

# OpenBSD, seguridad proactiva y firewalls

Francisco de Borja López Río - sistemas@codigo23.net

28 de marzo de 2005

## **Resumen**

Hoy en día, Internet supone un importante impulso para la mayoría de las empresas a nivel mundial. Conectividad en tiempo real, acceso a datos e información no accesible de otra forma, o no tan rápido al menos, publicidad y un sin fin más de ventajas atraen cada día a más y más empresas de casi todos los sectores. Es por esto que cada día es más importante también la seguridad a nivel informático. El flujo de información privada, confidencial a través de internet implica que es necesario utilizar herramientas que nos permitan trabajar mejor, pero manteniendo toda la seguridad necesaria.

Es en este entorno donde el sistema operativo OpenBSD cumple sus funciones posiblemente mejor que cualquier otra alternativa. En este documento pretendo dar una pequeña idea de lo que es OpenBSD, de por qué es considerado como posiblemente el sistema operativo más seguro del mundo del software libre y en especial, por qué es una solución muy aconsejable para el montaje de firewalls.

# Índice

<b>1. ¿Que es OpenBSD?</b>	<b>4</b>
1.1. BSD Unix, Berkeley Software Distribution . . . . .	4
1.2. Distribuciones BSD: NetBSD, FreeBSD y OpenBSD . . . . .	4
1.3. Objetivos del proyecto OpenBSD . . . . .	4
<b>2. OpenBSD, seguro por defecto</b>	<b>6</b>
2.1. ¿Que significa seguridad proactiva? . . . . .	6
2.2. El modelo de seguridad de OpenBSD . . . . .	6
<b>3. Packet filtering con pf</b>	<b>7</b>
3.1. Firewalls de filtrado de paquetes . . . . .	7
3.2. ¿Que es pf? . . . . .	7
3.3. Ejemplo práctico: pymegal . . . . .	7
3.3.1. pymegal, estudio inicial . . . . .	7
3.3.2. Implantación del firewall, instalación y primeros pasos . . . . .	8
3.3.3. Integrando Pf, proteccion hacia dentro. . . . .	9
3.3.4. Ofreciendo servicios seguros a través de internet . . . . .	12
3.3.5. Mejorando la seguridad . . . . .	15
3.3.6. Seguridad total, proteccion desde dentro . . . . .	18
3.3.7. Ampliacion de personal, controlando el trafico generado . . . . .	21
3.3.8. ¿Que está pasando? . . . . .	25
3.4. Y la historia continua... . . . . .	28
<b>4. Bibliografía</b>	<b>28</b>
4.1. Libros . . . . .	28
4.2. Enlaces/URLS . . . . .	29

## 1. ¿Que es OpenBSD?

**OpenBSD** es un sistema de tipo **Unix**, descendiente de la rama de Unix desarrollada por la Universidad de Berkeley en California. Esta rama, conocida como **BSD Unix**, era un fork del Unix original creado en 1969 en los laboratorios Bell, distribuido por la Universidad de Berkeley en California, que introdujo un elevado numero de mejoras al sistema Unix en general, como el nuevo sistema de memoria virtual, la creacion de sockets o la implementación del protocolo TCP/IP

### 1.1. BSD Unix, Berkeley Software Distribution

Quizás la entidad que más aportó a Unix despues de los laboratorios Bell fue la **Universidad de Berkeley en California**. Desde aproximadamente 1977, el **C.S.R.G** (University of California at Berkeley's Computer Systems Research Group) se encargo de crear y desarrollar un fork del Unix original de los laboratorios Bell, al cual se le llamo **BSD** (Berkeley Software Distributions), que fue desarrollado en sucesivas versiones (1BSD, 2BSD, 3BSD y 4BSD) durante los años 70 y 80.

Durante todo este tiempo, las mejoras aportadas por las diferentes versiones de BSD, fueron incorporadas en el unix original, o adaptadas, algunos ejemplos son:

- Nuevo soporte de memoria virtual.
- Sockets.
- Incorporacion de los protocolos de red utilizados por D.A.R.P.A. (TCP/IP).
- Muchas de las llamadas al sistema o syscalls.
- La libreria termcap
- El verificador de codigo C lint

### 1.2. Distribuciones BSD: NetBSD, FreeBSD y OpenBSD

Tras varios años de investigación y desarrollo, el **C.S.R.G** consiguió a comienzos de los 90 publicar una version de su sistema 4.4BSD (**4.4BSD-Lite**) que no requería que el usuario que iba a utilizarlo dispusiese de una licencia de codigo fuente de Unix. De este sistema surgieron dos alternativas dentro del mundo del software libre que evolucionarion en base a él, **NetBSD** y **FreeBSD**, que aparecen sobre los años 1991 y 1992.

Uno de los primeros **committers** (termino designado a los colaboradores que tienen acceso de escritura al repositorio de codigo del proyecto.) del proyecto **NetBSD** junto a **Chris Demetriou** fue **Theo de Raadt**. Ambos comenzaron con el proyecto **NetBSD** con una máquina con CVS montada por **Chris** y trabajando sobre el codigo de lo que sería el primer **NetBSD**. El proyecto comenzó a crecer, varios desarrolladores fueron uniendose al grupo, entre ellos **Charles Hannum** del **M.I.T.** (Massachusetts Institute of Technology) con el que Theo tendría sus más y sus menos más adelante. A raíz de las diferencias con Hannum, Theo se separa definitivamente del proyecto **NetBSD**. A los pocos meses, crea un repositorio de codigo junto a algunos desarrolladores que no estaban de acuerdo con su exclusión del proyecto **NetBSD** y es en ese momento cuando nace el proyecto **OpenBSD**.

### 1.3. Objetivos del proyecto OpenBSD

En la página del proyecto podemos ver una lista de los objetivos globales del proyecto, pero quizás los mas interesantes sean:

- Proveer al usuario con acceso total a los fuentes desarrollados por el proyecto.
- Integrar cualquier tipo de codigo que disponga de una licencia aceptable, la idea es disponer de un codigo que sea accesible para cualquier usuario, tal y como se dice en el punto anterior, pero permitiendo al usuario que haga **lo que quiera y/o/u necesite** con ese codigo.

- Intentar llevar un seguimiento de posibles bugs y fallos y corregirlos antes que nadie, de esta forma se pretende conseguir el sistema operativo más seguro posible.
- Seguir siempre e implementar en la medida de lo posible los estandares (ANSI, POSIX, etc)
- Ser un proyecto totalmente apolitico.
- Sacar una version en CD cada 6 meses de desarrollo, lo que ayuda a financiar el proyecto.

Estos son tan sólo algunos de los objetivos del proyecto, aunque posiblemente el punto que mas se conoce del proyecto es el de conseguir **el sistema más seguro posible**, tratando de ser **el número 1 en cuanto a seguridad** en sistemas operativos.

El enfoque del proyecto en este sentido a sido claro desde las primeras versiones hasta la actual, mejorando release a release e incluyendo mejoras como la generación aleatoria de **PIDs** (Process ID o ID de proceso, que es el numero con el que podemos referenciar a cada uno de los procesos del sistema), la generación aleatoria de los números de secuencia iniciales de las conexiones TCP, la integración de algoritmos de criptografía en el kernel, la separacion de privilegios para la mayoría de daemons del sistema, el uso de systrace o propolice, etc.

Un vistazo a la pagina <http://www.openbsd.org/plus.html> nos da una idea de todas las mejoras que se han aportado a nivel de seguridad en su sistema por los desarrolladores de OpenBSD. Algo a tener en cuenta es que, en la mayoría de los casos, esas mejoras son adoptadas luego no sólo por los demas sistemas derivados del BSD original, si no tambien por otros sistemas como Linux, Solaris, etc.

## 2. OpenBSD, seguro por defecto

Decir que un sistema es seguro por defecto puede sonar arrogante, esto incluso ha hecho que determinadas secciones de la comunidad underground enfocasen sus puntos de mira en sistemas que funcionen bajo OpenBSD para demostrar que sus desarrolladores se equivocan, y que no existe el sistema perfectamente seguro.

Esto último, en mi opinion, es cierto ya que no hay sistema que pueda permanecer eternamente seguro a lo largo del tiempo, al menos no si no se lleva un mantenimiento adecuado del mismo.

Seguro por defecto, en el contexto del proyecto OpenBSD, significa un sistema que out-of-the-box como se suele decir, o recién instalado, sea lo más seguro posible sin necesidad de pasar un tiempo configurandolo en lo que se suele conocer en el lenguaje de shakespeare como *hardening*. Un sistema OpenBSD recién instalado no solo no tiene apenas servicios corriendo por defecto, si no que los que tiene estan configurados de forma que no supongan peligro de ataques externos. Los daemons que corran por defecto en el sistema, así como la mayoría de los que el usuario/administrador encargado del sistema pueda activar/instalar, lo hacen como un usuario sin privilegios, esto unido a que muchos de los servicios estan configurados para trabajar por defecto en entornos **chroot** hace que los problemas en caso de producirse algun fallo se reduzcan drasticamente. Estas dos son tan solo ejemplos de la innumerable cantidad de razones que hacen que los desarrolladores del proyecto OpenBSD puedan anunciar su sistema como **seguro por defecto**

### 2.1. ¿Que significa seguridad proactiva?

Seguridad proactiva, siempre hablando dentro del contexto del proyecto OpenBSD, significa que se intenta buscar fallos, auditar todo el código posible del sistema siempre buscando nuevos fallos, nuevos bugs, explorando con tecnicas que se van descubriendo en el campo de la seguridad informática, tratando de descubrir esos bugs, entiendo como se pueden originar y aplicando esos conocimientos al resto del código. De esta forma se consigue tener un sistema en constante fase de auditoría, aumentando la posibilidad de ser los primeros en encontrar un fallo en cualquier parte del sistema y no solo corrigiendola para OpenBSD, si no ayudando a otros proyectos a corregir esos fallos y mejorar la seguridad de sus sistemas en general.

