Apéndice B

Monitorización de tráfico

Índice del capítulo

B.1. Instalación del módulo *ROUTE*

B.1.1. Obtención de los paquetes

Como se adelantó en el punto 5.2.2, "*iptables y el módulo ROUTE*", de este documento en esta ocasión no hemos podido utilizar *paquetes Debian* para recoger las utilidades que hemos necesitado para efectuar la monitorización de tráfico mediante la retransmisión de los paquetes con *iptables*. A continuación vamos a seguir el proceso completo de obtención, configuración e instalación del los paquetes del ***kernel*** de Linux y de ***iptables*** para conseguir esta funcionalidad.

Comenzaremos obteniendo los distintos paquetes que necesitamos, descargándolos de Internet. Primero obtengamos el ***kernel***:

```
#> cd /usr/src

#> wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.15.1.tar.bz2
--11:56:33--  http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.15.1.tar.bz2
          => 'linux-2.6.15.1.tar.bz2'
Resolving www.kernel.org... 204.152.191.5, 204.152.191.37
Connecting to www.kernel.org[204.152.191.5]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 39,844,313 [application/x-bzip2]
```

```
100%[==============================>] 39,844,313    66.11K/s    ETA 00:00

12:03:15 (96.92 KB/s) - 'linux-2.6.15.1.tar.bz2' saved [39844313/39844313]

#> wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.15.1.tar.bz2.sign
--12:06:26--  http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.15.1.tar.bz2.
    sign
           => 'linux-2.6.15.1.tar.bz2.sign'
Resolving www.kernel.org... 204.152.191.37, 204.152.191.5
Connecting to www.kernel.org[204.152.191.37]:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 248 [application/pgp-signature]

100%[==============================>] 248           --.--K/s

12:06:27 (2.37 MB/s) - 'linux-2.6.15.1.tar.bz2.sign' saved [248/248]
```

A continuación obtendremos las fuentes de ***iptables***:

```
#> wget ftp://ftp.netfilter.org/pub/iptables/iptables-1.3.5.tar.bz2
--12:10:11--  ftp://ftp.netfilter.org/pub/iptables/iptables-1.3.5.tar.bz2
           => 'iptables-1.3.5.tar.bz2'
Resolving ftp.netfilter.org... 213.95.27.115
Connecting to ftp.netfilter.org[213.95.27.115]:21... connected.
Logging in as anonymous ... Logged in!
==> SYST ... done.    ==> PWD ... done.
==> TYPE I ... done.  ==> CWD /pub/iptables ... done.
==> PASV ... done.    ==> RETR iptables-1.3.5.tar.bz2 ... done.
Length: 191,820 (unauthoritative)

100%[==============================>] 191,820       76.84K/s

12:10:27 (76.63 KB/s) - 'iptables-1.3.5.tar.bz2' saved [191820]

#> wget ftp://ftp.netfilter.org/pub/iptables/iptables-1.3.5.tar.bz2.sig
--12:10:48--  ftp://ftp.netfilter.org/pub/iptables/iptables-1.3.5.tar.bz2.sig
           => 'iptables-1.3.5.tar.bz2.sig'
Resolving ftp.netfilter.org... 213.95.27.115
Connecting to ftp.netfilter.org[213.95.27.115]:21... connected.
Logging in as anonymous ... Logged in!
==> SYST ... done.    ==> PWD ... done.
==> TYPE I ... done.  ==> CWD /pub/iptables ... done.
==> PASV ... done.    ==> RETR iptables-1.3.5.tar.bz2.sig ... done.
Length: 65 (unauthoritative)

100%[==============================>] 65            --.--K/s

12:11:03 (33.71 KB/s) - 'iptables-1.3.5.tar.bz2.sig' saved [65]
```

Y en tercer lugar, obtengamos las de ***patch-o-matic***:

```
#> wget ftp://ftp.netfilter.org/pub/patch-o-matic-ng/snapshot/patch-o-matic-ng
    -20060206.tar.bz2
--11:58:51--  ftp://ftp.netfilter.org/pub/patch-o-matic-ng/snapshot/patch-o-matic
    -ng-20060206.tar.bz2
           => 'patch-o-matic-ng-20060206.tar.bz2'
Resolving ftp.netfilter.org... 213.95.27.115
Connecting to ftp.netfilter.org[213.95.27.115]:21... connected.
Logging in as anonymous ... Logged in!
==> SYST ... done.    ==> PWD ... done.
==> TYPE I ... done.  ==> CWD /pub/patch-o-matic-ng/snapshot ... done.
==> PASV ... done.    ==> RETR patch-o-matic-ng-20060206.tar.bz2 ... done.
Length: 364,284 (unauthoritative)

100%[==============================>] 364,284       31.86K/s    ETA 00:00

11:59:27 (20.04 KB/s) - 'patch-o-matic-ng-20060206.tar.bz2' saved [364284]

#> wget ftp://ftp.netfilter.org/pub/patch-o-matic-ng/snapshot/patch-o-matic-ng
    -20060206.tar.bz2.md5sum
--12:01:10--  ftp://ftp.netfilter.org/pub/patch-o-matic-ng/snapshot/patch-o-matic
    -ng-20060206.tar.bz2.md5sum
```

```
          => 'patch-o-matic-ng-20060206.tar.bz2.md5sum'
Resolving ftp.netfilter.org... 213.95.27.115
Connecting to ftp.netfilter.org[213.95.27.115]:21... connected.
Logging in as anonymous ... Logged in!
==> SYST ... done.    ==> PWD ... done.
==> TYPE I ... done.  ==> CWD /pub/patch-o-matic-ng/snapshot ... done.
==> PASV ... done.    ==> RETR patch-o-matic-ng-20060206.tar.bz2.md5sum ... done.
Length: 68 (unauthoritative)

100%[==============================>] 68            --.--K/s

12:01:28 (29.97 KB/s) - 'patch-o-matic-ng-20060206.tar.bz2.md5sum' saved [68]
```

