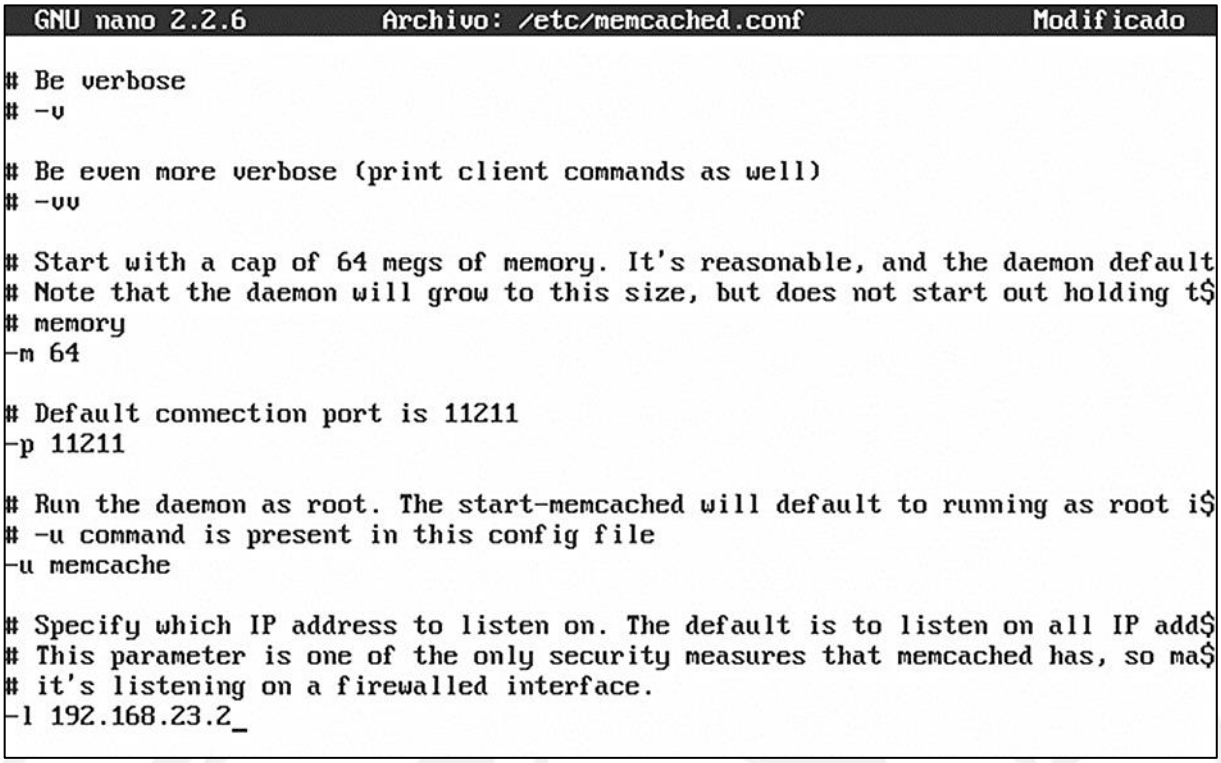


Figura 400. Cambio de la configuración de memcached

Obviamente, el otro servidor *backend* debe hacer una instalación y una modificación de la IP idéntica a la vista, pero con su propia IP (`192.168.23.20`). Una vez cambiados ambos, se debe reiniciar `memcached`: `sudo service memcached restart`

Ahora que el sistema distribuido de objetos está operativo, tenemos que indicar en todos los servidores que usen PHP que vamos a usarlo para guardar las sesiones de las aplicaciones que vamos a servir. Para ello, en cada máquina con PHP, tenemos que editar el fichero `/etc/php5/apache2/php.ini` y encontrar las directivas `session.save_handler` y `session.save_path`, de manera que su valor ahora sea "`memcache`" y la lista de servidores que tienen `memcached` operativo, respectivamente. Esto lo podemos ver en las siguientes imágenes:

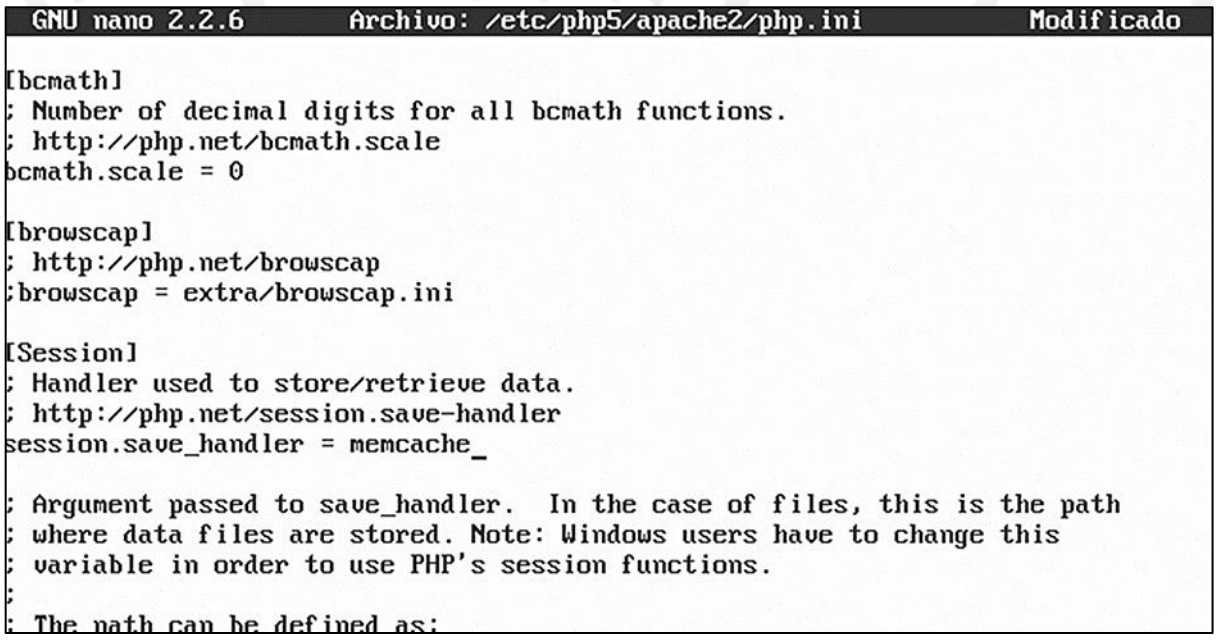


Figura 401. Definición de session.save_handler de memcached

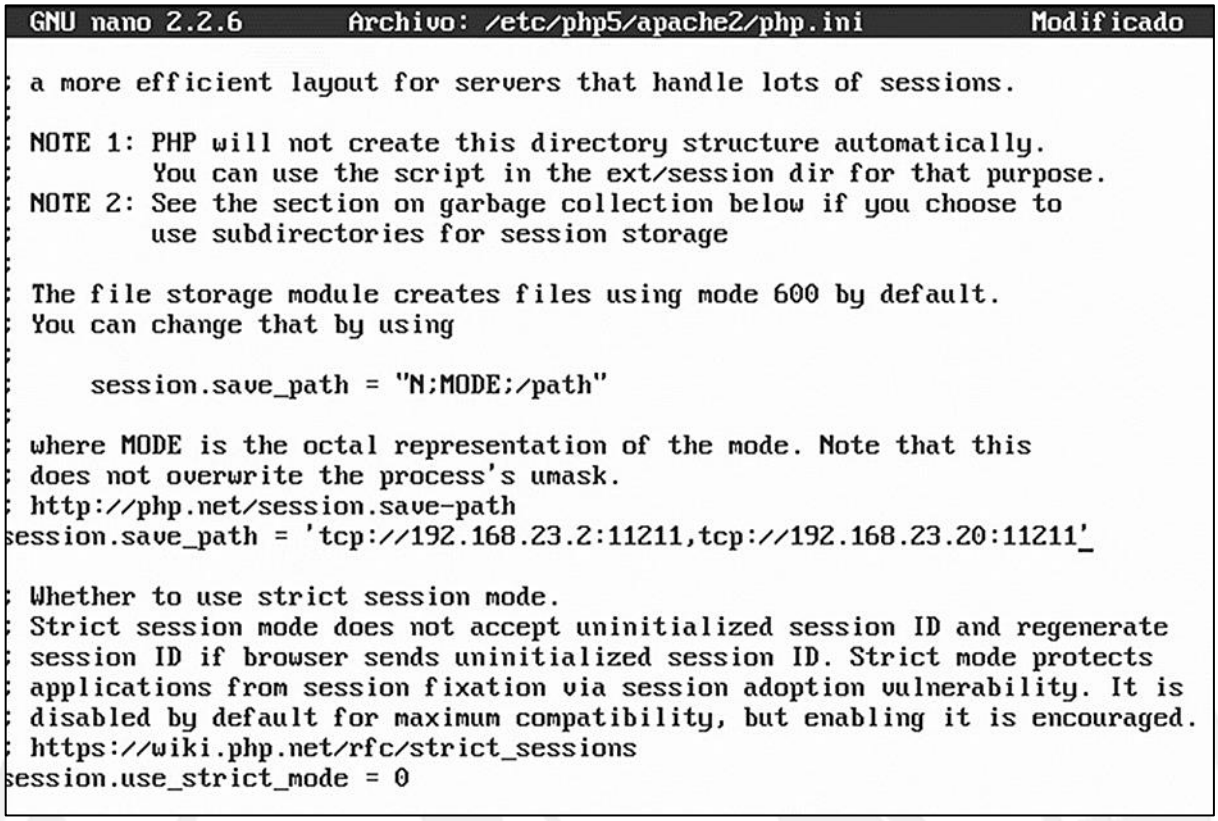


Figura 402. Definición de session.save_path de memcached

En la última directiva debemos recordar especificar el puerto `11211` en cada máquina, ya que es el puerto donde `memcached` escucha sus peticiones. Por supuesto, dicho puerto debe estar abierto en todas las máquinas:

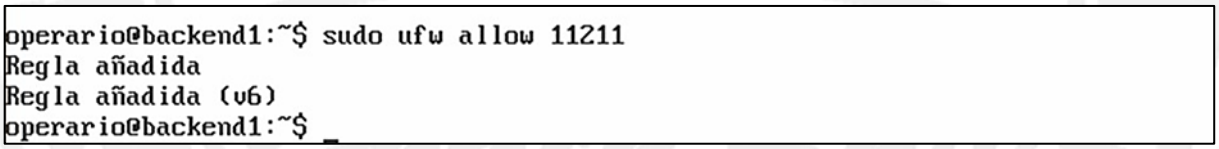


Figura 403. Abriendo el puerto de memcached en el firewall

Con el sistema de memoria distribuido en marcha y PHP correctamente configurado para que lo use para guardar sus sesiones, estamos muy cerca de poder usar la replicación de sesiones que perseguimos. El último paso que resta para ello es indicar a `memcached` que se comporte de forma redundante en caso de fallo. Esto lo podemos conseguir añadiendo un par de líneas al fichero `/etc/php5/mods-available/memcache.ini` (o ruta equivalente según la versión que hayáis instalado) en cada máquina, que indiquen precisamente esta circunstancia:

```
memcache.allow_failover=1
memcache.session_redundancy=3
```

```
  GNU nano 2.2.6   Archivo: /etc/php5/mods-available/memcache.ini   Modificado

; uncomment the next line to enable the module
extension=memcache.so

[memcache]
memcache.dbpath="/var/lib/memcache"
memcache.maxreclevel=0
memcache.maxfiles=0
memcache.archivememlim=0
memcache.maxfilesize=0
memcache.maxratio=0

memcache.allow_failover=1
memcache.session_redundancy=3
```

Figura 404. Modificando la configuración de memcached para soportar redundancia

El valor que puede parecer más inexplicable es el de `session_redundancy`. Este valor debe ser igual al nº de máquinas con `memcached` instalado + 1, motivo por el cual en nuestra instalación de pruebas el valor correcto es 3. Hecho esto, basta con reiniciar *Apache 2* para tener operativo el sistema de replicación de sesiones finalmente.

