Linux 2.4 NAT HOWTO

Rusty Russell, Mailingliste netfilter@lists.samba.org, ins Deutsche uebersetzt von Melanie Berg mel@sekurity.de

v1.0.1 Mon May 1 18:38:22 CST 2000

Dieses Dokument beschreibt, wie man Masquerading, transparente Proxies, Port Forwarding und andere Formen der Network Address Translation mit dem 2.4er Linuxkernel einsetzen kann. Diese deutsche Uebersetzung steht unter den Bedingungen der GNU General Public License (http://www.gnu.org/copyleft/gpl.html)

Contents

1 Einleitung

Willkommen, geschaetzter Leser,

Du bist dabei, in die faszinierende (und manchmal schreckliche) Welt der NAT einzutauchen: Network Address Translation, und dieses HOWTO wird etwas wie Dein Fuehrer zum 2.4er Kernel und weiter sein.

Mit Linux 2.4 wurde eine Infrastruktur fuer das Untersuchen von Paketen, genannt 'netfilter', eingefuehrt. Eine darauf aufbauende Schicht bietet NAT, komplett von den vorangegangenen Kerneln neu implementiert.

2 Wo ist die offizielle Website und Mailingliste?

Es gibt drei offizielle Seiten:

- Dank an *Penguin Computing* `http://netfilter.filewatcher.org/` .
- Dank an *The Samba Team and SGI* `http:/netfilter.samba.org/` .
- Dank an *Harald Welte* `http://netfilter.gnumonks.org/` .

Fuer die offizielle Netfilter-Mailingliste siehe Sambas Listserver

Netfilter List `http://www.netfilter.org/contact.html#list` .

2.1 Was ist Network Address Translation?

Gewoehnlich reisen Pakete in einem Netzwerk von ihrer Quelle (z.B. Dein Computer) zu ihrem Ziel (z.B. www.gnumonks.org) durch viele verschiedene Links: ungefaehr 19 von da, wo ich in Australien bin. Keiner dieser Links veraendert das Paket wirklich, sie schicken es einfach weiter.

Wenn einer dieser Links NAT machen wuerde, dann wuerde er die Quelle oder das Ziel des Paket veraendern, wenn es eintrifft. Wie Du Dir vorstellen kannst, wurde das System nicht entworfen, so zu arbeiten, also ist NAT immer etwas, was man mit Vorsicht behandeln sollte. Gewoehnlich wird sich der Link, der NAT macht, daran erinnern, wie er das Paket veraendert hat, und wenn ein Antwortpaket aus der anderen Richtung kommt, wird er genau das Umgekehrte darauf anwenden, und so funktioniert es.

2.2 Warum sollte ich NAT wollen?

In einer perfekten Welt wuerdest Du das gar nicht. In der Zwischenzeit sind hier die Gruende:

Modemverbindungen zum Internet

Die meisten Internetanbieter geben Dir eine einzelne Adresse, wenn Du Dich bei ihnen einwaehlst. Du kannst Pakete mit welcher Quelladresse auch immer verschicken, aber nur Pakete mit dieser Antwortadresse werden zu Dir zurueckkommen. Wenn Du mehrere verschiedene Maschinen (so wie ein Heim-Netzwerk) benutzen willst, um Dich durch diesen Link mit dem Internet zu verbinden, wirst Du NAT brauchen.

Dies ist die heute am meisten verbreitete Art von NAT, gewoehnlich in der Linuxwelt als 'Masquerading' bekannt. Ich nenne dies SNAT, weil die **Quell** ('source') Adresse des ersten Pakets veraendert wird.

Mehrere Server

Manchmal moechtest Du aendern, wohin einkommende Pakete in Deinem Netzwerk gehen sollen. Oft ist das so, weil Du (wie oben erwaehnt) nur eine IP-Adresse hast, Du moechtest den Leuten aber die Moeglichkeit geben, auch die Rechner hinter dem einen mit der 'echten' IP-Adresse zu erreichen. Du kannst das schaffen, wenn Du das Ziel von einkommenden Paketen aendern kannst.

