Control de acceso a la web

(Artículo original de Pedro Pablo Fábrega Martínez)

(Conversión de HTML a ODT y PDF por jamj2000 at gmail dot com – Enero 2011)

Índice de contenido

CONTROL DE ACCESO MEDIANTE SQUID

El filtrado de accesos es un tema que genera polémica en cuanto algunos lo pueden interpretar como una restricción o censura mientras que otros lo consideran como una protección. Esta no es la discusión que nos ocupa, aquí sólo pretendo dar una visión técnica sobre como establecer filtros de contenidos con la esperanza de que se haga buen uso de ella.

El filtrado de accesos es también una herramienta para mantener la seguridad y privacidad de los usuarios de nuestra red en tanto que podremos protegerlos de ciertos programas de spyware, popup y otras prácticas molestas de la web.

Para controlar los accesos primero, tenemos que saber definir las propiedades de las conexiones sobre las que queremos establecer algún tipo de control. Las acl de Squid permiten especificar acl según muy diversos criterios, por ejemplo:

1. Una dirección de red
2. Una dirección de máquina
3. Un rango de direcciones
4. Una URL de una conexión
5. Una franja horaria

CONCEPTO DE ACL EN SQUID

Un ACL es una definición de control de acceso, que en Squid se especifica mediante el parámetro acl según la siguiente sintaxis:

```
acl nombre_acl tipo_acl descripción ...
acl nombre_acl tipo_acl "fichero_de_descripciones" ...
```

Cuando usamos un "fichero_de_descripciones", cada descripción se corresponde con una línea del fichero.

TIPOS DE ACL

src

Especifica una dirección origen de una conexión en formatio IP/máscara.

Por ejemplo, utilizaremos una acl de tipo src para especificar la red local:

```
acl red_local src 192.168.1.0/24
```

También podemos especificar rangos de direcciones mediante una acl de tipo src:

```
acl jefes src 192.168.1.10-192.168.1.25/32
```

dst

Especifica una dirección destino de una conexión en formatio IP/máscara.

```
acl google_es dst 216.239.0.0/24
```

También podemos especificar hosts concretos mediante una acl de tipo dst:

```
acl google_es2 dst 216.239.59.104/32 216.239.39.104/32 216.239.57.104/32
```

Las definiciones son idénticas a las acl de tipo src salvo que se aplican al destino de las conexiones, no al origen.

srcdomain y dstdomain

Estos tipos de acl especifican un nombre de dominio.

En el caso de srcdomain es el dominio origen y se determina por resolución DNS inversa de la IP de la máquina, es decir, tendremos que tener bien configurado el DNS de la red local.

En el caso de dstdomain el nombre del dominio se comprueba con el dominio que se haya especificado en la petición de página web.

Por ejemplo:

```
acl google_com dstdomain google.com
```

srcdom_regex y dstdom_regex

Especifican una expresión regular que verifican los dominio origen o destino. La expresión regular hace distinción entre mayúsculas y minúsculas salvo que incluyamos la ción "-i" que evita dicha distinción.

Por ejemplo

```
acl google_todos dstdom_regex -i google\..*
```

Observamos como al incluir "-i" estamos indicando que no haga distinción entre mayúsculas y minúsculas.

time

Este tipo de acl perimite especificar una franja horaria concreta dentro de una semana. La sintaxis es la siguientes

```
acl nombre_acl_horaria time [dias-abrev] [h1:m1-h2:m2]
```

Donde la abreviatura del día es:

S - Sunday (domingo)

M - Monday (lunes)

T - Tuesday (martes)

W - Wednesday (miércoles)

H - Thursday (jueves)

F - Friday (viernes)

A - Saturday (sábado)

además la primera hora especificada debe ser menor que la segunda, es decir h1:m1 tiene que ser menor que h2:m2

Por ejemplo

```
acl horario_laboral time M T W H F 8:00-15:00
```

Estarímos especificando un horario de 8 a 15 y de lunes a viernes.

url_regex

Permite especificar expresiones regulares para comprobar una url completa, desde el http:// inicial.

Por ejemplo, vamos a establecer una acl que se verifique con todos los servidores cuyo nombre sea adserver:

```
url_regex serv_publicidad ^http://adserver.*
```

En otro ejemplo podemos ver una acl que verifique las peticiones de ficheros mp3:

```
url_regex ficheros_mp3 -i mp3$
```

Nota: ver expresiones regulares

referer_regex

Define una acl que se comprueba con el enlace que se ha pulsado para acceder a una determinada página. Cada petición de una página web incluye la dirección donde se ha pulsado para acceder. Si escribimos la dirección en el navegador entonces estaremos haciendo una petición directa.

