Le routage sélectif

Présentation du chapitre

Nous abordons ici un sujet relativement inclassable, tant il fait appel à de multiples outils.

L'objectif initial est de réaliser un "proxy" transparent pour le protocole POP3. Un serveur mandataire, destiné à intercepter le trafic POP3 au niveau applicatif pour y effectuer un contrôle antivirus, de façon tout à fait transparente pour les utilisateurs.

Pour réaliser cette opération, nous devons :

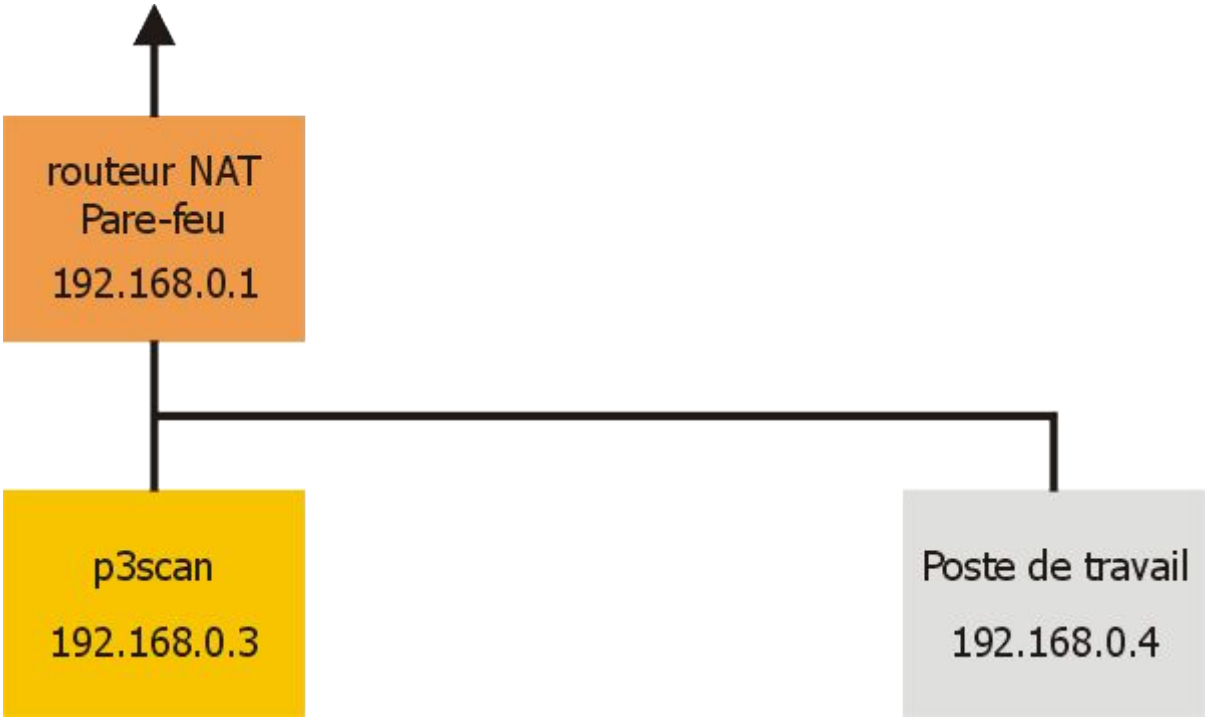
- installer un serveur mandataire (P3Scan),
- l'équiper d'un antivirus (Clamav),
- identifier les flux TCP qui transportent POP3 (Netfilter),
- les router vers le serveur mandataire par un système de routage sélectif (Iproute).

Le thème principal de cet exposé est de réaliser le routage sélectif, en utilisant les services de Netfilter et d'Iproute2. La solution de filtrage antivirus transparente est tout de même réalisée de bout en bout, mais nous n'analyserons pas en profondeur toutes les possibilités offertes par Clamav et P3scan.

Plan du chapitre

Topologie : ce dont nous disposons

Voici l'architecture sur laquelle nous allons monter ce système :

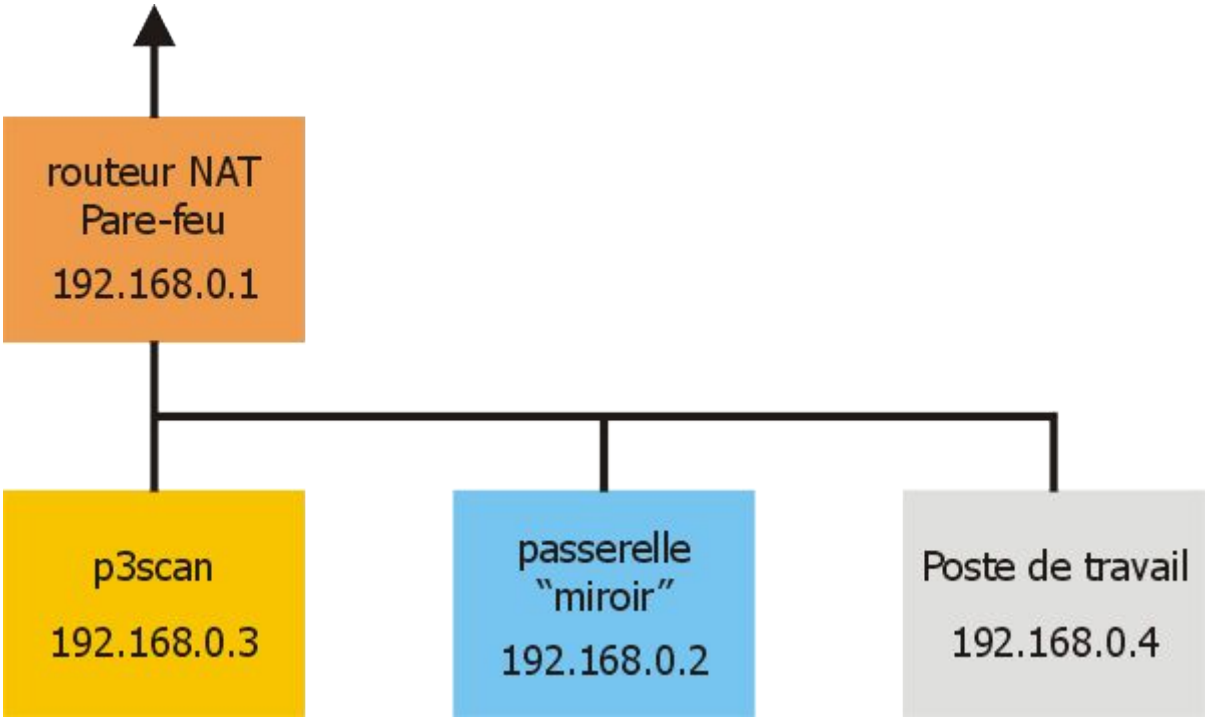


Nous disposons d'un ou de plusieurs postes de travail, qui accèdent à l'internet via un routeur NAT, faisant également office de pare-feu.