Eine bekannte Variation dessen nennt sich 'load-sharing': Eine grosse Anzahl von Paketen wird ueber eine Reihe von Maschinen veraendert, indem die Pakete 'aufgefaechert' werden. Diese Version von NAT wurde unter frueheren Linuxversionen Port-Forwarding genannt.

Transparente Proxies

Manchmal moechtest Du so tun, also ob jedes Paket, das durch Deinen Linuxrechner geht, fuer ein Programm auf dem Linuxrechner selbst bestimmt ist. Dies wird fuer transparente Proxies verwendet: ein Proxy ist ein Programm, das zwischen Deinem Netzwerk und der Aussen- welt steht und die Kommunikation dazwischen regelt. Der transparente Teil kommt daher, weil Dein Netzwerk nicht einmal weiss, dass es mit mit einem Proxy redet, es sei denn natuerlich, der Proxy funktioniert nicht.

Squid kann auf diese Art konfiguriert werden, unter frueherern Linux- versionen hiess das Umleiten (redirection) oder auch transparentes Proxying.

3 Die zwei Formen von NAT

Ich unterscheide NAT in zwei verschiedene Typen: **Source Nat** (SNAT) und **Destination** NAT (DNAT).

Wenn Du die Quelladresse des ersten Pakets aenderst, ist das Source NAT: Du veraenderst den Ursprung der Verbindung. Source NAT ist immer Post- Routing, es wirkt, gerade bevor das Paket in die Leitung geht. Masquerading ist eine spezielle Form von SNAT.

Wenn Du die Zieladresse des ersten Pakets anderst, ist das Destination NAT: Du veraenderst das Ziel, wohin die Verbindung geht. Destination NAT ist immer Pre-Routing, gerade wenn das Paket aus der Leitung kommt. Port-Forwarding, load-sharing und transparente Proxies sind alles Formen von DNAT.

4 Schnelle Uebersetzung vom 2.0er und 2.2er Kernel

Sorry an alle von Euch, die noch immer geschockt sind vom Uebergang von 2.0 (ipfwadm) auf 2.2 (ipchains). Es gibt gute und schlechte Neuigkeiten.

Zuerst einmal kannst Du ipfwadm und ipchains wie gewohnt weiterbenutzen. Um das zu tun, musst Du das 'ipchains.o' oder 'ipfwadm.o' Kernelmodul aus der letzten netfilter-Distribution laden (insmod). Diese beiden schliessen sich gegenseitig aus (Du bist gewarnt) und sollten nicht mit anderen netfilter-Modulen kombiniert werden.

Sobald eins dieser Module installiert ist, kannst Du ipchains und ipfwadm wie gewohnt benutzen, mit den folgenden Unterschieden:

- Das Masquerading Timeout mit ipchains -M -S, oder mit ipfwadm -M -S, zu setzen, bringt nichts. Da die neuen Timeouts der neuen NAT-Infrastruktur laenger sind, sollte das aber egal sein.
- Die init_seq, delta und previous_delta Felder in der ausfuehrlichen Masqueradingliste sind immer Null.
- Gleichzeitig die Zaehler auflisten und auf Null setzen (-Z -L) funktioniert nicht mehr: Die Zaehler werden nicht zurueckgesetzt.

Fuer Hacker:

- Du kannst jetzt auch Ports von 61000-65095 einbinden, sogar wenn Du Masquerading machst. Der Masquerading Code hatte frueher angenommen, dass alles im diesem Bereich freigehalten werden sollte, so dass Programme ihn nicht nutzen konnten.
- Der (undokumentierte) 'getsockname' Hack, welchen man nutzen konnte, um bei transparenten Proxies das wirkliche Ziel herauszufinden, funktioniert nicht mehr.
- Der (undokumentierte) 'bind-to-foreign-address' Hack ist auch nicht implementiert; dies wurde verwendet, um die Illusion von transparenten Proxies komplett zu machen.