### 2.2. El modelo de seguridad de OpenBSD

Como ya he comentado antes, el desarrollo de OpenBSD es continuado y se publica una release cada 6 meses. Además del desarrollo en si del proyecto, una parte de los desarrolladores se encargan unicamente de auditar tanto el código que lleva más tiempo formando parte del sistema como el que se va aportando de release en release. Esta auditoria de código es llevada por diferentes desarrolladores a lo largo de la vida del proyecto.

## 3. Packet filtering con pf

### 3.1. Firewalls de filtrado de paquetes

Los firewalls de filtrado de paquetes consisten en un software que, dependiendo de una serie de reglas establecidas, modifican el tráfico de red a través de un determinado dispositivo de red. Este tipo de firewalls son cada día más utilizados por su sencillez de configuración y su eficacia, además de disponer de diferentes tipos para los sistemas operativos basados en software libre, lo que hace que se pueda disponer de firewalls muy seguros y estables a un coste muy bajo.

Algunos ejemplos de software de filtrado de paquetes en el ámbito del software libre son **netfilter** para el kernel de Linux, o **ipf**, **ipfw** o **pf** para sistemas basados en BSD. Es en este último en el que voy a basar el resto del documento.

Este tipo de software de filtrado no solo nos permite denegar el paso de determinado tipo de paquetes desde nuestra red o hacia ella, si no que permiten la redirección, normalización, y logueo de los mismos.

### 3.2. ¿Que es pf?

Es el software de filtrado de paquetes creado y desarrollado por el equipo del proyecto OpenBSD. Desde la versión 3.0 de OpenBSD es la opción utilizada por defecto, en detrimento de ipf, que era el software utilizado hasta entonces.

Algunos de los puntos fuertes de pf son:

- Filtrado de paquetes IPv4 e IPv6
- NAT, Network Address Translation
- Normalización de paquetes
- Sets de reglas dinámicos
- Balanceo de carga
- Modulación de ancho de banda
- Logueo de paquetes
- Autenticación de usuarios contra el firewall

Pf es lo que se conoce como un *stateful packet filter* o, lo que es lo mismo, un software de filtrado de paquetes que controla el estado de las conexiones. Gracias a esto, en lugar de dejar pasar todo el tráfico hacia un determinado puerto, se puede permitir solo el paso del primer paquete, el resto del tráfico podrá pasar igualmente debido a que el firewall ha guardado información referente a esa conexión y sabe que paquetes pertenecen a la misma. Este tipo de configuración en un firewall previene, por ejemplo, determinado tipo de ataques basados en *spoofing*.

### 3.3. Ejemplo práctico: pymegal

#### 3.3.1. pymegal, estudio inicial

Pymegal es una empresa que se dedica a la gestión contable de pequeñas y medianas empresas del ámbito gallego. Ofrece sus servicios a un elevado número de clientes que ponen en sus manos todo el material económico y fiscal de sus respectivas empresas, por lo que en pymegal se hace uso y se almacena información que podríamos considerar delicada.

En esta empresa nos encontramos con una plantilla formada por siete personas, dos gestores que son los dueños de la empresa, tres administrativos que se encargan de hacer la mayor parte del trabajo y dos secretarios que trabajan de cara a los clientes.

En la oficina se trabaja con una red local que sigue una arquitectura cliente-servidor, con un servidor central

donde almacenan toda la información con la que trabajan y un ordenador por empleado. Para mejorar el servicio de cara a sus clientes, disponen de una conexión a internet que les permite estar en contacto electrónico permante con ellos, asi como ofrecer algunos servicios digitales como acceso a documentos en formato electronico.

### 3.3.2. Implantación del firewall, instalación y primeros pasos

Como hemos visto, el tipo de información con la que se trabaja en pymegal se puede clasificar como información muy delicada, por lo que es muy importante que el sistema informático en el que se basa todo su trabajo se encuentre protegido contra posibles ataques provenientes de Internet.

Para ello vamos a colocar entre la red local y su conexion a internet un firewall. Lo primero es instalar OpenBSD, lo cual no veremos aqui por que se sale del tema a cubrir por este documento, lo que si vamos a ver aqui es una serie de información sobre el sistema que nos será útil como referencia a lo largo del documento.

El firewall consistirá en una maquina **Pentium4**, corriendo **OpenBSD 3.6** con un kernel **GENERIC**:

```
OpenBSD nacht.e-shell.org 3.6 GENERIC#0 i386 Intel(R) Pentium(R) 4
CPU 1.70GHz ("GenuineIntel" 686-class)
```

El firewall dispone de dos interfaces o tarjetas de red, una conectada a la red local y otra conectada al enlace con internet, tambien dispone de una interfaz wireless 802.11b que utilizaremos para dar cobertura Inalámbrica dentro de la oficina:

```
# ifconfig -a
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33224
    inet 127.0.0.1 netmask 0xff000000
    inet6 ::1 prefixlen 128
    inet6 fe80::1%lo0 prefixlen 64 scopeid 0x5
fxp0: flags=8802<BROADCAST,SIMPLEX,MULTICAST> mtu 1500
    address: 00:02:3f:76:25:97
    media: Ethernet autoselect (none)
    status: no carrier
pflog0: flags=0<> mtu 33224
pfsync0: flags=0<> mtu 2020
enc0: flags=0<> mtu 1536
wi0: flags=8c43<UP,BROADCAST,RUNNING,OACTIVE,SIMPLEX,MULTICAST> mtu 1500
    address: 00:90:d1:06:53:f6
    ieee80211: nwid codigo23 1dBm (auto)
    media: IEEE802.11 autoselect (DS11)
    status: active
    inet 10.0.0.2 netmask 0xffffffff broadcast 10.0.0.255
    inet6 fe80::290:d1ff:fe06:53f6%wi0 prefixlen 64 scopeid 0x6
ne3: flags=8822<BROADCAST,NOTRAILERS,SIMPLEX,MULTICAST> mtu 1500
    address: 00:30:f1:08:20:5a
    media: Ethernet manual
```

Además de los dispositivos conectados a la intranet (**ne3**) y al enlace con Internet (**fxp0**), podemos ver dos dispositivos utilizados por pf, uno utilizado por **pflogd** para el sistema de logueo de paquetes (**pflog0**) y otro utilizado por **CARP** para la redundancia de firewalls (**pfsync0**). Tambien podemos observar el dispositivo **loopback** (**lo0**) y un dispositivo llamado **enc0** que podremos utilizar para filtrar trafico **IPSec** con pf.

En este caso nos encontramos con que entre el propio firewall y la conexion a internet tenemos un router proveido por la compañía de telecomunicaciones con la que se contrató el servicio, por lo que configuramos el router en modo monopuesto, simplemente enviando todo el tráfico que recibe a una dirección IP que

nosotros queramos. Esa dirección IP sera la dirección de nuestro firewall, al cual el router estará conectado directamente, sin ningun tipo de hub o switch por el medio. En este caso vamos a configurar el router para que envíe todo el tráfico a la dirección **192.168.2.2**, configurandolo para que su dirección dentro de la subred **192.168.2.0/24** sea la **192.168.2.1**.

Esta es una configuración muy habitual, aunque perfectamente podríamos eliminar el router de la configuración y simplemente conectar nuestro firewall en su lugar, configurando la conexión mediante **pppoe**, **pppoea** o el protocolo utilizado por el proveedor.

La red interna de la oficina se situa dentro del rango **192.168.1.0/24**, con las estaciones de trabajo numeradas **de 1 a n** siendo n el número máximo de equipos (siete en este caso). El servidor dispone de la dirección **192.168.1.50** y al firewall le vamos a aplicar la dirección **192.168.1.100**, esta será la dirección que hemos de configurar luego en todos los equipos como **puerta de enlace por defecto**.