La solution triviale serait d'installer notre proxy POP3 sur le routeur lui-même, mais ce n'est pas forcément une bonne solution, d'abord parce que nous verrons que Clamav consomme énormément de ressources, ensuite parce que ce routeur peut très bien être une "boîte noire" spécialisée, sur laquelle nous ne pouvons rien installer.

La solution qui reste est donc d'ajouter sur notre réseau local une machine dédiée au serveur mandataire et à l'antivirus. Nous trouverons bien un moyen d'y envoyer les flux POP3 à destination d'hôtes de l'internet.

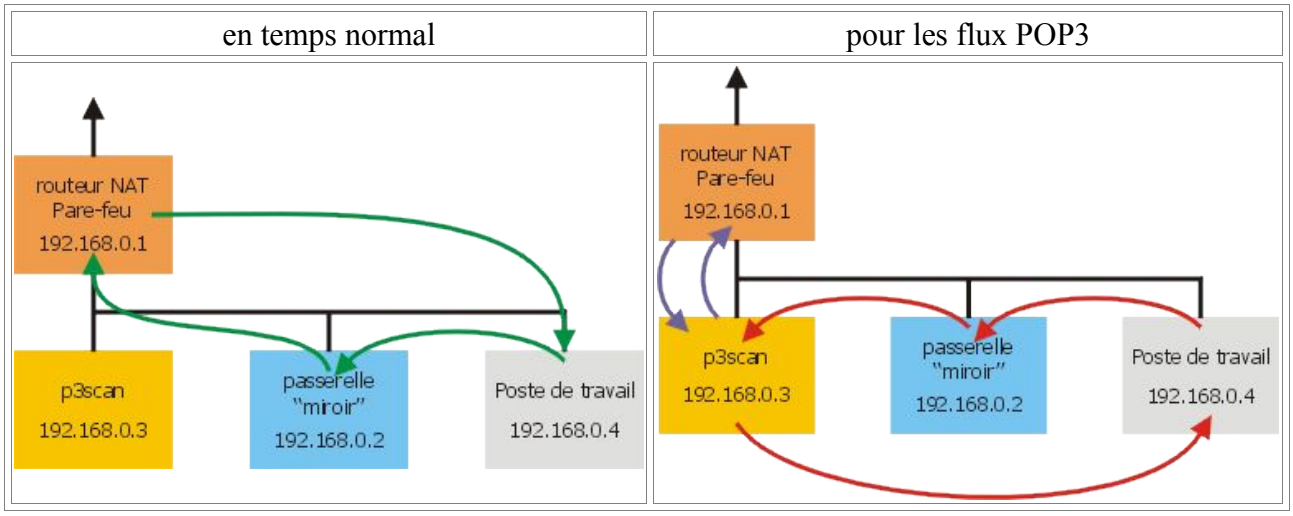
Cette architecture pourrait suffire, nous allons cependant faire plus "riche" en ajoutant encore une nouvelle machine :



La passerelle "miroir" va être configurée comme un routeur, mais avec une seule patte. En gros, tout flux sortant du ou des postes de travail vers l'internet sera dirigé dessus, à charge pour ce miroir de rediriger les flux :

- directement sur le routeur NAT en temps normal,
- sur le proxy P3scan pour les flux POP3.

Les chemins empruntés par les flux seront quelque chose de ce genre :



Rappelons qu'un serveur mandataire joue en quelque sorte le rôle d'une passerelle, mais au niveau de l'application et non au niveau IP. Le fait qu'il soit transparent va faire que :

- le client du LAN va croire qu'il s'adresse directement au serveur cible (pop.free.fr par exemple),
- les requêtes POP3 sont interceptées par le proxy, celui-ci les retransmettant pour son propre

compte au vrai serveur POP3.

- les réponses du vrai serveur POP3 arrivent donc tout naturellement sur le proxy,
- celui-ci effectue le filtrage antivirus sur les messages entrants,
- les retransmet au client du LAN comme s'ils arrivaient directement du vrai serveur POP3, si et seulement s'il n'y a pas eu de virus reconnu dans le message. Sinon, le proxy enverra au client un message d'alerte à la place du message original, qui sera gardé en quarantaine sur le serveur ou purement et simplement détruit.

Le client, en temps normal n'y verra que du feu, n'aura pas besoin de la moindre modification dans sa configuration. Tout se fait dans son dos.

L'intérêt d'ajouter cette passerelle intermédiaire a un double avantage :

- ça permet de mieux comprendre comment le système fonctionne,
- ça permettra d'ajouter facilement d'autres serveurs mandataires pour d'autres protocoles comme FTP ou HTTP ou même de permettre une répartition de la charge sur plusieurs serveurs, si le trafic est très important.

Pratiquement, nous verrons, lorsque nous aurons bien compris le mécanisme, que la fonction de routage intermédiaire dévolue à cette machine peut être reportée sur le mandataire POP3 lui-même, si l'on veut faire plus simple.

Dans la suite de cet exposé, la passerelle miroir et le proxy POP3 sont des machines Linux. Le routeur NAT aussi, mais ça n'a aucune importance dans la suite. Le ou les postes de travail peuvent être n'importe quoi, pourvu que ces n'importe quoi sachent exploiter un réseau TCP/IP.

Le routeur NAT, qui permet "in fine" d'atteindre l'internet sera vu comme un simple routeur, nous ne nous occuperons pas le moins du monde de sa configuration qui est supposée être déjà opérationnelle. Si vous voulez plus de détails sur la façon de réaliser un tel routeur, reportez-vous aux chapitres :

- Partage de connexion[1] pour le principe du NAT,
- Netfilter et IPtables[2] pour approfondir les règles de filtrage de paquets,
- La sécurité[3] pour mieux comprendre ce que l'on risque à se connecter à l'internet.

La distribution utilisée est une Debian "Sarge" (avec un noyau 2.6.8), encore en statut "testing" à l'heure où ces lignes sont écrites, mais qui finira bien un jour pas être déclarée "stable" à la place de la "Woody" vieillissante.

1 MASQUERADE : http://christian.caleca.free.fr/masquerade/
2 Netfilter et IPtables : http://christian.caleca.free.fr/netfilter/
3 Sécurité : http://christian.caleca.free.fr/securite/

Iproute 2

Avertissement

Ce qui suit suppose comme acquis tout ce qui est dit dans les chapitres relatifs au routage :

- Le routage[4]
- Exemples de routage[5]

Au bon vieux temps...

Tous les utilisateurs de GNU/Linux connaissent les commandes "route" et "ifconfig". Bien qu'elles soient toujours efficaces, les évolutions des fonctions de routage des noyaux 2.4.x et supérieurs ont fait qu'elles ne rendent plus compte que d'une partie des possibilités.