4.1 Ich will nur Masquerading! Hilfe!

Das ist das, was die meisten Leute wollen. Wenn Du durch eine PPP-Verbindung eine dynamische IP-Adresse hast (wenn Du das nicht weisst, dann hast Du eine), moechtest Du Deinem Rechner einfach sagen, dass alle Pakete, die aus Deinem internen Netzwerk kommen, so aussehen sollen, als ob sie von dem Rechner mit der PPP-Verbindung kommen wuerden.

```
# Das NAT-Modul laden (dies zieht all die andern mit).
modprobe iptable_nat

# In der NAT-Tabelle (-t nat) eine Regel fuer alle an ppp0 (-o ppp0)
# ausgehenden Pakete hinter dem Routing (POSTROUTING), die maskiert
# werden sollen, anhaengen (-A).
iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE

# IP-Forwarding aktivieren
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Beachte, dass Du hier keine Pakete filterst: hierzu lese das Packet- Filtering-HOWTO: Kombinieren von NAT und Paketfiltern.

4.2 Was ist mit ipmasqadm?

Das ist eine verzwicktere Sache, und ich habe mir hier keine grossen Sorgen um die Rueckwaerts-Kompatibilitaet gemacht. Um Port-Forwarding zu verwenden, kannst Du einfach 'iptables -t nat' benutzen. Unter Linux 2.2 haettest Du es zum Beispiel so machen koennen:

```
# Linux 2.2
# TCP-Pakete, die an 1.2.3.4 Port 8080 gehen, an 192.168.1.1 Port 80
# weiterleiten
ipmasqadm portfw -a -P tcp -L 1.2.3.4 8080 -R 192.168.1.1 80
```

Jetzt wuerdest Du folgendes tun:

```
# Linux 2.4
# Eine Pre-Routing (PREROUTING) Regel an die NAT-Tabelle (-t nat) an-
# haengen (-A), die besagt, dass alle TCP-Pakete (-p tcp) fuer 1.2.3.4
# (-d 1.2.3.4) Port 8080 (--dport) auf 192.168.1.1:80
# (--to 192.168.1.1:80) gemappt werden (-j DNAT).
iptables -A PREROUTING -t nat -p tcp -d 1.2.3.4 --dport 8080 \
-j DNAT --to 192.168.1.1:80
```

Wenn Du willst, dass diese Regel auch lokale Verbindung veraendert (ich meine, wenn sogar auf dem NAT-Rechner selbst ein Telnet auf 1.2.3.4 Port 8080 an 192.168.1.1 Port 80 geleitet wird), kannst Du diese Regel in die OUTPUT-Kette (fuer lokal ausgehende Pakete) einfuegen:

```
# 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
```

5 Kontrollieren, worauf man NAT anwendet

Du musst NAT-Regeln erstellen, die dem Kernel sagen, was fuer Verbindungen er aendern soll, und wie er sie aendern soll. Um das zu tun, setzen wir das vielseitige `iptables` Tool ein und sagen ihm durch das Angeben der '-t nat' Option, dass es die NAT-Tabelle aendern soll.

Die Tabelle der NAT-Regeln enthaelt drei Listen, die 'Ketten' genannt werden: Alle Regeln werden der Reihe nach untersucht, bis eine davon zutrifft. Die drei Ketten heissen PREROUTING (fuer Destination NAT, da die Pakete hereinkommen), POSTROUTING (fuer Source NAT, da die Pakete ausgehen) und OUTPUT (fuer Destination NAT von lokal generierten Paketen).

Wenn ich irgendein kuenstlerisches Talent haette, wuerde dieses Diagramm es ganz gut zeigen:

```
 -----                                     -----
/     \                                   /     \
PREROUTING -->[Routing ]----------------->POSTROUTING----->
```

