Linux 2.4 NAT COMO

Rusty Russell, lista de correo *netfilter@lists.samba.org*
Traducido por Ricardo J. Cárdenes Medina *a1402@dis.ulpgc.es*
v1.0.1 Lunes 1 de Mayo 18:38:22 CST 2000, traducción del 25 de Junio de 2000

Este documento describe cómo hacer el enmascarado (masqueradinq), proxy transparente, reenvío de puertos (port forwarding), y otras formas de Network Address Translation (Traducción de Direcciones de Red) con los núcleos 2.4 de Linux.

Índice

1. Introducción

Bienvenido, gentil lector.

Está a punto de sumergirse en el fascinante (y a veces horrendo) mundo del NAT (Network Address Translation), y este COMO va a ser su guía más o menos precisa para el núcleo 2.4 de Linux y posteriores.

En Linux 2.4, se ha introducido una infrastuctura para trastear con los paquetes, llamada «netfilter». Hay una capa por encima que proporciona NAT, completamente reescrita con respecto a anteriores núcleos.

2. ¿Dónde está el sitio web oficial y la lista?

Hay tres sitios oficiales:

- Gracias a Penguin Computing: *http://netfilter.filewatcher.org/* .
- Gracias a el equipo Samba y a SGI *http://netfilter.samba.org/* .
- Gracias a Harald Welte *http://netfilter.gnumonks.org/* .

La lista oficial de correo de netfilter está en el servidor de listas de Samba: *http://www.netfilter.org/contact.html#list*

2.1. ¿Qué es Network Address Translation?

Normalmente, los paquetes viajan en una red desde su origen (por ejemplo su ordenador) a su destino (como por ejemplo www.gnumonks.org) a través de varios enlaces diferentes: unos 19 desde donde yo estoy en Australia (esto lo dice Rusty, claro). Ninguno de estos enlaces altera realmente el paquete: simplemente lo envían un paso adelante.

Si uno de estos enlaces hiciera NAT, podría alterar el origen o destino del paquete según pasa a través suyo. Como puede imaginar, ésta no es la función para la que se diseñó el sistema, y por tanto NAT es siempre un tanto enrevesado. Normalmente, el enlace que esté haciendo NAT recordará cómo jugueteó con el paquete, para hacer la acción inversa con el paquete de respuesta, de manera que todo funciona como se esperaba.

2.2. Razones para usar NAT

En un mundo perfecto, no debería. Mientras tanto, las razones principales son:

Conexiones con módem a Internet

La mayoría de los PSI (Proveedor de Servicios de Internet) le dan una sola dirección IP cuando se conecta con ellos. Puede enviar paquetes con cualquier dirección que le plazca, pero sólo obtendrá respuestas a los paquetes con esa IP de origen. Si desea utilizar varias máquinas diferentes (como una red casera) para conectar a Internet a través de un enlace, necesita NAT.

Este es, de lejos, el uso más común de NAT hoy en día, conocido normalmente como «enmascaramiendo» (masquerading) en el mundo de Linux. Yo lo llamo SNAT, porque se cambia la dirección de **origen** (source) del primer paquete.

Varios servidores

Puede que quiera cambiar el destino de los paquetes que entran en su red. Con frecuencia esto se debe (como antes), a que sólo tiene una dirección IP, pero desea que la gente sea capaz de llegar a las máquinas detrás de la que tiene la IP «real». Si reescribe el destino de los paquetes entrantes, podrá conseguirlo.

Una variante común de esto es el balanceo de carga, en la cual se toma un cierto número de máquinas, repartiendo los paquetes entre ellas. Este tipo de NAT se llamó reenvío de puerto (port-forwarding) en anteriores versiones de Linux.

