Linux IPCHAINS-HOWTO

Paul Russell, *ipchains@rustcorp.com*
Version française par Arnaud Launay, *asl@launay.org*

v1.0.7, 12 mars 1999

Ce document décrit l'obtention, l'installation et la configuration du logiciel amélioré de chaînes pare-feu IP pour Linux, et donne quelques idées sur l'utilisation que vous pouvez en faire.

Contents

1 Introduction

Ceci est le Linux IPCHAINS-HOWTO ; voyez la section 1.4 (Où ?) pour le site principal, qui détient la dernière version. Vous devriez également lire le Linux NET-3-HOWTO. Les howtos IP-Masquerading, PPP-HOWTO, l'Ethernet-HOWTO et le Firewall HOWTO peuvent aussi être intéressants à lire (une fois de plus, la FAQ de alt.fan.bigfoot peut l'être aussi).

Si le filtrage des paquets vous semble dépassé, lisez la section 1.2 (Pourquoi ?), la section 2.3 (Comment ?), et lisez les titres de la section 4 (Chaînes de protection IP).

Si vous vous adaptez en partant d'`ipfwadm`, lisez la section 1 (Introduction), la section 2.3 (Comment ?), et les annexes des sections 8 (Différences entre ipchains et ipfwadm) et 9 (Utiliser le script 'ipfwadm-wrapper').

1.1 Qu'est ce que c'est ?

Le Linux `ipchains` est une réécriture du code de firewalling de l'IPv4 de Linux (qui avait été principalement emprunté à BSD) et une réécriture d'`ipfwadm`, lui même réécriture d'`ipfw` de BSD, je crois. Il est nécessaire pour administrer le filtrage des paquets IP dans les noyaux Linux à partir de la version 2.1.102.

1.2 Pourquoi ?

L'ancien code de firewalling de Linux ne pouvait gérer les fragments, utilisait des compteurs 32 bits (au moins sur Intel), ne permettait pas la spécification de protocoles autres que TCP, UDP ou ICMP, ne pouvait faire de grands changements atomiquement, ne permettait pas la spécification de règles inverses, avait quelques bizarreries, et pouvait être une catastrophe à gérer (le rendant propice aux erreurs des utilisateurs).

1.3 Comment ?

Actuellement le code se trouve dans le noyau principal à partir du 2.1.102. Pour la série des noyaux 2.0, vous devrez récupérer une correction pour le noyau sur une page web. Si votre noyau 2.0 est plus récent que la correction récupérée, la correction ancienne devrait fonctionner ; cette partie des noyaux 2.0 est relativement stable (la correction pour le noyau 2.0.34 fonctionne bien sur le noyau 2.0.35). Cependant, le correctif 2.0 est incompatible avec les correctifs pour l'ipportfw et l'ipautofw, je recommande donc de ne pas l'appliquer, à moins que vous n'ayez réellement besoin de l'une des fonctionnalités offertes par ipchains.

1.4 Où ?

La page officielle est la *Page des chaînes de filtrage IP de Linux* `http://www.rustcorp.com/linux/ipchains` .

Il y a une liste de diffusion pour les rapports d'erreurs, les discussions, le développement et l'utilisation. Vous pouvez rejoindre la liste de diffusion en envoyant un message contenant le mot "subscribe" à ipchains-request de rustcorp.com. Pour écrire à la liste utilisez 'ipchains' à la place de 'ipchains-request'.

2 Bases du filtrage de paquets

2.1 Qu'est ce que c'est ?

Tout le trafic circulant dans un réseau est envoyé sous la forme de **paquets**. Par exemple, pour charger ce paquetage (disons qu'il fait 50k) vous avez dû recevoir plus ou moins 36 paquets de 1460 octets chacun (pour prendre des valeurs au hasard).

Le début de chaque paquet précise où celui-ci va, d'où il vient, le type du paquet, et divers autres détails administratifs. Le début de chaque paquet est appelé l'**entête**. Le reste du paquet, contenant les données à transmettre, est couramment appelé le **corps**.

Quelques protocoles, comme le **TCP**, qui est utilisé pour le trafic web, mail et les connexions à distance, utilisent le concept de 'connexion' – avant que le moindre paquet de données ne soit envoyé, divers paquets de configuration (avec des entêtes spéciales) sont échangés en disant 'je veux me connecter', 'OK' et 'Merci'. Ensuite les paquets normaux sont échangés.

Un filtre de paquet est un logiciel qui regarde l'*entête* des paquets lorsque ceux-ci passent, et décide du destin du paquet entier. Il peut décider de le **refuser** (le supprimer comme s'il n'avait jamais été reçu), de l'**accepter** (le laisser circuler) ou de le **rejeter** (effet identique au refus, mais il est précisé à la source que le paquet n'a pas été accepté).

Sous Linux, le filtrage des paquets est inclus dans le noyau, et il y a diverses choses que nous pouvons faire avec les paquets, mais le principe général (regarder les entêtes et décider du destin du paquet) est toujours présent.

2.2 Pourquoi ?

Contrôle. Sécurité. Vigilance.

Contrôle :

Lorsque vous utilisez un ordinateur sous Linux pour connecter votre réseau interne à un autre réseau (disons, l'Internet) vous aurez l'opportunité de permettre certains types de trafics, et d'interdire les autres. Par exemple, l'entête d'un paquet contient l'adresse de destination du paquet, et vous pouvez ainsi éviter que des paquets aillent vers un certain endroit du réseau extérieur. Comme autre exemple, j'utilise Netscape pour accéder aux archives de Dilbert. Il y a des publicités provenant de doubleclick.net sur la page, et Netscape perd du temps en les chargeant gentiment. Dire au filtre des paquets de ne pas autoriser la circulation de paquets provenant ou allant vers doubleclick.net résoud le problème (il y a cependant de meilleurs moyens pour y parvenir).

Sécurité :

Lorsque votre machine Linux est le seul rempart entre le chaos de l'Internet et votre réseau propre et bien ordonné, il est utile de savoir que vous pouvez restreindre ce qui vient sonner à votre porte. Par exemple, vous pouvez autoriser tout ce qui sort de votre réseau, mais vous pouvez vous inquiéter du fort connu 'Ping of Death' pouvant provenir d'intrus extérieurs. Comme autre exemple, vous pouvez interdire aux personnes extérieures de se connecter en telnet sur votre machine Linux, même si tous vos comptes ont des mots de passe ; peut-être désirerez-vous (comme la plupart des gens) être un simple observateur sur l'Internet, et non un serveur (de bonne volonté ou non) – simplement en ne laissant personne se connecter, le filtrage de paquets rejetant tous les paquets entrants utilisés pour créer des connexions.

Vigilance :

Parfois, une machine mal configurée du réseau local décidera d'envoyer des paquets au monde extérieur. Il est sympathique de pouvoir spécifier au filtre de paquets de vous informer si quelque chose d'anormal se produit ; vous pourrez éventuellement y faire quelque chose, ou bien laisser libre cours à la simple curiosité.

2.3 Comment ?

2.3.1 Un noyau avec le filtrage de paquets

Vous aurez besoin d'un noyau disposant du nouveau code de chaînes de protection IP. Vous pouvez savoir si le noyau que vous utilisez actuellement en dispose en cherchant le fichier '/proc/net/ip_fwchains'. Si ce fichier existe, alors c'est tout bon.

Sinon, vous devez créer un noyau contenant le code de chaînes de protection IP. Tout d'abord, récupérez les sources du noyau que vous désirez. Si vous avez un noyau dont le numéro est supérieur ou égal à 2.1.102, vous n'aurez pas besoin de le corriger (il se trouve déjà inclus dans le noyau). Autrement, appliquez le correctif que vous trouverez sur la page web listée plus haut, et utilisez la configuration détaillée ci-dessous. Si vous ne savez pas comment le faire, ne paniquez pas – lisez le Kernel-HOWTO.

Les options de configuration que vous devez utiliser pour les *noyaux de série 2.0* sont :

```
CONFIG_EXPERIMENTAL=y
CONFIG_FIREWALL=y
CONFIG_IP_FIREWALL=y
CONFIG_IP_FIREWALL_CHAINS=y
```

Pour les séries de *noyaux 2.1* ou *2.2* :

```
CONFIG_FIREWALL=y
CONFIG_IP_FIREWALL=y
```

L'outil `ipchains` parle au noyau et lui dit quels paquets filtrer. À moins que vous ne soyez un programmeur, ou un curieux invétéré, c'est ainsi que vous contrôlerez le filtrage des paquets.