Pour exploiter au mieux les possibilités de routage des noyaux actuels, il faut installer le paquetage "iproute" (iproute2 dans d'autres distributions, comme la Mandrake).

Iproute2, allié à Netfilter, donne un couple capable de faire de grandes choses. Comme il faut bien commencer par un bout , voyons d'abord en quoi iproute2 peut nous aider.

Présentation rapide d'Iproute2

Les interfaces du réseau

Nous savons que la commande "ifconfig" nous montre les interfaces réseau en service sur notre machine :

```
betelgeuse:~# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:20:18:2D:D2:91
          inet addr:192.168.100.30  Bcast:192.168.100.255  Mask:255.255.255.0
          inet6 addr: fe80::220:18ff:fe2d:d291/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:19370201 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3279270 errors:0 dropped:0 overruns:0 carrier:0
          collisions:150124 txqueuelen:1000
          RX bytes:3979103861 (3.7 GiB)  TX bytes:376345956 (358.9 MiB)
          Interrupt:11 Base address:0x2000

eth1      Link encap:Ethernet  HWaddr 00:90:27:71:43:C7
          inet addr:192.168.0.250  Bcast:192.168.0.255  Mask:255.255.255.0
          inet6 addr: fe80::290:27ff:fe71:43c7/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:3129241 errors:0 dropped:0 overruns:0 frame:0
          TX packets:3344387 errors:0 dropped:0 overruns:0 carrier:21
          collisions:478776 txqueuelen:1000
          RX bytes:317538639 (302.8 MiB)  TX bytes:2507642937 (2.3 GiB)
          Interrupt:11 Base address:0x7000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
```

4 Routage : http://christian.caleca.free.fr/routage/
5 Exemples de routage : http://christian.caleca.free.fr/demoroutage/

```
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:28040 errors:0 dropped:0 overruns:0 frame:0
          TX packets:28040 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:3395473 (3.2 MiB)  TX bytes:3395473 (3.2 MiB)

ppp0      Link encap:Point-to-Point Protocol
          inet addr:80.8.154.12  P-t-P:80.8.128.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1492  Metric:1
          RX packets:6405 errors:0 dropped:0 overruns:0 frame:0
          TX packets:4791 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:3
          RX bytes:5209426 (4.9 MiB)  TX bytes:662798 (647.2 KiB)

tun1      Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
          inet addr:192.168.18.2  P-t-P:192.168.18.1  Mask:255.255.255.255
          UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1500  Metric:1
          RX packets:685 errors:0 dropped:0 overruns:0 frame:0
          TX packets:685 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:54460 (53.1 KiB)  TX bytes:73640 (71.9 KiB)
```

Très verbeux, nous avons pas mal d'informations sur les diverses interfaces. L'exemple est pris sur une passerelle qui assure la connexion à l'internet via un lien PPP, et qui présente également un lien vers un tunnel Open VPN (tun1).

Iproute dispose d'une commande : "ip" suivie de plusieurs arguments. Utilisons ici "ip addr" :

```
betelgeuse:~# ip addr list
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:20:18:2d:d2:91 brd ff:ff:ff:ff:ff:ff
    inet 192.168.100.30/24 brd 192.168.100.255 scope global eth0
    inet6 fe80::220:18ff:fe2d:d291/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:90:27:71:43:c7 brd ff:ff:ff:ff:ff:ff
    inet 192.168.0.250/24 brd 192.168.0.255 scope global eth1
    inet6 fe80::290:27ff:fe71:43c7/64 scope link
       valid_lft forever preferred_lft forever
4: sit0: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
254: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1492 qdisc pfifo_fast qlen 3
    link/ppp
    inet 80.8.154.12 peer 80.8.128.1/32 scope global ppp0
255: tun1: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/[65534]
    inet 192.168.18.2 peer 192.168.18.1/32 scope global tun1
```

Je vous laisse passer du temps à comparer les deux informations. Formulées de manières différentes (et pas forcément plus lisibles), ce sont, en partie, les mêmes.

Nous avons aussi à notre disposition les commandes :

ip link list, qui affiche la liste des interfaces :

```
betelgeuse:~# ip link list
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:20:18:2d:d2:91 brd ff:ff:ff:ff:ff:ff
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:90:27:71:43:c7 brd ff:ff:ff:ff:ff:ff
4: sit0: <NOARP> mtu 1480 qdisc noop
    link/sit 0.0.0.0 brd 0.0.0.0
```

```
913: ppp0: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1492 qdisc pfifo_fast qlen 3
    link/ppp
914: tun1: <POINTOPOINT,MULTICAST,NOARP,UP> mtu 1500 qdisc pfifo_fast qlen 100
    link/[65534]
```

ip neigh list, qui affiche la table ARP, un peu à la manière de la commande "arp -a" :

```
betelgeuse:~# ip neigh list
2001:4f8:0:2::8 dev eth0  nud failed
192.168.0.13 dev eth1 lladdr 00:20:18:2f:d1:8c nud stale
192.168.0.12 dev eth1 lladdr 00:20:18:2f:0e:35 nud reachable
192.168.0.11 dev eth1 lladdr 00:20:18:2a:f5:ca nud reachable
192.168.0.67 dev eth1 lladdr 00:90:96:c4:d9:e0 nud stale
192.168.0.10 dev eth1 lladdr 00:05:5d:4a:f1:c8 nud reachable
192.168.0.64 dev eth1 lladdr 00:40:05:de:68:c3 nud stale
```

Les routes

Voyons maintenant la commande "route" :

```
betelgeuse:~# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use Iface
192.168.18.1    0.0.0.0         255.255.255.255 UH    0      0        0 tun1
80.8.128.1      0.0.0.0         255.255.255.255 UH    0      0        0 ppp0
192.168.100.0   0.0.0.0         255.255.255.0   U     0      0        0 eth0
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0 eth1
172.16.0.0      192.168.18.1    255.255.0.0     UG    0      0        0 tun1
0.0.0.0         80.8.128.1      0.0.0.0         UG    0      0        0 ppp0
```

La commande "ip", suivie de l'argument "route", nous donne :

```
betelgeuse:~# ip route list
192.168.18.1 dev tun1  proto kernel  scope link  src 192.168.18.2
80.8.128.1 dev ppp0  proto kernel  scope link  src 80.8.154.12
192.168.100.0/24 dev eth0  proto kernel  scope link  src 192.168.100.30
192.168.0.0/24 dev eth1  proto kernel  scope link  src 192.168.0.250
172.16.0.0/16 via 192.168.25.1 dev tun1
default via 80.8.128.1 dev ppp0
```

Moins lisible, assurément. Mais plus riche en informations. Voyons ça de plus près.