### 3.3.3. Integrando Pf, proteccion hacia dentro.

Una vez instalado OpenBSD, tenemos que activar la opcion que permite al kernel del sistema hacer redirección de paquetes (**Packet Forwarding**), lo cual podemos hacer mediante la herramienta **sysctl**:

```
# sysctl -algrep forwarding
net.inet.ip.forwarding=0
net.inet6.ip6.forwarding=0
# sysctl -w net.inet.ip.forwarding=1
net.inet.ip.forwarding: 0 -> 1
#
```

En este caso tan sólo activamos la redirección de paquetes de tipo IPv4, ya que no deseamos tráfico IPv6 hacia la red y la red no va a generar tráfico de este tipo. Para que este cambio en la variable **net.inet.ip.forwarding** se establezca con cada arranque del firewall, tenemos que editar el fichero **/etc/sysctl.conf** y descomentar la siguiente linea:

```
#net.inet.ip.forwarding=1      # 1=Permit forwarding (routing) of packets
```

Lo siguiente es activar el soporte para pf, lo cual hacemos editando el fichero **/etc/rc.conf** y modificando las siguientes lineas:

```
pf=NO                # Packet filter / NAT
pf_rules=/etc/pf.conf # Packet filter rules file
pflogd_flags=        # add more flags, ie. "-s 256"
```

De forma que nos queden:

```
pf=YES              # Packet filter / NAT
pf_rules=/etc/pf.conf # Packet filter rules file
pflogd_flags=        # add more flags, ie. "-s 256"
```

Básicamente la linea que tenemos que cambiar es la primera, que activa el soporte para pf en el arranque. Las otras dos opciones especifican el fichero que se encargara de almacenar la configuración del firewall (**pf\_rules**) y parametros extra para el daemon de logueo (**pflogd\_flags**).

Tambien podemos activar pf directamente sin tener que reiniciar para que los cambios en **/etc/rc.conf** tengan efecto, en ese caso no sólo tenemos que activar pf, si no que deberíamos lanzar **pflogd** y levantar el dispositivo **pflog0** manualmente:

```
$ sudo pfctl -e
pf enabled
$ sudo ifconfig pflog0 up
$ sudo pflogd
$
```

Una vez habilitado pf, procedemos con la configuracion del firewall, lo primero es dar salida hacia Internet a los equipos de la oficina de pymegal, pero protegiendo la red de los datos que entran hacia ella, para ello vamos a usar un fichero `/etc/pf.conf` como el siguiente:

```
#
# Proveeido y configurado porCodigo23
# http://www.codigo23.net - sistemas@codigo23.net
# Marzo 2005

# Definicion de macros
#
# Interfaces
ext_if="fxp0"           # Interfaz conectada a internet
int_if="ne3"           # Interfaz conectada a la red local
loop="lo0"            # Interfaz loopback

# Redes
ext_net="192.168.2.0/24" # Subred conectada a la interfaz externa
int_net="192.168.1.0/24" # Subred conectada a la interfaz interna

# Hosts
ext_firewall="192.168.2.2" # Dirección ip de la interfaz externa
int_firewall="192.168.1.100" # Dirección ip de la interfaz interna
int_server="192.168.1.50" # Dirección ip del servidor

# Opciones
set optimization normal # Tiempo medio de ruptura de conexiones
set block-policy return # Las peticiones a puertos bloqueados son devueltas
set skip on $loop       # No tratar el dispositivo loopback con pf

# Normalizacion de trafico
scrub in on $ext_if all # Los paquetes fragmentados son reensamblados antes
                        # de continuar

# NAT
#
# Mapeamos la red interna hacia el exterior a traves de la
# interfaz de red externa.
nat on $ext_if from $int_net to any -> ($ext_if)

# Filtrado
#
# Bloqueamos todo por defecto
block in all

# Interfaz conectada a la red interna,
# aceptamos todo el trafico de entrada/salida
pass in quick on $int_if all
pass out quick on $int_if all
```

```
# Interfaz conectada al enlace con Internet, solo aceptamos
# tráfico icmp de entrada y tcp/udp de respuesta a los paquetes
# de salida.
pass in quick on $ext_if proto icmp all keep state
pass out quick on $ext_if from $int_net to any keep state
```

La estructura del fichero `/etc/pf.conf` es muy sencilla, toda línea que comienza por `#` es un comentario, por lo que no será parseada e interpretada por `pf`. Luego tenemos una serie de secciones que siguen un determinado orden. En la primera configuración de nuestro firewall no vamos a utilizar todas las secciones, sino solamente las siguientes:

1. **Macros:** Son variables definidas por el usuario, en este punto es interesante establecer, por ejemplo, variables identificando las interfaces de red sobre las que estableceremos el filtrado, subredes, direcciones IP de hosts, etc.  
Para definir una variable, hacemos uso de la fórmula `variable="valor"`.  
En nuestro fichero de configuración definimos tres bloques principales de variables. En el primero definimos cuáles van a ser las interfaces de red que vamos a utilizar, en el segundo definimos las subredes a las que estará conectado el firewall y en el tercero algunas direcciones IP que puedan ser de utilidad a la hora de crear las reglas de filtrado.  
Haciendo uso de estas variables conseguimos que, en un hipotético caso de ejemplo, si tuviésemos que cambiar la interfaz de red `fxp0` por otra con chipset diferente (`rl0`), no tendríamos que reeditar todas las reglas del firewall para cambiar ese dato, si no simplemente modificarlo en la definición de la variable.
2. **Opciones** Mediante estas opciones podemos modificar determinadas formas de actuación del firewall. Aquí podremos definir cosas como el tiempo de expiración de las conexiones, tiempos de espera o la política a seguir con los paquetes bloqueados (`drop/return`).  
En nuestro caso utilizamos tres opciones de configuración. En la primera establecemos que la política de actuación del firewall sea **normal**, lo que es más que aconsejable para la mayoría de las configuraciones. Dos valores más a tener en cuenta para este parámetro son **aggressive** y **conservative**, el primero corta conexiones no activas más rápidamente mientras que el segundo hace todo lo contrario.  
La segunda opción de configuración utilizada en nuestro firewall es **block-policy**, cuyos valores pueden ser **return** o **drop**. El primero enviará un mensaje de respuesta en caso de que se bloquee un paquete `tcp` o `udp`, mientras que el segundo simplemente descartará ese paquete.  
La tercera de estas opciones, **skip on**, la utilizamos para descartar interfaces de red cuyo tráfico no ha de ser procesado por `pf`.
3. **Normalización de tráfico** En este punto podemos establecer reglas que obliguen a que los paquetes no puedan estar fragmentados, es muy útil para evitar que entren en la red local determinado tipo de ataques (sobre todo de tipo scanner).  
En nuestro caso hacemos uso de la configuración por defecto, de forma que todos los paquetes que lleguen al firewall, fragmentados, son puestos en una cola donde se van reensamblando hasta que forman el paquete original y completo.
4. **NAT, redirecciones:** En esta sección se establecen las reglas que indican cómo mapear/traducir las peticiones salientes de la red local hacia internet o viceversa, así como establecer redirecciones de paquetes dependiendo de su procedencia/destino.  
En esta primera configuración, simplemente agregamos una regla que nos mapee todo el tráfico de la red local destinado a internet como si fuese tráfico generado por el firewall. Esto nos permite disponer de una conexión común a internet para toda una red local. Esto se conoce también como **masquerading** en Linux.
5. **Filtrado de paquetes:** En este punto es donde establecemos las reglas de filtrado. `Pf` evalúa estas reglas de arriba hacia abajo, es decir, la última regla coincidente es la aplicada. Cuando un paquete llega a este punto, va pasando por todas y cada una de las reglas que tenga el firewall definido, hasta la última regla aplicable a ese paquete, que será la utilizada.

Esto, no obstante, puede ser modificado, utilizando una palabra clave o keyword, **quick**, que indica que si una regla es aplicable a un paquete en concreto y lleva esa palabra clave, automáticamente se aplica esa regla y no se sigue recorriendo el listado de reglas.

En nuestro caso adoptamos una política de **bloqueo por defecto**, utilizando luego algunas reglas para permitir determinado tipo de tráfico. En este caso permitimos pasar todo el tráfico generado desde/para la subred local. En cuanto al tráfico en la interfaz de red externa, bloqueamos todo el tráfico que llega si se trata de paquetes **tcp** o **udp**, pero permitimos la entrada de tráfico **icmp** (Internet Control Message Protocol).

Además, permitimos el paso de todo el tráfico saliente de la red hacia internet. En el caso de las dos últimas reglas utilizamos la opción **keep state**, que almacena el estado de la conexión al pasar por el firewall, de esta forma aunque tengamos bloqueado todo el tráfico proveniente de Internet, el tráfico que llegue como respuesta a una petición originada en el firewall es aceptado.

Llegado este punto, utilizamos **pfctl** para recargar las reglas, limpiando las reglas cargadas en memoria y las posibles conexiones preestablecidas:

```
$ sudo pfctl -F all
rules cleared
nat cleared
0 tables deleted.
altq cleared
0 states cleared
source tracking entries cleared
pf: statistics cleared
pf: interface flags reset
$
```

Y disponemos de un firewall, simple y eficaz, que nos da salida a internet a todos los equipos de la oficina de pymegal, e impide a cualquier intruso conectarse a ningún tipo de servicio desde Internet.

### 3.3.4. Ofreciendo servicios seguros a través de internet

Una de las necesidades de pymegal es ofrecer determinados servicios a sus clientes a través de Internet. Entre estos servicios encontramos la consulta de datos online y la descarga de documentos. Además de estos servicios orientados a los clientes, dispondrán de correo electrónico centralizado en la oficina, vamos a ver como adaptar la configuración de nuestro firewall para permitir el acceso de los usuarios a estos servicios.

Para ofrecer estos servicios, se añade un nuevo equipo a la red local, con dirección IP **192.168.1.51**, que se utilizara como pequeño servidor de servicios externos. En este equipo tendremos un servidor web (**http/https**), un servidor **ftp** y un servidor de correo(**smtp/imap/pop3s**).