2.3.2 ipchains

L'outil `ipchains` insère et efface des règles dans la section de filtrage de paquets du noyau. Ce qui signifie que quoi que vous configuriez, tout sera perdu lors d'un redémarrage ; voyez la section 2.3.3 (Rendre les règles permanentes) pour savoir comment s'assurer que les règles seront restaurées au prochain lancement de Linux.

`ipchains` remplace `ipfwadm`, qui était utilisé par l'ancien code pare-feu. Il y a un ensemble de scripts utiles disponibles sur le site ftp d'ipchains :

ftp://ftp.rustcorp.com/ipchains/ipchains-scripts-1.1.2.tar.gz `ftp://ftp.rustcorp.com/ipchains/ipchains-scripts-1.1.2.tar.gz`

Cette archive contient un script shell appellé `ipfwadm-wrapper` qui vous autorisera à utiliser le filtrage de paquets comme avant. Vous ne devriez probablement pas utiliser ce script à moins que vous ne souhaitiez un moyen rapide de mettre à jour un système utilisant `ipfwadm` (ce script est plus lent, ne vérifie pas les arguments, etc.). Dans ce cas, vous n'avez pas non plus besoin de ce howto.

Voyez l'annexe 8 (Différences entre ipchains et ipfwadm) et l'annexe 9 (Utiliser le script 'ipfwadm-wrapper') pour des détails supplémentaires concernant `ipfwadm`.

2.3.3 Rendre les règles permanentes

Votre configuration actuelle de pare-feu est sauvée dans le noyau, et sera ainsi perdue lors d'un redémarrage. Je vous recommande d'utiliser les scripts 'ipchains-save' et 'ipchains-restore' pour rendre vos règles permanentes. Pour ce faire, configurez vos règles, puis utilisez (en tant que super-utilisateur) :

```
# ipchains-save > /etc/ipchains.rules
#
```

Créez un script comme le suivant :

```
#! /bin/sh
# Script to control packet filtering.

# If no rules, do nothing.
[ -f /etc/ipchains.rules ] || exit 0

case "$1" in
    start)
        echo -n "Turning on packet filtering:"
        /sbin/ipchains-restore < /etc/ipchains.rules || exit 1
        echo 1 > /proc/sys/net/ipv4/ip_forward
        echo "."
```

```
            ;;
    stop)
            echo -n "Turning off packet filtering:"
            echo 0 > /proc/sys/net/ipv4/ip_forward
            /sbin/ipchains -X
            /sbin/ipchains -F
            /sbin/ipchains -P input ACCEPT
            /sbin/ipchains -P output ACCEPT
            /sbin/ipchains -P forward ACCEPT
            echo "."
            ;;
    *)
            echo "Usage: /etc/init.d/packetfilter {start|stop}"
            exit 1
            ;;
esac

exit 0
```

Assurez vous que ceci est lancé suffisamment tôt dans la procédure de lancement. Dans mon cas (Debian 2.1), j'ai créé un lien symbolique appellé 'S39packetfilter' dans le répertoire '/etc/rcS.d' (il sera ainsi lancé avant S40network).

3 Je suis troublé ! Routage, camouflage, redirection de ports, ipautofw...

Ce HOWTO a pour sujet le filtrage de paquets. Ce filtrage permet la prise de décision concernant le destin d'un paquet : s'il est autorisé à passer ou non. Cependant, Linux étant le joujou pour bidouilleurs qu'il est, vous voudriez probablement en savoir un peu plus.

Un des problèmes est que le même outil ("ipchains") est utilisé pour contrôler à la fois le camouflage et le cache transparent, alors que ce sont des notions séparées du filtrage de paquets (l'implémentation actuelle de Linux les brouille tous les trois de façon inhabituelle, laissant l'impression qu'ils sont très proches).

Le camouflage et le cachage sont recouverts par des HOWTOs séparés, et les possibilités de redirection automatique et de redirection de ports sont contrôlées par des outils séparés, mais puisque de nombreuses personnes continuent à me harceler à leur propos, je vais ajouter un ensemble de scénarios classiques en indiquant les moments où chacun doit être utilisé. Les mérites de la sécurité de chacun de ces scénarios ne seront néanmoins pas discutés ici.