Por ejemplo vamos a establecer uan acl para todas las páginas a las que hayamos accedido pulsando en una ventana de búsqueda de google:

```
acl pincha_google referer_regex http://wwww.google.*
```

req_mime

Las acl de tipo req_mime se utilizan para comprobar el tipo de petición mime que realiza un cliente, y se puede utilzar para detectar ciertas descargas de ficheros o ciertas peticiones en túneles HTTP.

Esta acl sólo comprueba las peticiones que realiza el cliente, no comprueba la respuesta del servidor. Esto es importante para tener claro qué estamos haciendo y qué no.

Por ejemplo

```
acl subida req_mime_type -i ^multipart/form-data$
acl javascript req_mime_type -i ^application/x-javascript$
acl estilos req_mime_type -i ^text/css$
acl audiompeg req_mime_type -i ^audio/mpeg$
```

rep_mime_type

Este tipo de acl se utiliza para verificar el tipo de respuesta recibida por el proxy. Este tipo de acl, analiza una respuesta del servidor por lo que sólo le afectas las reglas de respuesta como **http_reply_access** y no las reglas **http_access** que se aplican a las peticiones.

Por ejemplo

```
acl javascript rep_mime_type -i ^application/x-javascript$
acl ejecutables rep_mime_type -i ^application/octet-stream$
acl audiompeg rep_mime_type -i ^audio/mpeg$
```

EJEMPLOS DE ACL

acl todos

Una acl que verifica todos los equipos de la red. Puede parecer un poco tonta, pero se va a utilizar posteriormente para establecer una política preestablecida para denegarlo o aceptarlo todo.

```
acl todos src 0.0.0.0/0.0.0.0
```

acl localhost

Nuestra propia maquina como origen de conexiones

```
acl localhost src 127.0.0.1/255.255.255.255
```

Nuestra propia máquina como destino de las conexiones

```
acl localhost_destino dst 127.0.0.0/8
```

http_access

Este es el parámetro que permite o deniega accesos a una o más acl.

La sintaxis de uso es:

```
http_access allow|deny [!]acl ...
```

Observamos como cada acl puede ir precedida por un signo "!" que indicaría que la acl no se verifica.

Por ejemplo, para permitir acceso fuera del horario laboral, según una acl que definimos anteriormente:

```
http_access allow ! horario_laboral
```

Para denegar el acceso en horario laboral

```
http_access deny horario_laboral
```

Para dar acceso completo a la red local

```
http_access allow red_local
```

Características de http_access

Si no hay ninguna línea de acceso la acción predeterminada es denegar la petición.

Si una petición no ha verificado ninguna línea de acceso, la ación que se realiza es la opuesta a la última línea de la lista. Si la última línea deniega entonces el valor predeterminado es permitir. Por este motivo es conveniente incluir una línea "deny all" o "allow all" al final de las listas de accesos para tener las cosas claras.

Es importante la forma en la cual añadimos los distinto parámetros **http_access** para determinar la forma en la que se comprueba. En primer lugar, todas las acl incluidas en en una cláusula http_access se comprueban y todas ellas tendrán que verificarse conjuntamente, es decir, como si estuvieran unidas por un operador AND. Después, los sucesivos parámetros **http_access** se evalúan individualmente, es decir, como si estuvieran unidos mediante un operador OR.

Por ejemplo

```
acl red1 src 192.168.0.0/24
acl red2 src 192.168.1.0/24
http_access allow red1 red2
```

permitiría el acceso a todas aquellas conexiones que procedieran a la vez de red1 y de red2, que probablemente no es lo que pretendemos. Para permitir acceso a las dos redes podríamos:

```
acl red1 src 192.168.0.0/24
```

```
acl red2 src 192.168.1.0/24
http_access allow red1
http_access allow red2
```

o también, de forma más simple:

```
acl redes src 192.168.0.0/24 192.168.1.0/24
http_access allow redes
```

EJEMPLOS DE ACL Y HTTP_ACCESS

Denegar las peticiones de ficheros de tipo exe, pif, bat y cmd

```
acl tipoexe url_regex -i exe$ pif$ bat$ cmd$
http_access deny tipoexe
```