Proxy transparente

Hay veces que deseará simular que cada paquete que pase por su máquina Linux esté destinado a un programa en la propia máquina. Esto se utiliza para hacer proxyes transparentes: un proxy es un programa que se pone entre su red y el mundo real, filtrando las comunicaciones entre ambos. La parte transparente se debe a que su red nunca tendrá por qué enterarse de que está comunicándose con un proxy, a menos, claro, que el proxy no funciones.

Se puede configurar Squid para que trabaje de esta manera, y a esto se le llamó redirección o proxy transparente en anteriores versiones de Linux.

3. Los dos tipos de NAT

Yo divido NAT en dos diferentes tipos: **Source NAT** (SNAT, por origen), y **Destination NAT** (DNAT, por destino).

Source NAT es cuando alteramos el origen del primer paquete: esto es, estamos cambiando el lugar de donde viene la conexión. *Source NAT* siempre se hace después del encaminamiento, justo antes de que el paquete salga por el cable. El enmascaramiento es una forma especializada de SNAT.

Destination NAT es cuando alteramos la dirección de destino del primer paquete: esto es, cambiamos la dirección a donde se dirige la conexión. DNAT siempre se hace antes del encaminamiento, cuando el paquete entra por el cable. El port forwarding (reenvío de puerto), el balanceo de carga y el proxy transparente son formas de DNAT.

4. Puesta al día rápida con respecto a los núcleos 2.0 y 2.2

Lo siento por aquellos que todavía estén aturdidos por la transición desde 2.0 (ipfwadm) a 2.2 (ipchains). Hay buenas y malas noticias.

Primero, puede seguir usando ipchains o ipfwadm como antes. Para hacerlo, necesita cargar los módulos del núcleo «ipchains.o» o «ipfwadm.o» que encontrará en la última distribución de netfilter. Son mutuamente exclusivos (está advertido), y no deberían combinarse con ningún otro módulo de netfilter.

Una vez haya instalado uno de estos módulos puede utilizar ipchains e ipfwadm con normalidad, excepto por las siguientes diferencias:

- Establecer los tiempos límite (timeout) con ipchains -M -S o ipfwadm -M -s no hace nada. Como los límites de tiempo con la nueva infrastructura NAT son más grandes, no debería haber problema.
- Los campos init_seq, delta y previous_delta en la lista ampliada de enmascaramiento (verbose masquerade listing) siempre son 0.
- Listar los contadores y ponerlos a cero al mismo tiempo «-Z -L» ya no funciona: los contadores no se pondrán a cero.

Los hackers también se darán cuenta de que:

- Ahora puede asociar un programa (bind) a los puertos 61000-65095 incluso si está haciendo enmascaramiento. El código de enmascarado asumía que no había nada en este rango, de manera que los programas no lo podían usar.
- El parche (no documentado) de «getsockname», que podían utilizar los programas de proxy transparente para averiguar el destino real de la conexión no funciona.
- El parche (no documentado) bind-to-foreign-address (asociado-a-una-dirección-externa) tampoco está implementado; se usaba para completar la ilusión del proxy transparente.

4.1. ¡Sólo quiero enmascarar! ¡Ayuda!

Esto es lo que la mayoría de la gente quiere. Si tengo una conexión PPP con IP dinámica (si no sabe lo que es, entonces tiene una), simplemente querré decirle a mi máquina que todos los paquetes que vengan de la red interna deberían aparentar venir de la máquina que tiene el enlace PPP.

```
# Cargue el módulo NAT (esto carga también los otros).
modprobe iptable_nat

# Agrega (-A) una regla a la tabla NAT (-t nat), después del
# encaminamiento (POSTROUTING) para todos los paquetes que salgan por
# ppp0 (-o ppp0) enmascarando la conexión (-j MASQUERADE).
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE

# Ponga en marcha el reenvío de IP (IP forwarding)
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Fíjese que no está haciendo filtrado de paquetes: para eso, lea el COMO de Filtrado de Paquetes: «Mezclando NAT con Filtrado de Paquetes».

5. ¿Qué pasa con ipmasqadm?

Este programa tiene un nicho de usuarios mucho más definido, de manera que no me he preocupado mucho de darle compatibilidad retroactiva. Puede usar «iptables -t nat» para hacer reenvío de puertos. De manera que, por ejemplo, en Linux 2.2 podría haber hecho:

```
#Linux 2.2
#Reenvía los paquetes TCP dirigidos al puerto 8080 de 1.2.3.4 al 80 de
#192.168.1.1
ipmasqadm portfw -a -P tcp -L 1.2.3.4 8080 -R 192.168.1.1 80
```

Ahora debería hacer:

```
# Linux 2.4
# Agrega una regla previa al encaminamiento (-A PREROUTING) a la tabla NAT
# (-t nat) de manera que los paquetes TCP (-p tcp) que vayan a 1.2.3.4
# (-d 1.2.3.4), puerto 8080 (--dport 8080) tengan una correspondencia de
# destino (-j DNAT) con 192.168.1.1, puerto 80 (--to 192.168.1.1:80).
iptables -A PREROUTING -t nat -p tcp -d 1.2.3.4 --dport 8080 \
        -j DNAT --to 192.168.1.1:80
```

Si desea que esta regla altere también las conexiones locales (aquellas que se originen en la propia máquina que hace NAT), puede insertar la misma regla en la cadena OUTPUT (que es para los paquetes locales de salida):

```
# Linux 2.4
iptables -A OUTPUT -t nat -p tcp -d 1.2.3.4 --dport 8080 \
        -j DNAT --to 192.168.1.1:80
```

6. Controlar qué cosas pasar por NAT

Necesita crear reglas NAT que le digan al núcleo qué conexiones cambiar, y cómo hacerlo. Para ello, usaremos la muy versátil herramienta `iptables`, y le diremos que altere la tabla de NAT usando la opción «-t nat».

La tabla de reglas NAT contiene tres listas llamadas «cadenas»: cada regla se examina por orden hasta que una coincide. Las tres cadenas se llaman PREROUTING (para *Destination NAT*, según los paquetes entran), POSTROUTING (para SOURCE NAT, según los paquetes salen), y OUTPUT (para *Destination NAT* con los paquetes generados en la propia máquina).

El siguiente diagrama lo ilustraría bastante bien si yo tuviese algo de talento artístico:

```
      _____                                     _____
     /     \                                   /     \
   PREROUTING -->[Decisión de   ]----------------->POSTROUTING----->
     \D-NAT/     [Encaminamiento]                  \S-NAT/
                     |                                ^
                     |                              __|__
                     |                             /     \
                     |                            | OUTPUT|
                     |                             \D-NAT/
                     |                                ^
                     |                                |
                     ----------> Proceso Local ----------
```

En cada uno de los puntos anteriores, cuando un paquete pasa miramos la conexión a la que está asociado. Si es una conexión nueva, comprobamos la cadena correspondiente en la tabla de NAT para ver qué hacer con ella. La respuesta que obtenemos se aplicará a cualquier paquete posterior de esa conexión.

6.1. Selección sencilla usando iptables

`iptables` toma cierto número de decisiones estándar que se listarán ahora. Todas las opciones con doble guión pueden ser abreviadas, siempre que `iptables` pueda distinguirlas de otras opciones posibles. Si el núcleo tiene la implementación de iptables como módulo, necesitará cargar el módulo ip_tables.o antes: «insmod ip_tables».

La opción más importante aquí es la opción de selección de tabla, «-t». Para todas las operaciones de NAT, querrá usar «-t nat» para la tabla NAT. La segunda más importante es «-A» para añadir una nueva regla al final de una cadena («-A POSTROUTING»), o «-I» para insertarla al principio («-I PREROUTING»).

Puede especificar el origen («-s» o «–source») y el destino («-d» o «–destination») de los paquetes sobre los que quiere hacer NAT. Estas opciones pueden ir seguidas por una IP sencilla (192.168.1.1), un nombre (www.gnumonks.org), o una dirección de red (192.168.1.0/24 o 192.168.1.0/255.255.255.0).

Puede especificar qué interfaz de entrada («-i» o «–in-interface») o de salida («-o» o «–out-interface») mirar, pero lo que puede especificar depende de en qué cadena esté poniendo la regla: en PREROUTING sólo puede elegir la interfaz de entrada, y en POSTROUTING (y OUTPUT) sólo la de salida. Si usa la equivocada, `iptables` le avisará con un mensaje de error.

6.2. Opciones más refinadas de selección de paquetes a toquetear.

Dije antes que se puede especificar una dirección de origen y destino. Si omite la opción de origen, entonces será cualquier dirección de origen. Si omite la de destino, será cualquier dirección de destino.

También puede indicar un protocolo específico («-p» o «–protocol»), como TCP o UDP; sólo los paquetes de este protocolo coincidirán con la regla. La razón principal para hacer esto es que especificar uno de los protocolos tcp o udp permite más opciones: específicamente las opciones «–source-port» y «–destination-port» (abreviadas «–sport» y «–dport»).

Estas opciones le permiten especificar que sólo los paquetes con un determinado origen y destino coincidirán con la regla. Esto es útil para redireccionar peticiones web (puertos TCP 80 u 8080) y dejar los demás paquetes tranquilos.

Estas opciones deben seguir a la «-p» (que tiene el efecto secundario de cargar la biblioteca compartida de extensión para ese protocolo). Puede usar números de puerto, o un nombre de fichero `/etc/services`.

Todos los diferentes parámetros por los que se puede seleccionar un paquete vienen enumerados con toda clase de dolorosos detalles en la página de manual (`man iptables`).

7. Cómo modificar los paquetes

De manera que ahora sabemos cómo elegir los paquetes que queremos modificar. Para completar nuestra regla, necesitamos decirle al núcleo exactamente qué queremos que haga con los paquetes.

7.1. Source NAT (Cambio de Origen)

Quiere hacer *Source NAT*; cambiar la dirección de origen de las conexiones a algo diferente. Esto se hace en la cadena POSTROUTING, justo antes de que sea enviado. Este es un detalle importante, ya que significa que cualquier otro servicio de la máquina Linux (encaminamiento, filtrado de paquetes) verá el paquete sin cambiar. También significa que se puede utilizar la opción «-o» (interfaz de salida).

El *Source NAT* se especifica indicando «-j SNAT», y la opción «–to-source» especifica una dirección IP, un rango de direcciones IP, y un puerto o rango de puertos opcionales (sólo con los protocolos UDP y TCP).

```
## Cambiar la dirección de origen por 1.2.3.4
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 1.2.3.4
```

```
## Cambiar la dirección de origen a 1.2.3.4, 1.2.3.5 o 1.2.3.6
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 1.2.3.4-1.2.3.6

## Cambiar la dirección de origen por 1.2.3.4, puertos 1-1023
# iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to 1.2.3.4:1-1023
```

7.1.1. Enmascaramiento

Hay un caso especializado de *Source NAT* denominado «enmascaramiento» (masquerading): sólo debería ser usado para direcciones IP asignadas de forma dinámica, tales como las de conexiones por llamada estándar (para direcciones IP estáticas, utilize el SNAT descrito anteriormente).

No es necesario escribir la dirección de origen de forma explícita con el enmascaramiento: utilizará la dirección de origen de la interfaz por la que el paquete está saliendo. Pero más importante aún, si el enlace cae, las conexiones (que se iban a perder de todas maneras) se olvidan, lo que significa que habrá menos follón cuando la conexión vuelva a la normalidad con una IP diferente.