Comprobaremos a continuación la integridad de los archivos descargados. Comencemos igualmente por el *kernel*. Necesitaremos obtener la firma de *kernel.org* para poder comprobar la integridad del archivo [32, 33]:

```
#> gpg --keyserver wwwkeys.pgp.net --recv-keys 0x517D0F0E
gpg: keyring '/root/.gnupg/secring.gpg' created
gpg: requesting key 517D0F0E from hkp server wwwkeys.pgp.net
gpg: /root/.gnupg/trustdb.gpg: trustdb created
gpg: key 517D0F0E: public key "Linux Kernel Archives Verification Key <
    ftpadmin@kernel.org>" imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1

#> gpg --verify linux-2.6.15.1.tar.bz2.sign linux-2.6.15.1.tar.bz2
gpg: Signature made Sun Jan 15 07:58:31 2006 CET using DSA key ID 517D0F0E
gpg: Good signature from "Linux Kernel Archives Verification Key <ftpadmin@kernel
    .org>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: C75D C40A 11D7 AF88 9981  ED5B C86B A06A 517D 0F0E
```

El proceso es idéntico para verificar la integridad de *iptables* [34]:

```
#> gpg --keyserver wwwkeys.pgp.net --recv-keys 0xCA9A8D5B
gpg: requesting key CA9A8D5B from hkp server wwwkeys.pgp.net
gpg: key CA9A8D5B: public key "Netfilter Core Team <coreteam@netfilter.org>"
    imported
gpg: no ultimately trusted keys found
gpg: Total number processed: 1
gpg:               imported: 1

gpg --verify iptables-1.3.5.tar.bz2.sig iptables-1.3.5.tar.bz2
gpg: Signature made Wed Feb  1 14:03:57 2006 CET using DSA key ID CA9A8D5B
gpg: Good signature from "Netfilter Core Team <coreteam@netfilter.org>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 02AC E2A4 74DD 09D7 FD45  2E2E 35FA 89CC CA9A 8D5B
```

Por último comprobaremos la integridad de *patch-o-matic-ng* [35]:

```
#> md5sum -c -v patch-o-matic-ng-20060206.tar.bz2.md5sum
patch-o-matic-ng-20060206.tar.bz2 OK
```

Ya solo falta descomprimir las fuentes de los tres archivos:

```
#> tar -xvjf linux-2.6.15.1.tar.bz2
...

#> tar -xvjf iptables-1.3.5.tar.bz2
...

#> tar -xvjf patch-o-matic-ng-20060206.tar.bz2
...
```

B.1.2. Aplicando *patch-o-matic-ng* para compatibilizar con el módulo *ROUTE*

En esta sección veremos el uso de *patch-o-matic-ng* para parchear tanto al *kernel* como a *iptables* y hacerlos compatibles con el módulo *ROUTE* [29, 30].

patch-o-matic-ng se maneja de una forma muy sencilla [31]. Se basa en una *script* que nos va preguntando sucesivamente por cada uno de los módulos que contiene si deseamos instalarlos o no. *patch-o-matic-ng* comprueba qué módulos están ya instalados y cuales no son compatibles para instalarlos (bien porque han quedado obsoletos o bien porque las fuentes que queremos parchear han quedado anticuadas) de manera que nos evita que cometamos errores de manera cómoda. *patch-o-matic-ng* tiene además divididos los módulos que contiene en varios paquetes distintos, como *updates*, *submitted*, *pending*, *base*, *extra* u *obsolete*. Como es evidente, el nombre del paquete hace referencia al estado en que se encuentran los módulos que contiene. En nuestro caso el módulo *ROUTE* está en el paquete *extra* [29].

Veamos su uso[1]:

```
#> cd /usr/src/patch-o-matic-ng-20060205/

#> ./runme extra --kernel-path=/usr/src/linux-2.6.15.1 --iptables-path=/usr/src/
    iptables-1.3.5
Loading patchlet definitions..... done
```

Podremos responder a cada pregunta de la *script* con *N* para no aplicar el parche, *Y* para aplicar el parche, *T* para comprobar si el parche podría ser aplicado sin problemas o con *Q* para finalizar la *script*, entre otras opciones. Nosotros no aplicaremos ningún parche hasta llegar a la pregunta sombre el módulo *ROUTE*:

```
Welcome to Patch-o-matic ($Revision: 4088 $)!

Kernel:   2.6.15, /usr/src/linux-2.6.15.1
Iptables: 1.3.5, /usr/src/iptables-1.3.5
Each patch is a new feature: many have minimal impact, some do not.
Almost every one has bugs, so don't apply what you don't need!
-------------------------------------------------------
Already applied: NETMAP iprange

Testing ROUTE... not applied
The ROUTE patch:
   Author: éCdric de Launois <delaunois@info.ucl.ac.be>
   Status: Experimental


  This option adds a 'ROUTE' target, which enables you to setup unusual
  routes. For example, the ROUTE lets you route a received packet through
  an interface or towards a host, even if the regular destination of the
  packet is the router itself. The ROUTE target is also able to change the
  incoming interface of a packet.

  The target can be or not a final target. It has to be used inside the
  mangle table.

  ROUTE target options:
  --oif    ifname     Send the packet out using 'ifname' network interface.
  --iif    ifname     Change the packet's incoming interface to 'ifname'.
  --gw     ip         Route the packet via this gateway.
```