3.1 Le guide du camouflage en 3 lignes par Rusty

Ces lignes présument que votre interface **externe** est appellée "ppp0". Utilisez ifconfig pour le vérifier, et ajustez selon votre goût.

```
# ipchains -P forward DENY
# ipchains -A forward -i ppp0 -j MASQ
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

3.2 Publicité gratuite : le zèle de WatchGuard

Vous pouvez acheter des pare-feu tout faits. Un excellent est le FireBox de WatchGuard. C'est excellent parce que je l'apprécie, parce qu'il est sécurisé, basé sur Linux, et parce qu'ils financent la maintenance d'ipchains ainsi que du nouveau code pare-feu (prévu pour le 2.3). En bref, WatchGuard me paye à manger lorsque je travaille pour vous. Donc, je vous prierai de prendre leur travail en compte.

http://www.watchguard.com `http://www.watchguard.com`

3.3 Configurations classiques de type pare-feu

Vous êtes petiteboite.com. Vous avez un réseau interne, et une connexion intermittente (PPP) simple à l'Internet (firewall.petiteboite.com a pour IP 1.2.3.4). Vous êtes en Ethernet sur votre réseau local, et votre machine personnelle s'appelle "mamachine".

Cette section illustrera les différents arrangements classiques. Lisez attentivement, car ils sont tous subtilement différents.

3.3.1 Réseaux privés : caches traditionnels

Dans ce scénario, les paquets venant d'un réseau privé ne traversent jamais l'Internet, et vice versa. Les adresses IP du réseau privé doivent être assignées en utilisant les adresses privées réservées par la RFC 1597 (càd 10.*.*.*, 172.16.*.* ou 192.168.*.*).

La seule méthode pour que les choses soient connectés à l'Internet est en se connectant au pare-feu, qui est la seule machine sur les deux réseaux qui sont connectés plus loin. Vous lancez un programme (sur le pare-feu) appellé un proxy pour ce faire (il y a des proxy (caches) pour le FTP, l'accès web, telnet, RealAudio, les News Usenet et autres services). Voyez le Firewall HOWTO.

Tous les services auxquels vous voulez que l'Internet puisse avoir accès doivent être sur le pare-feu (mais voyez 3.3.5 (Services internes limités) plus bas).

Exemple : autoriser l'accès web d'un réseau privé vers l'Internet.

1. On a assigné les adresses 192.168.1.* au réseau privé, avec mamachine étant 192.168.1.100, et l'interface Ethernet du pare-feu étant assignée à 192.168.1.1.
2. Un cache web (comme "squid") est installé et configuré sur le pare-feu, disons tournant sur le port 8080.
3. Netscape sur le réseau privé est configuré pour utiliser le pare-feu port 8080 comme cache.
4. Le DNS n'a pas besoin d'être configuré sur le réseau privé.
5. Le DNS doit être configuré sur le pare-feu.
6. Le réseau privé n'a pas besoin de disposer de route par défaut (passerelle).

Netscape sur mamachine lit http://slashdot.org.

1. Netscape se connecte sur le port 8080 du pare-feu, en utilisant le port 1050 de mamachine. Il demande la page web de "http://slashdot.org".
2. Le cache recherche le nom "slashdot.org", et obtient 207.218.152.131. Il ouvre alors une connexion sur cette adresse IP (en utilisant le port 1025 de l'interface externe du pare-feu), et demande la page au serveur web (port 80).

3. En recevant la page web par sa connexion au serveur web, le pare-feu copie les données vers la connexion de Netscape.
4. Netscape affiche la page.

C'est-à-dire que du point de vue de slashdot.org, la connexion est réalisée par 1.2.3.4 (interface PPP du pare-feu), port 1025, vers 207.218.152.131 (slashdot.org) port 80. Du point de vue de mamachine, la connexion est faite de 192.168.1.100 (mamachine) port 1050, vers 192.168.1.1 (interface Ethernet du pare-feu), port 8080.

3.3.2 Réseaux privés : caches transparents

Dans ce scénario, les paquets venant du réseau privé ne traversent jamais l'Internet, et vice versa. Les adresses IP du réseau privé doivent être assignées en utilisant les adresses privées réservées par la RFC 1597 (càd 10.*.*.*, 172.16.*.* ou 192.168.*.*).

La seule méthode pour que les choses soient connectées à l'Internet est en se connectant au pare-feu, qui est la seule machine sur les deux réseaux, qui sont connectés plus loin. Vous lancez un programme (sur le pare-feu) appellé un cache transparent pour ce faire ; le noyau envoie les paquets sortants au cache transparent au lieu de les envoyer plus loin (càd qu'il rend bâtard le routage).

Le cachage transparent signifie que les clients n'ont pas besoin de savoir qu'il y a un proxy dans l'histoire.

Tous les services que l'Internet peut utiliser doivent être sur le pare-feu (mais voyez 3.3.5 (Services internes limités) plus bas).

Exemple : autoriser l'accès web du réseau privé vers l'Internet.

1. On a assigné les adresses 192.168.1.* au réseau privé, avec mamachine étant 192.168.1.100, et l'interface Ethernet du pare-feu étant assignée à 192.168.1.1.
2. Un proxy web transparent (je présume qu'il existe des correctifs pour squid lui permettant d'opérer de cette façon, sinon, essayez "transproxy") est installé et configuré sur le pare-feu, disons tournant sur le port 8080.
3. On dit au noyau de rediriger les connexions sur le port 80 du proxy, en utilisant ipchains.
4. Netscape, sur le réseau privé, est configuré pour se connecter directement.
5. Le DNS doit être configuré sur le réseau privé (càd que vous devez faire tourner un serveur DNS de la même manière que le proxy sur le pare-feu).
6. La route par défaut (passerelle) doit être configuré sur le réseau privé, pour envoyer les paquets au pare-feu.

Netscape sur mamachine lit http://slashdot.org.

1. Netscape recherche le nom "slashdot.org", et obtient 207.218.152.131. Il ouvre alors une connexion vers cette adresse IP, en utilisant le port local 1050, et demande la page au serveur web (port 80).
2. Comme les paquets venant de mamachine (port 1050) et allant sur slashdot.org (port 80) passent par le pare-feu, ils sont redirigés sur le proxy transparent en attente sur le port 8080. Le cache transparent ouvre alors une connexion (en utilisant le port local 1025) vers 207.218.152.131 port 80 (vers lequel les paquets de départ allaient).
3. Alors que le cache reçoit la page web par sa connexion avec le serveur web, il copie les données vers la connexion avec Netscape.

4. Netscape affiche la page.

C'est à dire que du point de vue de slashdot.org, la connexion est réalisée par 1.2.3.4 (interface PPP du pare-feu) port 1025 vers 207.218.152.131 (slashdot.org) port 80. Du point de vue de mamachine, la connexion est faite à partir de 192.168.1.100 (mamachine) port 1050, vers 207.218.152.131(slashdot.org) port 80, mais il parle en fait au proxy transparent.

3.3.3 Réseaux privés : camouflage

Dans ce scénario, les paquets venant du réseau privé ne traversent jamais l'Internet sans traitement spécial, et vice versa. Les adresses IP du réseau privé doivent être assignées en utilisant les adresses privées réservées par la RFC 1597 (càd 10.*.*.*, 172.16.*.* ou 192.168.*.*).

Au lieu d'utiliser un cache, nous utilisons une spécificité spéciale du noyau nommée "camouflage" (masquerading). Le camouflage réécrit les paquets lorsqu'ils passent par le pare-feu, ce qui fait qu'ils semblent toujours venir du pare-feu lui-même. Il réécrit ensuite les réponses afin qu'elles semblent venir du destinataire originel.

Le camouflage dispose de modules séparés afin de gérer les protocoles "étranges", comme FTP, RealAudio, Quake, etc. Pour les protocoles vraiment difficiles à gérer, la spécificité de "redirection automatique" (auto forwarding) peut en gérer quelques-uns en configurant automatiquement la redirection de ports pour un ensemble donné de ports : voyez "ipportfw" (noyaux 2.0) ou "ipmasqadm" (noyaux 2.1 et supérieurs).

Tous les services auxquels vous voulez que l'Internet puisse avoir accès doivent être sur le pare-feu (mais voyez 3.3.5 (Services internes limités) plus bas).

Exemple : autoriser l'accès web du réseau privé sur l'Internet.

1. On a assigné les adresses 192.168.1.* au réseau privé, avec mamachine étant 192.168.1.100, et l'interface Ethernet du pare-feu étant assignée à 192.168.1.1.

2. Le pare-feu est configuré pour camoufler tous les paquets venant du réseau privé et allant sur le port 80 d'un hôte sur Internet.

3. Netscape est configuré pour se connecter directement.

4. Le DNS doit être configuré correctement sur le réseau privé.

5. Le pare-feu doit être la route par défaut (passerelle) du réseau privé.

Netscape sur mamachine lit http://slashdot.org.

1. Netscape recherche le nom "slashdot.org", et obtient 207.218.152.131. Il ouvre alors une connexion vers cette adresse IP, en utilisant le port local 1050, et demande la page au serveur web (port 80).

2. Comme les paquets venant de mamachine (port 1050) et allant sur slashdot.org (port 80) passent par le pare-feu, ils sont réécrits pour venir de l'interface PPP du pare-feu, port 65000. Le pare-feu a une adresse Internet valide (1.2.3.4), donc les paquets venant de slashdot.org sont routés correctement.

3. Lorsque les paquets venant de slashdot.org (port 80) sur firewall.petiteboite.com (port 65000) arrivent, ils sont réécrits pour aller sur mamachine, port 1050. La véritable magie du camouflage se trouve ici : il se souvient des paquets sortants réécrits afin de pouvoir réécrire les paquets entrants qui en sont la réponse.

4. Netscape affiche la page.

C'est à dire que du point de vue de slashdot.org, la connexion est réalisée de 1.2.3.4 (interface PPP du pare-feu), port 65000 vers 207.218.152.131 (slashdot.org) port 80. Du point de vue de mamachine, la connexion est faite entre 192.168.1.100 (mamachine) port 1050, et 207.218.152.131 (slashdot.org) port 80.

3.3.4 Réseaux publics

Dans ce scénario, votre réseau personnel fait partie de l'Internet : les paquets peuvent passer sans avoir à changer de réseau. Les adresses IP du réseau interne doivent être assignées en utilisant un bloc d'adresses IP, de manière à ce que le reste du réseau sache comment vous envoyer des paquets. Ceci implique une connexion permanente.

Dans ce rôle, le filtrage de paquets est utilisé pour restreindre les paquets qui peuvent être redirigés entre votre réseau et le reste de l'Internet, càd pour restreindre le reste de l'Internet à accéder uniquement au serveur web qui se trouve en interne.

Exemple : autoriser l'accès web du réseau privé vers l'Internet.

1. Votre réseau interne dispose du bloc d'adresses IP que vous avez enregistré, disons 1.2.3.*.
2. Le pare-feu est configuré pour autoriser tout le trafic.
3. Netscape est configuré pour se connecter directement.
4. Le DNS doit être configuré correctement sur votre réseau.
5. Le pare-feu doit être la route par défaut (passerelle) pour le réseau privé.

Netscape sur mamachine lit http://slashdot.org.

1. Netscape recherche le nom "slashdot.org", et obtient 207.218.152.131. Il ouvre alors une connexion vers cette adresse IP, en utilisant le port local 1050, et demande la page au serveur web, port 80.
2. Les paquets passent par votre pare-feu, comme ils passent par d'autres routeurs entre vous et slashdot.org.
3. Netscape affiche la page.

C'est à dire qu'il n'y a qu'une seule connexion : à partir de 1.2.3.100 (mamachine) port 1050, vers 207.218.152.131 (slashdot.org) port 80.

3.3.5 Services internes limités

Il y a quelques trucs que vous pouvez utiliser pour autoriser l'Internet à accéder à vos services internes, plutôt que de faire tourner vos services internes sur le pare-feu. Ils fonctionneront soit avec un proxy, soit avec une approche type camouflage pour les connexions externes.

L'approche la plus simple est de faire tourner un "redirecteur", qui est un cache de pauvre, attendant une connexion sur un port donné, et ouvrant une connexion sur un hôte et un port interne fixé, et copiant les données entre les deux connexions. Un exemple de ceci est le programme "redir". Du point de vue de l'Internet, la connexion est faite sur votre pare-feu. Du point de vue de votre serveur interne, la connexion est faite par l'interface interne du pare-feu sur le serveur.

Une autre approche (qui nécessite un noyau 2.0 corrigé pour ipportfw, ou un noyau 2.1 ou supérieur) est d'utiliser la redirection des ports du noyau. Il effectue le même travail que "redir" d'une manière différente : le noyau réécrit les paquets lorsqu'ils passent, en changeant leur adresse de destination et le port pour les faire pointer sur un port et un hôte interne. Du point de vue de l'Internet, la connexion est faite sur votre pare-feu. Du point de vue de votre serveur interne, une connexion directe est réalisée entre l'hôte Internet et votre serveur.

3.4 Pour plus d'informations sur le camouflage

David Ranch a écrit un excellent howto tout neuf sur le camouflage, qui en grande partie recouvre ce howto. Vous pouvez le trouver sur *http://www.ecst.csuchico.edu/˜dranch/LINUX/index-LINUX.html* `http://www.ecst.csuchico.edu/~dranch/LINUX/index-LINUX.html`

J'espère pouvoir bientôt le trouver hébergé sous les auspices du LDP (Linux Documentation Project), sur *http://www.metalab.unc.edu/LDP* `http://www.metalab.unc.edu/LDP` .

La page officielle du camouflage se trouve sur *http://ipmasq.cjb.net* `http://ipmasq.cjb.net` .

4 Chaînes de protection IP

Cette section décrit tout ce que vous avez réellement besoin de savoir pour construire un filtre de paquets adapté à vos besoins.

4.1 Comment les paquets traversent les filtres

Le noyau commence avec trois listes de règles : ces listes sont appellées **chaînes de protection** ou juste **chaînes**. Ces trois chaînes sont appellées **input** (entrée), **output** (sortie) et **forward** (transmission). Lorsqu'un paquet arrive (disons, par une carte Ethernet), le noyau utilise la chaîne `input` pour décider de son destin. S'il survit à ce passage, alors le noyau décide où envoyer le paquet par la suite (ceci est appelé **routage**). S'il est destiné à une autre machine, il consulte alors la chaîne de `transmission`. Enfin, juste avant que le paquet ne ressorte, le noyau consulte la chaîne de `sortie`.

Une chaîne est une vérification de **règles**. Chaque règle dit 'si l'entête de ce paquet ressemble à ceci, alors voilà quoi faire de ce paquet'. Si la règle ne vérifie pas le paquet, alors la règle suivante dans la chaîne est consultée. Enfin, s'il n'y a plus de règles à consulter, alors le noyau regarde la chaîne **police** pour décider ce qu'il doit faire. Dans un système orienté sécurité, cette police dit généralement au noyau de rejeter ou de refuser le paquet.

Pour les fans de l'art ASCII, ceci montre le chemin complet d'un paquet arrivant à une machine.