```
## Enmascarar todo lo que salga por ppp0.
# iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

7.2. Destination NAT (Cambio de destino)

Esto se hace en la cadena PREROUTING, según entra el paquete; esto significa que cualquier otro servicio de la máquina con Linux (encaminamiento, filtrado de paquetes) verá el paquete yendo a su destino «real» (el definitivo). Esto significa que se puede utilizar la opción «-i» (interfaz de entrada).

Para alterar el destino de un paquete generado de forma local (en la máquina que hace el NAT), se debe usar la cadena OUTPUT, pero esto es más inusual.

Destination NAT se especifica utilizando «-j DNAT», y la opción «–to-destination» especifica una dirección IP, un rango de direcciones IP, y un puerto o rango de puertos opcionales (sólo para los protocolos UDP y TCP).

```
## Cambia la dirección de destino por 5.6.7.8
# iptables -t nat -A PREROUTING -i eth1 -j DNAT --to 5.6.7.8

## Cambia la dirección de destino por 5.6.7.8, 5.6.7.9 o 5.6.7.10.
# iptables -t nat -A PREROUTING -i eth1 -j DNAT --to 5.6.7.8-5.6.7.10

## Cambia la dirección de destino del tráfico web por 5.6.7.8,
## puerto 8080.
# iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth1 \
        -j DNAT --to 5.6.7.8:8080

## Redirige los paquetes locales que van a 1.2.3.4 hacia el dispositivo
## loopback.
# iptables -t nat -A OUTPUT -d 1.2.3.4 -j DNAT --to 127.0.0.1
```

7.2.1. Redirección

Hay un caso especializado de *Destination NAT* llamado redirección: es una simple conveniencia que es exactamente lo mismo que hacer DNAT, pero con la dirección de la interfaz de entrada.

```
## Envía el tráfico que entra dirigido al puerto 80 (web) a nuestro
## proxy squid (transparente)
# iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 \
        -j REDIRECT --to-port 3128
```

7.3. Correspondencias (mappings) en profundidad

Hay algunas sutilezas de NAT con las que la mayoría de la gente no tendrá que enfrentarse nunca. Las he documentado aquí para los más curiosos.

7.3.1. Selección de múltiples direcciones de un rango dado

Si se da un rango de direcciones IP, la dirección a usar se elegirá basándose en la menos utilizada para las conexiones de las que sabe nuestra máquina. Esto nos permite hacer un balanceo de carga primitivo.

7.3.2. Crear correspondencias NAT nulas

Puede utilizar el objetivo «-j ACCEPT» para dejar que una conexión pase sin que se produzca ninguna conversión NAT.

7.3.3. Comportamiento NAT estándar

El comportamiento estándar es alterar la conexión lo menos posible, dentro de los límites de la regla dada por el usuario. Esto significa que no cambiaremos el puerto a menos que nos veamos obligados.

7.3.4. Correspondencia implícita del puerto de origen

Incluso cuando no se pide NAT para una conexión, puede ser que haya un cambio de puerto de forma implícita, si otra conexión lo está usando ya. Consideremos el caso del enmascaramiento, que es lo más común:

1. Se establece una conexión web entre la máquina 192.1.1.1, puerto 1024 y www.netscape.com, puerto 80.
2. Esta conexión es enmascarada por otra máquina para que utilice su dirección como origen (1.2.3.4).
3. La máquina enmascaradora intenta hacer la conexión web con www.netscape.com puerto 80, desde 1.2.3.4 (la dirección de su interfaz interna), puerto 1024.
4. El código de NAT alterará la dirección de origen de una segunda conexión al puerto 1025, de manera que no pueda haber solapado.

Cuando se produce este cambio implícito de origen, los puertos se dividen en tres clases:

- Puertos por debajo del 512.
- Puertos entre el 512 y el 1023 (inclusive).
- Puertos 1024 y superiores.

Nunca se hará corresponder un puerto con otro de clase diferente.