Denegar el acceso a todos los servidores cuyo nombre comience por "popup", "banner" o "ads"

```
acl incordios url_regex http://popup.* http://ads.* http://ads.*
http_access deny incordios
```

Denegar todos los ficheros de tipo exe que provengan de virus_fijo.com

```
acl anti_exe url_regex -i exe$
acl vfijo dstdomain virus_fijo.com
http_access deny anti_exe vfijo
```

Denegar las conexiones a las url completas o incompletas que hay en el fichero "/etc/squid/lista_negra_1.txt"

```
acl ln1 url_regex "/etc/squid/lista_negra_1.txt"
http_access deny ln1
```

PROTEGER EL ANCHO DE BANDA

Cada usuario tiene tendencia a utilizar el 100% del ancho de banda disponible, no sé si esto es una ley escrita, si no debería estarlo.

Pues ahora que tenemos instalada la red y el acceso mediante proxy cache vamos a intentar regular el consumo de ancho de banda. Resulta interesante que nadie pueda cometer abusos y consumir todos los recursos de la red a costa del resto de usuarios. La regulación del ancho de banda se puede llevar a cabo utilizando distintos criterios: la máquina origen, la dirección o página destino o el tipo de transferencia.

Por un lado podemos intentar discriminar el origen de los datos, resumiendo que los jefe dispongan de mejor conexión que los "pringaos". Resumiendo, vamos a asignar distintos anchos de banda a distintos rangos de direcciones IP.

También se no puede dar el caso de que haya consultas masivas a unos determinados dominios o páginas y puede ser práctico delimitar el ancho de banda asignado a ese dominio.

En general el volumen de tráfico que genera el acceso a páginas web en html es bastante bajo, lo que realmente consume un volumen apreciable del ancho de banda son ficheros de sonido e imagen y una medida posible podría ser limitar de alguna forma la transferencia de estos tipos de ficheros.

Otra posibilidad sería que limitar el ancho de banda sólo para descargas de ficheros grandes y dejar un mayor ancho de banda a ficheros pequeños para que la navegación por páginas html sea más rápida.

Mediante Squid podemos establecer límites al ancho de banda mediante una de sus características de configuración denominada "delay pools". Establecemos una definición de regulación de ancho de banda, establecemos una ACL y asociamos la regulación a la ACL.

PARÁMETROS DE CONFIGURACIÓN

Cada una de las reglas de regulación está definida por un número entero que vamos a utilizar para indentificarlo en los distintos parámetros.

delay_pools

Esta parámetro se utiliza para definir cuantas reglas de regulación vamos a definir.

delay_class

Este parámetro toma dos argumentos el primero el identificador de la regla y en segundo lugar el tipo (clase) de la regla.

El primero argumento es un número entero para identificar la regla.

El segundo argumento puede ser 1,2 ó 3 para indicar uno de estos tipos de reglas.

delay_parameters

establece los valores de la regla. Los argumentos de este parámetro son parejas de valores velocidad/tamaño, donde velociad es un número entero que indica una velocidad en bytes por segundo (B/s), y tamaño indica el número de bytes de reserva que se transmiten antes de aplicar la velocidad de transferencia.

Es decir cada pareja especifica el número de bytes de margen que se permiten antes de se haga efectiva la restricción de velocidad.

Los valores no tienen por qué ser múltiplos de 2.

TIPOS DE REGLAS

Hay tres clases de regulaciones, con características diferentes.

Clase 1

La clase uno consiste en un canal individual compartido por todos los usuarios. Es la clase más simple, todas las tasas de descarga van juntas y lo único que tenemos que especificar es la velocidad, en bytes por segundo, y el número de bytes a partir de los cuales tiene que retardar la descarga.

Ejemplo: Supongamos que queremos que la velocidad de descarga de los ficheros .wmv que ocupen más de 1Mb sea de 8k/s.

```
delay_pools 1
delay_class 1 1
delay_parameters 1 8192/1048576
acl ficheros_wmv url_regex \.wmv$
delay_access 1 allow ficheros_wmv
```

Cuando en esa red, algún equipo haya bajado más de 1Mb correspondiente a ficheros .wmv entonces la descarga total de ficheros .wmv se hará a una velocidad de 8k/s. Si quisiéramos establecer este límite para los hosts de una red deberíamos elegir la clase 2.