```
        ----------------------------------------------------------------------------
        |           ACCEPTER/                              interface lo |
        v           REDIRIGER                _________                  |
--> S --> V --> ______ --> D --> ~~~~~~~~ -->| Chaîne  |----> _______ -->
    o     a    |chaîne|    e    {Décision}   |transfert|     |Chaîne |ACCEPTER
    m     l    |entrée|    m    {Routage }   |_________| --->|sortie |
    m     i    |______|    a     ~~~~~~~~         |      | ->|_______|
    e     d       |        s        |             |      | |    |
    |     i       |        q        |           NON/     | |    |
    |     t       v        u        v          REJET     | |    v
    |     é     NON/       e    Processus                | |   NON/
    |     |     REJET      r      Local                  | |  REJET
    |     v                |        ---------------------- |
    v    NON               \ ------------------------------/
```

```
NON
```

Voici une description point par point de chaque partie :

Somme (checksum) :

C'est un test vérifiant si le paquet n'a pas été corrompu d'une manière ou d'une autre. S'il l'a été, il est refusé.

Validité (sanity) :

Il y a en fait un de ces tests sanitaires avant chaque chaîne de protection, mais les chaînes d'entrée sont les plus importantes. Quelques paquets malformés peuvent rendre confus le code de vérification des règles, et ceux-ci sont refusés ici (un message est envoyé au syslog si ceci arrive).

Chaîne d'entrée (input chain) :

C'est la première chaîne de protection qui teste le paquet. Si le verdict de la chaîne n'est ni `DENY` ni `REJECT`, le paquet continue son chemin.

Demasquerade :

Si le paquet est une réponse à un paquet précédemment masqué, il est démasqué, et envoyé directement à la chaîne de `sortie`. Si vous n'utilisez pas le masquage IP, vous pouvez mentalement supprimer ceci du diagramme.

Décision routage (Routing decision) :

Le champ de destination est examiné par le code de routage, pour décider si le paquet doit aller vers un processus local (voir processus local plus bas) ou transmis à une machine distante (voyez les chaînes de renvoi plus bas).

Processus local (Local process) :

Un processus tournant sur la machine peut recevoir des paquets après l'étape de décision de routage, et peut envoyer des paquets (qui passent par l'étape de décision de routage, puis traversent la chaîne de sortie).

Interface lo :

Si les paquets venant d'un processus local sont destinés à un autre processus local, alors ils passeront par la chaîne de sortie en utilisant l'interface lo, puis reviendront par la chaîne d'entrée en utilisant la même interface. L'interface lo est généralement nommée interface loopback.

local :

Si le paquet n'a pas été créé par un processus local, alors la chaîne de transmission est vérifiée, sinon le paquet se dirige vers la chaîne de sortie.

forward chain :

Cette chaîne est traversée par tout paquet qui tente de passer par cette machine vers une autre.

output chain :

Cette chaîne est traversée par tous les paquets juste avant qu'ils ne soient envoyés à l'extérieur.

4.1.1 Utiliser ipchains

Tout d'abord, vérifiez que vous avez la version d'ipchains à laquelle se réfère ce document :

```
$ ipchains --version
ipchains 1.3.9, 17-Mar-1999
```

Notez que je recommande l'utilisation du 1.3.4 (qui ne dispose pas des options longues comme '–sport'), ou du 1.3.8 et suivants ; ils sont en effet très stables.

ipchains dispose d'une page de manuel plutôt bien détaillée (`man ipchains`), et si vous avez besoin de plus de détails en particulier, vous pouvez consulter l'interface de programmation (`man 4 ipfw`), ou le fichier `net/ipv4/ip_fw.c` dans les sources des noyaux 2.1.x, qui est (bien évidemment) la référence.

Il y a également une carte de référence rapide par Scott Bronson dans le paquetage source, aux formats PostScript (TM) a4 et US letter.

Il y a plusieurs choses différentes que vous pouvez faire avec `ipchains`. Tout d'abord les opérations servant à gérer les chaînes entières. Vous commencez avec trois chaînes intégrées `input`, `output` et `forward` que vous ne pouvez effacer.

1. Créer une nouvelle chaîne (-N) ;
2. Supprimer une chaîne vide (-X) ;
3. Changer la police d'une chaîne intégrée (-P) ;
4. Lister les règles d'une chaîne (-L) ;
5. Supprimer les règles d'une chaîne (-F) ;
6. Mettre à zéro les compteurs de paquets et d'octets sur toutes les règles d'une chaîne (-Z).

Il y a plusieurs moyens pour manipuler les règles à l'intérieur d'une chaîne :

1. Ajouter une nouvelle règle à une chaîne (-A) ;
2. Insérer une nouvelle règle à une position quelconque de la chaîne (-I) ;
3. Remplacer une règle à une position quelconque de la chaîne (-R) ;
4. Supprimer une règle à une position quelconque de la chaîne (-D) ;
5. Supprimer la première règle vérifiée dans la chaîne (-D).

Il y a quelques opérations pour le masquage, qui se trouvent dans `ipchains` dans l'attente d'un bon endroit pour les mettre :

1. Lister les connexions masquées actuelles (-M -L) ;
2. Configurer les valeurs de timeout (-M -S) (mais voyez id="no-timeout" name="Je ne peux pas modifier les temps d'attente du camouflage !">).

La fonction finale (et peut-être la plus utile) vous permet de vérifier ce qui arriverait à un paquet donné s'il avait à traverser une chaîne donnée.

4.1.2 Opérations sur une règle simple

Ceci est le B.A.-Ba d'ipchains ; manipuler des règles. Plus généralement, vous utiliserez probablement les commandes d'ajout (-A) et de suppression (-D). Les autres (-I pour l'insertion et -R pour le remplacement) sont des simples extensions de ces concepts.

Chaque règle spéficie un ensemble de conditions que le paquet doit suivre, et ce qu'il faut faire s'il les suit (une "destination"). Par exemple, vous pouvez vouloir refuser tous les paquets ICMP venant de l'adresse IP 127.0.0.1. Donc, dans ce cas nos conditions sont que le protocole doit être ICMP et que l'adresse source doit être 127.0.0.1. Notre destination est "DENY" (rejet).

127.0.0.1 est l'interface "loopback", que vous avez même si vous n'avez de connexion réseau réelle. Vous pouvez utiliser le programme "ping" pour générer de tels paquets (il envoie simplement un paquet ICMP de type 8 (requête d'écho) à qui tous les hôtes coopératifs doivent obligeamment répondre avec un paquet ICMP de type 0 (réponse à un écho)). Ceci le rend utile pour les tests.