Avec "route", nous pensions qu'il n'y avait qu'une table de routage. En réalité, il y en a plusieurs. Dans la terminologie iproute2, les tables sont exploitées en fonction de règles ("rules" en anglais). Nous pouvons avoir un aperçu d'un routage "standard" de la façon suivante :

```
betelgeuse:/etc/iproute2# ip rule list
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
```

Tout ceci voudrait dire qu'il y a au moins trois tables de routage, "local", "main" et "default", qui seraient auscultées pour router tous les paquets entrant dans le routeur, quelle que soit leur provenance ("from all") et dans l'ordre indiqué par l'index numérique placé juste avant chaque règle ?

Absolument, c'est bien comme ça que ça se passe.

Et donc, il serait possible, avec la commande ip, de voir ce qu'il y a dans chacune de ces tables ?

Oui. Il suffit d'en préciser le nom :

```
betelgeuse:/etc/iproute2# ip route list table main
```

```
192.168.18.1 dev tun1  proto kernel  scope link  src 192.168.18.2
80.8.128.1 dev ppp0  proto kernel  scope link  src 80.8.154.12
192.168.100.0/24 dev eth0  proto kernel  scope link  src 192.168.100.30
192.168.0.0/24 dev eth1  proto kernel  scope link  src 192.168.0.250
172.16.0.0/16 via 192.168.25.1 dev tun1
default via 80.8.128.1 dev ppp0
```

C'est la même que tout à l'heure !

Oui, c'est la même. Preuve que l'antique commande "route" ne nous dit pas tout, puisque avant "main", il y a "local", et que cette table n'est pas vide du tout :

```
betelgeuse:/etc/iproute2# ip route list table local
broadcast 192.168.100.0 dev eth0  proto kernel  scope link  src 192.168.100.30
broadcast 192.168.0.255 dev eth1  proto kernel  scope link  src 192.168.0.250
broadcast 127.255.255.255 dev lo  proto kernel  scope link  src 127.0.0.1
local 80.8.154.12 dev ppp0  proto kernel  scope host  src 80.8.154.12
local 192.168.0.250 dev eth1  proto kernel  scope host  src 192.168.0.250
local 192.168.18.2 dev tun1  proto kernel  scope host  src 192.168.18.2
broadcast 192.168.100.255 dev eth0  proto kernel  scope link  src 192.168.100.30
broadcast 192.168.0.0 dev eth1  proto kernel  scope link  src 192.168.0.250
broadcast 127.0.0.0 dev lo  proto kernel  scope link  src 127.0.0.1
local 192.168.100.30 dev eth0  proto kernel  scope host  src 192.168.100.30
local 127.0.0.1 dev lo  proto kernel  scope host  src 127.0.0.1
local 127.0.0.0/8 dev lo  proto kernel  scope host  src 127.0.0.1
```

Mais comme vous le voyez, cette table ne concerne que le routage local et les "broadcasts", rien de bien palpitant. Quant à la table "default", elle est vide par défaut.

Mais alors, il y aurait la possibilité de créer des tables qui ne seraient utilisées que dans certaines conditions ?

C'est une partie de l'intérêt et de la puissance du modèle Iproute2. Parce qu'en plus de ça, Iproute2 sait faire plein d'autres choses. Pour vous en convaincre, le mieux est de lire la documentation officielle : Advanced Routing HOWTO, dont il existe une version en français[6]. Il n'est bien entendu pas question ici de voir tout ce qu'il est possible de faire, l'objectif est juste de pouvoir réaliser un routage sélectif en fonction d'un numéro de port TCP.

Mise en place d'une règle de routage

Dans ce qui suit, la machine qui va servir de passerelle miroir s'appelle "saturne".

La liste des tables qui existent sur votre machine se trouve dans le fichier /etc/iproute2/rt_tables. Vous ne pourrez pas créer de règles (rules) associées à une table qui n'est pas référencée dans ce fichier. Comme nous aurons besoin d'une table de routage spécifique pour le protocole POP3, nous allons créer une entrée supplémentaire dans ce fichier qui, après modification, aura l'allure suivante :

```
saturne:~# cat /etc/iproute2/rt_tables
#
# reserved values
#

200     pop3
255     local
254     main
253     default
0       unspec
#
# local
```

6 http://www.traduc.org/docs/HOWTO/vf/Adv-Routing-HOWTO.html

```
#
1       inr.ruhep
```

Cette manipulation, en elle même n'apporte rien aux règles en vigueur :

```
saturne:~# ip rule list
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default
```

Si nous voulons ajouter une règle, il faudra le dire explicitement de la façon suivante (prenez-le comme une recette pour le moment) :

```
saturne:~# ip rule add fwmark 6e table pop3
```

Ceci a pour effet d'ajouter une règle, que nous comprendrons mieux par la suite. Disons pour le moment que, lorsqu'un paquet contient la marque 0x6e (valeur hexadécimale), il devra être routé en fonction des informations contenues dans la table de routage "pop3".

```
saturne:~# ip rule list
0:      from all lookup local
32765:  from all fwmark       6e lookup pop3
32766:  from all lookup main
32767:  from all lookup default
```

Nous voyons effectivement apparaître une ligne supplémentaire dans la liste des règles.

Mais la table de routage pop3 est vide. Il faut la peupler un petit peu. En réalité, qu'avons-nous besoin de faire, en fonction de la topologie donnée ? Il nous suffit de dire que pour ces paquets, la route par défaut n'est pas 192.168.0.1, mais 192.168.0.3, adresse ip du serveur mandataire. Faisons le :

```
ip route add default via 192.168.0.3 dev eth0 table pop3
```

Bien. Pour vérifier :

```
saturne:~# ip route list table pop3
default via 192.168.0.3 dev eth0
```

Et si nous regardons la table de routage principale :

```
saturne:~# ip route list table main
192.168.0.0/24 dev eth0  proto kernel  scope link  src 192.168.0.2
default via 192.168.0.1 dev eth0
```

Autrement dit, si tout se passe comme prévu, tous les paquets sortant du poste de travail (qui est configuré pour voir 192.168.0.2 comme passerelle par défaut) seront aiguillés vers 192.168.0.1 (la vraie passerelle vers l'internet), sauf les paquets marqués "6e", qui, eux, seront aiguillés vers 192.168.0.3 (le futur proxy transparent POP3).

Pour pouvoir vérifier si tout ça fonctionne, il faut commencer par pouvoir marquer ces paquets avec le label "6e", ce que nous n'avons pas encore fait.

Pour réaliser cette opération, nous aurons recours à Netfilter, avec IPtables.

Netfilter : charcuter les datagrammes IP

Avertissement

Pour ce qui suit, il est fortement recommandé de savoir ce que sont Netfilter et IPtables, autrement dit d'avoir lu le chapitre : "Netfilter et IPtables"[7].