[1] Nótese que al parecer hubo un error en el empaquetado de esta versión de *patch-o-matic-ng* ya que el nombre del paquete y del directorio que se crea no son coincidentes.

```
  --continue        Route the packet and continue traversing the rules.
  --tee             Route a copy of the packet, but continue traversing
                    the rules with the original packet, undisturbed.

  Note that --iif, --continue, and --tee, are mutually exclusive.

  Examples :

  # To force all outgoing icmp packet to go through the eth1 interface
  # (final target) :
  iptables -A POSTROUTING -t mangle -p icmp -j ROUTE --oif eth1

  # To tunnel outgoing http packets and continue traversing the rules :
  iptables -A POSTROUTING -t mangle -p tcp --dport 80 -j ROUTE --oif tunl1 --
       continue

  # To forward all ssh packets to gateway w.x.y.z, and continue traversing
  # the rules :
  iptables -A POSTROUTING -t mangle -p tcp --dport 22 -j ROUTE --gw w.x.y.z --
       continue

  # To change the incoming network interface from eth0 to eth1 for all icmp
  # packets (final target) :
  iptables -A PREROUTING -t mangle -p icmp -i eth0 -j ROUTE --iif eth1

  # To copy (duplicate) all traffic from and to a local ECHO server
  # to a second box (nonfinal target)
  iptables -A PREROUTING -t mangle -p tcp --dport 7 -j ROUTE --gw 1.2.3.4 --tee
  iptables -A POSTROUTING -t mangle -p tcp --sport 7 -j ROUTE --gw 1.2.3.4 --tee

-------------------------------------------------------
Do you want to apply this patch [N/y/t/f/a/r/b/w/q/?]
```

A esta pregunta responderemos *Y*, y en el siguiente módulo ya podremos responder *Q* para finalizar la *script*.

```
Excellent! Source trees are ready for compilation.
```

Podemos ver como para la versión de *iptables* seleccionada algunos de los módulos incluidos en el paquete *extra* ya están incluidos como indica la sentencia «*Already applied: NETMAP iprange*». Esto es una muestra del rápido desarrollo y evolución a la que están sometidas tanto *iptables* como *patch-o-matic-ng*, y que hacen difícil a veces conseguir versiones compatibles de varios módulos entre sí y con un mismo *iptables* y *kernel*. Así mismo, la evolución de *patch-o-matic-ng* es todavía más violenta y caótica, desapareciendo y reapareciendo módulos nuevos y viejos de una versión a la siguiente.

A pesar de estos problemas, como ve se este proceso de parcheado de las fuentes es muy simplificado por *patch-o-matic-ng*, y ahora podemos pasar a compilar tanto el *kernel* como *iptables*.

B.1.3. Compilación e instalación del *kernel*

Pasaremos ahora a compilar el *kernel* [36] compatible con el módulo *ROUTE*. Primero será necesario obtener un par de paquetes necesarios para poder configurar y compilar el *kernel*:

```
#> apt-get install build-essential libncurses5-dev
```

No es nuestro objetivo centrarnos en el estudio de dichos paquetes. Instalandos ambos, usaremos un menú presentado en consola para activar como módulo (en lugar de establecerlo como *built-in*) el módulo *ROUTE* en el *kernel*.

```
#> cd /usr/src/linux-2.6.15.1

#> make menuconfig
```

Con la última orden, se nos mostrará un menú como el mostrado en la Figura B.1.

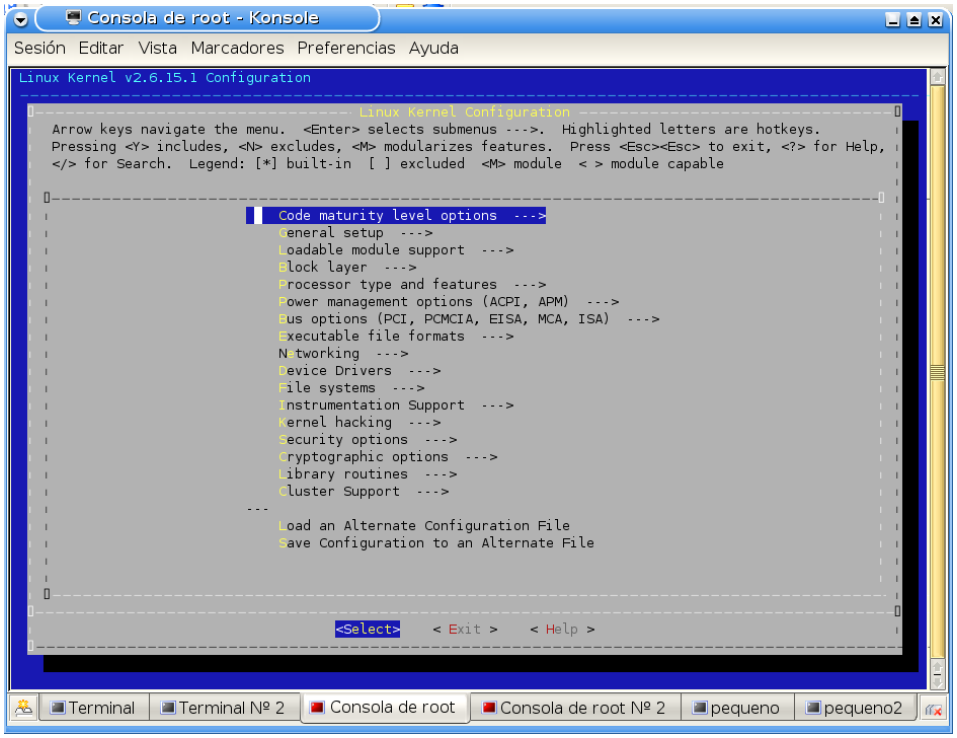

Figura B.1: Menú de configuración del *kernel* en modo consola

Gracias a este menú se pueden configurar las distintas opciones que componen nuestro *kernel*. Concretamente, el módulo *ROUTE* que deseamos configurar se encuentra en:

```
Networking
        Networking Options
                Network packet filtering (replaces ipchains)
                        IP: Netfilter Configuration
                                ROUTE target support (NEW)
```

Tras activarlo como módulo (M), podremos salir del menú grabando la configuración del nuevo *kernel*. Tras esto solo nos resta compilar e instalar:

```
#> cd /usr/src/linux-2.6.15.1

#> make all
...

#> make modules_install
...

#> cp /usr/src/linux-2.6.15.1/System.map /boot/System.map-2.6.15.1-ROUTE
```

```
#> cp /usr/src/linux-2.6.15.1/arch/i386/boot/bzImage /boot/vmlinuz-2.6.15.1-ROUTE

#> mkinitrd -o /boot/initrd.img-2.6.15.1-ROUTE 2.6.15.1
```