```
\D-NAT/     [Entscheidung]               \S-NAT/
                  |                          ^
                  |                        __|__
                  |                       /     \
                  |                      | OUTPUT|
                  |                       \D-NAT/
                  |                          ^
                  |                          |
                  -------->Lokaler Prozess------
```

Wenn ein Paket durchgeht, schauen wir an jedem der obigen Punkte nach, zu was fuer einer Verbindung es gehoert. Wenn es eine neue Verbindung ist, sehen wir in der entsprechenden Kette der NAT-Tabelle nach, was zu tun ist. Die Antwort, die wir erhalten, wird auf alle weiteren Pakete dieser Verbindung angewendet.

5.1 Einfache Auswahl mit iptables

`iptables` benoetigt eine Reihe von Standardoptionen, die weiter unten aufgelistet werden. Die Optionen mit einem doppelten Gedankenstrich koennen abgekuerzt werden, solange `iptables` sie danach noch von den anderen Optionen unterscheiden kann. Wenn Dein Kernel `iptables` als Modul unterstuetzt, wirst Du das 'iptables.o' Modul zuerst laden muessen: 'insmod iptables.o'.

Die wichtigste Option ist hier die, mit der man die Tabelle auswaehlen kann, '-t'. Fuer alle NAT Operationen wirst Du '-t nat' verwenden wollen, um in die NAT-Tabelle zu schreiben. Die zweitwichtigste Option ist das '-A', mit dem man eine neue Regel an das Ende einer Kette anhaengen kann (z.B. '-A POSTROUTING'), oder '-I', um eine Regel am Anfang einer Kette einzufuegen (z.B. '-I PREROUTING').

Du kannst die Quelle ('-s' oder '–source') und das Ziel ('-d' oder ('–destination') eines Pakets bestimmen, auf das Du NAT anwenden willst. Diesen Angaben kann eine einzelne IP-Adresse (z.B. 192.168.1.1), ein Name (z.B. www.gnumonks.org) oder ein Netzwerkadresse (z.B. 192.168.1.0/24 oder 192.168.1.0/255.255.255.0) folgen.

Du kannst die Schnittstelle bestimmen, an der Pakete eingehen ('-i' oder '–in-interface') oder ausgehen ('-o' oder '–out-interface'), aber welche von beiden haengt davon ab, in welche Kette Du diese Regel einfuegst: Bei der PREROUTING-Kette kannst Du nur die eingehende Schnittstelle waehlen, und bei der POSTROUTING-Schnittstelle (OUTPUT) nur die ausgehende. Wenn Du die falsche waehlst, wird `iptables` Dir eine Fehlermeldung geben.

5.2 Genauere Auswahl der betreffenden Pakete

Ich habe weiter oben gesagt, dass Du eine Quell- und eine Zieladresse bestimmen kannst. Wenn Du die Quelladresse weglaesst, wird jegliche Adresse zutreffend sein. Wenn Du die Zieladresse weglaesst, wird jegliche Zieladresse zutreffend sein.

Du kannst auch ein bestimmtes Protokoll ('-p' oder '–protocol') angeben, so wie TCP oder UDP; nur auf Pakete dieses Typs wird die Regel zutreffen. Der Hauptgrund hierfuer besteht darin, dass das Bestimmen eines Protokolls Extra-Optionen erlaubt: insbesondere die '–source-port' und die '–destination-port' Optionen (abgekuerzt als '-sport' und '-dport').

Diese Optionen erlauben Dir, zu bestimmen, dass eine Regel nur auf Pakete mit einem bestimmten Quell- oder Zielport zutrifft. Dies ist nuetzlich fuer umgeleitete Web-Anfragen (TCP-Port 80 und 8080) und laesst andere Pakete ausser Acht.

Diese Optionen muessen der '-p' Option folgen (welche den Nebeneffekt hat, dass die Erweiterungen fuer die shared libraries fuer das ent- sprechende Protokoll geladen werden). Du kannst Portnummern verwenden oder Namen aus der /etc/services Datei.

All die verschiedenen Eigenschaften, nach denen Du Pakete auswaehlen kannst, werden in schmerzhaften Einzelheiten detailliert in der Man-Page beschrieben (`man iptables`).

6 Wie die Pakete veraendert Werden sollen

Jetzt wissen wir also, wie wir die Pakete, die wir veraendern wollen, auswaehlen koennen. Um unsere Regel zu vervollstaendigen, muessen wir dem Kernel sagen, was genau er mit dem Paket tun soll.

6.1 Source NAT

Du moechtest Source NAT machen; veraendere die Quelladresse von Paketen zu etwas anderem. Dies wird in der POSTROUTING-Kette gemacht, kurz bevor das Paket schliesslich geschickt wird; dies ist ein wichtiges Detail, da es bedeutet, dass alles andere auf dem Linuxrechner selbst (Routing, Paketfilter) das unveraenderte Paket sehen wird. Es bedeutet auch, dass die '-o' (ausgehende Schnittstelle) Option verwendet werden kann.

Source NAT wird durch '-j SNAT' bestimmt und die '–to-source' Option bestimmt eine IP-Adresse, eine Reihe von IP-Adressen, und einen optionalen Port oder eine Reihe von Ports (nur fuer UDP und TCP Protokolle).