```
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes
64 bytes from 127.0.0.1: icmp_seq=0 ttl=64 time=0.2 ms

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0.2/0.2/0.2 ms
# ipchains -A input -s 127.0.0.1 -p icmp -j DENY
# ping -c 1 127.0.0.1
PING 127.0.0.1 (127.0.0.1): 56 data bytes

--- 127.0.0.1 ping statistics ---
1 packets transmitted, 0 packets received, 100% packet loss
#
```

Vous pouvez voir ici que le premier ping réussit (le "-c 1" dit à ping de n'envoyer qu'un seul paquet).

Puis nous ajoutons (-A) à la chaîne d'entrée ("input"), une règle spécifiant que tous les paquets provenant de 127.0.0.1 ("-s 127.0.0.1") avec le protocole ICMP ("-p ICMP") doivent être refusés ("-j DENY").

Ensuite nous testons notre règle, en utilisant le second ping. Il y aura une pause avant que le programme ne se termine, attendant une réponse qui ne viendra jamais.

Nous pouvons supprimer la règle avec l'un des deux moyens. Tout d'abord, puisque nous savons que c'est la seule règle de la chaîne d'entrée, nous pouvons utiliser une suppression numérotée, comme dans :

```
# ipchains -D input 1
#
```

Pour supprimer la règle numéro 1 de la chaîne d'entrée.

La deuxième possibilité est de copier la commande -A, mais en remplaçant le -A par -D. C'est utile lorsque vous avez une chaîne complexe de règles et que vous ne voulez pas avoir à les compter afin de savoir que c'est la règle 37 que vous voulez supprimer. Dans ce cas, nous pouvons utiliser :

```
# ipchains -D input -s 127.0.0.1 -p icmp -j DENY
#
```

La syntaxe de -D doit avoir exactement les mêmes options que la commande -A (ou -I ou -R). S'il y a des règles multiples identiques dans la même chaîne, seule la première sera supprimée.

4.1.3 Spécifications du filtrage

Nous avons vu l'utilisation de "-p" pour spécifier le protocole, et de "-s" pour spécifier l'adresse souce, mais il y a d'autres options que nous pouvons utiliser pour spécifier les caractéristiques des paquets. Ce qui suit en est une description exhaustive.

Spécifier les adresses IP source et destination Les adresses IP source (-s) et destination (-d) peuvent être spécifiées de quatre façons différentes. La façon la plus classique est d'utiliser le nom complet, comme "localhost" ou "www.linuxhq.com". La deuxième méthode est de spécifier l'adresse IP, comme "127.0.0.1".

Les deux autres méthodes permettent la spécification d'un groupe d'adresses IP, comme "199.95.207.0/24" ou "199.95.207.0/255.255.255.0". Toutes deux spécifient toutes les adresses IP de 192.95.207.0 à 192.95.207.255, incluse ; les chiffres suivant le "/" indiquent quelles parties de l'adresse IP sont significatives. "/32" ou "/255.255.255.255" sont les défauts (vérifient toutes les adresses IP). Pour ne spécifier aucune adresse IP, "/0" peut être utilisé, par exemple :

```
# ipchains -A input -s 0/0 -j DENY
#
```

Ceci est rarement utilisé, car l'effet produit par cette ligne de commande est le même que celui obtenu en ne spécifiant pas l'option "-s".

Spécifier l'inversion De nombreuses options, incluant les options "-s" et "-d" peuvent avoir leurs propres arguments précédés par "!" (prononcé "non") pour ne vérifier que les adresses n'étant PAS équivalentes à celles données. Par exemple, "-s ! localhost" vérifiera tout paquet ne provenant PAS de localhost.

Spécifier le protocole Le protocole peut être spécifié en utilisant l'option "-p". Le protocole peut être soit un nombre (si vous connaissez les valeurs numériques des protocoles pour IP), soit le nom des cas spéciaux parmis "TCP", "UDP" ou "ICMP". La casse n'est pas prise en compte, donc "tcp" fonctionne aussi bien que "TCP".

Le nom du protocole peut être préfixé par un "!", pour l'inverser, comme dans "-p ! TCP".

Spécifier les ports UDP et TCP Pour les cas spéciaux où un protocole TCP ou UDP est spécifié, il peut y avoir un argument supplémentaire indiquant le port TCP ou UDP, ou un intervalle (inclusif) de ports (mais voyez 4.1.3 (Utiliser les fragments) plus bas). Un intervalle est représenté en utilisant le caractère ":", par exemple "6000:6010", qui couvre 11 ports, du 6000 au 6010, de manière inclusive. Si la borne inférieure est omise, alors elle se met par défaut à 0. Si la borne supérieure est omise, elle est considérée par défaut comme étant 65535. Ainsi, pour spécifier les connexions TCP venant des ports inférieurs à 1024, la syntaxe pourrait être "-p TCP -s 0.0.0.0/0 :1023". Les numéros de ports peuvent être spécifiés par leur nom, par exemple "www".

Notez que la spécification du port peut être précédée par un "!", qui l'inverse. Ainsi, pour spécifier tous les paquets TCP, SAUF un paquet WWW, vous pourriez spécifier

```
-p TCP -d 0.0.0.0/0 ! www
```

Il est important de réaliser que spécifier

```
-p TCP -d ! 192.168.1.1 www
```

est très différent de

```
-p TCP -d 192.168.1.1 ! www
```

La première ligne spécifie tout paquet TCP dirigé vers le port WWW de n'importe quelle machine sauf 192.168.1.1. La seconde spécifie toute connexion TCP vers tout port de 192.168.1.1 sauf le port WWW.

Enfin, le cas suivant spécifie tout, sauf le port WWW et la machine 192.168.1.1 :

```
-p TCP -d ! 192.168.1.1 ! www
```

Spécifier les types et codes ICMP ICMP permet aussi un argument optionnel, mais comme l'ICMP ne dispose pas de ports (ICMP a un **type** et un **code**), ils ont une signification différente.

Vous pouvez les spécifier par les noms ICMP (utilisez `ipchains -h icmp` pour lister les noms disponibles) après l'option "-s", ou en tant que type et code ICMP numérique, où le type suit l'option "-s" et le code suit l'option "-d".

Les noms ICMP sont relativement longs : vous avez uniquement besoin de suffisamment de lettres pour rendre chaque nom distinct des autres.

Voici un petit résumé de quelques-uns des paquets ICMP les plus communs :