Por tanto, ya podemos hacer pruebas de la infraestructura para ver si funciona. Las pruebas se realizarán con esta sencilla aplicación `.php` que implementa un contador de visitas usando la sesión, y que imprime tanto la sesión como la IP del servidor que sirve la página:

Fichero contador_visitas.php:

```
<?php
    header('Content-Type: text/plain');      session_start();
    if(!isset($_SESSION['visit']))
    {
        echo "Esta es tu primera visita\n";
        $_SESSION['visit'] = 0;
    }
    else echo "Hasta ahora me has visitado: ".$_SESSION['visit'] veces. "\n";

    $_SESSION['visit']++;

    echo "IP del servidor: ".$_SERVER['SERVER_ADDR'] . "\n";
    print_r($_COOKIE);
?>
```

Dado que no hemos incluido el balanceador de carga en esta infraestructura, vamos a hacer las pruebas de otra forma. Lo primero es comprobar a través del host *Windows* que la aplicación efectivamente funciona:

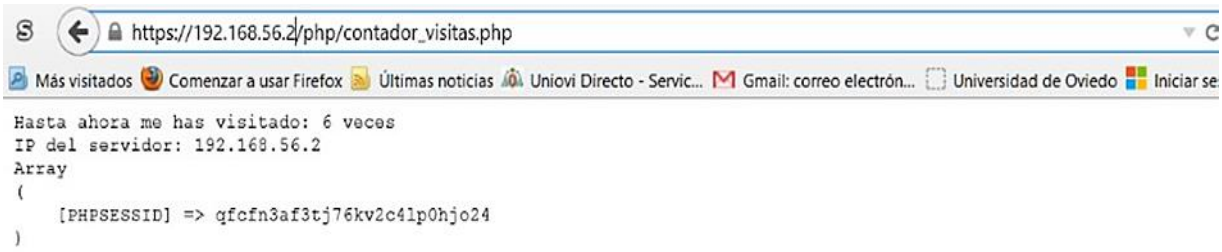


Figura 405. Petición a la página PHP cuya sesión se guarda en memcached

Si ahora copiamos la cookie con el valor de la sesión (valor de `PHPSESSID`) y se lo enviamos con `curl` al otro servidor haciendo una petición HTTP a la misma página, vemos que la respuesta es que la sesión se ha mantenido, ya que el contador de visitas no parte de cero:

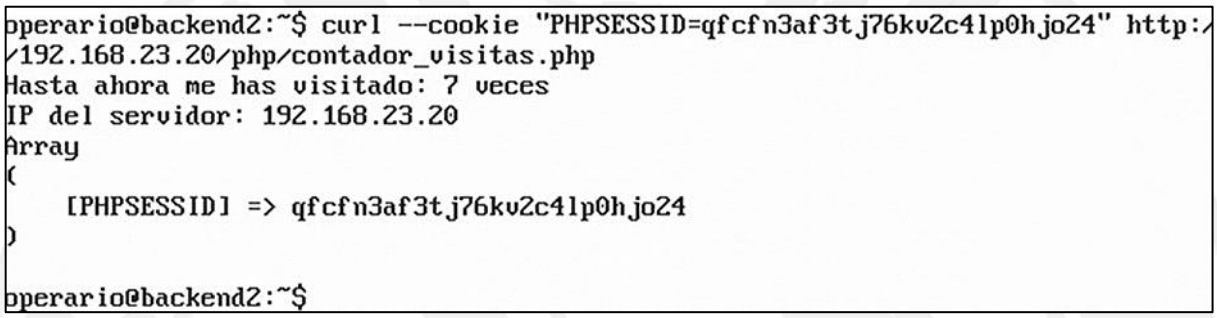


Figura 406. Petición al otro servidor

Por supuesto, una petición al primer servidor de nuevo tiene en cuenta la visita que acabamos de hacer al segundo:

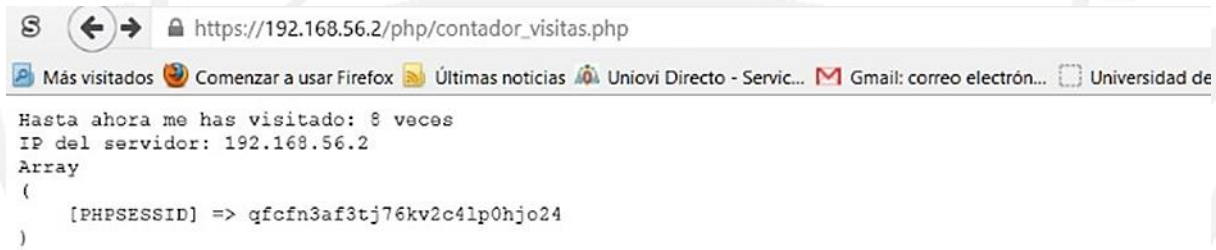


Figura 407. Petición a backend1 con sesión actualizada

Vamos a probar a continuación que el sistema soporta caídas de algún servidor. Para ello vamos a parar el servicio `memcached` en `backend1` con `sudo service memcached stop`:

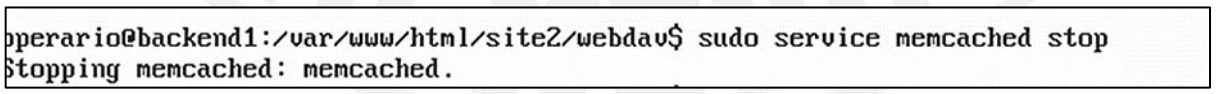


Figura 408.Parando memcached en backend1

Y vamos a continuación a hacer una petición a `backend1`:

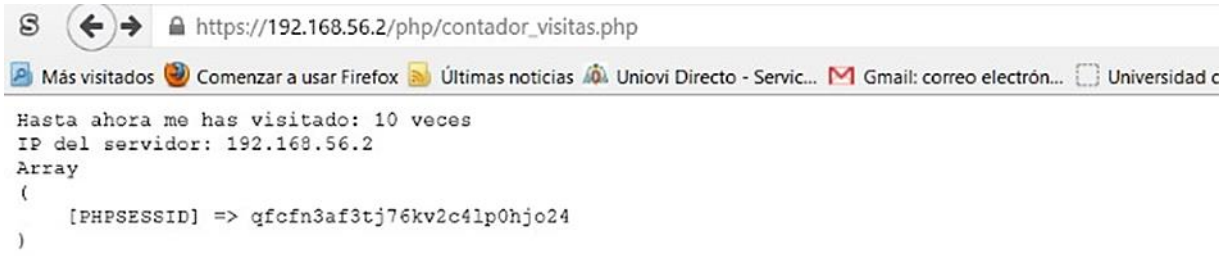


Figura 409. Petición a backend1 con su servicio memcached inoperativo

Como podemos apreciar, sigue funcionando sin problemas gracias a la copia de los datos de la sesión que se ha replicado en `backend2`.

Para finalizar con esta sección, queda un último aspecto por contemplar, y es **la seguridad del servicio** `memcached`. Aunque se esté ejecutando en una red privada, cualquier intrusión en la misma puede poner en riesgo los datos que manipula. Para dotar a la infraestructura de la mayor seguridad posible en este aspecto, vamos a establecer unas reglas de *firewall* que hagan que sólo unas determinadas máquinas autorizadas puedan conectarse al puerto de `memcached` en cada uno de los servidores que lo tiene instalado. En nuestro caso solo habrá una máquina autorizada, `backend2`, por lo que solo a ella se le permitirá conectarse a dicho puerto. Obviamente, tenemos que hacer lo contrario en `backend2`, permitiendo solo a `backend1`. Sobre `backend1`, para ello ejecutamos la siguiente orden:

```
sudo ufw allow from 192.168.23.20 to any port 11211
```

Al hacer esto, el estado del firewall debe ser el siguiente:

```
operario@backend1:~$ sudo ufw allow from 192.168.23.20 to any port 11211
Regla añadida
operario@backend1:~$ sudo ufw status
Estado: activo

Hasta                       Acción      Desde
-----                       ------      -----
80                          ALLOW       Anywhere
443                         ALLOW       Anywhere
8080                        ALLOW       Anywhere
22                          ALLOW       Anywhere
11211                       ALLOW       Anywhere
11211                       ALLOW       192.168.23.20
80 (v6)                     ALLOW       Anywhere (v6)
443 (v6)                    ALLOW       Anywhere (v6)
8080 (v6)                   ALLOW       Anywhere (v6)
22 (v6)                     ALLOW       Anywhere (v6)
11211 (v6)                  ALLOW       Anywhere (v6)

operario@backend1:~$ sudo ufw delete allow 11211
Regla eliminada
Regla eliminada (v6)
```

Figura 410. Estado del firewall tras limitar el tráfico al puerto 11211

Ahora ya podemos deshacernos de la regla que permitía el tráfico al puerto `11211` a todo el mundo, quedando el estado final del *firewall* de la siguiente forma:

```
operario@backend1:~$ sudo ufw delete allow 11211
Regla eliminada
Regla eliminada (v6)
operario@backend1:~$ sudo ufw status
Estado: activo

Hasta                      Acción      Desde
-----                      ------      -----
80                         ALLOW       Anywhere
443                        ALLOW       Anywhere
8080                       ALLOW       Anywhere
22                         ALLOW       Anywhere
11211                      ALLOW       192.168.23.20
80 (v6)                    ALLOW       Anywhere (v6)
443 (v6)                   ALLOW       Anywhere (v6)
8080 (v6)                  ALLOW       Anywhere (v6)
22 (v6)                    ALLOW       Anywhere (v6)
```