```
## Quelladresse auf 1.2.3.4 aendern
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 1.2.3.4

## Quelladresse auf 1.2.3.4, 1.2.3.5, oder 1.2.3.5 aendern
# iptables -t nat -A POSTROUTING -o eth0 -j SNAT --to 1.2.3.4-1.2.3.6

## Quelladresse zu 1.2.3.4, Ports 1 - 1023, aendern
# iptables -t nat -A POSTROUTING -p tcp -o eth0 -j SNAT --to \
# 1.2.3.4:1-1023
```

6.1.1 Masquerading

Es gibt einen Spezialfall von Source NAT, der Masquerading genannt wird: es sollte nur fuer dynamisch zugeordnete IP-Adressen verwendet werden wie bei normalen Waehlverbindungen (Benutze bei statischen IP-Adressen SNAT weiter oben).

Beim Masquerading musst Du die Quelladresse nicht explizit angeben: es wird die Quelladresse der Schnittstelle nehmen, an der das Paket ausgeht. Wichtiger ist, dass, wenn der Link unterbrochen wird, die Verbindungen (die jetzt sowieso verloren sind) vergessen werden, was weniger Stoerungen bedeutet, wenn die Verbindung mit einer neuen IP-Adresse wieder aufgebaut wird.

```
## Maskiere alles, was an ppp0 ausgeht
# iptables -t nat -A POSTROUTING -o ppp0 -j MASQUERADE
```

6.2 Destination NAT

Dies wird in der PREROUTING-Kette erledigt, wenn das Paket gerade eingegangen ist; das bedeutet, dass alles andere auf dem Linuxrechner selbst (Routing, Paketfilter) das Paket zum 'wirklichen' Ziel gehen sehen wird. Es bedeutet auch, dass die '-i' Option (eingehende Schnittstelle) verwendet werden kann.

Um das Ziel von lokal generierten Paketen zu aendern, kann auch die OUTPUT-Kette benutzt werden, das ist aber eher ungewoehnlich.

Destination NAT wird durch '-j DNAT' bestimmt und die '–to-destination' Option bestimmt eine IP-Adresse, eine Reihe von IP-Adressen, und einen optionalen Port oder eine Reihe von Ports (nur fuer UDP und TCP Protokolle).