```
Numéro  Nom                      Utilisé par

0       echo-reply               ping
3       destination-unreachable Tout trafic TCP/UDP
5       redirect                 Routage, si pas de démon de routage
8       echo-request             ping
11      time-exceeded            traceroute
```

Notez que les noms ICMP ne peuvent être précédés de "!" pour le moment.

NE PAS, SURTOUT NE PAS bloquer tous les paquets ICMP de type 3 ! (voir 5.2.1 (Paquets ICMP) plus bas).

Spécifier une interface L'option "-i" spécifie le nom d'une **interface** à vérifier. Une interface est le périphérique physique d'où vient le paquet, ou bien par où sort ce paquet. Vous pouvez utiliser la commande `ifconfig` pour lister les interfaces qui sont "up" (càd en fonctionnement).

L'interface pour les paquets entrants (ie. les paquets traversant la chaîne d'`entrée`) est considérée comme étant l'interface d'où les paquets proviennent. Logiquement, l'interface des paquets sortants (les paquets traversant la chaîne de `sortie`) est l'interface où ils vont. L'interface pour les paquets traversant la chaîne de `retransmission` est également l'interface par laquelle ils sortiront ; une décision plutôt arbitraire à mon sens.

Il est parfaitement autorisé de spécifier une interface qui n'existe pas au moment de la spécification ; la règle ne vérifiera rien jusqu'à ce que l'interface soit mise en place. Ceci est extrêmement utile pour les connexions ppp intermittentes (habituellement les interfaces du type `ppp0`) et autres.

En tant que cas spécial, un nom d'interface se finissant par un "+" vérifiera toutes les interfaces (qu'elles existent à ce moment ou non) qui commencent par cette chaîne. Par exemple, pour spécifier une règle qui vérifiera toutes les interfaces PPP, l'option `-i ppp+` pourrait être utilisée.

Le nom d'interface peut être précédé par un "!" pour vérifier un paquet qui ne vérifie PAS l'(les) interface(s) spécifiée(s).

Spécifier uniquement des paquets TCP SYN Il est parfois utile d'autoriser des connexions TCP dans une direction, mais pas dans l'autre. Par exemple, vous pouvez vouloir autoriser les connexions vers un serveur WWW externe, mais pas les connexions venant de ce serveur.

L'approche naïve serait de bloquer les paquets TCP venant du serveur. Malheureusement, les connexions TCP utilisent des paquets circulant dans les deux sens pour fonctionner.

La solution est de bloquer uniquement les paquets utilisés pour la demande d'une connexion. Ces paquets sont nommés paquets **SYN** (ok, techniquement ce sont des paquets avec le drapeau SYN mis, et les drapeaux FIN et ACK supprimés, mais nous les appellerons des paquets SYN). En interdisant seulement ces paquets, nous pouvons stopper toute demande de connexion dans l'oeuf.

L'option "-y" est utilisée pour celà : elle est valide seulement pour les règles qui spécifient TCP comme leur protocole. Par exemple, pour spécifier une demande de connexion TCP venant de 192.168.1.1 :

```
-p TCP -s 192.168.1.1 -y
```

Une fois de plus, ce drapeau peut être inversé en le faisant précéder par un "!", qui vérifie tout paquet autre que ceux d'initialisation de connexion.

Utiliser les fragments Parfois, un paquet est trop large pour rentrer dans le câble en une seule fois. Lorsque ceci arrive, le paquet est divisé en **fragments**, et envoyé en plusieurs paquets. Le receveur réassemble les fragments pour reconstruire le paquet en entier.

Le problème avec les fragments se situe dans certaines des spécifications listées ci-dessus (en particulier, le port source, le port de destination, le type et le code ICMP, ou le drapeau TCP SYN), qui demandent au noyau de jeter un regard sur le début du paquet, qui est contenu seulement dans le premier fragment.

Si votre machine est la seule connexion vers un réseau extérieur, vous pouvez demander à votre noyau Linux de réassembler tous les fragments qui passent par lui, en compilant le noyau avec l'option `IP: always defragment` mise à "Y". Ceci évite proprement la plupart des problèmes.

D'autre part, il est important de comprendre comment les fragments sont traités par les règles de filtrage. Toute règle de filtrage qui demande des informations dont nous ne disposons pas ne vérifiera *rien*. Ceci signifie que le premier fragment est traité comme tout autre paquet. Le deuxième fragment et les suivants ne le seront pas. Ainsi, une règle `-p TCP -s 192.168.1.1 www` (spécifiant un port source de "www" ne vérifiera jamais un fragment (autre que le premier fragment). La règle opposée `-p TCP -s 192.168.1.1 ! www` ne fonctionnera pas non plus.

Cependant, vous pouvez spécifier une règle spéciale pour le deuxième fragment et les suivants, en utilisant l'option "-f". Évidemment, il est illégal de spécifier un port TCP ou UDP, un type ou un code ICMP, ou un drapeau TCP SYN dans une règle de fragment.

Il est également autorisé de spécifier une règle qui ne s'applique *pas* au deuxième fragment et aux suivants, en plaçant un "!" avant le "-f".

Habituellement, il est considéré comme sûr de laisser passer le deuxième fragment et les suivants, puisque le filtrage s'effectuera sur le premier fragment, et préviendra donc le réassemblage sur la machine cible. Cependant, des bogues, connus, permettent de crasher des machines en envoyant de simples fragments. À vous de voir.

À noter pour les gourous réseau : les paquets malformés (TCP, UDP et paquets ICMP trop courts pour que le code pare-feu puisse lire les ports ou les types et codes ICMP) sont également traités comme des fragments. Seuls les fragments TCP débutant en position 8 sont supprimés explicitement par le code pare-feu (un message doit apparaître dans le syslog si celà arrive).

Par exemple, la règle suivante supprimera tout fragment allant sur 192.168.1.1 :

```
# ipchains -A output -f -d 192.168.1.1 -j DENY
#
```

4.1.4 Effets de bord du filtrage

OK, maintenant nous connaissons tous les moyens pour vérifier un paquet en utilisant une règle. Si un paquet vérifie une règle, les choses suivantes arrivent :

1. Le compteur d'octets de cette règle est augmenté de la taille de ce paquet (entête et autres) ;
2. Le compteur de paquets de cette règle est incrémenté ;
3. Si la règle le demande, le paquet est enregistré ;
4. Si la règle le demande, le champ "Type Of Service" du paquet est modifié ;
5. Si la règle le demande, le paquet est marqué (sauf dans la série des noyaux 2.0) ;
6. La destination de la règle est examinée afin de décider ce que nous ferons ensuite du paquet.

Pour la diversité, je détaillerai ceux-ci en ordre d'importance.

Spécifier une destination Une **destination** dit au noyau ce qu'il doit faire d'un paquet qui vérifie une règle. ipchains utilise "-j" (penser à "jump-to") pour la spécification de la destination. Le nom de la cible doit comporter moins de 8 caractères, et la casse intervient : "RETOUR" et "retour" sont totalement différents.

Le cas le plus simple est lorsqu'il n'y a pas de destination spécifiée. Ce type de règle (souvent appellée règle de "comptage") est utile pour simplement compter un certain type de paquet. Que cette règle soit vérifiée ou non, le noyau examine simplement la règle suivante dans la chaîne. Par exemple, pour compter le nombre de paquets venant de 192.168.1.1, nous pouvons faire ceci :

```
# ipchains -A input -s 192.168.1.1
#
```

(En utilisant "ipchains -L -v" nous pouvons voir les compteurs de paquets et d'octets associés à chaque règle).

Il y a six destinations spéciales. Les trois premières, `ACCEPT`, `REJECT` et `DENY` sont relativement simples. `ACCEPT` autorise le passage du paquet. `DENY` supprime le paquet comme s'il n'avait jamais été reçu. `REJECT` supprime le paquet, mais (si ce n'est pas un paquet ICMP) génère une réponse ICMP vers la source pour lui dire que la destination n'est pas accessible.

La suivante, `MASQ` dit au noyau de camoufler le paquet. Pour que ceci fonctionne, votre noyau doit être compilé avec le camouflage IP intégré. Pour les détails sur ceci, voyez le Masquerading-HOWTO et l'annexe 8 (Différences entre ipchains et ipfwadm). Cette destination est valide uniquement pour les paquets qui traversent la chaîne `forward`.

L'autre destination majeure spéciale est `REDIRECT` qui demande au noyau d'envoyer un paquet vers un port local au lieu de là où il était destiné. Ceci peut être spécifié uniquement pour les règles spécifiant TCP ou UDP en tant que protocole. Optionnellement, un port (nom ou numéro) peut être spécifié après "-j REDIRECT" qui redirigera la paquet vers ce port particulier, même si celui-ci était dirigé vers un autre port. Cette destination est valide uniquement pour les paquets traversant la chaîne `input`.

La dernière destination spéciale est la `RETURN` qui est identique à une terminaison immédiate de la chaîne (voir 4.1.4 (Choisir une police) plus bas).

Toute autre destination indique une chaîne définie par l'utilisateur (décrite dans 4.1.4 (Opérations sur une chaîne entière) plus bas). Le paquet traversera tout d'abord les règles de cette chaîne. Si cette chaîne ne décide pas du destin du paquet, lorsque la traversée de cette chaîne sera achevée, la traversée reprendra sur la règle suivante de la chaîne courante.

Il est temps de faire encore un peu d'art ASCII. Considérons deux (étranges) chaînes : `input` (la chaîne intégrée) et `Test` (une chaîne définie par l'utilisateur).