7.3.5. Qué sucede cuando NAT falla

Si no hay manera de hacer corresponder una conexión de forma única tal como ha pedido el usuario, será descartada. Esto se aplica también a los paquetes que no pueden ser clasificados como parte de una conexión, porque están malformados, o nos hemos quedado sin memoria, etc.

7.3.6. Múltiples correspondencias (mappings), solapado y colisiones

Podemos tener reglas NAT que asignen paquetes dentro del mismo rango. El código NAT es suficientemente inteligente para evitar colisiones. Por tanto, tener dos reglas que hagan corresponder las direcciones de origen 192.168.1.1 y 192.168.1.2 respectivamente con 1.2.3.4 es correcto.

Aún más, se pueden hacer correspondencias sobre direcciones reales y en uso, siempre y cuándo estas direcciones pasen también a través de nuestra máquina. De manera que si tenemos una red asignada (1.2.3.0/24), pero tenemos una red interna que está usando esas direcciones y otra con Direcciones Privadas de Internet 192.168.1.0./24, podemos hacer NAT con las direcciones de origen 192.168.1.0/24 dentro del rango de la red 1.2.3.0, sin temor a colisiones:

```
# iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth1 \
        -j SNAT --to 1.2.3.0/24
```

La misma lógica se aplica a las direcciones que usa la propia caja que hace NAT: ésta es la manera en que funciona el enmascaramiento (compartiendo las direcciones de la interfaz entre los paquetes enmascarados y los paquetes «reales» que vienen de la propia máquina).

Aún más, podemos direccionar los mismos paquetes a varios objetivos diferentes, y serán compartidos. Por ejemplo, si no queremos direccionar nada usando 1.2.3.5, podríamos hacer:

```
# iptables -t nat -A POSTROUTING -s 192.168.1.0/24 -o eth1 \
        -j SNAT --to 1.2.3.0-1.2.3.4 --to 1.2.3.6-1.2.3.254
```

7.3.7. Alterar el destino de conexiones generadas de forma local

Si se cambia el destino de un paquete generado de forma local (esto es, usando la cadena OUTPUT), y eso hace que el paquete salte a una interfaz diferente, entonces se cambia también la dirección de origen a la de esa interfaz. Por ejemplo, cambiar el destino de un paquete de loopback para que salga por eth0 tendrá como resultado que también se cambie la dirección de origen, 127.0.0.1, por la de eth0. Al contrario que con el resto de cambios, esto se hace de forma inmediata. Naturalmente, ambos cambios son invertidos cuando llega la respuesta del paquete.

8. Protocolos especiales

A algunos protocolos no les gusta pasar por NAT. Se deben escribir dos extensiones para cada uno de ellos; uno para el seguimiento de las conexiones de ese protocolo, y otro para el propio NAT.

Junto con la distribución de netfilter, hay módulos para el ftp: ip_conntrack_ftp.o e ip_nat_ftp.o. Si inserta estos módulos en el núcleo (o los compila de forma permanente), entonces podrá hacer NAT sobre conexiones FTP. Si no lo hace, entonces sólo podrá utilizar FTP pasivo, e incluso eso no será del todo fiable si hace algo más que un sencillo *Source Nat*.

9. Defectos del NAT

Si hace NAT sobre una conexión, todos los paquetes que pasen en **ambas** direcciones (dentro y fuera de la red) deberán hacerlo a través de la máquina que hace NAT, ya que si no, no funcionará de forma fiable. En particular, el código de seguimiento de conexiones ensambla los fragmentos, lo que significa que no sólo la conexión no será fiable, sino que los paquetes pueden terminar por no llegar, ya que los fragmentos quedarán retenidos.

10. Agradecimientos

Gracias antes que nada a WatchGuard, y a David Bonn, quien creyó en la idea de netfilter lo suficiente para darme soporte mientras trabajaba en ello.

Y al resto del mundo que soportó mis cabreos mientras aprendía sobre los horrores de NAT, especialmente a aquellos que leyeron mi diario.

Rusty.