En un entorno ideal, se crearia una **DMZ** o **Zona demilitarizada** donde el servidor externo estaría aislado, de forma que se minimizasen los daños en caso de una intrusión, para este simple ejemplo vamos a conectar el servidor directamente a la red con los demás equipos y el servidor interno.

El nuevo fichero de configuración, después de añadir lo necesario, queda de la siguiente forma:

```
#
# pymegal firewall
#
# Proveeido y configurado porCodigo23
# http://www.codigo23.net - sistemas@codigo23.net
# Marzo 2005

# Definicion de macros
#
# Interfaces
ext_if="fxp0"                # Interfaz conectada a internet
```

```

int_if="ne3"                # Interfaz conectada a la red local
loop="lo0"                 # Interfaz loopback

# Redes
ext_net="192.168.2.0/24"   # Subred conectada a la interfaz externa
int_net="192.168.1.0/24"  # Subred conectada a la interfaz interna

# Hosts
ext_firewall="192.168.2.2" # Direccion ip de la interfaz externa
int_firewall="192.168.1.100" # Direccion ip de la interfaz interna
int_server="192.168.1.50"  # Direccion ip del servidor interno
ext_server="192.168.1.51"  # Direccion ip del servidor externo

# Opciones
set optimization normal    # Tiempo medio de ruptura de conexiones
set block-policy return    # Las peticiones a puertos bloqueados son devueltas
set skip on $loop         # No tratar el dispositivo loopback con pf

# Normalizacion de trafico
scrub in on $ext_if all    # Los paquetes fragmentados son reensamblados antes
                           # de continuar

# NAT
#
# Mapeamos la red interna hacia el exterior a traves de la
# interfaz de red externa.
nat on $ext_if from $int_net to any -> ($ext_if)

# redirecciones, redireccionamos todos los servicios a la maquina definida
# como ext_server
rdr on $ext_if proto tcp from any to $ext_firewall port 80 -> $ext_server port 80
rdr on $ext_if proto tcp from any to $ext_firewall port 443 -> $ext_server port 443
rdr on $ext_if proto tcp from any to $ext_firewall port 25 -> $ext_server port 25
rdr on $ext_if proto tcp from any to $ext_firewall port 995 -> $ext_server port 995
rdr on $ext_if proto tcp from any to $ext_firewall port 993 -> $ext_server port 993

# Filtrado
#
# Bloqueamos todo por defecto
block in all

# Interfaz conectada a la red interna,
# aceptamos todo el trafico de entrada/salida
pass in quick on $int_if all
pass out quick on $int_if all

# Interfaz conectada al enlace con Internet, solo aceptamos
# tráfico icmp de entrada y tcp/udp de respuesta a los paquetes
# de salida.
pass in quick on $ext_if proto icmp all keep state

```

```

pass in quick on $ext_if proto tcp from any to $ext_server port { 80, 443, 25, 995, 993 } keep state
pass in quick on $ext_if proto tcp from any to $ext_firewall port 21 keep state
pass out quick on $ext_if from $int_net to any keep state

```

Los cambios más importantes se encuentran en las secciones de **NAT y redirecciones** y **filtrado de paquetes**, aparte de agregar el nuevo servidor externo en la sección de **macros**. En la sección de NAT utilizamos reglas **rdr** para establecer redirecciones de puertos, de forma que todo el tráfico que el firewall reciba con destino a los puertos de los servicios que se ofrecen, se redirige al equipo en el que están instalados y configurados esos servicios, en este caso el definido por la variable **ext\_server**. Hay que tener en cuenta que tras pasar por las reglas que establecen las redirecciones, los paquetes son enviados a la sección de filtrado, aunque esto puede obviarse utilizando la palabra clave **pass** directamente en las reglas **rdr**:

```
rdr pass on $ext_if proto tcp from any to $ext_firewall port 80 -> $ext_server port 80
```

En la sección de **filtrado de paquetes** hemos de establecer las reglas necesarias para que el firewall permita que el tráfico fluya del exterior al servidor. Esto puede hacerse en una sola regla, o en varias reglas, una por servicio. En este caso hacemos uso de una única regla para todos los servicios redireccionados hacia el servidor. Otra opción sería establecer previamente, en la sección de **macros**, una variable con los servicios cuyos puertos han de ser abiertos, de forma que luego la regla nos quedase más o menos así:

```

openports="{ 80, 443, 25, 995, 993 }"
...
pass in quick on $ext_if proto tcp from any to $ext_server port $openports keep state

```

En caso de utilizar una macro para todos los puertos que vamos a necesitar, podemos utilizar dicha macro no sólo en la sección de packet filtering, si no también en las redirecciones:

```
rdr on $ext_if proto tcp from any to $ext_firewall port $openports -> $ext_server
```

Algo a tener en cuenta en esta configuración, es que no hemos establecido redirección para el servicio **ftp**, y que la regla de filtrado que permite el paso de tráfico hacia el puerto 21 no permite el paso de tráfico hacia el servidor, si no hacia el propio firewall. Esto es por que vamos a utilizar un **proxy** para las conexiones ftp. El motivo para tomar esta decisión viene dada por el largo historial de problemas del protocolo de transferencia de ficheros y los firewalls de filtrado, sobre todo cuando se trata de transferencias en **modo activo**. Este proxy funciona de forma totalmente transparente, por lo que lo único que tenemos que hacer es configurarlo para que escuche peticiones en el puerto 21 del firewall y redirija las peticiones al servidor ftp real. Para lanzar el servicio de proxy ftp (llamado **ftp-proxy**), no tenemos más que añadir la siguiente línea al fichero de configuración de **inetd**, **/etc/inetd.conf**:

```
192.168.2.2:21 stream tcp nowait root /usr/libexec/ftp-proxy ftp-proxy -R 192.168.1.51:21
```

Llegado este punto, utilizamos **pfctl** para recargar las reglas, limpiando las reglas cargadas en memoria y las posibles conexiones preestablecidas:

```

$ sudo pfctl -F all

rules cleared
nat cleared
0 tables deleted.
altq cleared
0 states cleared
source tracking entries cleared
pf: statistics cleared
pf: interface flags reset
$

```

Y ya disponemos de un firewall que además de compartir la conexión a internet entre los diferentes equipos de la red interna, y protegerlos de accesos desde Internet, también se encarga de redirigir el tráfico que llega para los diferentes servicios que pymegal ofrece a sus clientes de forma remota.

### 3.3.5. Mejorando la seguridad

Vamos a ver ahora como securizar un poco más las conexiones que pasan a través de nuestro firewall, cubriendo una serie de puntos a aplicar a las reglas utilizadas hasta el momento. El nuevo fichero de configuración queda de la siguiente forma:

```
#
# pymegal firewall
#
# Proveeido y configurado porCodigo23
# http://www.codigo23.net - sistemas@codigo23.net
# Marzo 2005

# Definicion de macros
#
# Interfaces
ext_if="fxp0"           # Interfaz conectada a internet
int_if="ne3"           # Interfaz conectada a la red local
loop="lo0"            # Interfaz loopback

# Redes
ext_net="192.168.2.0/24" # Subred conectada a la interfaz externa
int_net="192.168.1.0/24" # Subred conectada a la interfaz interna

# Hosts
ext_firewall="192.168.2.2" # Direccion ip de la interfaz externa
int_firewall="192.168.1.100" # Direccion ip de la interfaz interna
int_server="192.168.1.50" # Direccion ip del servidor interno
ext_server="192.168.1.51" # Direccion ip del servidor externo

# Services
openports="{ 80, 443, 25, 995, 993 }"

# Opciones
set optimization normal # Tiempo medio de ruptura de conexiones
set block-policy return # Las peticiones a puertos bloqueados son devueltas
set skip on $loop       # No tratar el dispositivo loopback con pf

# Normalizacion de trafico
scrub in on $ext_if all # Los paquetes fragmentados son reensamblados antes
                        # de continuar

# NAT
#
# Mapeamos la red interna hacia el exterior a traves de la
# interfaz de red externa.
nat on $ext_if from $int_net to any -> ($ext_if)

# redirecciones, redireccionamos todos los servicios a la maquina definida
# como ext_server
rdr on $ext_if proto tcp from any to $ext_firewall port $openports -> $ext_server
```

```

# Filtrado
#
# Bloqueamos todo por defecto
block in all

# Antispoof, para las interfaces externa e interna
antispoof quick for { $int_if, $ext_if }

# Interfaz conectada a la red interna,
# aceptamos todo el trafico de entrada/salida
pass in quick on $int_if all
pass out quick on $int_if all

# Interfaz conectada al enlace con Internet, solo aceptamos
# tráfico icmp de entrada y tcp/udp de respuesta a los paquetes
# de salida.
pass in quick on $ext_if proto icmp all keep state
pass in quick on $ext_if proto tcp from any to $ext_server port $openports flags S/SA synproxy state
pass in quick on $ext_if proto tcp from any to $ext_firewall port 21 flags S/SA keep state
pass out quick on $ext_if from $int_net to any modulate state

```

Comenzaremos controlando los **flags** de las conexiones **TCP**. En las cabeceras de cada paquete TCP podemos encontrar una serie de marcas o flags que indican, entre otras cosas, a que momento de la conexión pertenece cada paquete (inicio, transmisión o finalización). Cada flag tiene un significado y un mismo paquete TCP puede llevar varios activados, a continuación vemos un listado de los flags con su representación en pf y una descripción:

- **FIN** (F) Indica el fin de una conexión
- **SYN** (S) Implica una petición de comienzo de conexión
- **RST** (R) Cierra una conexión abruptamente
- **PUSH** (P) Indica que el paquete es enviado inmediatamente
- **ACK** (A) Indica el reconocimiento de la recepción de un paquete
- **URG** (U) Implica urgencia
- **ECE** (E) Indica congestión en la transmisión de datos
- **CWR** (W) Indica una reducción de la congestión en la transmisión.