```
      ‘input’                           ‘Test’
-----------------------------    -----------------------------
| Règle1: -p ICMP -j REJECT |    | Règle1: -s 192.168.1.1    |
|---------------------------|    |---------------------------|
| Règle2: -p TCP -j Test    |    | Règle2: -d 192.168.1.1    |
|---------------------------|    -----------------------------
| Règle3: -p UDP -j DENY    |
-----------------------------
```

Considérons un paquet TCP venant de 192.168.1.1, allant vers 1.2.3.4. Il pénètre dans la chaîne `input`, et est testé par la Règle1 - pas de correspondance. La Règle2 correspond, et sa destination est `Test`, donc la règle suivante à examiner est le début de `Test`. La Règle1 de `Test` correspond, mais ne spécifie pas de destination, donc la règle suivante est examinée (Règle2). Elle ne correspond pas, nous avons donc atteint la fin de la chaîne. Nous retournons alors à la chaîne `input`, dont nous avons juste examiné la Règle2, et nous examinons alors la Règle3, qui ne correspond pas non plus.

Le chemin suivi par le paquet est donc le suivant :

```
                        v    ---------------------------
 ‘entrée’               |   /    ‘Test’                v
------------------------|--/    -----------------------|----
| Règle1                | /|    | Règle1               |   |
|-----------------------|/-|    |----------------------|---|
| Règle2                /  |    | Règle2               |   |
|--------------------------|    -----------------------v----
| Règle3                /--+---------------------------/
------------------------|---
                        v
```

Voyez la section 5.1 (Comment organiser vos règles pare-feu) pour les moyens d'utiliser efficacement les chaînes définies par l'utilisateur.

Enregistrement des paquets C'est un effet de bord que la vérification d'une règle peut avoir ; vous pouvez enregistrer les paquets vérifiés en utilisant l'option "-l". Vous n'aurez généralement pas besoin de ceci pour les paquets habituels, mais ce peut être une option très utile si vous désirez être tenu au courant des événements exceptionnels.

Le noyau enregistre cette information de la manière suivante :

```
Packet log: input DENY eth0 PROTO=17 192.168.2.1:53 192.168.1.1:1025
  L=34 S=0x00 I=18 F=0x0000 T=254
```

Ce message d'information est prévu pour être concis, et contient des informations techniques qui ne sont utiles qu'aux gourous réseau, mais qui peuvent néanmoins être intéressantes pour le commun des mortels. Il se décompose de la façon suivante :

1. 'input' est la chaîne qui contenait la règle correspondant au paquet, et qui a causé l'apparition du message.
2. 'DENY' est ce que la règle a dit au paquet de faire. Si ceci est un '-' alors la règle n'a pas du tout affecté le paquet (une règle de comptage).
3. 'eth0' est le nom de l'interface. Puisque ceci était la chaîne d'entrée, celà signifie que le paquet vient de 'eth0'.
4. 'PROTO=17' signifie que le paquet était de protocole 17. Une liste des numéros de protocoles est donnée dans '/etc/protocols'. Les plus communs sont 1 (ICMP), 6 (TCP) et 17 (UDP).
5. '192.168.2.1' signifie que l'adresse IP source du paquet était 192.168.2.1.
6. ':53' signifie que le port source était le port 53. En regardant dans '/etc/services' on voit que ceci est le port 'domain' (càd que c'est probablement une réponse DNS). Pour UDP et TCP, ce numéro est le port source. Pour ICMP, c'est le type ICMP. Pour les autres, ce sera 65535.
7. '192.168.1.1' est l'adresse IP de destination.
8. ':1025' signifie que le port de destination était 1025. Pour UDP et TCP, ce numéro est le port de destination. Pour ICMP, il s'agit du code ICMP. Pour les autres, ce sera 65535.
9. 'L=34' signifie que le paquet avait une longueur totale de 34 octets.
10. 'S=0x00' est le champ "Type Of Service" (divisez par 4 pour obtenir le Type of Service utilisé par ipchains).
11. 'I=18' est l'identificateur de l'IP.
12. 'F=0x0000' est l'offset du fragment 16 bits, avec les options. Une valeur débutant par '0x4' ou '0x5' signifie que le bit "Don't Fragment" (ne pas fragmenter) est mis. '0x2' ou '0x3' signifie que le bit "More Fragments" (des fragments suivent) est mis ; attendez vous à recevoir plus de fragments après. Le reste du nombre est le décalage de ce fragment, divisé par 8.
13. 'T=254' est la durée de vie du paquet. On soustrait 1 à cette valeur à chaque saut (hop), et on débute généralement à 15 ou 255.
14. '(#5)' il peut y avoir un numéro entre parenthèses sur les noyaux les plus récents (peut-être après le 2.2.9). Il s'agit du numéro de la règle qui a causé l'enregistrement du paquet.

Sur les systèmes Linux standards, la sortie du noyau est capturée par klogd (démon d'information du noyau) qui le repasse à syslogd (démon d'information du système). Le fichier '/etc/syslog.conf' contrôle le comportement de syslogd, en spécifiant une destination pour chaque "partie" (dans notre cas, la partie est le "noyau") et un "niveau" (pour ipchains, le niveau utilisé est "info").

Par exemple, mon /etc/syslog.conf (Debian) contient deux lignes qui vérifient 'kern.info' :

```
kern.*                          -/var/log/kern.log
*.=info;*.=notice;*.=warn;\
        auth,authpriv.none;\
        cron,daemon.none;\
        mail,news.none          -/var/log/messages
```

Ceci signifie que les messages sont dupliqués dans '/var/log/kern.log' et '/var/log/messages'. Pour plus de détails, voyez 'man syslog.conf'.

Manipuler le type de service Il y a quatre bits utilisés par eux-mêmes dans l'entête IP, appelés les bits de **Type of Service** (TOS). Ceux-ci ont pour effet de modifier la manière dont les paquets sont traités : les quatres bits sont "Minimum Delay", "Maximum Throughput", "Maximum Reliability" et "Minimum Cost". Un seul de ces bits est autorisé à être placé. Rob van Nieuwkerk, l'auteur du code de gestion du TOS, les décrit comme suit :

> Tout spécialement, le "Minimum Delay" est important pour moi. Je le mets pour les paquets "interactifs" dans mon routeur principal (Linux). Je suis derrière une connexion modem 33,6k. Linux rend les paquets prioritaires en 3 queues. De cette façon, j'obtiens des performances interactives acceptables tout en faisant de gros transferts de fichiers en même temps. (Celà pourrait même être encore mieux s'il n'y avait pas de grosse queue dans le pilote série, mais le temps de latence est maintenu en dessous des 1,5 secondes pour l'instant).

Note : bien entendu, vous n'avez aucun contrôle sur les paquets arrivants ; vous pouvez seulement contrôler la priorité des paquets qui quittent votre machine. Pour négocier les priorités de chaque côté, un protocole comme RSVP (dont je ne connais rien) doit être utilisé.

L'utilisation la plus commune est de placer les connexions telnet et contrôle du ftp en "Minimum Delay" et les données FTP en "Maximum Throughput". Ceci peut être fait de la manière suivante :

```
ipchains -A output -p tcp -d 0.0.0.0/0 telnet -t 0x01 0x10
ipchains -A output -p tcp -d 0.0.0.0/0 ftp -t 0x01 0x10
ipchains -A output -p tcp -s 0.0.0.0/0 ftp-data -t 0x01 0x08
```

L'option "-t" prend deux paramètres supplémentaires, tous les deux en hexadécimal. Ceci permet un contrôle complexe des bits du TOS : le premier masque est ANDé avec le TOS actuel du paquet, et ensuite le deuxième masque est XORé avec lui. Si celà est trop confus, utilisez simplement le tableau suivant :