Figura 411. Estado final del firewall en backend1

Una vez hecho lo propio en `backend2`, solo debemos probar que el sistema sigue funcionando y que no hemos cometido ningún error a la hora de cerrar puertos:

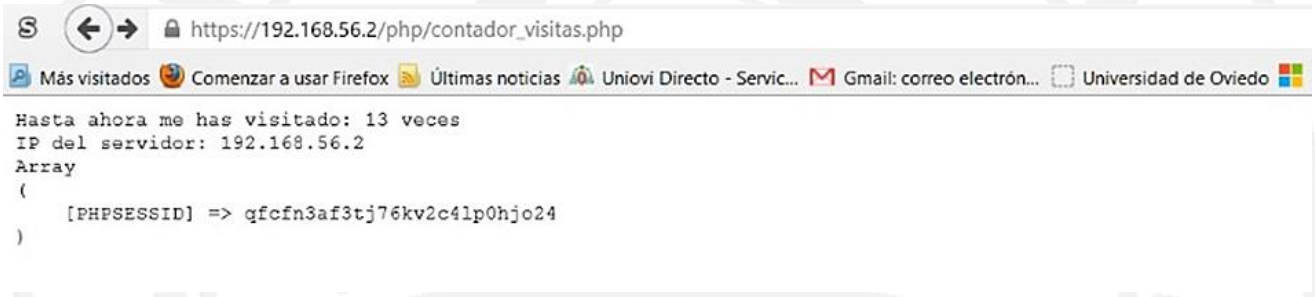


Figura 412. memcached funcionando y con configuración segura

Clúster *Tomcat* horizontal con replicación de sesiones bajo *Linux*

La infraestructura que vamos a plantear en esta sección es una granja web para aplicaciones JSP que se ejecuten bajo el servidor *Tomcat*. Tendremos N servidores que ejecutan una aplicación determinada (todos la misma, para que el clúster sea coherente) y por otro lado un servidor *Apache 2* que haga de balanceador de carga de los *Tomcat*. En lugar de un *reverse proxy*, como en el caso anterior, usaremos la tecnología de *clustering* y replicación de sesiones que este producto trae integrada, usando dos servidores Tomcat distintos. La principal ventaja, al igual que el sistema similar a este que hemos visto con PHP, es que **la sesión de la aplicación se replicará a todos los nodos**.

Por tanto, en caso de que uno deje de funcionar otro ocupará su lugar, pero con la ventaja de que, al tener una réplica de la sesión de cada usuario, **el usuario podrá seguir trabajando en el punto exacto donde estaba** (mantendrá su *login*, preferencias y cualquier dato que guardase en su sesión). Por tanto, al igual que en el caso que hemos visto de PHP con `memcached` un fallo en uno de los nodos sea completamente inapreciable para los clientes. La principal diferencia entre esta arquitectura y la anterior es que no necesitamos un producto externo como `memcached` para hacerla operativa: todo va integrado con *Tomcat*. La siguiente imagen ilustra la arquitectura que se va a crear en esta sección:

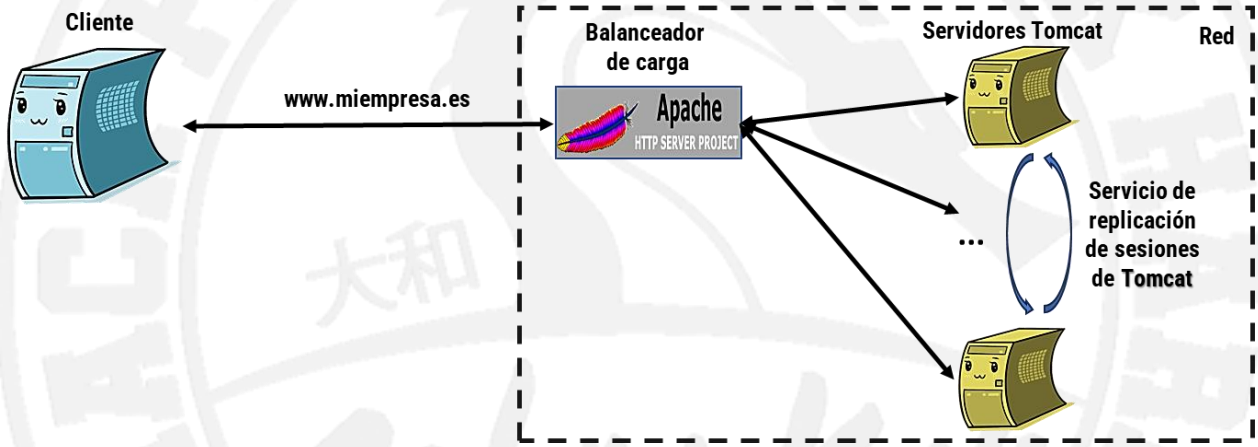


Figura 413. Ejemplo de la arquitectura a construir

Preparación de la Infraestructura

Para construir la arquitectura propuesta usaremos los nodos `backend1` y `backend2` como servidores *Tomcat*. `backend1` debe tener una instalación de *Tomcat* tal y como lo vimos en una sección anterior que trataba precisamente de cómo dar soporte a aplicaciones JSP. `backend2` es un clon de `backend1`, siguiendo los procedimientos de clonado de máquinas vistos. En cuanto al balanceador de carga *Apache 2*, podríamos ubicarlo en una tercera máquina (por ejemplo, `frontend1`), pero para ahorrar recursos vamos a instalarlo en `backend1` (donde también está el primer *Tomcat* del clúster) de manera que esta máquina sea a la vez balanceador y nodo de servicio (dicho de otra forma, en el diagrama anterior el balanceador y el primer nodo están instalados en la primera máquina). Nótese que esto es solo a modo de prueba, una instalación de esta arquitectura en un entorno real estaría mejor configurada si *Apache 2* y los *Tomcat* están en máquinas diferentes.

Sobre esta base, hay que realizar una serie de modificaciones para que la arquitectura funcione correctamente. Estas son:

- La página que usamos para la instalación de *Tomcat* en una sección anterior usaba como prueba un contador de visitas. No obstante, este contador de visitas guardaba su información en el objeto

`Application`. Cambiaremos el código de la página para que en su lugar use el objeto `Session`, como se ve en la siguiente imagen. Por otro lado, aunque en circunstancias normales el código de todas las webs del clúster debería ser el mismo, en nuestro caso la página de prueba indicará desde que nodo del clúster se sirve para probar su funcionamiento correcto. Recordamos que las aplicaciones de *Tomcat* se sirven desde `/var/lib/tomcat7/webapps`:

```
<%@ page import="java.io.*, java.util.*" %>

<html>
	<head>
		<title>Contador de visitas en JSP</title>
	</head>
	<body>
	<p>Tomcat 1</p>
	<%
		Integer visitas = (Integer)session.getAttribute("contador");
		if ((visitas == null) || (visitas == 0)) {
			out.println("Es tu primera visita a la página");
			visitas = 1;
		}
		else
		{
			visitas += 1;
			out.println("Es tu visita num. " + visitas);
		}
		session.setAttribute("contador", visitas);
	%>
	</body>
</html>
```

```
 GNU nano 2.2.6          Archivo: index.jsp                    Modificado

<%@ page import="java.io.*, java.util.*" %>

<html>
        <head>
                <title>Contador de visitas en JSP</title>
        </head>
        <body>
        <p>Tomcat 1</p>
        <%
                Integer visitas = (Integer)application.getAttribute("contador");
                if ((visitas == null) || (visitas == 0)) {
                        out.println("Es tu primera visita a la pagina");
                        visitas = 1;
                }
                else
                {
                        visitas += 1;
                        out.println("Es tu visita num. " + visitas);
                }
                session.setAttribute("contador", visitas);
        %>
        </body>
</html>
```

Figura 414. Página de pruebas del clúster Tomcat

Convertirnos el *Apache 2* de `backend1` en un mero balanceador de carga para *Tomcat*, de forma que no pueda servir nada más. De esta forma este *Apache 2* no sirve ninguna otra web y solamente se usará para balancear la carga entre ambos *Tomcat*. Como esto quiere decir que *Apache 2* no va a tener ningún *host* virtual operativo, si tuviéramos que hacer alguna configuración particular sobre `/var/www/html` u otros subdirectorios (como restricción de acceso por IP, etc.) podemos incluirla en el fichero `apache2.conf`.

Para inhabilitar *Apache 2* de esta forma hay que hacer los siguientes pasos, aunque debe tenerse en cuenta que es posible que ahora al acceder a la ruta raíz del servidor se muestre un listado de directorios, por lo que debemos prohibirlo o bien asignar el clúster a la ruta raíz de *Apache 2*:

- Ejecutamos `sudo a2dissite default`
- Eliminamos o comentamos la línea `NameVirtualHost *: 80` del fichero `/etc/apache2/ports.conf` si la hubiera.
- Finalmente, reiniciamos *Apache 2*

Paramos o desinstalamos el *Apache 2* que tendrá `backend2` por ser un clon de `backend1`, ya que no lo vamos a usar en este ejercicio (`sudo service apache2 stop`).

Puesta en marcha del clúster

Para poner en marcha el clúster de *Tomcat* tenemos que modificar el fichero `worker.properties` de `/etc/apache2 en backend1` que creamos en la instalación inicial de *Tomcat* para que especifique los dos nodos que forman parte del clúster, dejándolo como se muestra a continuación. Cada unidad de trabajo de *Tomcat* se denomina `worker`. Nótese que hemos usado las IPs de los nodos con *Tomcat* (`backend1` y `backend2`) y que en ambos casos nos conectamos al mismo puerto `8009` para contactar con el *Tomcat* instalado en cada máquina. Los nombres en `balanced_workers` deben ser los mismos que usemos en las entradas `worker.tomcatX`. El fichero quedaría como sigue:

```
worker.list=loadbalancer,jk-status
worker.jk-status.type=status    worker.tomcat1.port=8009    worker.tomcat1.host=192.168.23.2
worker.tomcat1.type=ajp13  worker.tomcat1.lbfactor=1

worker.tomcat2.port=8009     worker.tomcat2.host=192.168.23.20     worker.tomcat2.type=ajp13
worker.tomcat2.lbfactor=1

worker.loadbalancer.type=lb
worker.loadbalancer.balanced_workers=tomcat1,tomcat2  worker.loadbalancer.sticky_session=1
```

En nuestro caso hemos comentado el contenido que ya tenía el fichero y hemos añadido el nuevo que aparece en el cuadro anterior. Puede verse como el *worker* que nos interesa (al que hemos llamado por ejemplo `loadbalancer`) tiene el tipo `lb`, y que mediante la propiedad `balanced_workers` se le especifican los nodos *Tomcat* que pertenecen al clúster (pudiendo añadir todos los que tengamos). Otro detalle importante es que, aunque en este ejemplo estamos usando una máquina con el balanceador + un nodo *Tomcat*, separar ambos en dos máquinas es trivial, dado que los nodos *Tomcat* se especifican por IP.