Estos dos últimos flags han sido incorporados en el 2001 para disponer de un control de congestiones en la transmisión de tráfico TCP.

En las reglas de filtrado de nuestro firewall podemos incluir una opción más, que filtre esas marcas o flags según nuestras necesidades. Esto suele aplicarse sobre todo a paquetes de inicio de conexión, de forma que controlemos que el primer paquete que recibimos no tenga activados flags que no sean necesarios y si, por contra, puedan suponer algún tipo de problema. Determinado tipo de **ataques de denegación de servicio** (DOS o Denial Of Service) hacen uso de una combinación de flags incorrecta en el mecanismo de establecimiento de conexión para conexiones tcp. Otro tipo de ataques muy comun es el uso de estos flags para intentar sobrepasar las reglas de determinados firewalls.

En las reglas de filtrado de pf controlamos los flags de cada paquete con la opción **flags x/y**, donde **y** es una lista de flags a tener en cuenta y **x** una lista de flags que han de encontrarse activas en el paquete para que la regla sea aplicable. Todos los flags que vayan en la cabecera del paquete, que no se encuentren

en la lista **y** serán ignorados a efectos del firewall. Aquellos flags que se encuentren en **y**, pero no en **x**, han de estar desactivados en la cabecera del paquete para que la regla sea aplicable.

En nuestro caso, aplicamos los siguientes cambios a las reglas de filtrado:

```
pass in quick on $ext_if proto tcp from any to $ext_server port $openports flags S/SA keep state
pass in quick on $ext_if proto tcp from any to $ext_firewall port 21 flags S/SA keep state
```

Con lo cual establecemos que solo pasarán aquellos paquetes que tengan las combinaciones **SYN**, **SYN+PUSH** y **SYN+RST** en la cabecera del paquete. Esto, combinado con el uso de **keep state** garantiza que la conexión se establecerá correctamente y que, una vez establecida, toda la transmisión se llevará a cabo correctamente.

Hemos visto que el uso de la opción **keep state** es muy útil para llevar un control de qué paquetes pertenecen a cada conexión y permitir el paso o no de esos paquetes dependiendo de si son respuestas a una conexión preestablecida o no. Pf dispone de dos tipos más de seguimiento de conexiones, **modulate state** y **synproxy state**. El uso de ambas opciones es idéntico al de **keep state** y el uso de **modulate state** implica **keep state** y el de **synproxy state** implica **modulate state** y **keep state**.

La opción **modulate state** hace que cada respuesta a un paquete procesado por la regla de filtrado que incluya esta opción, contenga un **ISN** (Initial sequence number o número inicial de secuencia) generado aleatoriamente. Esto es una medida excelente contra determinados ataques que se basan en la predictibilidad con la que determinados stacks tcp/ip generan dichos números de secuencia.

La opción **synproxy state** añade una funcionalidad más a nuestro firewall. En lugar de redirigir simplemente paquetes entre el origen y el destino, un cliente accediendo al servicio de publicación online y nuestro servidor web por ejemplo, pf se encargará del establecimiento de conexión con el origen. Si esta conexión se efectúa correctamente, entonces el resto de los paquetes de esa conexión son enviados a su destino. Lo que hace pf exactamente es manejar la conexión TCP inicial, de forma que el firewall recibe el primer paquete con el flag **SYN** activado, responde al origen de la conexión con un paquete de respuesta (**SYN+ACK**) y espera un tercer paquete del origen (**ACK** antes de pasar ningún paquete al destino, el servidor web en nuestro caso).

La utilización de estas dos opciones está sujeta al tipo de red en la que se utilice el firewall. Por ejemplo, **synproxy state** es recomendable en casos en los que dispongamos de servicios accesibles desde fuera de la red, sabiendo que el software que ofrece esos servicios puede ser vulnerable a ataques de tipo **SYN flood**. El uso de **modulate state** está igualmente sujeto a que, los equipos situados en la red interna dispongan de algún sistema operativo cuyo stack tcp/ip envíe paquetes cuyos ISNs sean fácilmente predecibles.

En nuestro caso vamos a aplicar **synproxy state** para las conexiones recibidas para los servicios que pymegal ofrece a través de Internet y **modulate state** para las emitidas por los equipos de la red local.

```
pass in quick on $ext_if proto tcp from any to $ext_server port $openports flags S/SA synproxy state
pass in quick on $ext_if proto tcp from any to $ext_firewall port 21 flags S/SA keep state
pass out quick on $ext_if from $int_net to any modulate state
```

En el caso del servicio ftp no utilizamos **synproxy** ya que será el propio proxy ftp el encargado de manejar las conexiones en lugar del servidor real.

Ya hemos visto como controlar los flags de las cabeceras de los paquetes TCP que llegan al firewall, como controlar el establecimiento de conexiones entre un origen fuera de la red y un destino dentro y viceversa, así como la opción para *securizar* los números de secuencia iniciales de paquetes TCP que son enviados a través del firewall. Vamos a ver ahora como agregar una regla a nuestro firewall que evite posibles ataques de tipo **spoofing** de una forma sencilla.

Un ataque de estas características se produce cuando un atacante es capaz de *secuestrar* o *robar* una sesión entre dos máquinas. El atacante ha de ser capaz de monitorizar una conexión establecida entre dos puntos, averiguar los números de secuencia de los paquetes enviados entre ambos puntos. Es entonces cuando el atacante se hace pasar por uno de los dos puntos implicados en la conexión, asignándose su misma dirección IP e inyectando en la conexión *secuestrada* sus propios paquetes sin que el punto con el que ha establecido la conexión sea capaz de detectar el cambio.

Ante este tipo de ataques un buen remedio es el uso de la opción **modulate state** que, como hemos visto antes, hace que los números de secuencia sean prácticamente imposibles de averiguar, pero pf nos ofrece una regla especial, **antispoof**, que nos permite establecer de una forma rápida unas reglas que impiden que recibamos tráfico de direcciones IP que no debieramos recibir. En nuestro caso esto se traduce en:

```
# Antispoof, para las interfaces externa e interna
antispoof quick for { $int_if, $ext_if }
```

Esa única regla es interpretada por Pf como si aplicásemos estas cuatro:

```
block drop in on ! $int_if inet from 192.168.1.0/24 to any
block drop in from 192.168.1.100 to any
block drop in on ! $ext_if inet from 192.168.2.0/24 to any
block drop in from 192.168.2.2 to any
```

De forma que cualquier paquete recibido de una dirección IP que supuestamente tendría que venir de la red interna, desde Internet o viceversa será automáticamente descartado. Además, se bloquea cualquier paquete proveniente de las direcciones IP del propio firewall desde fuera del propio equipo.

Llegado este punto, utilizamos de nuevo **pfctl** para recargar las reglas:

```
$ sudo pfctl -F all

rules cleared
nat cleared
0 tables deleted.
altq cleared
0 states cleared
source tracking entries cleared
pf: statistics cleared
pf: interface flags reset
$
```

### 3.3.6. Seguridad total, protección desde dentro

En la mayoría de los casos, los firewalls suelen configurarse para proteger una red de posibles ataques desde el exterior, dejando los accesos desde la propia red hacia Internet u otras redes totalmente abiertos. Este era nuestro caso hasta este momento. No solamente es importante proteger la red de posibles ataques externos, si no que es igualmente importante controlar que tipo de tráfico se genera desde nuestros propios equipos. Esto puede servirnos para detectar anomalías, como un posible atacante que haya ganado acceso a alguno de nuestros equipos, el servidor externo por ejemplo. También podemos encontrarnos en el caso de que por normas de trabajo, los empleados no deban de tener acceso a determinado tipo de protocolos (ed2k, kaza, msn messenger, irc, etc).

Imaginémonos el peor de los casos, donde un posible atacante haya conseguido entrar en nuestro servidor de correo y tenga un acceso de shell desde el cual pueda lanzar ataques contra otras máquinas en Internet. Si nuestro firewall está correctamente configurado podremos minimizar el problema, evitando que el atacante utilice nuestras máquinas como base para lanzar más ataques.