En el anterior ejemplo hemos supuesto que hay un único canal de regulación en el parámetro delay_pools. Ahora vamos a ver un ejemplo con dos canales de clase 1. Queremos que el tráfico de la red local no tenga límite de transferencia, mientras que para las conexiones a internet vamos a utilizar un total de 256Kbits/s.

```
delay_pools 2
delay_class 1 1
delay_parameters 1 -1/-1
acl red_local src 192.168.0.0/24
delay_access 1 allow red_local
delay_class 2 1
delay_parameters 2 32768/1024
acl resto all src 0.0.0.0/0.0.0.0
delay_access 2 allow resto
```

La tasa de transferencia viene especificada en bytes por segundo, por ejemplo convirtiendo a otras unidades uno de los parámetros de delay_parameters:

32768 B/s = 32 KB/s = 32 x 8 Kb/s (bits) = 256 Kb/s = 256 Kbits/s

El valor 1024 es un número de bytes lo suficientemente pequeño para que se alcance pronto y se estabilice el ancho de banda.

El valor -1 indica ilimitado.

Las regulaciones de clase 1 están diseñadas para evitar que la acl correspondiente consuma todo el ancho de banda, es decir definen la suma máxima de los anchos de banda de todos los clientes, pero no evita que un cliente pueda consumir ese ancho de banda a costa de otro; todos os clientes comparten un máximo ancho de banda.

Clase 2

La clase dos que consiste en un canal común con 256 canales individuales. Es decir tenemos un canal global y dentro de él otros 256 canales que se aplican a las máquinas de una red de clase C.

Ahora vamos a ver otro ejemplo con dos canales, en este caso ambos de de clase 2. Ahora también que el tráfico de la red local no tenga límite de transferencia, ni global ni por IP, mientras que para las conexiones a internet vamos a utilizar un total de 256Kbits/s y a cada máquina de la red le asigna un máximo de 64kbits/s.

```
delay_pools 2
delay_class 1 2
```

```
delay_parameters 1 -1/-1 -1/-1
acl red_local src 192.168.0.0/24
delay_access 1 allow red_local
delay_class 2 2
delay_parameters 2 32768/1024 8192/1024
acl resto all src 0.0.0.0/0.0.0.0
delay_access 2 allow resto
```

Este ejemplo es muy parecido al anterior de clase 1, sólo tenemos que agregarle una pareja de valores al parámetro "delay_parameters". La primera pareja de este parámetro especifica el ancho de banda global, mientras que la segunda pareja especifica el ancho de banda por host.

Hay que tener en cuenta que los clientes están limitados por el tamaño del canal más pequeño, por lo que no tiene sentido que el canal común tenga un tamaño menor que los canales individuales.

Clase 3

La clase tres que define un canal común que contiene 256 canales de red, 65536 canales individuales. Muy parecida a la clase dos pero par redes de clase B.

Hay que tener en cuenta que los clientes están limitados por el tamaño del canal más pequeño, por lo que no tiene sentido que el canal común tenga un tamaño menor que los canales individuales.

EJEMPLOS DE CONTROL DE ANCHO DE BANDA

Limitación global del ancho de banda

Tenemos una conexión con internet de 4Mbits y queremos que el ancho de banda dedicado a consulta de páginas web desede las IP comprendidas entre 192.168.5.100 hasta 192.168.5.199 sea de 1Mbits máximo.

1Mb/s = 128kB/s = 131072 B/s

4Mb/s = 512kB/s = 524288 B/s

```
delay_pools 1
delay_class 1 1
delay_parameters 1 131072/8192
acl lista src 192.168.5.100-192.168.5.199/32
delay_access 1 allow lista
```

Limitación de ancho de banda por red y host

Tenemos una conexión con internet de 4Mbits y queremos que el ancho de banda dedicado a consulta de páginas web desede las IP comprendidas entre 192.168.5.100 hasta 192.168.5.199 sea de 1Mbits máximo, de forma que cada una de las máquinas no pueda consumir más de 512Kbits/s.