Tras esto, solo nos queda añadir este *kernel* al archivo `/boot/grub/menu.lst` las siguientes líneas para habilitar este núcleo en el arranque del sistema:

`/boot/grub/menu.lst` (líneas añadidas)

```
    title           Debian GNU/Linux, kernel 2.6.15.1-ROUTE
    root            (hd0,1)
    kernel          /boot/vmlinuz-2.6.15.1-ROUTE root=/dev/hda2 ro
    initrd          /boot/initrd.img-2.6.15.1-ROUTE
5   savedefault
    boot
```

B.1.3.1. Creando un *paquete Debian* del *kernel*

Como alternativa a lo anterior, se ofrece la posibilidad de crear un *paquete Debian* del *kernel* [37]. El motivo de crear este paquete es facilitar el transporte de este *kernel* a más sistemas, sin tener que pasar por los pesados pasos de compilación de la sección anterior. Este sistema es recomendado encarecidamente, ya que no solo tiene las ventajas anteriores sino que nos permite almacenar el *kernel* y nos ahorra tener que volver a repetir el proceso (muy costoso en términos de tiempo).

Para empezar será necesario crear un enlace simbólico al directorio donde reside el *kernel* ya configurado. Para ello podemos ejecutar lo siguiente:

```
#> ln -s /usr/src/linux /usr/src/linux-2.6.15.1
```

Con esto tendremos el directorio `/usr/src/linux` que será un enlace al directorio real donde se ubica nuestro *kernel*.

También necesitaremos descargar e instalar el siguiente paquete:

```
#> apt-get install kernel-package
```

El paquete ***kernel-package*** dispone de una serie de utilidades que facilitan la creación de *paquetes Debian* de *kernels* para los sistemas. Nosotros sólo vamos a mencionar los comandos y parámetros necesarios para obtener el *paquete Debian* del *kernel* deseado, sin centrarnos especialmente en todas sus capacidades.

Lo primero será tener el *kernel* adecuadamente configurado, como se hizo en la sección B.1.3, "*Compilación e instalación del kernel*". También limpiaremos los resultados de la compilación de la sección anterior para enfatizar el hecho de que la construcción del paquete del *kernel* es independiente a la compilación anterior. Para ello se pueden utilizar los comandos:

```
#> cd /usr/src/linux

#> make menuconfig
...

#> make clean
...
```

Ya estamos preparados para crear el paquete del *kernel*. Con el programa *make-kpkg* incluído dentro del paquete ***kernel-package*** primero nos aseguraremos de limpiar la imagen del *kernel* que vamos a empaquetar de posibles intentos anteriores (fallidos o no) de empaquetado. Para ello ejecutaremos:

```
#> cd /usr/src/linux

#> make-kpkg clean
...
```

Finalmente crearemos el paquete del *kernel*. Para poder diferenciar la versión que estamos compilando de otras versiones compiladas y, sobre todo, para evitar dañar la versión del *kernel* que esté actualmente siendo ejecutada en el sistema (en el caso de ser la misma versión que la que vamos a empaquetar) utilizaremos el parámetro `--append-to-version` de *make-kpkg*. Esto nos permitirá evitar los problemas anteriormente citados además de permitirnos diferenciar con comodidad distintas versiones de *kernels* que hayamos empaquetado. El comando para empaquetar el *kernel* sería el siguiente:

```
#> make-kpkg --initrd --bzimage --append-to-version=-route kernel_image
...
```

La ejecución de este comando nos creará en el directorio padre del directorio donde es ejecutado, es decir en `/usr/src`, el *paquete Debian* correspondiente:

```
linux-image-2.6.15.1-route_2.6.15.1-route-10.00.Custom_i386.deb
```

La forma de utilizar estos paquetes es muy sencilla: se instalarán con el gestor de paquetes *dpkg* de los sistemas **Debian Sarge**. En su instalación el paquete ubicará en los lugares adecuados los módulos del *kernel* (típicamente en `/lib/modules`)y los demás archivos, y realizará las modificaciones adecuadas al archivo `/boot/grub/menu.lst` que permitirán arrancar el nuevo *kernel*, todo ello de forma automática y transparente.

B.1.4. Compilación en instalación de *iptables*

El siguiente paso será actualizar la versión de *iptables* del sistema sustituyéndola por una versión compatible con el módulo *ROUTE*. Dado que en el sistema **Debian Sarge** viene ya una versión de *iptables* instalada, lo primero que haremos será eliminarla del sistema al completo para evitar posibles problemas de cruce de versiones:

```
#> apt-get remove --purge iptables
Reading Package Lists... Done
Building Dependency Tree... Done
The following packages will be REMOVED:
  iptables*
0 upgraded, 0 newly installed, 1 to remove and 1 not upgraded.
Need to get 0B of archives.
After unpacking 1270kB disk space will be freed.
Do you want to continue? [Y/n] Y
(Reading database ... 103701 files and directories currently installed.)
Removing iptables ...
```

Tras esto, podemos pasar a compilar e instalar *iptables*, algo que se hace rápidamente:

```
#> cd /usr/src/iptables-1.3.5

#> make KERNEL_DIR=/usr/src/linux-2.6.15.1
...

#> make install KERNEL_DIR=/usr/src/linux-2.6.15.1
...
```

Con esto ya tenemos el nuevo *iptables* compatible con el módulo *ROUTE* instalado en nuestro sistema. Tras esto podremos reiniciar el sistema con el nuevo *kernel* para activar la compatibilidad con *ROUTE*.

B.1.4.1. Creando un *paquete Debian* de *iptables*

El proceso de creado de un *paquete Debian* para *iptables* [38, 39] difiere bastante del sistema para hacer un paquete del *kernel* que se estudió en la sección B.1.3.1, "*Creando un paquete Debian del kernel*". Los motivos de querer hacer un *paquete Debian* son los mismos: aumentar la facilidad en el transporte e instalación de los paquetes necesarios en todas las máquinas de nuestro sistema de monitorización.