Por otro lado, hemos habilitado un *worker* más, llamado `jk-status` y cuyo tipo (`status`) está predefinido por `mod_jk`, que nos permitirá posteriormente ver el estado del clúster desde la web:

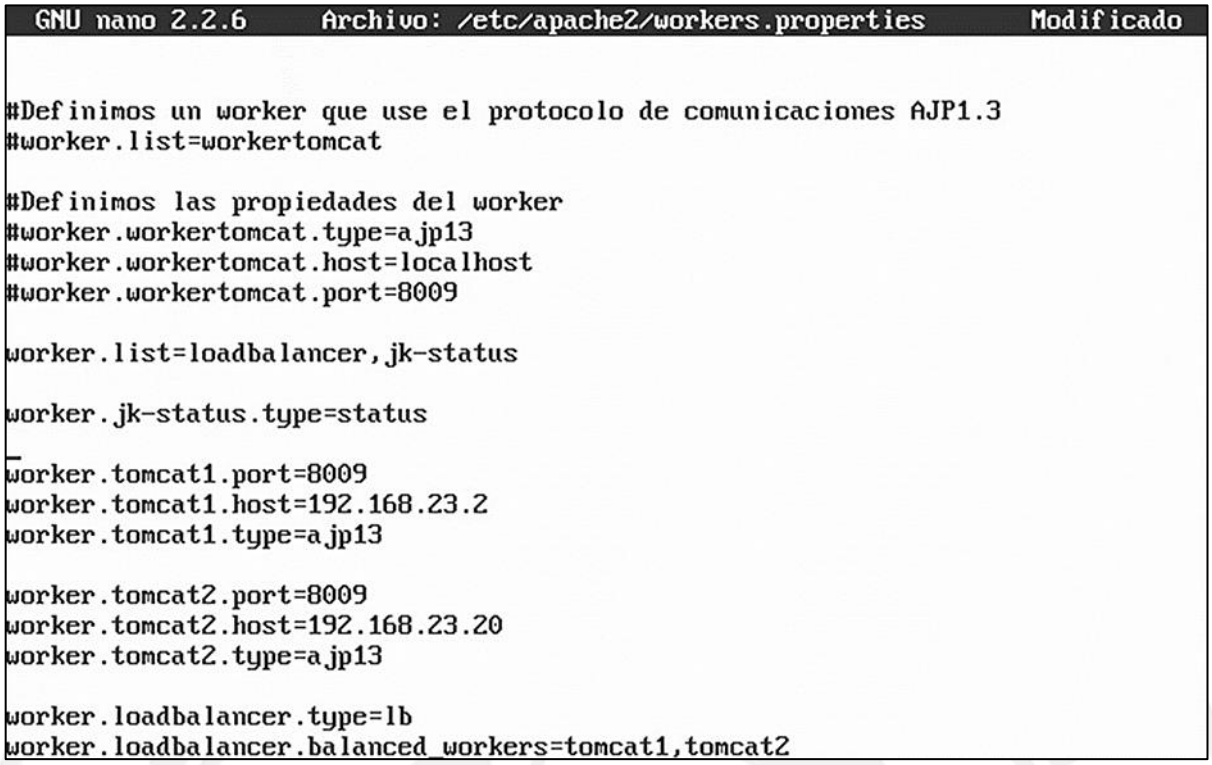


Figura 415. Fichero worker.properties de backend1

Este primer paso solo hay que hacerlo en `backend1`, puesto que será el balanceador de carga (por otro lado, `backend2` tiene *Apache 2* desactivado). Hecho esto, la siguiente operación sería añadir las siguientes líneas al final de `apache2.conf` para configurar el comportamiento del `mod_jk` y hacer que se cargue el fichero `worker.properties` creado, aunque esta línea ya debería estar presente si hemos hecho la instalación de *Tomcat* propuesta anteriormente:

```
#Ruta del worker.properties de antes
JkWorkersFile /etc/apache2/worker.properties
```

Ahora debemos editar el fichero `jk.conf`, la configuración del módulo `mod_jk`. En este fichero tenemos que hacer operaciones más importantes, como por ejemplo establecer la ruta donde se dará acceso al clúster *Tomcat* y también habilitar una ruta para ver el estatus del clúster si quisiésemos tenerlo accesible vía web. Mediante `JkMount`, en la imagen siguiente vemos que el balanceador de carga se ejecuta cuando accedamos a la URI `http://<ip del servidor Apache>/contadorVisitas` y también que se ha definido una localización nueva, `/jkstatus`, que permite ver el estado actual del clúster de servidores (nótese que hemos hecho que solo nuestro host *Windows* pueda ver dicha página limitándolo por IP):

```
 GNU nano 2.2.6    Archivo: /etc/apache2/mods-enabled/jk.conf      Modificado

    # or copy it logically there by including "JkMountCopy On"
    # in the virtual host.
    # Add an appropriate authentication method here!
    <Location /jk-status>
        # Inside Location we can omit the URL in JkMount
        JkMount jk-status
        Order deny,allow
        Deny from all
        Allow from 192.168.56.1
    </Location>
#    <Location /jk-manager>
#        # Inside Location we can omit the URL in JkMount
#        JkMount jk-manager
#        Order deny,allow
#        Deny from all
#    </Location>

    JkMount /contadorVisitas/ loadbalancer
```

Figura 416. Puntos de acceso para el clúster y el estatus del clúster

Aunque hemos eliminado todos los hosts virtuales de *Apache 2*, conviene quitar el `JkMount` del site por defecto que podríamos tener de la instalación de *Tomcat* anterior, como se ve en la siguiente imagen:

```
 GNU nano 2.2.6 Archivo: ...apache2/sites-enabled/000-default.conf

<VirtualHost *:80>
        # The ServerName directive sets the request scheme, hostname and port t$
        # the server uses to identify itself. This is used when creating
        # redirection URLs. In the context of virtual hosts, the ServerName
        # specifies what hostname must appear in the request's Host: header to
        # match this virtual host. For the default virtual host (this file) this
        # value is not decisive as it is used as a last resort host regardless.
        # However, you must set it for any further virtual host explicitly.
        #ServerName www.example.com

        ServerAdmin webmaster@localhost
        DocumentRoot /var/www/html

        JkMount /contadorVisitas* loadbalancer
```

Figura 417. Eliminar JkMount previos a la misma ruta

La idea de estas modificaciones es que Apache balancee la carga que vaya dirigida a la ruta `/contadorVisitas` y a todo fichero o ruta que depende de la misma entre ambos *Tomcat*. Por ejemplo, si nos conectamos a http://192.168.56.2/contadorVisitas, *Apache 2* decidirá si esa petición la atiende el *Tomcat* de la máquina *master* (`56.2`, ella misma) o el del otro nodo (`56.20`). Recordemos que podemos añadir todas las rutas balanceadas que necesitemos sobre todas las aplicaciones definidas en los *Tomcat* (se recuerda que `/contadorVisitas` ya tiene una aplicación *Tomcat* asociada en la máquina desde la que partimos).

No obstante, esta configuración por sí sola no es suficiente para tener el clúster operativo. Otro paso que debemos dar es decirle a *Tomcat* que las aplicaciones a servir a través del clúster son "clusterizables" (es decir, declaramos que pueden funcionar correctamente dentro de un clúster). Para ello debemos crear unos ficheros de configuración en rutas específicas dentro de la carpeta donde está alojada la aplicación que le digan a *Tomcat* que esta aplicación lo va a ser. En la siguiente imagen se resaltan los directorios concretos que permiten este comportamiento. Creamos pues las carpetas `WEB-INF` y `META-INF` dentro de

la ruta donde esté alojada la aplicación, ya que la aplicación de pruebas, al ser tan sencilla, no necesitaba previamente la existencia de estos directorios:

```
operario@backend1:/var/lib/tomcat7/webapps/contadorVisitas$ sudo mkdir WEB-INF
operario@backend1:/var/lib/tomcat7/webapps/contadorVisitas$ sudo mkdir META-INF
operario@backend1:/var/lib/tomcat7/webapps/contadorVisitas$ ls -la
total 24
drwxr-xr-x 4 root    root    4096 ago 31 10:56 .
drwxrwxr-x 4 tomcat7 tomcat7 4096 ago 20 13:42 ..
-rw-r--r-- 1 root    root     459 ago 20 13:47 index.jsp
-rw-r--r-- 1 root    root     459 ago 20 13:51 index.txt
drwxr-xr-x 2 root    root    4096 ago 31 10:56 META-INF
drwxr-xr-x 2 root    root    4096 ago 31 10:56 WEB-INF
operario@backend1:/var/lib/tomcat7/webapps/contadorVisitas$ _
```

Figura 418. Carpetas WEB-INF y META-INF de una aplicación Tomcat

Adicionalmente, en `/usr/share/tomcat7/webapps/contadorVisitas/WEB-INF` crearemos el fichero `web.xml` con el siguiente contenido:

```
 <?xml version="1.0" encoding="ISO-8859-1"?>
 <web-app xmlns="http://java.sun.com/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"        version="2.5">
       <distributable/>
 </web-app>
```

Y también, en `/usr/share/tomcat7/webapps/contadorVisitas/META-INF` crearemos el fichero `context.xml` con el siguiente contenido:

```
<?xml version="1.0" encoding="UTF-8"?>
<Context distributable="true" />
```

Ahora que tenemos configurado el balanceador de carga, las aplicaciones que vamos a balancear y hemos marcado dichas aplicaciones como aptas para ser servidas dentro de un clúster, tenemos que indicar a cada *Tomcat* del clúster que va a funcionar precisamente como un clúster. Para ello, hay que modificar el fichero `/etc/tomcat7/server.xml` de cada *Tomcat* para incluir unos cambios de acuerdo con los siguientes criterios:

Añadir un atributo `jvmRoute` a la etiqueta `Engine` cuyo valor sea el mismo que el nombre asignado al nodo en el `worker.properties`. En nuestro caso sería así para el `backend1`: `<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat1">`

```
GNU nano 2.2.6        Archivo: /etc/tomcat7/server.xml        Modificado

<!-- Define an AJP 1.3 Connector on port 8009 -->

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />


<!-- An Engine represents the entry point (within Catalina) that processes
     every request.  The Engine implementation for Tomcat stand alone
     analyzes the HTTP headers included with the request, and passes them
     on to the appropriate Host (virtual host).
     Documentation at /docs/config/engine.html -->

<!-- You should set jvmRoute to support load-balancing via AJP ie :
<Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">
-->
<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat1">

  <!--For clustering, please take a look at documentation at:
      /docs/cluster-howto.html  (simple how to)
      /docs/config/cluster.html (reference documentation) -->
  <!--
  <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
  -->
```

Figura 419. server.xml de backend1

```
GNU nano 2.2.6        Archivo: /etc/tomcat7/server.xml        Modificado

<!-- Define an AJP 1.3 Connector on port 8009 -->

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />


<!-- An Engine represents the entry point (within Catalina) that processes
     every request.  The Engine implementation for Tomcat stand alone
     analyzes the HTTP headers included with the request, and passes them
     on to the appropriate Host (virtual host).
     Documentation at /docs/config/engine.html -->

<!-- You should set jvmRoute to support load-balancing via AJP ie :
<Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">
-->
<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat2">

  <!--For clustering, please take a look at documentation at:
      /docs/cluster-howto.html  (simple how to)
      /docs/config/cluster.html (reference documentation) -->
  <!--
  <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"/>
  -->

  <!-- Use the LockOutRealm to prevent attempts to guess user passwords
```

Figura 420. server.xml de backend2

Hecho esto, queda la parte más difícil, que consiste en completar la sección de gestión de clústeres añadiendo el contenido que se muestra en el próximo cuadro. Esta es la configuración recomendada por el fabricante para la creación de un clúster de tamaño moderado como el que estamos montando. Partes importantes de la configuración del clúster son:

- **El tráfico de control del clúster** (qué nodos están "vivos", nodos que entran en servicio en un momento dado, cuándo hace falta replicar sesiones, etc.) se envía **por una dirección *multicast* compartida por todos los nodos del clúster**. Esta IP es la que determina si un *Tomcat* es miembro o no del clúster, y se configura en la sección `<Membership>` del fichero `server.xml`, como veremos más abajo. Por ese motivo la IP *multicast* del ejemplo `228.0.0.4` debe ser la misma en ambos nodos.
- Además, una dirección *multicast* no puede funcionar **si no establecemos una ruta para la misma por alguno de los interfaces de red**, es decir, tenemos que indicar al SO que esa dirección es válida y por qué tarjeta de red tendrá que encaminar el tráfico destinado a ella en ambas máquinas (en este caso la tarjeta de nuestra red interna, aunque podría ser otra red dedicada exclusivamente a este tipo de tráfico). Para ello ejecutamos (en ambas máquinas): `route add -host 228.0.0.4 dev eth1`

```
operario@backend1:/etc/tomcat7$ sudo route add -host 228.0.0.4 dev eth1
```

Figura 421. Añadiendo una ruta multicast para comunicación entre los nodos del clúster

- **Si se reinician las máquinas** y esta ruta desaparece (ejecutar el comando `route`), conviene introducir este comando en algún script de inicialización.
- **El atributo** `address` del tag `<Receiver>` es la IP del propio servidor (debemos pues cambiarla en cada servidor del clúster). Esto es así porque es el canal a través del cual se reciben los datos de las sesiones de los usuarios.

El resto de los elementos del fichero son módulos necesarios para que el clúster haga su labor correctamente, pero que su explicación conlleva meterse en temas técnicos complejos propios de *Tomcat* y excede los objetivos de este libro. Se puede encontrar la explicación de todos ellos en la documentación oficial referenciada más abajo. La sección `Cluster` de ambos *Tomcat* debería quedar como sigue. Se ha marcado en negrita el valor que debe cambiar entre un servidor *Tomcat* y el otro (es decir, la IP del `Reciever`, que en `backend1` será `23.2` y en `backend2` será `23.20`):

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat1">
  <!--For clustering, please take a look at documentation at: /docs/cluster-howto.html
(simple how to) /docs/config/cluster.html (reference documentation) -->

  <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster" channelSendOptions="8">
     <Channel className="org.apache.catalina.tribes.group.GroupChannel">
      <Membership className="org.apache.catalina.tribes.membership.McastService"
address="228.0.0.4" port="45564" frequency="500" dropTime="3000"/>
       <Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
         <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
       </Sender>
       <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
address="192.168.23.2" port="4000" autoBind="100" selectorTimeout="5000" maxThreads="50"/>
      <Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
      <Interceptor
```

```
className="org.apache.catalina.tribes.group.interceptors.MessageDispatch15Intercep tor"/>
   </Channel>
   <Valve className="org.apache.catalina.ha.tcp.ReplicationValve" filter=""/>
   <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>
   <ClusterListener
className="org.apache.catalina.ha.session.JvmRouteSessionIDBinderListener"/>
   <ClusterListener className="org.apache.catalina.ha.session.ClusterSessionListener"/>
</Cluster>
```

```
GNU nano 2.2.6              Archivo: server.xml

<Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">
-->
<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat1">

  <!--For clustering, please take a look at documentation at:
      /docs/cluster-howto.html  (simple how to)
      /docs/config/cluster.html (reference documentation) -->

    <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
             channelSendOptions="8">

      <Channel className="org.apache.catalina.tribes.group.GroupChannel">
        <Membership className="org.apache.catalina.tribes.membership.McastS$
                    address="228.0.0.4"
                    port="45564"
                    frequency="500"
                    dropTime="3000"/>
        <Receiver className="org.apache.catalina.tribes.transport.nio.NioRe$
                  address="192.168.23.2"
                  port="4200"
                  autoBind="100"
                  selectorTimeout="5000"
                  maxThreads="6"/>

        <Sender className="org.apache.catalina.tribes.transport.Replication$
```

Figura 422. Extracto de la sección <Cluster> del server.xml de backend1

```
GNU nano 2.2.6        Archivo: /etc/tomcat7/server.xml          Modificado

<!-- You should set jvmRoute to support load-balancing via AJP ie :
<Engine name="Catalina" defaultHost="localhost" jvmRoute="jvm1">
-->
<Engine name="Catalina" defaultHost="localhost" jvmRoute="tomcat2">

  <!--For clustering, please take a look at documentation at:
      /docs/cluster-howto.html  (simple how to)
      /docs/config/cluster.html (reference documentation) -->

    <Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
             channelSendOptions="8">

      <Channel className="org.apache.catalina.tribes.group.GroupChannel">
        <Membership className="org.apache.catalina.tribes.membership.McastS$
                    address="228.0.0.4"
                    port="45564"
                    frequency="500"
                    dropTime="3000"/>
        <Receiver className="org.apache.catalina.tribes.transport.nio.NioRe$
                  address="192.168.23.20"
                  port="4200"
                  autoBind="100"
                  selectorTimeout="5000"
                  maxThreads="6"/>
```

Figura 423. Extracto de la sección <Cluster> del server.xml de backend2

Finalmente, en el fichero `context.xml` hay que habilitar un gestor de sesiones. Si no indicamos lo contrario, por defecto se usará el gestor de sesiones llamado `DeltaManager`. Se recomienda consultar la documentación oficial (ver final de la sección) para ver en qué consiste y qué otras opciones hay, pero para nuestro ejemplo usar este es adecuado. Lo que si debemos hacer es decirle al gestor de sesiones elegido es que no borre las sesiones si un nodo cae, para que así la replicación pueda producirse correctamente. Aunque esta configuración podría estar también presente dentro de sección `Cluster` del fichero `server.xml`, parece ser que ciertas versiones de *Tomcat 7* tienen un bug que impide que sea leída correctamente si lo situamos ahí en lugar de en el fichero `context.xml`. El fichero `context.xml` mencionado debe ser así en ambos nodos *Tomcat*:

```
GNU nano 2.2.6       Archivo: /etc/tomcat7/context.xml

 WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 See the License for the specific language governing permissions and
 limitations under the License.
-->
<!-- The contents of this file will be loaded for each web application -->
<Context>

    <!-- Default set of monitored resources -->
    <WatchedResource>WEB-INF/web.xml</WatchedResource>

    <!-- Uncomment this to disable session persistence across Tomcat restarts -$
    <!--
    <Manager pathname="" />
    -->

    <!-- Uncomment this to enable Comet connection tacking (provides events
         on session expiration as well as webapp lifecycle) -->
    <!--
    <Valve className="org.apache.catalina.valves.CometConnectionManagerValve" />
    -->

    <Manager className="org.apache.catalina.ha.session.DeltaManager"
                expireSessionsOnShutdown="false"
                notifyListenersOnReplication="true" />
</Context>