```
## Zieladresse zu 5.6.7.8 aendern
# iptables -t nat -A PREROUTING -i eth1 -j DNAT --to 5.6.7.8

## Zieladresse zu 5.6.7.8, 5.6.7.9 oder 5.6.7.10 aendern
# iptables -t nat -A PREROUTING -i eth1 -j DNAT --to 5.6.7.8-5.6.7.10

## Aendern der Zieladresse von Webtraffic auf 5.6.7.8 Port 8080
# iptables -t nat -A PREROUTING -p tcp --dport 80 -i eth1 \
  -j DNAT --to 5.6.7.8:8080

## Lokale Pakete fuer 1.2.3.4 an das Loopback umleiten
# iptables -t nat -A OUTPUT -d 1.2.3.4 -j DNAT --to 127.0.0.1
```

6.2.1 Umadressierung (Redirection)

Es gibt einen speziellen Fall von Destination NAT, der Redirection genannt wird: Es ist eine einfache Bequemlichkeit, die genau das gleiche tut wie NAT auf der eingehenden Schnittstelle.

```
## Eingehenden Webtraffic an Port 80 an unseren (transparenten) Squid-
#  Proxy weiterleiten
# iptables -t nat -A PREROUTING -i eth1 -p tcp --dport 80 \
-j REDIRECT --to-port 3128
```

6.3 Mappings genauer betrachtet

Es ist gibt ein paar subtile Einzelheiten bei NAT, um die sich die meisten Leute nie werden kuemmern muessen. Fuer die Neugierigen sind sie hier dokumentiert.

6.3.1 Auswahl von mehrere Adressen in einer Reihe

Wenn eine Reihe von IP-Adressen gegeben ist, wir diejenige ausgewaehlt, die im Moment am wenigsten fuer IP-Verbindungen, von denen die Maschine weiss, benutzt wird. Dies macht primitives 'load-balancing' moeglich.

6.3.2 Ein Null NAT Mapping erstellen

Du kannst das '-j ACCEPT' Ziel verwenden, um eine Verbindung zuzulassen, ohne dass irgendein NAT stattfindet.

6.3.3 Standard NAT-Verhalten

Gewoehnlich veraendert man eine Verbindung so wenig wie moeglich, entsprechend den Vorgaben einer durch den Benutzer gegebenen Regel. Das bedeutet, dass wir Ports nicht 're-mappen' werden, solange wir es nicht unbedingt tun muessen.

6.3.4 Implizites Quellport-Mappen

Sogar, wenn fuer eine Verbindung kein NAT benoetigt wird, kann Quellport- veraenderung stillschweigend auftreten, wenn eine andere Verbindung ueber die neue gemappt wurde. Stell Dir den Fall von Masquerading vor, der recht gewoehnlich ist:

1. Eine Webverbindung von einem Rechner 192.168.1.1 Port 1024 ist zu www.netscape.com Port 80 aufgebaut.
2. Dies wird von einem Masquerading-Rechner maskiert, um 1.2.3.4 als Quelle zu verwenden.
3. Der Masqerading-Rechner versucht, von 1.2.3.4 (die Adresse seiner externen Schnittstelle) Port 1024, eine Webverbindung zu www.netscape.com aufzubauen.
4. Damit die Verbindung sich nicht ueberschneidet, wird der NAT-Code die Quelle der zweiten Verbindung auf 1025 aendern.

Wenn dieses implizite Quell-Mapping auftaucht, werden Ports in drei Klassen aufgeteilt:

- Ports unter 512.
- Ports zwischen 512 und 1023.
- Ports ab 1024.

Ports werden niemals implizit in eine andere Klasse gemappt.

6.3.5 Was passiert, wenn NAT versagt

Wenn es keine Moeglichkeit gibt, eine Verbindung einheitlich zu mappen wie es der Benutzer verlangt, so wird sie verworfen werden. Dies trifft auch auf Pakete zu, die nicht als Teil einer Verbindung klassifiziert werden konnten, weil sie beschaedigt sind, oder der Rechner nicht genug Speicher hat, etc.

6.3.6 Mehrere Mappings, Overlaps und Clashes

Du kannst NAT-Regeln haben, die Pakete in denselben Bereich mappen; der NAT-Code ist clever genung, um Zusammenstoesse zu vermeiden. Es ist also okay, zwei Regeln zu haben, die die Quelladressen 192.168.1.1 und 192.168.1.2 jeweils auf 1.2.3.4 mappen.

Ausserdem kannst Du ueber wirklich verwendete IP-Adressen mappen, solange diese Adressen auch durch den Mapping-Rechner muessen. Wenn Du also ein zugewiesenes Netzwerk (1.2.3.0/24) hast, aber auch ein internes Netzwerk, das dieselben Adressen benutzt, und eins, das private Internet Adressen (192.168.1.0/24) verwendet, kannst Du die 192.168.1.0/24-er Adressen auf das 1.2.3.0/24-er Netzwerk mappen, ohne Dir Sorgen um Zusammenstoesse machen zu muessen:

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

Dieselbe Logik kann auf Adressen angewandt werden, die der NAT-Rechner selbst benutzt: So funktioniert Masquerading (indem die Adressen der Schnittstellen von maskierten Paketen mit den 'wirklichen' Paketen, die durch den Rechner gehen, geteilt werden).

Ausserdem kannst die dieselben Pakete auf viele verschiedene Ziele mappen, und sie werden aufgeteilt werden. Du koenntest zum Beispiel, wenn Du nichts ueber 1.2.3.5 mappen willst, folgendes tun:

```
# 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
```

6.3.7 Das Ziel von lokal-generierten Verbindungen veraendern

Wenn das Ziel eines lokal-generierten Pakets geaendert wird (ich meine durch die OUTPUT-Kette) und das bewirkt, dass das Paket durch eine andere Schnittstelle muss, wird die Quelladresse auch zu der Adresse der Schnittstelle geaendert. Wenn Du zum Beispiel das Ziel eines Loopback-Pakets auf eth0 aenderst, wird die Quelle auch von 127.0.0.1 zur Adresse von eth0 geaendert werden; im Gegensatz zu anderen Source-Mappings geschieht das im selben Augenblick. Natuerlich wird beides wieder umgekehrt, wenn Antwortpakete eintreffen.

7 Spezielle Protokolle

Manche Protokolle werden nicht gern geNATted. Fuer jedes dieser Protokolle muessen zwei Erweiterungen geschrieben werden; eine fuer das Connection-Tracking des Protokolls, und eine fuer das eigentliche NAT.

In der netfilter-Distibution gibt es zur Zeit Module fuer FTP: ip_conntrack_ftp.o und ip_nat_ftp.o. Wenn Du diese Module mit insmod in den Kernel laedst (oder sie permanent hineinkompilierst), sollte NAT auf FTP-Verbindungen funktionieren. Wenn Du das nicht tust, kannst Du nur passives FTP verwenden, und sogar das koennte nicht zuverlaessig funktionieren, wenn Du mehr als einfaches Source-NAT machst.

8 Einsprueche gegen NAT

Wenn Du NAT auf einer Verbindung machst, muessen alle Pakete in **beide** Richtungen (in und aus dem Netzwerk) durch den NAT-Rechner, sonst wird es nicht zuverlaessig funktionieren. Im Besonderen heisst das, dass der connection tracking Code Fragmente wieder zusammensetzt, was bedeutet, dass Deine Verbindung nicht nur unzuverlaessig sein wird, sondern koennten Pakete sogar ueberhaupt nicht durchkommen, da Fragmente zurueckgehalten werden.

9 Danke

Danke zuerst an Watchguard, und an David Bonn, der stark genug an die netfilter-Idee geglaubt hat, um mich waehrend meiner Arbeit daran zu unterstuetzen.

Danke auch allen anderen, die meine Wortschwaelle ertragen mussten, als ich ueber die unschoenen Dinge von NAT gelernt habe. Besonders auch denen, die mein Tagebuch gelesen haben.

Rusty.