Empezaremos obteniendo los paquetes necesarios con la siguiente orden:

```
#> apt-get install dh-make fakeroot
```

No es nuestro objetivo centrarnos en el uso y funcionamiento de las utilidades comprendidas en dichos paquetes. Tras ello, nos moveremos al directorio de las fuentes de *iptables* ya descomprimidas, donde limpiaremos los objetos ya compilados. También modificaremos el nombre del directorio de las fuentes para diferenciar el paquete resultante del paquete original:

```
#> mv /usr/src/iptables-1.3.5 /usr/src/iptables-1.3.5-ROUTE

#> cd /usr/src/iptables-1.3.5-ROUTE

#> make clean
```

Ahora ejecutaremos el comando *dh_make* que acabamos de obtener en el paquete ***dh-make*** que nos preparará una serie de archivos para poder realizar posteriormente nuestro paquete.

```
#> dh_make -e rmicmir@gmail.com -c gpl -p iptables-1.3.5-route -f ../iptables
     -1.3.5.tar.bz2

Type of package: single binary, multiple binary, library, kernel module or cdbs?
 [s/m/l/k/b] s

Maintainer name : Rafael Mico Miranda
Email-Address   : rmicmir@gmail.com
Date            : Wed,  7 Jun 2006 12:35:14 +0200
Package Name    : iptables-1.3.5-ROUTE
Version         : 1.3.5-ROUTE
License         : gpl
Type of Package : Single
Hit <enter> to confirm:
Done. Please edit the files in the debian/ subdirectory now. You should also
check that the iptables-1.3.5-ROUTE Makefiles install into $DESTDIR and not in /
     .
```

Esto nos creará un directorio `debian` dentro de nuestra estructura de las fuentes de *iptables*, es decir, nos creará el directorio `/debian` dentro de la ruta de `/usr/src/iptables-1.3.5-ROUTE`. En él encontraremos una serie de archivos que habrá que editar para configurar correctamente el paquete. Ha sido necesario realizar modificaciones en los siguientes ficheros:

```
/usr/src/iptables-1.3.5-ROUTE/Makefile
/usr/src/iptables-1.3.5-ROUTE/Rules.make
/usr/src/iptables-1.3.5-ROUTE/debian/changelog
/usr/src/iptables-1.3.5-ROUTE/debian/control
/usr/src/iptables-1.3.5-ROUTE/debian/copyright
/usr/src/iptables-1.3.5-ROUTE/debian/rules
```

Los archivos `Makefile` y `Rules.make` han sido editados para arreglar problemas en la creación del paquete. Las modificaciones realizadas han sido, respectivamente, las siguientes:

`/usr/src/iptables-1.3.5-ROUTE/Makefile` (modificado)

```
    ifndef KERNEL_DIR
15  KERNEL_DIR=/usr/src/linux
    endif
    IPTABLES_VERSION:=1.3.5
    OLD_IPTABLES_VERSION:=1.3.4

20

    LIBDIR:=/lib
    BINDIR:=/sbin
    MANDIR:=/usr/share/man
    INCDIR:=/include
25
    # directory for new iptables releases
    RELEASE_DIR:=/tmp
```

`/usr/src/iptables-1.3.5-ROUTE/Rules.make` (modificado)

```
    # Have to handle extensions which no longer exist.
    clean: $(EXTRA_CLEANS)
            rm -f $(SHARED_LIBS) $(EXTRAS) $(EXTRAS_EXP) $(SHARED_LIBS:%.so=%_sh.o)
10          rm -f extensions/initext.c extensions/initext6.c
            @find . -name '*.[ao]' -o -name '*.so' | xargs rm -f

    install: all $(EXTRA_INSTALLS)
            @if [ -f /usr/local/bin/iptables ];\
15          then echo 'Erasing iptables from old location (now "$(BINDIR)").';\
```

```
        rm -f /usr/local/bin/iptables;\
        fi

install-experimental: $(EXTRA_INSTALLS_EXP)
```

Las modificaciones realizadas sobre los archivos citados emplazados en `/debian` (dentro de `/usr/src/iptables-1.3.5-ROUTE`) son mucho más simples:

`/usr/src/iptables-1.3.5-ROUTE/debian/changelog` (modificado)

```
iptables-1.3.5-route (1.3.5-ROUTE-1) experimental; urgency=low

  * Initial release
  * This is my first Debian package.
  * Adjusted the Makefile to fix $DESTDIR problems.


 -- Rafael Mico Miranda <rmicmir@gmail.com>  Wed,  7 Jun 2006 12:35:14 +0200
```

`/usr/src/iptables-1.3.5-ROUTE/debian/control` (modificado)

```
Source: iptables-1.3.5-route
Section: net
Priority: optional
Maintainer: Rafael Mico Miranda <rmicmir@gmail.com>
Build-Depends: debhelper (>= 4.0.0)
Standards-Version: 3.6.2

Package: iptables-1.3.5-route
Architecture: any
Depends: ${shlibs:Depends}, ${misc:Depends}
Replaces: iptables
Description: iptables with patch-o-matic ROUTE target
 iptables-1.3.5 with patch-o-matic ROUTE target
```

`/usr/src/iptables-1.3.5-ROUTE/debian/copyright` (modificado)

```
This package was debianized by Rafael Mico Miranda <rmicmir@gmail.com> on
Wed,  7 Jun 2006 12:35:14 +0200.

It was downloaded from ftp://ftp.netfilter.org/pub/iptables/iptables-1.3.5.tar.
    bz2
  and sources were modified with
  ftp://ftp.netfilter.org/pub/patch-o-matic-ng/snapshot/patch-o-matic-ng
      -20060206.tar.bz2

Copyright Holder: netfilter.org

License:

   This package is free software; you can redistribute it and/or modify
   it under the terms of the GNU General Public License as published by
   the Free Software Foundation; either version 2 of the License, or
   (at your option) any later version.

   This package is distributed in the hope that it will be useful,
   but WITHOUT ANY WARRANTY; without even the implied warranty of
   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
   GNU General Public License for more details.

   You should have received a copy of the GNU General Public License
   along with this package; if not, write to the Free Software
   Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301 USA

On Debian systems, the complete text of the GNU General
Public License can be found in '/usr/share/common-licenses/GPL'.
```