^G Ver ayuda  ^O Guardar    ^R Leer Fich ^Y RePág.    ^K Cortar Tex^C Pos actual
^X Salir      ^J Justificar^W Buscar     ^V Pág. Sig. ^U PegarTxt  ^T Ortografía
```

Figura 424. Fichero context.xml de ambos Tomcat

En todos los ficheros de configuración que hemos visto se han usado nºs de puerto para ciertos servicios. Ahora mismo estos puertos están cerrados en el *firewall* de cada máquina, por lo que **conviene abrirlos en ambos *Tomcat*** para que la infraestructura funcione correctamente. La forma de abrir puertos ya la hemos visto (`sudo ufw allow <nº de puerto>`) y tenemos que abrir los puertos `4000`, `4200`, `8009` y `45564`.

Ahora ya podemos por fin reiniciar ambos *Tomcat* tras hacer estos cambios y estamos en condiciones de probar si el clúster funciona correctamente.

Pruebas de funcionamiento

Las pruebas de funcionamiento son sencillas puesto que básicamente accederemos a la IP del clúster y, gracias a que la página en el clúster delata qué nodo la sirve, veremos quien ha considerado oportuno Apache que sirva las peticiones inicialmente. En esta imagen podemos ver el resultado de la ejecución de múltiples peticiones sobre la web del clúster:

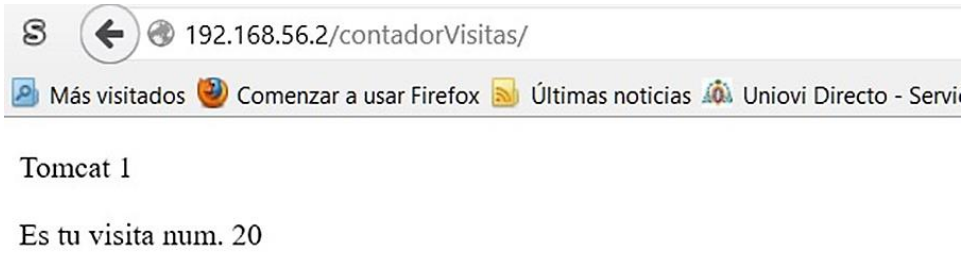


Figura 425. Peticiones sobre el clúster

Ya que nos está sirviendo el primer nodo, podemos parar el *Tomcat* que se ejecuta en el mismo (`sudo /etc/init.d tomcat7 stop`) y hacer una nueva petición al clúster, ocurriendo esto:

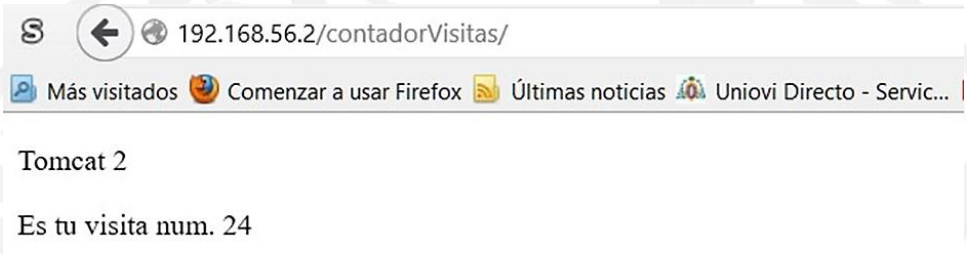


Figura 426. Clúster operativo con un nodo fuera de servicio

Es importante ver como el contador de visitas sigue contando a partir del valor que tenía la aplicación antes de que se parase el nodo que la servía, y que este proceso es completamente transparente al usuario (salvo quizá, un ligero retardo en la respuesta la primera vez que `backend2` se hace cargo de servir las peticiones).

Por otro lado, si quisiésemos saber en qué estado está el clúster en cualquier momento, podemos usar la página de status que el propio *Tomcat* provee para estos casos y para la que hemos creado una localización accesible (`/jk-status`):

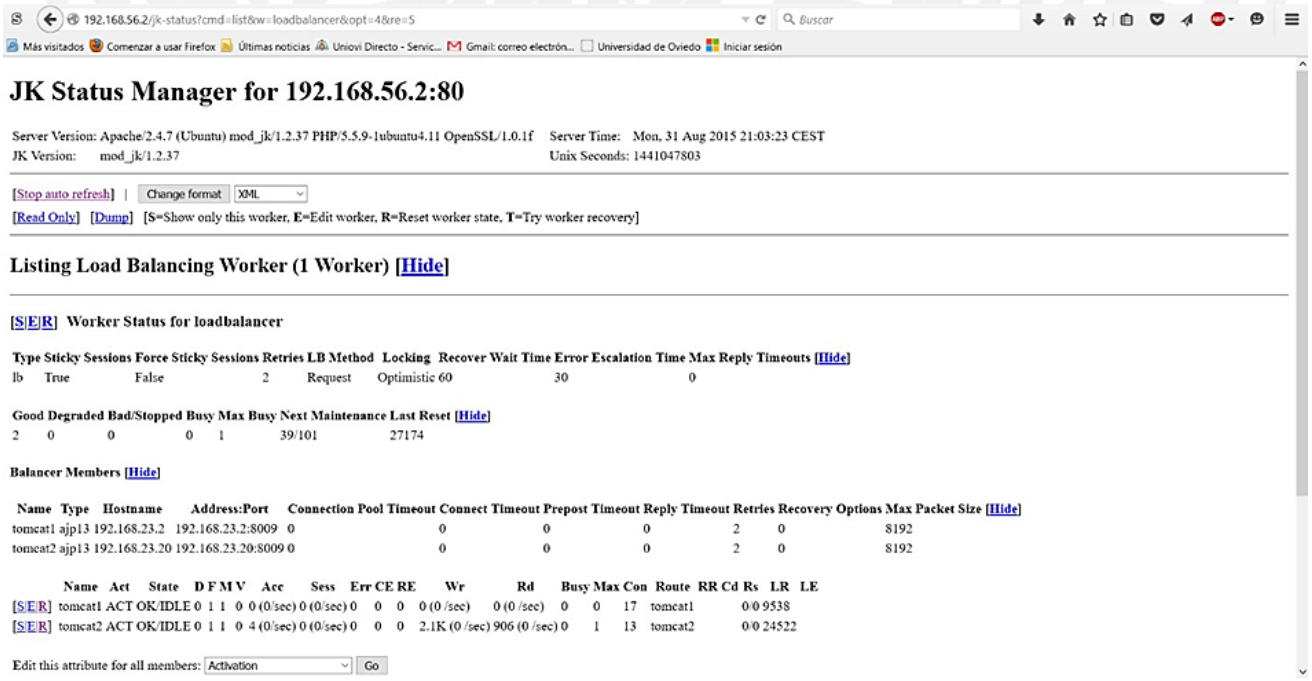


Figura 427. Página de estatus del clúster

Llegados a este punto, ya hemos visto cómo se instala y pone en marcha un clúster *Tomcat* con balanceador *Apache 2* y replicación automática de sesiones. Esta infraestructura podemos usarla con cualquier aplicación que use sesiones sin más que incluirla como aplicación *Tomcat* en la ruta adecuada y permitir su acceso desde *Apache 2*. No obstante, la gestión de clústeres con *Tomcat* tiene más opciones. Para saber más datos y opciones de cómo montar un clúster *Tomcat* + *Apache 2*, se recomienda consultar la documentación oficial: http://tomcat.apache.org/tomcat-7.0-doc/cluster-howto.html. Aquí podremos encontrar aspectos de configuración adicionales importantes, especialmente un gestor de sesiones más adecuado para clústeres grandes.

Clúster *Tomcat* vertical con replicación de sesiones bajo *Linux*