La política a seguir para esta configuración es la misma que en el caso de los paquetes provenientes de Internet, bloqueamos todos los paquetes por defecto y tan sólo permitimos el paso de paquetes dirigidos

a determinado tipo de puertos y servicios, con lo que el fichero de configuración nos queda de la siguiente forma:

```
#
# pymegal firewall
#
# Proveeido y configurado porCodigo23
# http://www.codigo23.net - sistemas@codigo23.net
# Marzo 2005

# Definicion de macros
#
# Interfaces
ext_if="fxp0"           # Interfaz conectada a internet
int_if="ne3"           # Interfaz conectada a la red local
loop="lo0"            # Interfaz loopback

# Redes
ext_net="192.168.2.0/24" # Subred conectada a la interfaz externa
int_net="192.168.1.0/24" # Subred conectada a la interfaz interna

# Hosts
ext_firewall="192.168.2.2" # Direccion ip de la interfaz externa
int_firewall="192.168.1.100" # Direccion ip de la interfaz interna
int_server="192.168.1.50" # Direccion ip del servidor interno
ext_server="192.168.1.51" # Direccion ip del servidor externo

# Services
# Servicios que ofrecemos de cara a Internet
openports="{ 80, 443, 25, 995, 993 }"
# Servicios que permitimos que usen nuestros usuarios
avaliableports="{ 22, 80, 443, 25, 995, 993 }"

# Tablas
# Usuarios Privilegiados
table <privips> persist

# Opciones
set optimization normal # Tiempo medio de ruptura de conexiones
set block-policy return # Las peticiones a puertos bloqueados son devueltas
set skip on $loop # No tratar el dispositivo loopback con pf

# Normalizacion de trafico
scrub in on $ext_if all # Los paquetes fragmentados son reensamblados antes
# de continuar

# NAT
#
# Mapeamos la red interna hacia el exterior a traves de la
# interfaz de red externa.
```

```

nat on $ext_if from $int_net to any -> ($ext_if)

# redirecciones, redireccionamos todos los servicios a la maquina definida
# como ext_server
rdr on $ext_if proto tcp from any to $ext_firewall port $openports -> $ext_server

# Filtrado
#
# Bloqueamos todo por defecto
block in all

# Antispoof, para las interfaces externa e interna
antispoof quick for { $int_if, $ext_if }

# Interfaz conectada a la red interna,
# aceptamos todo el trafico de entrada/salida
pass in quick on $int_if all
pass out quick on $int_if all

# Interfaz conectada al enlace con Internet, solo aceptamos
# tráfico icmp de entrada y tcp/udp de respuesta a los paquetes
# de salida.
pass in quick on $ext_if proto icmp all keep state
pass in quick on $ext_if proto tcp from any to $ext_server port $openports flags S/SA synproxy state
pass in quick on $ext_if proto tcp from any to $ext_firewall port 21 flags S/SA keep state
pass out quick on $ext_if from <privips> to any modulate state
pass out quick on $ext_if from $int_net to any ports $availableports modulate state

```

Hemos definido una nueva macro **availableports** que tiene una lista de los puertos a los que permitiremos establecer conexiones desde la red local. Los más habituales se encuentran en nuestro fichero de configuración, permitiendo a los usuarios navegar por internet (tanto http como lugares cifrados con ssl, https), enviar correo, consultarlo por pop e imap, etc.

Además, en este caso utilizamos una option de pf que no habíamos visto antes, **las tablas**. Las tablas son similares a las macros que hemos estado utilizando hasta ahora, pero incorporan una ventaja muy interesante, la posibilidad de modificar su contenido dinámicamente mientras el firewall esta funcionando, sin tener que recargar las reglas del mismo. Otra diferencia notable es que estas tablas se utilizan solamente con direcciones IP o direcciones de subredes.

En nuestro caso utilizamos una tabla llamada **privips**, que contendrá una lista de direcciones IP de equipos privilegiados que seguirán disponiendo de un acceso total a internet, en lugar de los puertos y servicios especificados en **availableports**.

```

# Tablas
# Usuarios Privilegiados
table <privips> persist
...
pass out quick on $ext_if from <privips> to any modulate state
pass out quick on $ext_if from $int_net to any ports $availableports modulate state

```

En el momento de definir una tabla podemos definirla con dos opciones **persist**, que es la que utilizamos en nuestro caso y hace que el kernel mantenga la tabla en memoria incluso si esta vacía, y **const**, que implica que la tabla no podrá ser modificada en tiempo real. Aparte de crear tablas en el propio fichero de configuración, estas pueden ser creadas directamente con la herramienta **pfctl**, la cual podemos usar para manejar las tablas. Por ejemplo, para ver un listado de direcciones almacenadas en una tabla:

```
$ sudo pfctl -t privips -T show
  192.168.1.1
  192.168.1.2
$
```

Para eliminar una dirección de la tabla:

```
$ sudo pfctl -t privips -T delete 192.168.1.2
1/1 addresses deleted.
$ sudo pfctl -t privips -T show
  192.168.1.1
$
```

Y para agregar una dirección a la tabla:

```
$ sudo pfctl -t privips -T add 192.168.1.2
1/1 addresses added.
$ sudo pfctl -t privips -T show
  192.168.1.1
  192.168.1.2
$
```

En este caso hacemos uso de una tabla para controlar dinámicamente direcciones IP de equipos a los que se les permite conectarse a cualquier puerto/servicio a través del firewall, lo cual puede ser útil para habilitar un acceso puntual para un determinado equipo en un determinado momento (videoconferencia, transferencia de ficheros a través de rsync, etc). Otras utilidades interesantes para las tablas en un firewall con pf es el uso de las mismas como blacklists donde almacenar direcciones IP que nos interese bloquear directamente sin llegar a pasar por el resto de reglas del firewall.

Llegado este punto, utilizamos de nuevo **pfctl** para recargar las reglas:

```
$ sudo pfctl -F all

rules cleared
nat cleared
0 tables deleted.
altq cleared
0 states cleared
source tracking entries cleared
pf: statistics cleared
pf: interface flags reset
$
```

### 3.3.7. Ampliación de personal, controlando el tráfico generado

Hasta este punto disponemos de un firewall funcional, que no sólo ofrece un servicio de compartición transparente de una conexión a Internet para varios equipos en red, si no que les protege de posibles ataques provenientes de Internet y controla la redirección de tráfico dependiendo de determinados servicios, orígenes y destinos.

Vamos a ver ahora una faceta de pf que se sale un poco de la protección de las conexiones, pero que se complementa a la perfección con ella, el control de ancho de banda. Imaginémoslo el caso de pymegal donde, como ya hemos dicho, trabajan 7 personas. De esas 7 personas tenemos 2 directivos, 3 administrativos y 2 secretarios, todos ellos compartiendo la misma conexión a Internet de 1Mbps/512Kbps (download/upload). Como ya hemos visto también, los que se encargan de la mayor parte del trabajo son los administrativos, mientras que los secretarios están más para trabajos de cara al cliente (atención telefónica, soporte)

y los dos directivos se encargan de reuniones, decisiones importantes, firmar documentos, etc. Ante esto, parece lógico pensar que quien más necesitará el ancho de banda de la conexión serán los 3 administrativos.

En esta situación, nuestro firewall puede configurarse de modo que los equipos desde los que trabajan esos administrativos tengan garantizado un porcentaje del ancho de banda de la conexión, e incluso que tengan preferencia sobre los demás equipos.

En este caso vamos a asumir lo siguiente (teniendo en cuenta que la subred local es 192.168.1.0:

- 192.168.1.1 y 2 son las direcciones IP de los equipos de los directivos
- 192.168.1.3, 4 y 5 son las direcciones IP de los equipos de los administrativos
- 192.168.1.6 y 7 son las direcciones IP de los equipos de los secretarios
- Vamos a establecer una configuración mediante la cual cada *departamento* tendrá un ancho de banda máximo asignado.

Teniendo lo anterior en cuenta, el fichero de configuración de nuestro firewall se ve ampliado:

```
#
# pymegal firewall
#
# Proveeido y configurado porCodigo23
# http://www.codigo23.net - sistemas@codigo23.net
# Marzo 2005

# Definicion de macros
#
# Interfaces
ext_if="fxp0"           # Interfaz conectada a internet
int_if="ne3"           # Interfaz conectada a la red local
loop="lo0"            # Interfaz loopback

# Redes
ext_net="192.168.2.0/24" # Subred conectada a la interfaz externa
int_net="192.168.1.0/24" # Subred conectada a la interfaz interna

# Hosts
ext_firewall="192.168.2.2" # Direccion ip de la interfaz externa
int_firewall="192.168.1.100" # Direccion ip de la interfaz interna
int_server="192.168.1.50" # Direccion ip del servidor interno
ext_server="192.168.1.51" # Direccion ip del servidor externo

# Services
# Servicios que ofrecemos de cara a Internet
openports="{ 80, 443, 25, 995, 993 }"
# Servicios que permitimos que usen nuestros usuarios
avaliableports="{ 22, 80, 443, 25, 995, 993 }"

# Tablas
# Usuarios Privilegiados
table <privips> persist
table <direc> const { 192.168.1.1, 192.168.1.2 }
table <admin> const { 192.168.1.3, 192.168.1.4, 192.168.1.5 }
table <secre> const { 192.168.1.5, 192.168.1.6 }
```

```

# Opciones
set optimization normal          # Tiempo medio de ruptura de conexiones
set block-policy return          # Las peticiones a puertos bloqueados son devueltas
set skip on $loop                # No tratar el dispositivo loopback con pf

# Normalizacion de trafico
scrub in on $ext_if all          # Los paquetes fragmentados son reensamblados antes
                                # de continuar

# Control de ancho de banda.
altq on $ext_if cbq bandwidth 1Mb queue { direc, admin, secre }
queue direc bandwidth 40% priority 5 cbq (default)
queue admin bandwidth 50% priority 6 cbq (red)
queue secre bandwidth 10% priority 4 cbq (red)

# NAT
#
# Mapeamos la red interna hacia el exterior a traves de la
# interfaz de red externa.
nat on $ext_if from $int_net to any -> ($ext_if)

# redirecciones, redireccionamos todos los servicios a la maquina definida
# como ext_server
rdr on $ext_if proto tcp from any to $ext_firewall port $openports -> $ext_server

# Filtrado
#
# Bloqueamos todo por defecto
block in all

# Antispoof, para las interfaces externa e interna
antispoof quick for { $int_if, $ext_if }

# Interfaz conectada a la red interna,
# aceptamos todo el trafico de entrada/salida
pass in quick on $int_if all
pass out quick on $int_if all

# Interfaz conectada al enlace con Internet, solo aceptamos
# tráfico icmp de entrada y tcp/udp de respuesta a los paquetes
# de salida.
pass in quick on $ext_if proto icmp all keep state
pass in quick on $ext_if proto tcp from any to $ext_server port $openports flags S/SA synproxy state
pass in quick on $ext_if proto tcp from any to $ext_firewall port 21 flags S/SA keep state
pass out quick on $ext_if from <privips> to any modulate state
pass out quick on $ext_if from <direc> to any ports $availableports modulate state queue direc
pass out quick on $ext_if from <admin> to any ports $availableports modulate state queue admin
pass out quick on $ext_if from <secre> to any ports $availableports modulate state queue secre