La table "mangle"

Cette table, peu évoquée dans le chapitre indiqué ci-dessus, va nous servir ici. Son but est justement de pouvoir placer des marques sur les paquets qui circulent. Elle dispose au moins de la chaîne PREROUTING, celle qui nous intéresse ici. Autrement dit, ce qui suit doit être réalisable avec tout noyau 2.4.x ou supérieur.

Nous devons en effet marquer les paquets avant le routage, puisque le routage va dépendre justement de cette marque.

Que devons-nous faire exactement ?

Nous devons, sur la passerelle miroir, marquer avant routage les paquets qui contiennent un port de destination égal à 110 (en décimal), le port par défaut sur lequel écoutent les serveurs POP3 :

```
iptables -t mangle -A PREROUTING -i eth0 -p tcp -m tcp --dport 110 -j MARK --set-mark 0x6e
```

Pourquoi la marque 0x6e (le 0x signifie que l'on s'exprime en hexadécimal) ? En réalité, la valeur n'a aucune importance, pourvu qu'on utilise la même avec iproute2, lors de la création de la règle de routage. 0x6e = 110, c'est juste un choix mnémotechnique.

A ce niveau nous avons sur notre passerelle "miroir" :

- les paquets à destination du port 110 qui sont marqués, avec iptables,
- la table de routage spécifique à cette marque qui est renseignée avec une route par défaut qui pointe sur le futur proxy POP3

Si cette passerelle est configurée pour effectuer le routage (/proc/sys/net/ipv4 = 1), elle devrait donc remplir son office.

Passons maintenant à la machine destinée à recevoir le proxy. Pour l'instant, il y a juste un système installé. Il n'est pas nécessaire qu'il soit configuré pour router, mais pour tester ce que nous avons fait jusqu'ici, ce sera utile.

Voyons son état :

```
mercure:~# ip rule list
0:      from all lookup local
32766:  from all lookup main
32767:  from all lookup default

mercure:~# ip route list
192.168.0.0/16 dev eth0  proto kernel  scope link  src 192.168.0.3
```

7 Netfilter et IPtables : http://christian.caleca.free.fr/netfilter/

```
default via 192.168.0.1 dev eth0
```

Vous l'avez compris, sur la maquette, cette machine s'appelle mercure. Elle n'a aucune règle de routage particulière et donc, sa seule route par défaut est 192.168.0.1, la "vraie" passerelle vers l'internet.

Pour les besoins de la manipulation qui suit, nous activons le routage :

```
mercure:~# echo 1 > /proc/sys/net/ipv4/ip_forward
```

Et maintenant, nous avons besoin d'un outil un peu spécial. Un outil qui trace les routes, mais qui le fait en utilisant TCP (et non UDP ou ICMP, comme le traceroute "classique"), sur un port de son choix. Cet outil n'est bien sûr pas installé par défaut, mais est disponible sur la Debian Sarge, avec un "apt-get install tcptraceroute".

Dans un premier temps, un tcptraceroute sur le port 80 de www.free.fr :

```
janus:~# tcptraceroute www.free.fr 80
Selected device eth0, address 192.168.0.4 for outgoing packets
Tracing the path to www.free.fr (213.228.0.42) on TCP port 80, 30 hops max
 1  192.168.0.2 (192.168.0.2)  0.275 ms  0.136 ms  0.121 ms
 2  192.168.0.1 (192.168.0.1)  0.769 ms
    193.253.160.3 (193.253.160.3)  68.483 ms  59.903 ms
 3  80.10.215.197 (80.10.215.197)  84.734 ms  58.070 ms  59.270 ms
 4  pos6-0.nraub303.Aubervilliers.francetelecom.net (193.252.99.70)  216.432 ms  215.634 ms  216.115
ms
 5  pos13-0.ntaub301.aubervilliers.francetelecom.net (193.252.161.82)  57.044 ms  55.925 ms  54.911
ms
 6  pos0-0-0-0.noaub101.aubervilliers.francetelecom.net (193.252.103.85)  56.687 ms  56.202 ms
57.633 ms
 7  193.252.161.98 (193.252.161.98)  58.090 ms  57.025 ms  57.829 ms
 8  p19-6k-2-v806.routers.proxad.net (212.27.50.161)  56.652 ms  55.946 ms  56.285 ms
 9  vlq-6k-2-v800.intf.routers.proxad.net (212.27.50.5)  58.152 ms  57.447 ms  57.984 ms
10  vlq-6k-1-po1.intf.routers.proxad.net (212.27.50.2)  57.969 ms  57.473 ms  60.456 ms
11  www1.free.fr (213.228.0.42) [open]  56.886 ms  58.319 ms  62.456 ms
```

Notez qu'à l'aller, le paquet issu de notre machine de test (janus, sous Debian Sarge elle aussi), passent par notre passerelle miroir 192.168.0.2, puis par le routeur NAT 192.168.0.1.

Voyons maintenant un tcptraceroute sur le port 110 de pop.free.fr :

```
janus:~# tcptraceroute pop.free.fr 110
Selected device eth0, address 192.168.0.4 for outgoing packets
Tracing the path to pop.free.fr (212.27.42.14) on TCP port 110, 30 hops max
 1  192.168.0.2 (192.168.0.2)  0.285 ms  0.122 ms  0.119 ms
 2  192.168.0.3 (192.168.0.3)  0.784 ms  0.229 ms  0.214 ms
 3  192.168.0.1 (192.168.0.1)  0.847 ms  0.585 ms  0.578 ms
 4  193.253.160.3 (193.253.160.3)  67.374 ms  58.665 ms
    80.10.215.197 (80.10.215.197)  58.922 ms
 5  pos1-0.nraub303.aubervilliers.francetelecom.net (193.252.103.170)  62.153 ms  63.039 ms  60.602
ms
 6  pos13-0.ntaub301.aubervilliers.francetelecom.net (193.252.161.82)  59.383 ms  55.566 ms  57.070
ms
 7  pos0-0-0-0.noaub101.aubervilliers.francetelecom.net (193.252.103.85)  57.811 ms  57.416 ms
59.551 ms
 8  193.252.161.98 (193.252.161.98)  56.101 ms  56.321 ms  57.904 ms
 9  p19-6k-2-v806.routers.proxad.net (212.27.50.161)  61.040 ms  55.864 ms  58.123 ms
10  vlq-6k-2-v800.intf.routers.proxad.net (212.27.50.5)  58.282 ms  58.115 ms  57.634 ms
11  vlq-6k-1-po1.intf.routers.proxad.net (212.27.50.2)  59.048 ms  58.657 ms  59.208 ms
12  pop4-q.free.fr (212.27.42.14) [open]  59.107 ms  58.373 ms  63.638 ms
```

Si le paquet issu de janus passe d'abord par la passerelle miroir 192.168.0.2, il est ensuite dirigé sur mercure, notre futur proxy POP3 dont l'adresse est 192.168.0.3. Comme celui-ci n'est pas encore configuré, mais qu'il a le routage activé, il transmet alors le paquet au routeur NAT 192.168.0.1.