Lo que hemos visto hasta ahora es lo que se denomina un clúster horizontal (varios *Tomcat* en diferentes máquinas). Un clúster vertical es lo mismo, pero hay varios Tomcat instalados dentro de la misma máquina. Si quisiéramos montar un clúster vertical, habría que hacer una variante del anterior ejercicio, pero usando un solo servidor. La principal ventaja de un clúster vertical es que permite explotar más los recursos de la máquina, aunque a costa de una menor disponibilidad en caso de fallos. Los pasos a dar son muy similares a los vistos, con las siguientes diferencias:

- Los *Tomcats* involucrados **tienen una instalación independiente**, en rutas diferentes y con *scripts* de inicialización y parada particulares a los mismos.
- *Apache* 2 y *Tomcats* **van en la misma máquina**, por lo que sus distintos servicios tienen que escuchar en puertos diferentes. A modo de guía de cómo gestionar los puertos se recomienda usar este enlace: http://www.easywayserver.com/implementation-tomcat-clustering.htm

El resto de las operaciones son las mismas que en el caso anterior, aunque teniendo en cuenta que sólo usamos una máquina para gestionarlo todo.

Otras arquitecturas compuestas

Aunque a lo largo de este documento hemos visto arquitecturas que contemplan varias máquinas siempre desde el punto de vista del ahorro de recursos a la hora de hacer las pruebas, eso no quiere decir que no podamos crear arquitecturas mucho más grandes y complejas combinando los elementos vistos para hacer arquitecturas que cumplan con objetivos determinados.

En esta sección vamos a ver una serie de diagramas que son algunos ejemplos de cómo combinar todos los contenidos vistos para crear arquitecturas más grandes que cumplan mejor con propósitos determinados.

Por supuesto, no son las únicas posibles, solo ejemplos adaptados a determinadas necesidades comunes. Una vez que se tiene el dominio de los elementos vistos, es posible "dejar volar la imaginación" y hacer combinaciones de estos elementos adaptadas a cualquier necesidad que tengamos, siempre y cuando, claro, contemos con el hardware adecuado. En todos los ejemplos presentados las máquinas marcadas con un globo terráqueo (*proxies* fortificados con las herramientas vistas) son las que serían visibles desde el exterior.

Empresa con redes para empleados y redes para la web corporativa separadas

Esta arquitectura muestra cómo sería una empresa donde existe una red para que sus empleados hagan su trabajo cotidiano y naveguen a través de un *proxy* (libremente o con restricciones de dominios). Por

otro lado, las páginas web de la empresa, desarrolladas en tres lenguajes: ASP .NET, PHP y JSP, están en un *hosting* completamente separado, en una red independiente y no relacionada con la anterior. Esta red está gobernada por un *reverse proxy* que pasará la petición al servidor adecuado en función de la aplicación que se solicite. Existe además un servidor por lenguaje en lugar de uno para todos ellos.

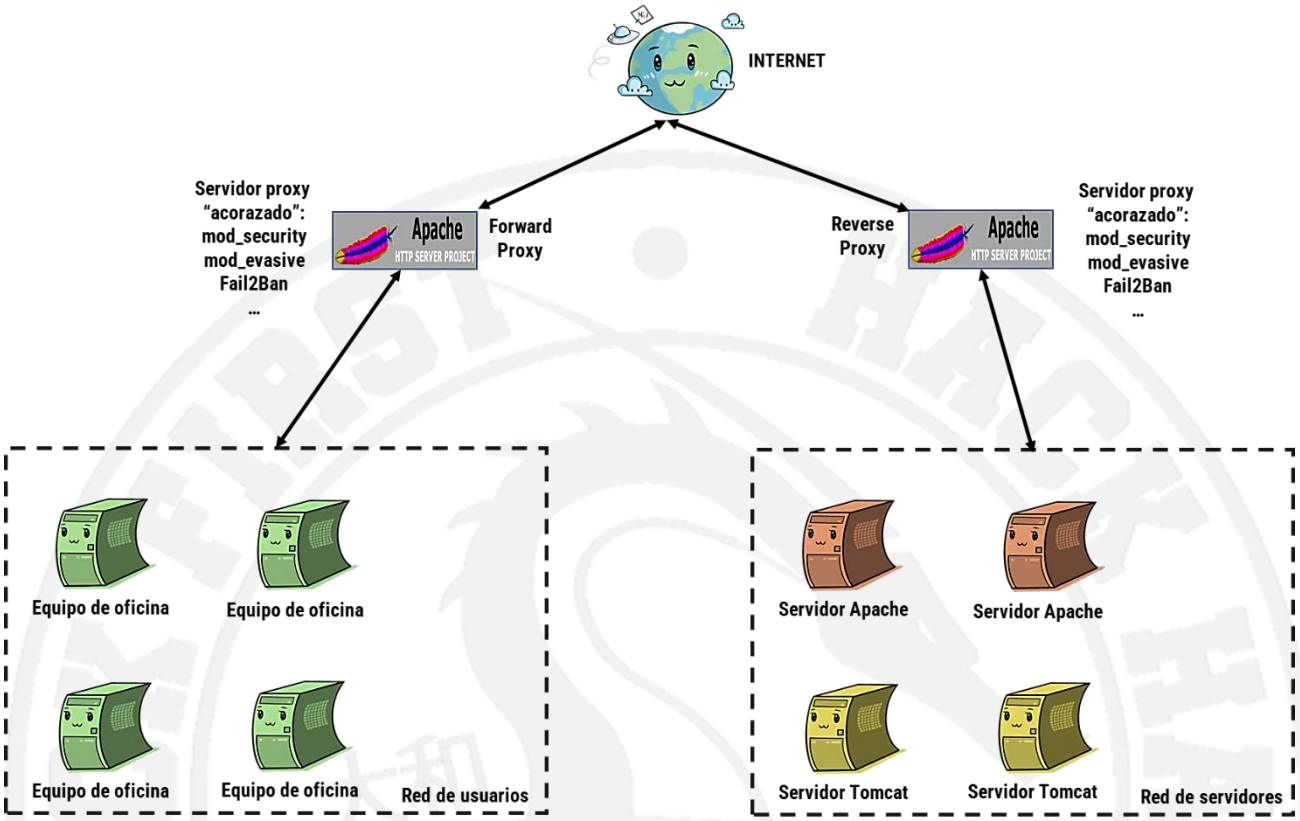


Figura 428. Empresa con redes para empleados y redes para la web corporativa separadas

Empresa de desarrollo de *software* con red de desarrollo y producción

Este esquema muestra una empresa que tiene contratados dos equipos de programadores. Uno de ellos desarrolla sobre C# (ASP .NET) y el otro en JSP, con su propia red para desarrollo y pruebas. Desde dicha red se puede navegar por Internet de forma libre o limitada gracias a un proxy forward. Una vez que un equipo de desarrollo ha terminado su trabajo, se conecta a una máquina llamada "*master* de réplicas" residente en una red independiente de producción, gobernada por un reverse proxy. Esta máquina tiene dos características:

- El *reverse proxy* no tiene una ruta para ella, no es accesible desde el exterior.
- Posee un servicio `rsync` al que se conectan el resto de los nodos de producción para actualizar automáticamente las webs servidas cuando hay una nueva versión que se ha liberado desde la red de desarrollo.

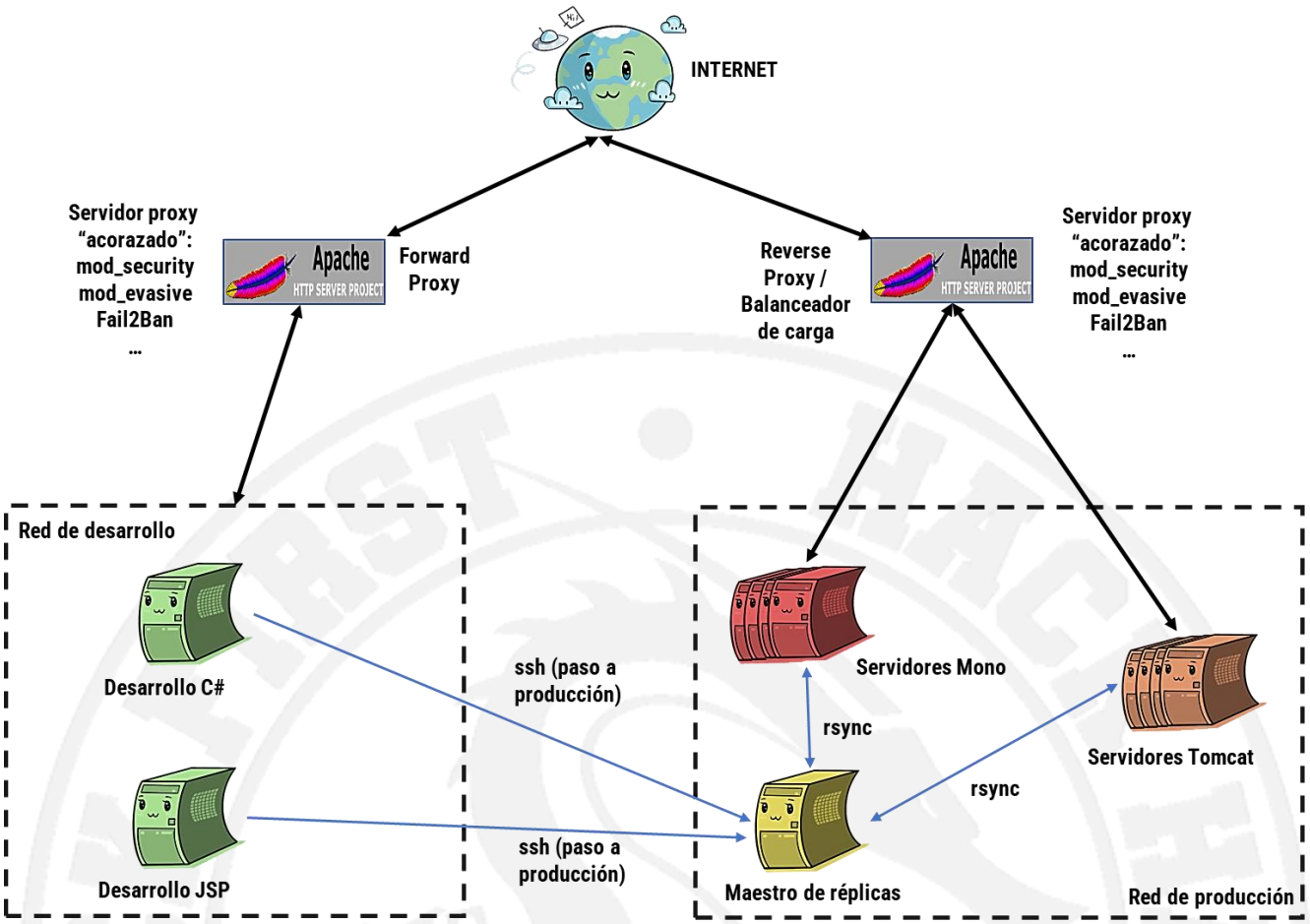


Figura 429. Empresa de desarrollo de software con red de desarrollo y red de producción

Infraestructura de balanceo de carga y alta disponibilidad para altas cargas de trabajo

Este esquema está formado por N balanceadores de carga para N grupos de M nodos, todos sirviendo la misma aplicación. Dicho de otro modo, tenemos N granjas web sirviendo la misma aplicación, sometida a una gran carga de tráfico. Para elegir entre un clúster u otro, se usa la técnica de *Round-Robin DNS*, al ser indiferente para el usuario a cuál de ellos acceder. Lógicamente, en caso de que hubiera sesiones habría que guardarlas en algún lugar centralizado o bien usar replicación, algo que no se contempla en este diagrama.

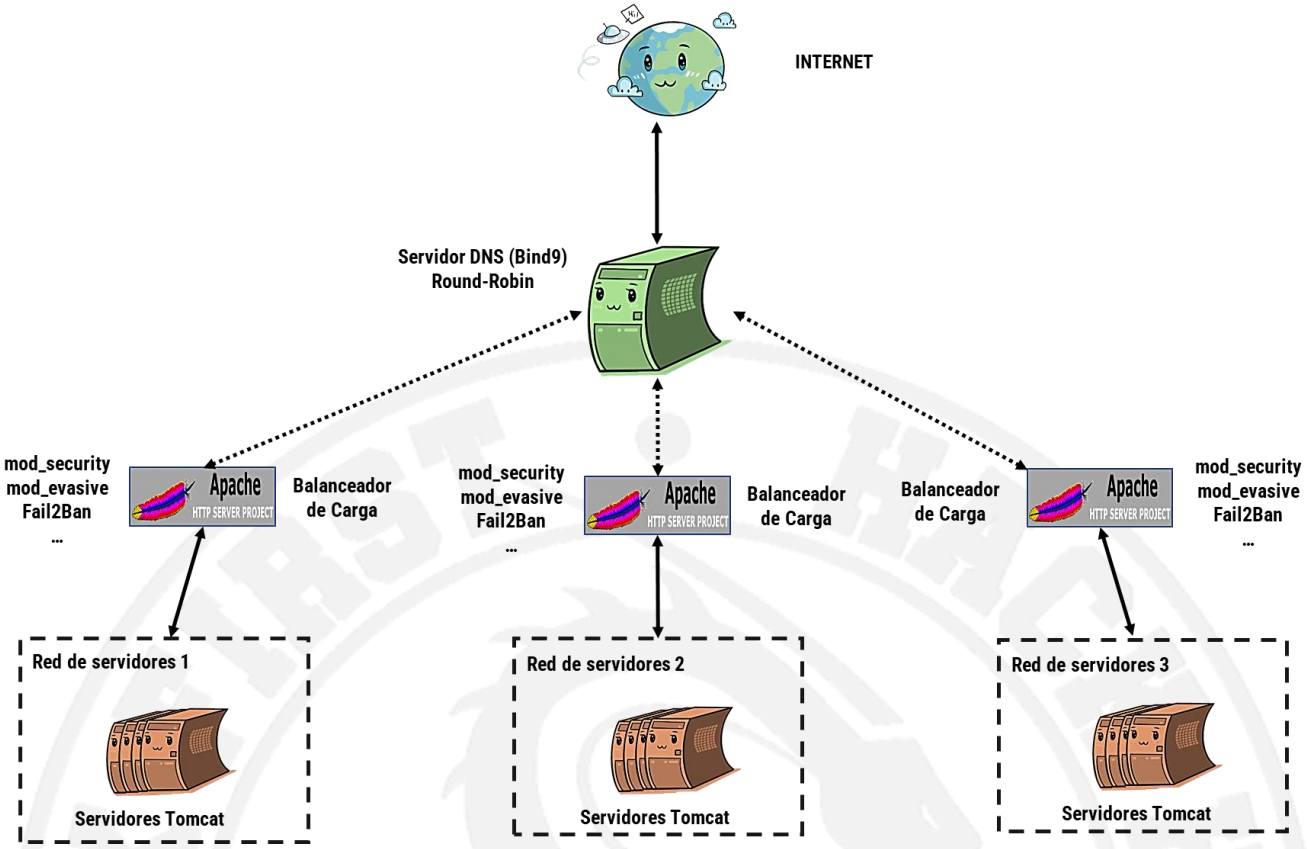


Figura 430. . Infraestructura de balanceo de carga y alta disponibilidad para altas cargas de trabajo

Infraestructura de servicios realizados en varias tecnologías ofertados en Internet

Este esquema consiste en un *reverse proxy* que sirve de única puerta de acceso a 3 aplicaciones diferentes, desarrolladas en tres tecnologías diferentes y que además están en clúster. La fortificación de los *proxys* se ha movido a los balanceadores, con la idea de que sea viable que puedan dar sus servicios de forma pública en lugar de solo a través del *reverse proxy* de una manera sencilla.

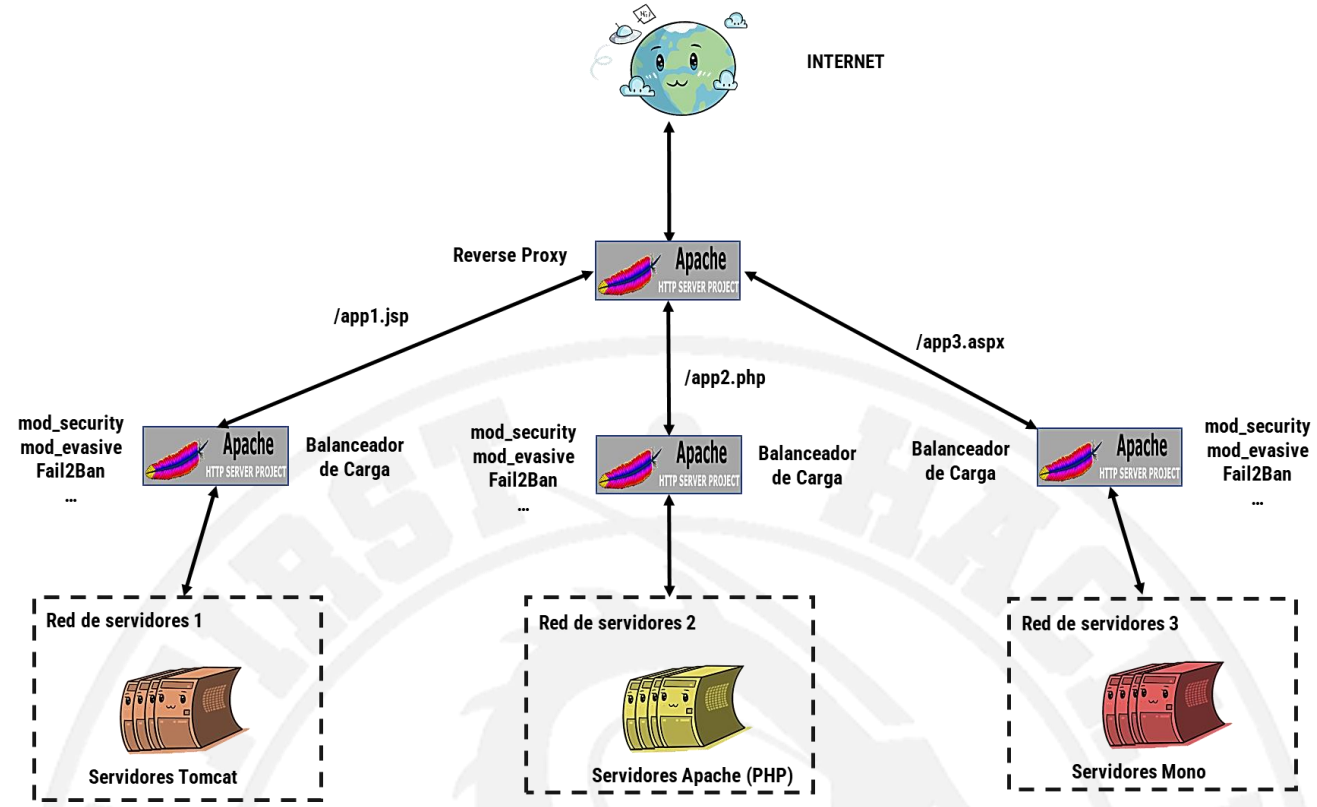


Figura 431. Infraestructura de servicios realizados en varias tecnologías ofertados en Internet

Redes internas de una empresa donde el acceso de los usuarios a recursos está controlado

Este esquema ilustra dos redes de usuarios separadas de una misma empresa (podrían ser dos sedes en diferentes países, provincias…), ambas con salida controlada a Internet por un *forward proxy*. El acceso de los recursos de una red a otra no es libre, sino controlado por dos *forward proxy* (de la red 1 a la red 2 y viceversa) que marcan qué recursos son accesibles y qué recursos no lo son entre ambas.

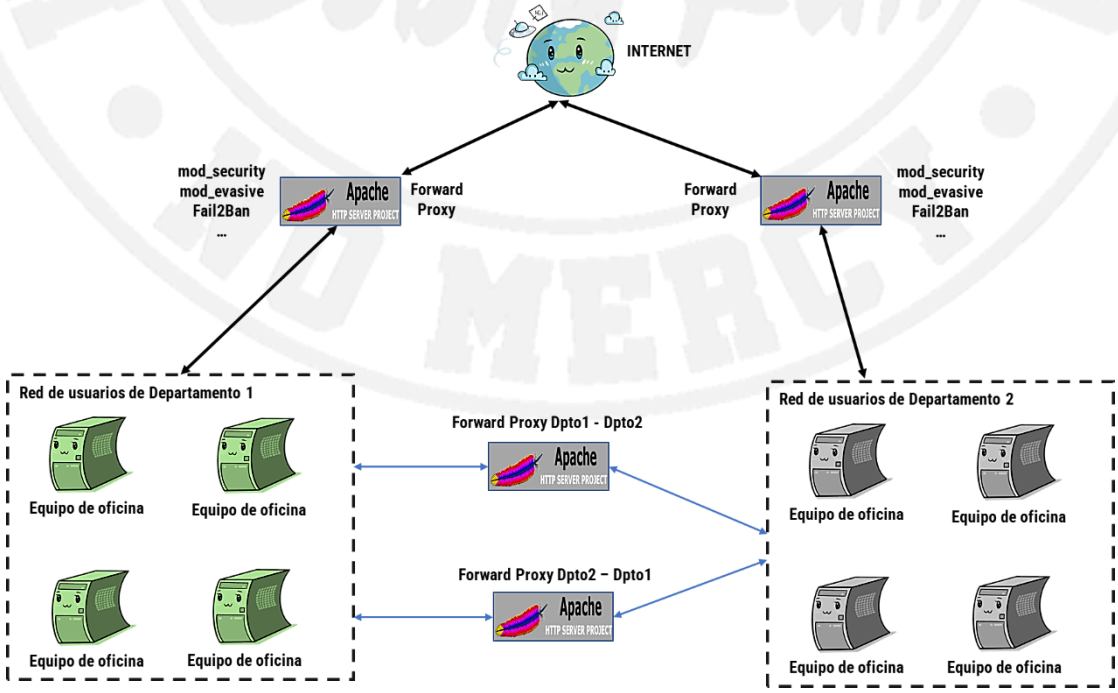


Figura 432. Redes internas de una empresa donde el acceso de los usuarios a recursos está controlado

Mashup de Servicios

Este esquema ilustra una gran aplicación web proveedora de servicios que necesita los servicios de tres empresas distintas. Cada empresa proporciona varios servicios de diferentes tipos y la empresa los integra todos en una misma aplicación web, llamándolos cuando es necesario. Lógicamente, esta aplicación no es el único cliente de esos servicios, sino que se ofertan públicamente para cualquiera que necesite usarlos.

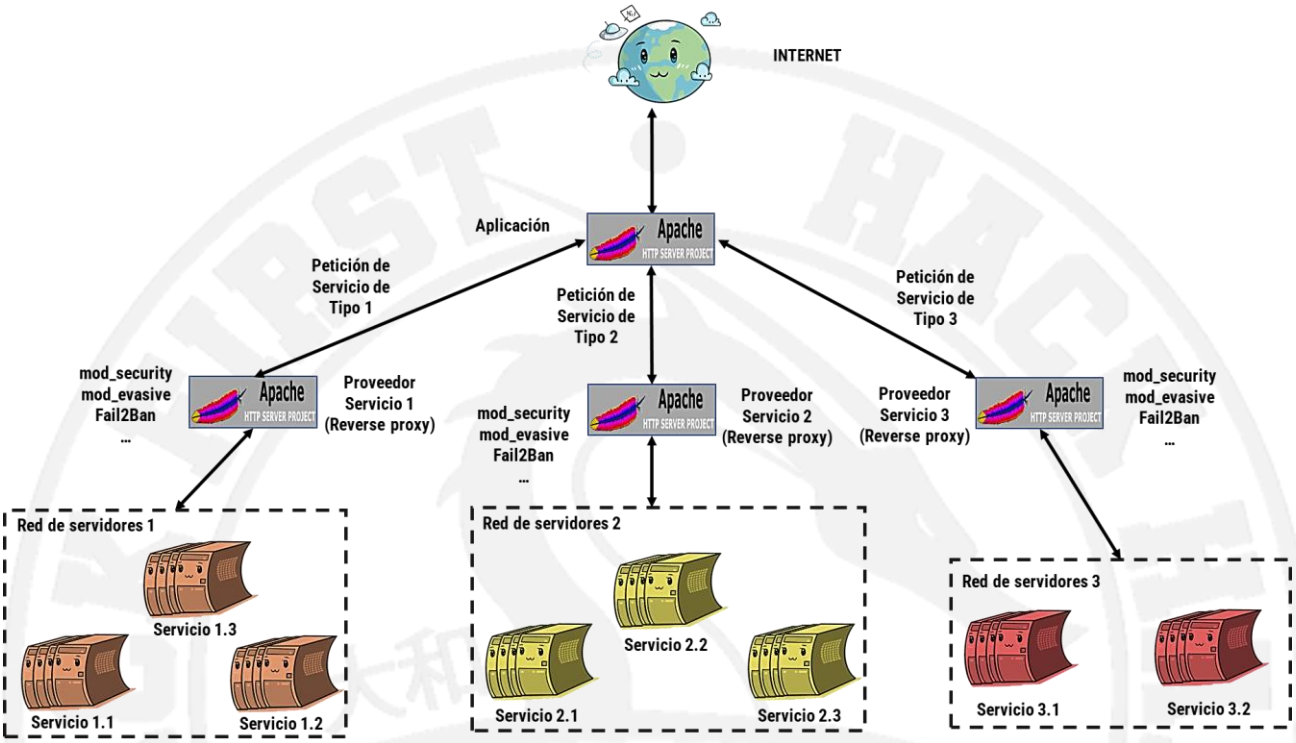


Figura 433. Mashup de Servicios