Hecho esto, podemos pasar a compilar y empaquetar nuestro paquete, lo que haremos con la siguiente orden:

```
#> cd /usr/src/iptables-1.3.5-route

#> dpkg-buildpackage -rfakeroot
```

Esto nos creará en el directorio padre `/usr/src` los siguientes archivos:

```
iptables-1.3.5-route_1.3.5-ROUTE-1.dsc
iptables-1.3.5-route_1.3.5-ROUTE-1.tar.gz
iptables-1.3.5-route_1.3.5-ROUTE-1_i386.changes
iptables-1.3.5-route_1.3.5-ROUTE-1_i386.deb
```

El que nos interesa en cuestión es el archivo `.deb` que es el paquete instalable. Tras instalarlo con *dpkg* podemos ver qué archivos ha emplazado, y se puede comprobar que la ubicación es idéntica a la de un *iptables* normal.

```
#> /usr/src# dpkg -L iptables-1.3.5-route
/.
/usr
/usr/bin
/usr/sbin
/usr/share
/usr/share/man
/usr/share/man/man8
/usr/share/man/man8/iptables-save.8.gz
/usr/share/man/man8/ip6tables.8.gz
/usr/share/man/man8/iptables.8.gz
/usr/share/man/man8/iptables-restore.8.gz
/usr/share/doc
/usr/share/doc/iptables-1.3.5-route
/usr/share/doc/iptables-1.3.5-route/README.Debian
/usr/share/doc/iptables-1.3.5-route/copyright
/usr/share/doc/iptables-1.3.5-route/changelog.Debian.gz
/sbin
/sbin/iptables
/sbin/iptables-save
/sbin/iptables-restore
/sbin/ip6tables
/lib
/lib/iptables
/lib/iptables/libipt_ah.so
/lib/iptables/libipt_addrtype.so
/lib/iptables/libipt_comment.so
/lib/iptables/libipt_connlimit.so
/lib/iptables/libipt_connmark.so
/lib/iptables/libipt_conntrack.so
/lib/iptables/libipt_dscp.so
/lib/iptables/libipt_ecn.so
/lib/iptables/libipt_esp.so
/lib/iptables/libipt_hashlimit.so
/lib/iptables/libipt_helper.so
/lib/iptables/libipt_icmp.so
/lib/iptables/libipt_iprange.so
/lib/iptables/libipt_length.so
/lib/iptables/libipt_limit.so
/lib/iptables/libipt_mac.so
/lib/iptables/libipt_mark.so
/lib/iptables/libipt_multiport.so
/lib/iptables/libipt_owner.so
/lib/iptables/libipt_physdev.so
/lib/iptables/libipt_pkttype.so
/lib/iptables/libipt_policy.so
/lib/iptables/libipt_realm.so
/lib/iptables/libipt_rpc.so
/lib/iptables/libipt_sctp.so
/lib/iptables/libipt_standard.so
/lib/iptables/libipt_state.so
/lib/iptables/libipt_tcp.so
/lib/iptables/libipt_tcpmss.so
/lib/iptables/libipt_tos.so
/lib/iptables/libipt_ttl.so
/lib/iptables/libipt_udp.so
```

```
/lib/iptables/libipt_unclean.so
/lib/iptables/libipt_CLASSIFY.so
/lib/iptables/libipt_CONNMARK.so
/lib/iptables/libipt_DNAT.so
/lib/iptables/libipt_DSCP.so
/lib/iptables/libipt_ECN.so
/lib/iptables/libipt_LOG.so
/lib/iptables/libipt_MARK.so
/lib/iptables/libipt_MASQUERADE.so
/lib/iptables/libipt_MIRROR.so
/lib/iptables/libipt_NETMAP.so
/lib/iptables/libipt_NFQUEUE.so
/lib/iptables/libipt_NOTRACK.so
/lib/iptables/libipt_REDIRECT.so
/lib/iptables/libipt_REJECT.so
/lib/iptables/libipt_SAME.so
/lib/iptables/libipt_SNAT.so
/lib/iptables/libipt_TARPIT.so
/lib/iptables/libipt_TCPMSS.so
/lib/iptables/libipt_TOS.so
/lib/iptables/libipt_TRACE.so
/lib/iptables/libipt_TTL.so
/lib/iptables/libipt_ULOG.so
/lib/iptables/libipt_CLUSTERIP.so
/lib/iptables/libipt_ROUTE.so
/lib/iptables/libipt_connbytes.so
/lib/iptables/libipt_dccp.so
/lib/iptables/libipt_recent.so
/lib/iptables/libipt_string.so
/lib/iptables/libip6t_connmark.so
/lib/iptables/libip6t_eui64.so
/lib/iptables/libip6t_hl.so
/lib/iptables/libip6t_icmpv6.so
/lib/iptables/libip6t_length.so
/lib/iptables/libip6t_limit.so
/lib/iptables/libip6t_mac.so
/lib/iptables/libip6t_mark.so
/lib/iptables/libip6t_multiport.so
/lib/iptables/libip6t_owner.so
/lib/iptables/libip6t_physdev.so
/lib/iptables/libip6t_policy.so
/lib/iptables/libip6t_standard.so
/lib/iptables/libip6t_state.so
/lib/iptables/libip6t_tcp.so
/lib/iptables/libip6t_udp.so
/lib/iptables/libip6t_CONNMARK.so
/lib/iptables/libip6t_HL.so
/lib/iptables/libip6t_LOG.so
/lib/iptables/libip6t_NFQUEUE.so
/lib/iptables/libip6t_MARK.so
/lib/iptables/libip6t_TRACE.so
/lib/iptables/libip6t_REJECT.so
/lib/iptables/libip6t_ROUTE.so
/lib/iptables/libip6t_ah.so
/lib/iptables/libip6t_esp.so
/lib/iptables/libip6t_frag.so
/lib/iptables/libip6t_ipv6header.so
/lib/iptables/libip6t_hbh.so
/lib/iptables/libip6t_dst.so
/lib/iptables/libip6t_rt.so
```