Nous avons bien réussi à mettre en évidence que saturne, la passerelle miroir, effectue un routage différent suivant que le trafic est du http (port 80) ou du POP3 (port 110).

Le proxy

Installation de Clamav

Clamav est un antivirus sous licence GPL, qui fonctionne sur le principe d'une base de données de signatures, comme la plupart des antivirus commerciaux. Son installation ne pose aucun problème particulier sur une Debian Sarge. Les scripts d'installation vous aideront à configurer correctement cet antivirus, en choisissant :

- le miroir le plus proche pour les mises à jour de la base de donnée des virus connus,
- le mode de mise à jour, plusieurs modes possibles, dont un démon qui vérifie périodiquement auprès du serveur, la mise à disposition d'une mise à jour de la base.

Pensez à installer tous les outils de décompression nécessaires au bon fonctionnement de l'antivirus sur les pièces jointes compressées. Apt vous indiquera la liste des paquetages supplémentaires suggérés et recommandés, s'ils ne sont déjà présents.

Clamav semble être un bon antivirus, dont le principal avantage est d'être sous licence GPL et le principal inconvénient, celui de consommer énormément de ressources CPU. Il ne sait pas éradiquer un virus dans un document (ici, un e-mail). Tout ce qu'il sait faire, c'est identifier un virus connu. Vous pourrez alors choisir de mettre le message en quarantaine ou de le détruire, mais c'est P3Scan qui se chargera de ça.

Clamav tourne par défaut sous le nom d'un utilisateur fictif : "clamav", créé lors de l'installation du paquetage.

Installation de P3Scan

P3Scan est le logiciel proxy. Il permet non seulement d'y greffer Clamav, ou d'autres antivirus non libres, mais encore spamassassin, si vous souhaitez en profiter pour filtrer les spams.

Il s'installe aussi simplement que clamav avec les apt, mais il faudra ici revenir un petit peu sur son fichier de configuration.

Pour que P3Scan fonctionne, il faut le configurer pour qu'il utilise au moins les services d'un antivirus.

Voici l'allure du fichier de configuration :

```
mercure:~# cat /etc/p3scan/p3scan.conf | grep ^[^#]
pidfile = /var/run/p3scan/p3scan.pid
maxchilds = 10
ip = 0.0.0.0
port = 8110
user = p3scan
notifydir = /var/spool/p3scan/notify
virusdir = /var/spool/p3scan
justdelete
scannertype = basic
scanner = /usr/bin/clamscan --no-summary
virusregexp = .*: (.*) FOUND
template = /etc/p3scan/p3scan-fr.mail
subject = [virus] dans un message pour vous:
notify = Pour information, le message a été détruit.
```

Le "grep ^[^#]" est une astuce qui permet d'éliminer toutes les lignes de commentaire (qui commencent par un #). Pour comprendre cette incantation, il faut comprendre la syntaxe des expressions régulières, c'est assez spécial...

Vous pourrez, au choix, conserver en quarantaine les messages vérolés (dans /var/spool/p3scan) ou tout simplement les détruire (directive "justdelete")

La directive "template" définit le chemin d'accès au message type qui se substituera au message infecté. P3scan propose dans /etc/p3scan une liste de messages types en différentes langues, que vous pouvez aisément personnaliser.

La ligne "subject" définit l'objet de ce message, qui sera complété par le nom du virus.

La ligne "notify" apparaît dans le message si "justdelete" est actif.

Voici (par anticipation, parce que pour l'instant, ça ne fonctionne pas encore), un exemple de texte source d'un message reçu (ici, avec l'option "justdelete" inactive) :

```
From - Sat Feb 05 14:29:34 2005
X-Account-Key: account2
X-UIDL: 1083005466.9058
X-Mozilla-Status: 0001
X-Mozilla-Status2: 00000000
Return-Path: <webmaster@divx-overnet.com>
Received: from mwinf1008.wanadoo.fr (mwinf1008.wanadoo.fr)
        by mwinb0704 (SMTP Server) with LMTP; Sat, 05 Feb 2005 14:30:25 +0100
X-Sieve: Server Sieve 2.2
Received: from me-wanadoo.net (localhost [127.0.0.1])
        by mwinf1008.wanadoo.fr (SMTP Server) with ESMTP id 56A2B6000384
        for <caleca.christian@free.fr>; Sat,  5 Feb 2005 14:30:25 +0100 (CET)
Received: from free.fr (12.254.100-84.rev.gaoland.net [84.100.254.12])
        by mwinf1008.free.fr (SMTP Server) with ESMTP id EC80B600038E
        for <caleca.christian@free.fr>; Sat,  5 Feb 2005 14:30:17 +0100 (CET)
X-ME-UUID: 20050205133018968.EC80B600038E@mwinf1008.free.fr
From: webmaster@divx-overnet.com
To: caleca.christian@free.fr
Date: Sat, 05 Feb 2005 14:30:34 +0100
Subject: [Virus] dans un message pour vous: Worm.SomeFool.AA-2
MIME-Version: 1.0
Content-Transfer-Encoding: 8bit
Content-Type: text/plain;
       charset="iso-8859-1"

Bonjour caleca.christian.
Ce corps de message a été généré automatiquement par P3Scan, qui fonctionne sur
la passerelle mercure pour scanner tous les courriels entrants.

Il remplace le corps du message contenant un VIRUS qui vous était adressé!
Ce message vous est envoyé à la place du courriel infecté, .

==========
Nom du virus:
   Worm.SomeFool.AA-2

Expéditeur du courriel:
   webmaster@divx-overnet.com

Envoyé à:
   caleca.christian@free.fr

Date:
   Sat, 05 Feb 2005 14:30:34 +0100

Object:
   Information

Informations de connexion:
   POP3 from 192.168.0.10:1395 to 193.252.22.90:110
```

```
Le fichier original du message est archivé sur le serveur sous le nom de :
   /p3scan.ZzReH5
==============

--
P3Scan 2.1
par Jack S. Lai <laitcg@cox.net>
.
```

Que manque-t-il pour finir ?

Nous avons :

- le routage sélectif qui est opérationnel, nous l'avons testé,
- l'antivirus installé,
- le proxy p3scan également installé.

Mais...

Il faut tout de même rediriger les flux POP3 (port 110) entrants sur mercure vers le port du proxy (8110 dans notre configuration). Il nous faudra donc écrire quelque règle "iptables" bien sentie sur mercure.