Es como el ejemplo anterior, pero añadiendo restricciones adicionales a cada una de las máquinas, por lo que tendremos que elegir la clase 2:

```
delay_pools 1
delay_class 1 2
delay_parameters 1 131072/8192 65536/8192
acl lista src 192.168.5.100-192.168.5.199/32
delay_access 1 allow lista
```

Limitación de ancho de banda por servidor

Queremos limitar el ancho de banda a 4KB/s y que cada máquina consuma a lo sumo 2KB/s para todas las conexiones que se hagan a los servidores de banner cuyo nombre empieza por ad.

```
delay_pools 1
delay_class 1 2
delay_parameters 1 4096/4096 2048/2048
acl publicidad url_regex http://ad.*
delay_access 1 allow publicidad
```

Ancho de banda ilimitado para la red local

Garantizar un ancho de banda ilimitado para los accesos a la red local

```
delay_pools 1
delay_class 1 2
delay_parameters 1 -1/-1 -1/-1
acl redlocal url_regex -i 192.168
delay_access 1 allow redlocal
```

Establecer privilegios de acceso

Establecer privilegios en la red

```
delay_pools 3
delay_class 1 1
delay_class 2 1
```

```
delay_class 3 1
delay_parameters 1 -1/-1
delay_parameters 2 131072/8192
delay_parameters 3 65536/8192
acl jefes src 192.168.5.1-192.168.5.50/32
acl subjefes src 192.168.5.51-192.168.5.99/32
acl resto src 0/0
delay_access 1 allow jefes
delay_access 2 allow subjefes
delay_access 3 allow resto
```

Ancho de banda en horarios distintos

Limitar el ancho de banda en una franja de horario

```
acl todos src 0/0
acl laboral time 08:30-16:30
delay_pools 2
delay_class 1 2
delay_parameters 1 -1/-1 -1/-1
delay_access 1 allow todos
delay_class 2 2
delay_parameters 2 131072/8192 65536/8192
delay_access 2 allow laboral
delay_access 2 deny !laboral
delay_access 2 allow todos
```

CONFIGURACIÓN DE LOS NAVEGADORES CLIENTES

Esta configuración exige que cada navegador esté configurado para utilizar el proxy. Pero claro, por una lado es un engorro tener que configurar uno a uno todos los equipos de la red y por otro, tenemos la posibilidad de que la configuración TCP/IP le da salida a través de una pasarela.

Si a pesar de todo queremos esta configuración, tendremos que decirle al navegador que utilice como proxy el equipo que alberga Squid y el puerto el 3128. El puerto se puede cambiar en el fichero de configuración. Otro puerto que se suele usar para proxy es el 8080, pero ¿para qué cambiarlo?

PROXY TRANSPARENTE

Como queremos evitar tener que configurar los navegadores cliente y queremos

evitar posibles puertas traseras de salida vamos a configurar un proxy transparente.

¿Qué es un proxy transparente? Es un proxy que no necesita ninguna configuración especial en los clientes. Se denomina transparente porque el cliente en realidad no sabe que lo está usando, es transparente para él.

Cómo configurar el proxy transparente, pues en primer lugar tenemos que configurar el cortafuegos para que reenvíe todas las peticiones que se hagan a un puerto 80 hacia el puerto 3128 que utiliza Squid. Es decir, capturamos todas las peticiones que se hagan a un servidor http y se las enviamos a Squid para que él se encarge del resto.

Estas son las reglas de iptables. La primera para redrigir las peticiones al proxy la siguiente para rechazar el resto de los reenvíos.

Si queremos que las páginas web pasen por el proxy con squid que está en 192.168.5.254, pero el resto siga con acceso normal, pondríamos:

```
iptables -t nat -A PREROUTING -i eth1 -p tcp -m tcp --dport 80 -j DNAT --to-destination 192.168.5.254:3128
```

Si el equipo con squid está en la misma pasarela entonces podemos poner:

```
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j REDIRECT --to-port 3128
```

Cuidado, de las opciones anteriores sólo debemos escoger una de ellas.

Si sólo queremos salida para visitar págias web, entonces pondremos:

Primero garantizamos el tráfico DNS:

```
iptables -A FORWARD -p tcp -m tcp --dport 53 -j ACCEPT
iptables -A FORWARD -p udp -m udp --dport 53 -j ACCEPT
iptables -A FORWARD -p tcp -m tcp --sport 53 -j ACCEPT
iptables -A FORWARD -p udp -m udp --sport 53 -j ACCEPT
```

En el siguiente caso suponemos que el equipo con squid está en 192.168.5.254:

```
iptables -A PREROUTING -i eth1 -p tcp -m tcp --dport 80 -j DNAT --to-destination 192.168.5.254:3128
iptables -t nat -A FORWARD -j REJECT --reject-with icmp-port-unreachable
```

Si el equipo con squid está en la misma pasarela entonces podemos poner:

```
iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 -j REDIRECT --to-port 3128
iptables -A FORWARD -j REJECT --reject-with icmp-port-unreachable
```