```

Desde la version 3.3 de OpenBSD, Pf integra Altq (Alternate Queueing), que nos permite modelar las

colas en las que los paquetes son almacenados al llegar al firewall, de forma que en lugar de seguir un patrón **FIFO** (First In, First Out o *el que primero llega, antes se va*, podamos establecer reglas y prioridades en base a los orígenes, destinos y tipo de paquetes que estemos tratando en cada momento. La implementación de Altq integrada en OpenBSD permite el uso de tres tipos diferentes de algoritmos:

- **cbq** *Class Based Queueing* o *cola basada en clases*. Las colas se organizan con una estructura de árbol, de forma que cada cola puede tener subcolas, cada una con un ancho de banda y una prioridad asociadas. Con cbq no sólo podemos especificar anchos de banda y prioridades, si no que podemos establecer reglas de compartición de ancho de banda entre las diferentes colas.
- **priq** *Priority Queueing* o *encolado por prioridad*, mucho más sencillo que cbq, este algoritmo nos permite tan sólo establecer colas que no podrán tener subcolas y que dispondrán de una prioridad de 0 a 15. Aquellas colas con prioridad más elevada son tratadas primero.
- **hfsc** *Hierarchical Fair Service Curve* o *curva de servicio razonablemente jerárquica*. Este algoritmo estará presente en la próxima release de OpenBSD (OpenBSD 3.7) y es muy similar a cbq, sólo que además soporta *compartición de enlaces* (*link-sharing* y *garantía de servicio en tiempo real* gracias al uso de una curva de servicio basada en el modelo **QoS**.

En nuestro caso utilizaremos **cbq**. Primero definimos cual será la interfaz sobre la que aplicaremos el control de ancho de banda:

```
# Control de ancho de banda.  
altq on $ext_if cbq bandwidth 1Mb queue { direc, admin, secre }
```

Aquí establecemos que el control se efectuará sobre la interfaz externa del firewall, definiendo un ancho de banda máximo de 1Mbps y tres colas que utilizaremos luego para asignarlas con los equipos de cada uno de los tres segmentos comentados antes.

Una vez definido esto, pasamos a detallar las colas que acabamos de inicializar:

```
queue direc bandwidth 40% priority 5 cbq (red)  
queue admin bandwidth 50% priority 6 cbq (red)  
queue secre bandwidth 10% priority 4 cbq (default)
```

Definimos las tres colas, asignándoles un porcentaje de ancho de banda sobre el total de 1Mbps y una prioridad (cuanto mayor sea el número, mayor prioridad). Para las colas que aplicaremos a los directivos y a los administrativos agregamos el uso de **red** (**Random Early Detection**), que es un algoritmo de eliminación de congestiones que se encarga de monitorizar la carga de las colas y, en caso de que una cola estuviese a punto de llenarse de paquetes, actúa descartando paquetes de forma que la cola no se sature.

Una vez hemos establecido la interfaz sobre la que actuaremos, así como las colas, ya sólo nos queda asignar las colas sobre las reglas de filtrado que consideremos adecuadas. En nuestro caso hemos reemplazado la regla final que permitía el paso de paquetes desde la red local hacia Internet, por tres reglas, cada una aplicada a una tabla previamente creada que contiene una lista con las direcciones IP sobre la que aplicaremos el control de ancho de banda:

```
...  
table <direc> const { 192.168.1.1, 192.168.1.2 }  
table <admin> const { 192.168.1.3, 192.168.1.4, 192.168.1.5 }  
table <secre> const { 192.168.1.5, 192.168.1.6 }  
...  
pass out quick on $ext_if from <direc> to any ports $availableports modulate state queue direc  
pass out quick on $ext_if from <admin> to any ports $availableports modulate state queue admin  
pass out quick on $ext_if from <secre> to any ports $availableports modulate state queue secre
```

De esta forma conseguimos que el 50% de la conexión (500Kbps) esté garantizada para los administrativos, el 40% de la misma para los directivos y el 10% para los secretarios. En este caso establecemos las tablas que referencian a que equipos se aplicada cada cola como tablas **const**, lo que implica

que no pueden ser modificadas en tiempo real, pero esto podría cambiarse, lo que nos permitiría agregar un equipo de, por ejemplo, un secretario a la lista de equipos que tendrían el 50 % del caudal garantizado.

Llegado este punto, utilizamos de nuevo **pfctl** para recargar las reglas:

```
$ sudo pfctl -F all

rules cleared
nat cleared
0 tables deleted.
altq cleared
0 states cleared
source tracking entries cleared
pf: statistics cleared
pf: interface flags reset
$
```

### 3.3.8. ¿Que está pasando?

Hemos llegado a un punto en el que disponemos de un firewall bastante completo: NAT, filtrado, Altq. En estos momentos todo está funcional y seguro pero, no sería interesante disponer de algún tipo de información que poder controlar cada cierto tiempo para ver el rendimiento del firewall, si hay ataques, cuales de estos se repiten, que tal funciona el algoritmo de control de ancho de banda, etc?

Uno de los aspectos más importantes de cualquier firewall es que pueda ser monitorizable, que podamos disponer de toda la información posible y, a poder ser, de la forma más sencilla de ver posible. En este punto tenemos que diferenciar dos tipos de información, por un lado información referente al rendimiento del firewall y por otro información sobre los paquetes, conexiones, etc que maneja el firewall en si.

Vamos a ver primero como monitorizar el tráfico que pasa por el firewall. El primer paso es utilizar una opción que no hemos visto por el momento, **log**, que puede establecerse para aquellas reglas que consideremos especialmente sensibles o aquellas cuyos paquetes sean de mayor interés para nosotros. Para empezar es interesante aplicar la opción **log** a todos aquellos paquetes que sean rechazados, ya que entre esos podremos encontrar muchos escaneos, intentos de acceso o intentos de recogida de información ilícitos. Hay que tener en cuenta que esto puede generar unos ficheros de log importantemente grandes, pero es una buena forma de controlar los intentos de acceso.

Para empezar a loguear todos los paquetes que son bloqueados, basta con agregar la opción **log** de la forma:

```
# Bloqueamos todo por defecto
block in log all

# Antispoof, para las interfaces externa e interna
antispoof log quick for { $int_if, $ext_if }
...
pass in log quick on $ext_if proto tcp from any to $ext_server port $openports flags S/SA synproxy s
pass in log quick on $ext_if proto tcp from any tp $ext_firewall port 21 flags S/SA keep state
pass out log quick on $ext_if from <privips> to any modulate state
```

En nuestro caso establecemos la opción **log** en todos los paquetes bloqueados, en aquellos que sean bloqueados por la regla anti-spoofing, en los paquetes recibidos para los servicios que ofrecemos de cara al exterior y para todos aquellos paquetes provenientes de direcciones IP en la tabla de direcciones privilegiadas.

Los paquetes que cumplan alguna de las reglas en las que hemos puesto la opción **log** serán enviados al interfaz **pflog0**, el cual es monitorizado por el daemon **pflogd**, que a su vez escribe en **/var/log/pflog** la información sobre el paquete en formato binario. Si la regla tiene alguna de las opciones **keep**, **modulate** o

synproxy state habilitada, tan sólo se loguea el primer paquete, el que establece la conexión, a menos que se use la opción **log-all** en lugar de **log**, en ese caso se loguean todos los paquetes de la conexión. Esto es lo que sucede con aquellas reglas que no tienen habilitada ninguna de las opciones mencionadas.