B.1.5. Activación del módulo *ROUTE*

En este momento, tras seguir las secciones B.1.1, B.1.2, B.1.3 y B.1.4, ya disponemos de un sistema compatible con el módulo *ROUTE* de *iptables*. Ahora veremos el uso que daremos del módulo para retransmitir los paquetes hacia la máquina que alberga el sistema de monitorización.

La mayor parte de la documentación sobre el módulo *ROUTE* ya ha sido mostrada durante la aplicación de *patch-o-matic-ng*, como se puede ver en la sección B.1.2, "*Aplicando patch-o-matic-ng para compatibilizar con el módulo ROUTE*", y podemos consultar una ayuda más resumida de la siguiente forma:

```
#> iptables -j ROUTE --help
...
ROUTE target v1.11 options:
    --oif       ifname   Route packet through 'ifname' network interface
    --iif       ifname   Change packet's incoming interface to 'ifname'
    --gw        ip       Route packet via this gateway 'ip'
    --continue           Route packet and continue traversing the
                         rules. Not valid with --iif or --tee.
    --tee                Duplicate packet, route the duplicate,
                         continue traversing with original packet.
                         Not valid with --iif or --continue.
```

Nosotros queremos aprovecharnos de la opción `--tee` [29, 30] que es la que permite duplicar un paquete mientras que el paquete original sigue siendo procesado por el resto de tablas de *iptables* de forma transparente a la duplicación. Como puede verse, *ROUTE* es un objetivo, un destino al que pueden enviarse los paquetes en *iptables* al igual que se mandarían al objetivo `ACCEPT` o a otra cadena definida por el usuario. Además de esto *ROUTE* es un objetivo que solo podrá ser utilizado dentro de la tabla `mangle` de *iptables*, la tabla utilizada para la alteración específica de paquetes.

Suponiendo que el sistema remoto es un sistema que actúa como cortafuegos de una red o *DMZ*, para analizar el tráfico que atraviesa el sistema será conveniente establecer la orden de retransmisión en la subtabla `FORWARD` de la tabla `mangle` del sistema remoto.

Ahora hacemos la suposición siguiente acerca de la estructura de la red:

- El sistema de monitorización se aloja en el sistema con dirección IP 192.168.100.21.

Con esto ya basta ejecutar el siguiente comando en el sistema remoto para para retransmitir con *iptables* [28] hacia nuestro sistema de monitorización:

```
#> iptables -t mangle -I FORWARD -j ROUTE --gw 192.168.100.21 --tee
```

Sin embargo, si deseamos ver todo el tráfico que se intenta enviar un *host* concreto, en este caso 192.168.100.22, deberíamos hacer la retransmisión antes que en la subtabla `FORWARD` por si las reglas de filtrado actuasen antes de dicha subtabla eliminando paquetes que el *host* sí está enviando:

```
#> iptables -t mangle -I PREROUTING -s 192.168.100.22 -j ROUTE --gw
     192.168.100.21 --tee

#> iptables -t mangle -I POSTROUTING -d 192.168.100.22 -j ROUTE --gw
     192.168.100.21 --tee
```

El que la retransmisión de paquetes se realice con *iptables* nos abre un abanico de posibilidades adicionales, permitiéndonos realizar retransmisiones mucho más específicas haciendo uso de sus capacidades, pero por norma general desearemos saber exactamente qué hace un *host* concreto, por lo que habitualmente se retransmitirá todo su tráfico.

A modo de detalle final, se adjunta la Figura B.2 donde se matiza la ubicación que deberán tener en la red tanto el módulo *ROUTE*, con su correspondiente *kernel* e *iptables* compatibles, y el *sniffer* o sistema que analizará el tráfico.

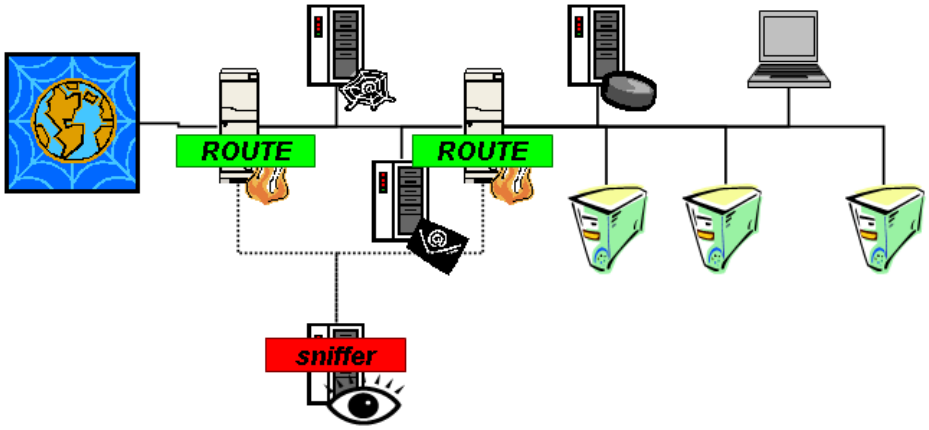

Figura B.2: Ubicación del módulo *ROUTE*

B.2. Instalación y configuración del paquete *ssh*

B.2.1. Instalación de *ssh*

El paquete ***ssh*** se instala por defecto en la instalación base del sistema **Debian Sarge**. Para comprobar que se encuentra instalado podemos hacer lo siguiente:

```
#> dpkg -l ssh
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Installed/Config-files/Unpacked/Failed-config/Half-installed
|/ Err?=(none)/Hold/Reinst-required/X=both-problems (Status,Err: uppercase=bad)
||/ Name     Version              Description
+++-========-====================-====================
ii  ssh      3.8.1p1-8.sarge.4    Secure rlogin/rsh/rcp replacement (OpenSSH)
```

En caso de que no apareciese como instalado, siempre podemos instalarlo con su correspondiente orden de *apt*:

```
#> apt-get install ssh
```

Los contenidos del paquete se podrán consultar con la siguiente orden:

```
#> dpkg -L ssh
```

B.2.2. Configuración de *ssh*

Su archivo de configuración se encuentra en `/etc/ssh/ssh_config`, y no vamos a realizar ninguna modificación en ella.