Quelque chose du genre :

```
iptables -t nat -A PREROUTING -i eth0 -d ! 192.168.0.0/24 -p tcp --dport 110 -j REDIRECT --to-ports
8110
```

Nous sommes en "PREROUTING", donc sitôt que le paquet TCP entre par eth0, s'il est à destination du port 110, il est redirigé sur le port 8110, sans changer l'adresse IP du destinataire (ce qui n'aurait pas été le cas avec une règle utilisant le DNAT, condition indispensable pour que P3Scan puisse savoir à quel vrai serveur POP3 il doit s'adresser). Il sera donc intercepté par le proxy, à condition que ce ne soit pas à destination de l'une de nos machines du LAN (c'est une précaution, pas indispensable, du moins pour p3scan).

Le résultat est assez amusant (et même vertigineux) à voir avec un tcptraceroute :

```
janus:~# tcptraceroute pop.wanadoo.fr 110
Selected device eth0, address 192.168.0.4 for outgoing packets
Tracing the path to pop.wanadoo.fr (193.252.22.68) on TCP port 110, 30 hops max
 1  192.168.0.2 (192.168.0.2)  0.282 ms  0.125 ms  0.118 ms
 2  pop.wanadoo.fr (193.252.22.68) [open]  1.087 ms  0.231 ms  0.209 ms
```

Quand je vous disais que c'est amusant... Le client, janus, qui fait un traceroute vers pop.wanadoo.fr a l'impression qu'il a atteint sa cible au deuxième "hop", c'est à dire, lorsqu'il atteint le proxy (mercure).

Fonctionnement

Ethereal va encore frapper, pour bien voir ce qu'il se passe au niveau de mercure, le proxy POP3. Juste pour vérifier le fonctionnement du proxy, depuis janus, nous allons ouvrir une session POP3 sur pop.free.fr, avec telnet, ce sera plus folklorique. Pendant ce temps, ethereal récupèrera tout ce qui passe sur l'unique interface eth0 de la machine.

Voici l'intégralité du dialogue dans la console telnet :

```
janus:~# telnet pop.free.fr 110
Trying 212.27.42.11...
Connected to pop1-q.free.fr.
Escape character is '^]'.
+OK <31488.1107616735@pop1-q.free.fr>
user caleca.christian
+OK
pass haben_non
+OK
stat
+OK 1121 25539581
quit
+OK
Connection closed by foreign host.
```

Vraiment le strict minimum, juste pour voir comment ça se passe au niveau de mercure.

```
No.     Source          Destination     Protocol Info
      1 192.168.0.4     212.27.42.12    TCP      33273 > pop3 [SYN] Seq=0 Ack=0
      2 212.27.42.12    192.168.0.4     TCP      pop3 > 33273 [SYN, ACK] Seq=0 Ack=1
      3 192.168.0.4     212.27.42.12    TCP      33273 > pop3 [ACK] Seq=1 Ack=1

# le client janus (192.168.0.4) a ouvert une connexion TCP avec ce qu'il croit être
# le serveur pop.free.fr (212.27.42.12). En réalité, il l'a ouverte avec mercure (192.168.0.3)
# mais ça ne se voit pas. C'est normal, c'est fait pour être "transparent"

      4 192.168.0.3     212.27.42.12    TCP      39232 > pop3 [SYN] Seq=0 Ack=0
      5 212.27.42.12    192.168.0.3     TCP      pop3 > 39232 [SYN, ACK] Seq=0 Ack=1
      6 192.168.0.3     212.27.42.12    TCP      39232 > pop3 [ACK] Seq=1 Ack=1

# A son tour, mercure (192.168.0.3) ouvre une connexion TCP avec le vrai pop.free.fr.
# Cette fois-ci, il n'y a pas de tromperie.

      7 212.27.42.12    192.168.0.3     POP      Response: +OK <14870.1107617952@pop2-q.free.fr>
      8 192.168.0.3     212.27.42.12    TCP      39232 > pop3 [ACK] Seq=1 Ack=40

# Le vrai serveur pop.free.fr, croyant avoir affaire avec un vulgaire client POP3
# envoie à notre proxy la réponse +OK (voir le protocole pop3 pour plus d'informations)

      9 212.27.42.12    192.168.0.4     POP      Response: +OK <14870.1107617952@pop2-q.free.fr>
     10 192.168.0.4     212.27.42.12    TCP      33273 > pop3 [ACK] Seq=1 Ack=40

# notre proxy, qui se fait toujours passer pour le vrai pop.free.fr, répercute alors cette
# réponse à notre client janus (192.168.0.4) qui continue à croire qu'il discute
# avec le vrai serveur POP3

     11 192.168.0.4     212.27.42.12    POP      Request: user caleca.christian
     12 212.27.42.12    192.168.0.4     TCP      pop3 > 33273 [ACK] Seq=40 Ack=24

# notre client, toujours berné, envoie alors le nom d'utilisateur, à ce qu'il croit
# toujours être son serveur POP3 (alors qu'il s'agit bien sûr de notre proxy)

     13 192.168.0.3     212.27.42.12    POP      Request: user caleca.christian
     14 212.27.42.12    192.168.0.3     TCP      pop3 > 39232 [ACK] Seq=40 Ack=24

# Et notre proxy (192.168.0.3) de répéter la chose au vrai serveur POP3
# qui continue à croire qu'il discute avec son client.

     15 212.27.42.12    192.168.0.3     POP      Response: +OK
     16 192.168.0.3     212.27.42.12    TCP      39232 > pop3 [ACK] Seq=24 Ack=46

# Il lui répond alors +OK

     17 212.27.42.12    192.168.0.4     POP      Response: +OK
     18 192.168.0.4     212.27.42.12    TCP      33273 > pop3 [ACK] Seq=24 Ack=46

# Bien entendu, notre proxy qui continue à se faire passer pour pop.free.fr
# vis à vis de janus, lui répercute la réponse.

     19 192.168.0.4     212.27.42.12    POP      Request: pass haben_non
     20 212.27.42.12    192.168.0.4     TCP      pop3 > 33273 [ACK] Seq=46 Ack=39
```

```
# Enhardi par ce succès, janus envoie alors le mot de passe au supposé pop.free.fr
# (qui n'est autre, vous le devinez, que le proxy mercure)...

    21 192.168.0.3    212.27.42.12   POP      Request: pass haben_non
    22 212.27.42.12   192.168.0.3    TCP      pop3 > 39232 [ACK] Seq=46 Ack=39

# Lequel mercure, qui continue inlassablement à se faire passer pour un client
# POP3 "normal", répète au vrai serveur POP3.

    23 212.27.42.12   192.168.0.3    POP      Response: +OK
    24 192.168.0.3    212.27.42.12   TCP      39232 > pop3 [ACK] Seq=39 Ack=52

# Lequel, ne trouvant rien de spécial à redire, se contente de répondre +OK

    25 212.27.42.12   192.168.0.4    POP      Response: +OK
    26 192.168.0.4    212.27.42.12   TCP      33273 > pop3 [ACK] Seq=39 Ack=52

# et notre proxy de continuer à jouer les péroquets.
# Et ainsi de suite... Je vous laisse finir de commenter ce "listing"

    27 192.168.0.4    212.27.42.12   POP      Request: stat
    28 212.27.42.12   192.168.0.4    TCP      pop3 > 33273 [ACK] Seq=52 Ack=45
    29 192.168.0.3    212.27.42.12   POP      Request: stat
    30 212.27.42.12   192.168.0.3    TCP      pop3 > 39232 [ACK] Seq=52 Ack=45
    31 212.27.42.12   192.168.0.3    POP      Response: +OK 1122 25540691
    32 192.168.0.3    212.27.42.12   TCP      39232 > pop3 [ACK] Seq=45 Ack=71
    33 212.27.42.12   192.168.0.4    POP      Response: +OK 1122 25540691
    34 192.168.0.4    212.27.42.12   TCP      33273 > pop3 [ACK] Seq=45 Ack=71
    35 192.168.0.4    212.27.42.12   POP      Request: quit
    36 212.27.42.12   192.168.0.4    TCP      pop3 > 33273 [ACK] Seq=71 Ack=51
    37 192.168.0.3    212.27.42.12   POP      Request: quit
    38 212.27.42.12   192.168.0.3    POP      Response: +OK
    39 192.168.0.3    212.27.42.12   TCP      39232 > pop3 [ACK] Seq=51 Ack=77
    40 212.27.42.12   192.168.0.4    POP      Response: +OK
    41 192.168.0.4    212.27.42.12   TCP      33273 > pop3 [ACK] Seq=51 Ack=77
    42 212.27.42.12   192.168.0.3    TCP      pop3 > 39232 [FIN, ACK] Seq=77 Ack=51
    43 212.27.42.12   192.168.0.4    TCP      pop3 > 33273 [FIN, ACK] Seq=77 Ack=51
    44 192.168.0.3    212.27.42.12   TCP      39232 > pop3 [FIN, ACK] Seq=51 Ack=78
    45 192.168.0.4    212.27.42.12   TCP      33273 > pop3 [FIN, ACK] Seq=51 Ack=78
    46 212.27.42.12   192.168.0.4    TCP      pop3 > 33273 [ACK] Seq=78 Ack=52
    47 212.27.42.12   192.168.0.3    TCP      pop3 > 39232 [ACK] Seq=78 Ack=52
```