El formato con el que pflogd almacena la información sobre los paquetes que son logueados es un formato binario, por lo que no podemos leer esos ficheros como si de un log normal se tratase (como /var/log/messages por ejemplo). Para monitorizar los paquetes logueados utilizamos una conocida herramienta de captura de paquetes, **tcpdump**, que nos permite tanto revisar el fichero pflog a posteriori, como leer en tiempo real la información que es enviada a pflog0 como si se tratase de un interfaz de red más del sistema. Tcpdump es una herramienta que nos ofrece un montón de posibilidades a la hora de su utilización, vamos a ver los usos más comunes junto a pf.

Para monitorizar, en tiempo real, lo que sucede en nuestro firewall, utilizaremos directamente tcpdump sobre pflog0:

```
$ sudo tcpdump -n -e -ttt -i pflog0
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0
```

Vemos que tcpdump nos muestra un aviso, no tiene mayor importancia, nos está avisando de que pflog0 es un interfaz sin una dirección IP asociada, pero pflog0 no necesita una dirección IP, como el resto de los interfaces de red. En este caso hemos llamado a tcpdump con las siguientes opciones:

- **-n** Utilizando este parámetro al llamar a tcpdump, eludimos la resolución de nombres para las direcciones IP de las que provienen y a las que van los paquetes. Esto puede ser una ventaja ya que el procesado de información es más rápido, pero en la mayoría de los casos el poder disponer de una referencia visual por nombre en lugar de una dirección IP puede ser más intuitivo.
- **-e** Esta opción hace que tcpdump nos muestre el número de la regla que ha procesado el paquete, así como el tipo de regla que es (block in, pass in, etc)
- **-ttt** Con esta opción, tcpdump mostrará, para cada paquete logueado, día y mes en formato timestamp.
- **-i** Esta opción va seguida del nombre del interfaz sobre el que queremos que tcpdump lea, en este caso, **pflog0**

Esta forma de uso de tcpdump es muy completa, pero también nos muestra mucha información por pantalla, podemos llamar a tcpdump de la siguiente forma para obtener un resultado más *compacto*:

```
$ sudo tcpdump -q -i pflog0
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0
```

También podemos hacer uso de determinados filtros a la hora de llamar a tcpdump, de forma que filtremos el resultado por origen, puerto o subred:

```
$ sudo tcpdump -n -e -ttt -i pflog0 host equipo1.pymegal.com
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0
$ sudo tcpdump -n -e -ttt -i pflog0 port 80
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0
$ sudo tcpdump -n -e -ttt -i pflog0 net 192.168.1.0
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0
$ sudo tcpdump -n -e -ttt -i pflog0 'host equipo1.pymegal.com and port 80'
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0
```

El resultado de la primera llamada serían solo los paquetes enviados/recibidos a/desde **equipo1.pymegal.com**, la segunda llamada mostraría aquellos paquetes cuyo destino sea el puerto estandar utilizado para servidores web y la tercera llamada mostraría aquellos paquetes cuyo origen o destino fuese la subred **192.168.1.0**.

El cuarto ejemplo nos muestra cómo combinar estos filtros para, por ejemplo, monitorizar los paquetes relacionados con **equipo1.pymegal.com**, pero sólo aquellos destinados al **puerto 80**.

Además de monitorizar directamente el interfaz pflog0, podemos hacer uso de la capacidad de tcpdump de leer contenidos de un fichero para ver, a posteriori, los paquetes logueados previamente por pflogd, para ello no tenemos más que cambiar el parámetro **-i interfaz** por **-r fichero**, de la forma:

```
$ sudo tcpdump -n -e -ttt -r /var/log/pflog
```

Todas las opciones comentadas anteriormente son aplicables igualmente cuando leemos ficheros en lugar de monitorizar un interfaz directamente. Tcpdump tiene muchas más opciones que las comentadas aquí, por lo que la lectura de su manual es más que aconsejable.

Hemos hablado de como monitorizar paquetes manejados por el firewall directamente, lo que nos permite tener un control sobre qué paquetes son bloqueados, cuales enviados desde un determinado origen o a un determinado destino. Vamos a ver ahora como monitorizar el firewall en cuanto a uso del mismo, recursos del sistema, utilización, etc.

Mediante la herramienta de control de pf, **pfctl**, que ya hemos visto anteriormente, podemos obtener información del estado del firewall. Utilizando el parámetro **-s** al llamar a pfctl, obtenemos información detallada de cada una de las diferentes partes de la configuración del firewall (nat, filtering, colas, tablas, etc). Por ejemplo, para ver el estado de las conexiones almacenadas por el firewall gracias a las opciones keep, modulate o synproxy state, podemos utilizar:

```
$ sudo pfctl -s state
```

Mientras que para ver información sobre el uso de pf en el sistema, podemos utilizar la siguiente expresión:

```
$ sudo pfctl -s info
Status: Enabled for 0 days 00:06:12          Debug: Urgent
```

```
Hostid: 0x71019b0e
```

State Table	Total	Rate
current entries	0	
searches	174	0.5/s
inserts	0	0.0/s
removals	0	0.0/s
Counters		
match	174	0.5/s
bad-offset	0	0.0/s
fragment	0	0.0/s
short	0	0.0/s
normalize	0	0.0/s
memory	0	0.0/s
bad-timestamp	0	0.0/s
congestion	0	0.0/s
ip-option	4	0.0/s
proto-cksum	0	0.0/s
state-mismatch	0	0.0/s
state-insert	0	0.0/s
state-limit	0	0.0/s

```

src-limit          0          0.0/s
synproxy           0          0.0/s

```

También podemos ver el uso de memoria e información acerca de la configuración de **timeouts** o tiempos de espera:

```

$ sudo pfctl -s memory
states      hard limit 10000
src-nodes   hard limit 10000
frags       hard limit 5000
$ sudo pfctl -s timeout
tcp.first           120s
tcp.opening         30s
tcp.established     86400s
tcp.closing         900s
tcp.finwait         45s
tcp.closed          90s
tcp.tsdiff          30s
udp.first           60s
udp.single          30s
udp.multiple        60s
icmp.first          20s
icmp.error          10s
other.first         60s
other.single        30s
other.multiple      60s
frag                30s
interval            10s
adaptive.start      0 states
adaptive.end        0 states
src.track           0s

```

Además de pfctl disponemos de algunas herramientas, que no forman parte del sistema base de OpenBSD, como pfstat, pftop o symon que nos permiten ver más información e incluso la generación de gráficas estadísticas.

### 3.4. Y la historia continua...

Además de lo que hemos visto hasta aquí, pf dispone de muchas otras ventajas, como la integración de spamd, un sistema antispam que pretende no sólo filtrar el correo basura, si no contraatacar a la maquina spammer consumiendo sus recursos; authpf para autenticación a través del propio firewall o el uso de la característica que permite a pf filtrar determinadas conexiones por nombre de usuario o grupo.

En este documento tan sólo hemos visto una pequeña parte de lo que es posible realizar con pf, y espero que haya servido para que os sintais animados a leer un poco más sobre ello, instalar OpenBSD y probar el que está considerado como posiblemente el sistema operativo más seguro del mundo.

Pf está en continuo desarrollo, con lo que cada nueva versión de OpenBSD trae consigo mejoras, nuevas funcionalidades y reimplementaciones de antiguas funcionalidades mejoradas, es por esto que es sobre OpenBSD donde se puede sacar el máximo rendimiento a pf, aunque ya se encuentra portado a los otros dos **flavours** BSD más conocidos, FreeBSD y NetBSD.

## 4. Bibliografía

### 4.1. Libros

- **Secure Architectures with OpenBSD** de Brandon Palmer y Jose Nazario, editorial Addison-

Wesley, abril de 2004

- **Building Firewalls with OpenBSD and PF, 2nd Edition** de Jacek Artymiak, 2003
- **The Design and Implementation of the 4.4BSD Operating System** de Marshall Kirk McKusick, Keith Bostic, Michael J. Karels y John S. Quarterman, editorial Addison-Wesley, 1996.

## 4.2. Enlaces/URLS

- **Wikipedia:** <http://wikipedia.org>
- **Proyecto OpenBSD:** <http://www.openbsd.org>
- **OpenBSD Journal:** <http://undeadly.org>
- **Proyecto FreeBSD:** <http://www.freebsd.org>
- **Proyecto NetBSD:** <http://www.netbsd.org>
- **Onlamp BSD:** <http://www.onlamp.com/bsd>
- **Pf, OpenBSD packet filter:** <http://www.benzedrine.cx/pf.html>
- **OpenBSD FAQ, pf:** <http://www.openbsd.org/faq/pf/index.html>
- **Pf blog:** <https://solarflux.org/pf-b/>
- **Pf forum:** <http://screamingelectron.org/forum/forumdisplay.php?f=50>
- **Firewalling with OpenBSD's PF packet filter:** <http://www.bgnett.no/peter/pf/en/index.html>
- **Understanding Packet Filter** [http://www.aei.ca/pmatulis/pub/obsd\\_pf.html](http://www.aei.ca/pmatulis/pub/obsd_pf.html)
- **TCP rfc:** <http://www.faqs.org/rfcs/rfc793.html>
- **The Addition of Explicit Congestion Notification (ECN) to IP**  
<http://www.faqs.org/rfcs/rfc3168.html>
- **Google:** <http://www.google.com>