Como se vio en la sección B.1.5, "*Activación del módulo ROUTE*", será necesario realizar una acción sobre el sistema remoto que contiene las versiones modificadas del *kernel* y de *iptables* para activar el módulo *ROUTE*. El funcionamiento normal de *ssh* se basa en una autenticación de usuario y clave para acceder a la sesión en el sistema remoto. En nuestro caso queremos eliminar la necesidad de la contraseña para que las acciones de control puedan realizarse de una forma más transparente de manera que no se necesitase la interacción de una persona para introducir la clave, pero manteniendo unos niveles de seguridad mínimos.

Por ello, vamos a proceder a crear un par de llaves cliente-servidor [40, 41] para poder autenticar de forma cómoda al sistema que alberga el sistema de monitorización (que de cara a la aplicación *ssh* hará las veces de cliente) contra el sistema que retransmitirá el tráfico (que hará las veces de servidor). Estas llaves identificarán al cliente en el servidor y permitirán la acción de control sin tener que mediar una clave en el proceso, lo que es ideal para ser utilizado luego en sistemas más automatizados.

Sí hay que resaltar que para poder manejar *iptables* en un sistema es necesario tener los privilegios de `root`, así que en lo que sigue tendremos que tomar el papel de ese usuario y las conexiones del cliente a los servidores deberán hacerse basándose en la cuenta de `root`.

Comenzaremos creando en el cliente, la máquina que retransmitirá el tráfico, el par de llaves cliente servidor:

(Sistema de monitorización)

```
#> cd /root/.ssh

#> ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
fe:19:7b:86:ac:32:84:6e:4f:f9:33:ce:e5:1d:c7:29 root@debian

#> ls -lash
total 20K
4.0K drwx------   2 root root 4.0K Feb 12 16:18 .
4.0K drwxr-xr-x  14 root root 4.0K Feb 12 16:16 ..
4.0K -rw-------   1 root root  668 Feb 12 16:18 id_rsa
4.0K -rw-r--r--   1 root root  601 Feb 12 16:18 id_rsa.pub
4.0K -rw-r--r--   1 root root  239 Feb 12 16:16 known_hosts
```

Para que no nos solicite claves en un futuro será necesario establecer el *passphrase* en blanco. La ejecución del comando anterior nos ha generado los archivos `id_rsa` e `id_rsa.pub`, donde el segundo archivo es el que contiene la llave pública. Ahora tenemos que distribuir la llave del cliente a los sistemas remotos que albergan los *kernels* e *iptables* modificados, de manera que todos

tengan la misma llave y podamos conectarnos a todos ellos desde el sistema de monitorización. Para ello distribuiremos el archivo `id_rsa.pub` a los sistemas remotos y lo añadiremos a su archivo `authorized_keys` en cada sistema remoto:

(Sistema remoto)

```
#> cd /root/.ssh

#> cat id_rsa.pub >> autorized_keys
```

Una vez hecho esto podemos efectuar fácilmente los comandos remotos, ya que el servidor que contiene el archivo `authorized_keys` actualizado recibirá del cliente la llave adecuada que realizará la autenticación. En el ejemplo siguiente estaríamos ordenando al sistema remoto 192.168.100.24 que retransmita el tráfico que la atraviesa a 192.168.100.21 que sería nuestro sistema de monitorización en el que se ejecutaría un *sniffer*, como se detalla en la Figura B.3:

(Sistema remoto)

```
#> ssh root@192.168.100.24 iptables -t mangle -I FORWARD -j ROUTE --gw
     192.168.100.21 --tee
```

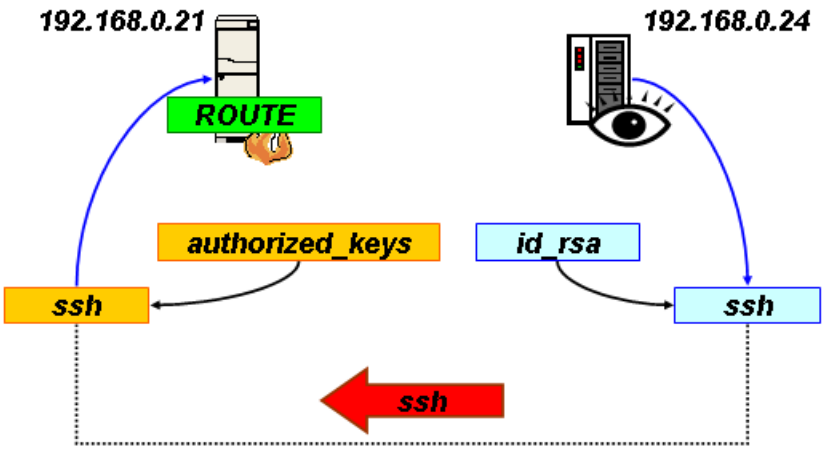

Figura B.3: Uso de *ssh* y sus llaves para la ejecución de comandos

Merece la pena recordar que la autenticación con el par de llaves cliente-servidor autentifica a un usuario concreto, en este caso `root`, entre los dos sistemas, por ello hay que emplazar los archivos en los directorios de `/root/.ssh` tanto en los servidores (sistemas remotos) como en el cliente (sistema de monitorización).