Donc, pour récapituler :

> Le client janus croit tout le temps qu'il dialogue avec pop.free.fr, alors qu'en réalité, il dialogue avec mercure, le proxy transparent. Pour janus, l'adresse IP de son serveur est toujours celle de pop.free.fr.
>
> Le vrai serveur pop.free.fr, tout le temps du dialogue, discute en réalité avec notre proxy, en croyant que c'est un vulgaire client POP3.
>
> Le proxy, quant à lui, se contente de répéter d'un bord à l'autre ce qu'il reçoit :
>
> - en se faisant passer pour un client du côté du vrai serveur, avec sa vraie adresse IP,
> - en se faisant passer pour le vrai serveur POP3 du côté du client, en "usurpant" l'adresse IP du vrai serveur POP3.

Finalement aucune des deux extrémités ne se rend compte qu'il y a un serveur mandataire entre les deux, et c'est bien le but d'un proxy transparent.

Conclusions

L'objectif de ce chapitre était de donner un exemple d'application du marquage de paquets avec Netfilter, associé à un routage sélectif avec Iproute2, en se servant de ces possibilités pour réaliser un proxy transparent pour le protocole POP3.

Nous n'avons pas vu tout ce qu'il est possible de faire avec la table "mangle" de Netfilter, ni tout ce qu'il est possible de tirer d'Iproute2. Parmi les choses possibles, il y a l'équilibrage de charge entre protocoles, la ventilation du trafic entre réseaux sur plusieurs routes par défaut, si l'on dispose par exemple de plusieurs accès à l'internet, la surveillance et la protection contre certains types de "DoS" et d'autres choses encore.

Nous avons rapidement vu le logiciel P3scan pour réaliser le serveur mandataire, sans pour autant en exploiter toutes les possibilités. Nous aurions pu y greffer aussi Spamassassin pour effectuer un filtrage de spams.

Nous avons vu, non moins rapidement, le logiciel antivirus Clamav, pour l'utiliser à travers P3scan. Mais Clamav est un antivirus à part entière, qui peut également servir à contrôler le contenu des mémoires de masse, et peut aussi se greffer sur d'autres applications.

Parmi les applications très proches de ce que nous avons fait, c'est à dire en utilisant des techniques tout à fait similaires, nous pourrions citer :

- la mise en place d'un serveur proxy transparent pour le FTP, avec filtrage antivirus ; Frox est un mandataire FTP bien adapté à cette possibilité,
- la mise en place d'un proxy transparent pour HTTP, avec Squid, que l'on peut également coupler à Clamav ; il est même possible d'utiliser Squid avec SquidGuard et Clamav, à la condition d'utiliser Clamav à travers SquidGuard.

Si ce genre d'aventures vous tente, pensez tout de même que Clamav, comme Spamassassin sont des logiciels très gourmands en ressources CPU et prévoyez les machines en conséquence.

L'architecture qui est proposée dans cet exposé offre l'avantage de pouvoir facilement multiplier les machines hébergeant les serveurs mandataires.

Les liens utiles pour aller plus loin :

- le HOWTO[8] du routage avancé et du contrôle de trafic sous Linux,
- les documentations du projet Netfilter[9],
- le site officiel de l'antivirus Clamav[10],
- le site officiel du proxy POP3 P3scan[11],
- le site officiel du proxy FTP Frox[12],
- le site officiel du proxy HTTP Squid[13],

8 http://www.traduc.org/docs/HOWTO/vf/Adv-Routing-HOWTO.html
9 Netfilter : http://www.netfilter.org/documentation/
10 Clamav : http://www.clamav.net/
11 P3scan : http://p3scan.sourceforge.net/
12 Frox : http://frox.sourceforge.net/
13 Squid : http://www.squid-cache.org/

- le site officiel du filtre SquidGuard[14],
- le site officiel de SpamAssassin[15],
- SquidClamAV Redirector[16] est un script en python qui permet d'utiliser Clamav avec Squid,
- Willowbark[17] est un script cgi en perl qui permet d'utiliser Clamav avec SquidGuard.

14 SquidGuard : http://www.squidguard.org/
15 SpamAssassin : http://spamassassin.apache.org/index.html
16 SquidClamAV : http://www.jackal-net.at/tiki-index.php
17 Willowbark : http://www.aerospacesoftware.com/willowbark-howto.html