```
Nom du TOS              Valeur          Utilisations typiques

Minimum Delay           0x01 0x10       ftp, telnet
Maximum Throughput      0x01 0x08       ftp-data
Maximum Reliability     0x01 0x04       snmp
Minimum Cost            0x01 0x02       nntp
```

Andi Kleen revient sur ce point pour ce qui suit (modérément édité pour la postérité) :

> Peut-être serait-il utile d'ajouter une référence au paramète txqueuelen de ifconfig dans la discussion sur les bits TOS. La longueur par défaut de la queue d'un périphérique est réglée pour les cartes ethernet, sur les modems elle est trop longue et rend le fonctionnement de la planification 3 bandes (qui place la queue en fonction du TOS) sous-optimal. C'est donc une bonne idée de le configurer avec une valeur comprise entre 4 et 10 sur un modem ou un lien RNIS b simple : sur les périphériques rapide une queue plus longue est nécessaire. C'est un problème des 2.0 et 2.1, mais dans le 2.1 c'est une option de ifconfig (dans les nettools récents), alors que dans le 2.0 il nécessite une modification des sources des pilotes du périphérique pour le changer.

Ainsi, pour voir les bénéfices maximum des changements de TOS pour les liaisons modem PPP, ajoutez un 'ifconfig $1 txqueuelen' dans votre script /etc/ppp/ip-up. Le nombre à utiliser dépend de la vitesse du modem et de la taille du tampon du modem ; voici à nouveau les idées d'Andi :

> La meilleure valeur pour une configuration donnée nécessite de l'expérimentation. Si les queues sont trop courtes sur le routeur, alors les paquets sauteront. Bien sûr, on peut toujours

gagner même sans changer le TOS, c'est juste que le changement du TOS aide à gagner les bénéfices sur les programmes non coopératifs (mais tous les programmes Linux standards sont coopératifs).

Marquage d'un paquet Ceci permet des interactions complexes et puissantes avec la nouvelle implémentation de Quality of Service d'Alexey Kuznetsov, ainsi que de la redirection basée sur le marquage dans la dernière série de noyaux 2.1. Des détails supplémentaires vont arriver puisque nous l'avons dans la main. Cette option est toutefois ignorée dans la série des noyaux 2.0.

Opérations sur une chaîne entière Une des options les plus utiles d'ipchains est la possibilité de regrouper des règles dans des chaînes. Vous pouvez appeller les chaînes de la manière qui vous plaît, tant que les noms ne sont pas ceux des chaînes intégrées (`input`, `output` et `forward`) ou des destinations ((`MASQ`, `REDIRECT`, `ACCEPT`, `DENY`, `REJECT` ou `RETURN`). Je suggère d'éviter complètement les noms en majuscules, car je pourrais les utiliser pour des extensions futures. Le nom de la chaîne ne doit pas dépasser 8 caractères.

Créer une nouvelle chaîne Créons une nouvelle chaîne. Étant un type imaginatif, je l'appellerai `test`.

```
# ipchains -N test
#
```

C'est aussi simple. Maintenant vous pouvez y rajouter des règles, comme détaillé ci-dessus.

Supprimer une chaîne La suppression d'une chaîne est tout aussi simple.

```
# ipchains -X test
#
```

Pourquoi "-X" ? Eh bien, toutes les bonnes lettres étaient déjà prises.

Il y a quelques restrictions à la suppression des chaînes : elles doivent être vides (voir 4.1.4 (Vider une chaîne) plus bas) et elles ne doivent pas être la destination d'une quelconque règle. Vous ne pouvez pas supprimer les chaînes intégrées.

Vider une chaîne Il y a un moyen simple de vider toutes les règles d'une chaîne, en utilisant la commande "-F".

```
# ipchains -F forward
#
```

Si vous ne spécifiez pas de chaîne, alors *toutes* les chaînes seront vidées.

Afficher une chaîne Vous pouvez afficher toutes les règles d'une chaîne en utilisant la commande "-L".

```
# ipchains -L input
Chain input (refcnt = 1): (policy ACCEPT)
target     prot opt    source                destination           ports
ACCEPT     icmp -----  anywhere              anywhere              any
# ipchains -L test
Chain test (refcnt = 0):
```

```
target     prot opt    source                destination           ports
DENY       icmp -----  localnet/24           anywhere              any
#
```

La "refcnt" listée pour `test` est le nombre de règles qui ont `test` comme destination. Ceci doit être égal à zéro (et la chaîne doit être vide) avant que cette chaîne ne puisse être supprimée.

Si le nom de la chaîne est omis, toutes les chaînes sont listées, même les chaînes vides.

Il y a trois options qui peuvent accompagner "-L". L'option "-n" (numérique) est très utile et empêche `ipchains` d'essayer de vérifier les adresses IP, ce qui (si vous utilisez DNS comme la plupart des gens) causera de longs délais si votre DNS n'est pas configuré proprement, ou si vous filtrez les requêtes DNS. Ceci affichera également les ports par leur numéro plutôt que par leur nom.

L'option "-v" vous montrera tous les détails des règles, comme les compteurs de paquets et d'octets, les masques de TOS, l'interface, et les marques de paquets. Autrement, ces valeurs seront omises. Par exemple :

```
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
 pkts bytes target prot opt   tosa tosx ifname  mark  source    destination  ports
   10   840 ACCEPT icmp ----- 0xFF 0x00 lo            anywhere  anywhere     any
```

Notez que les compteurs de paquets et d'octets sont affichés en utilisant les suffixes "K", "M" ou "G" pour 1000, 1.000.000 et 1.000.000.000, respectivement. En utilisant également l'option "-x" (développe les nombres), ipchains affichera les nombres en entier, quelque soit leur taille.

Remise à zéro des compteurs Il est parfois utile de pouvoir remettre à zéro les compteurs. Ceci peut être fait par l'option "-Z" (compteurs à Zéro). Par exemple :

```
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
 pkts bytes target prot opt   tosa tosx ifname  mark  source    destination  ports
   10   840 ACCEPT icmp ----- 0xFF 0x00 lo            anywhere  anywhere     any
# ipchains -Z input
# ipchains -v -L input
Chain input (refcnt = 1): (policy ACCEPT)
 pkts bytes target prot opt   tosa tosx ifname  mark  source      destination   ports
    0     0 ACCEPT icmp ----- 0xFF 0x00 lo            anywhere    anywhere      any
#
```

Le problème de cette approche est que parfois vous voudrez connaître les valeurs du compteur tout de suite après la remise à zéro. Dans l'exemple ci-dessus, quelques paquets peuvent passer entre les commandes "-L" et "-Z". Pour cette raison, vous pouvez utiliser le "-L" et le "-Z" *ensemble*, pour remettre à zéro les compteurs tout en les lisant. Malheureusement, si vous faîtes ceci, vous ne pouvez opérer sur une seule chaîne : vous devrez lister et remettre à zéro toutes les chaînes en une seule fois.

```
# ipchains -L -v -Z
Chain input (policy ACCEPT):
 pkts bytes target prot opt   tosa tosx ifname  mark  source      destination   ports
   10   840 ACCEPT icmp ----- 0xFF 0x00 lo            anywhere    anywhere      any

Chain forward (refcnt = 1): (policy ACCEPT)
Chain output (refcnt = 1): (policy ACCEPT)
Chain test (refcnt = 0):
```

```
    0     0 DENY   icmp ----- 0xFF 0x00  ppp0         localnet/24 anywhere      any
# ipchains -L -v
Chain input (policy ACCEPT):
 pkts bytes target prot opt   tosa tosx ifname  mark  source      destination   ports
   10   840 ACCEPT icmp ----- 0xFF 0x00 lo            anywhere    anywhere      any

Chain forward (refcnt = 1): (policy ACCEPT)
Chain output (refcnt = 1): (policy ACCEPT)
Chain test (refcnt = 0):
    0     0 DENY   icmp ----- 0xFF 0x00 ppp0          localnet/24 anywhere     any
#
```

Choisir une police Nous avons disserté sur ce qui se passe lorsqu'un paquet atteint la fin de la chaîne intégrée, lorsque nous avons discuté sur la manière dont un paquet parcourait les chaînes dans 4.1.4 (Spécifier une destination) plus haut. Dans ce cas, la **police** de la chaîne détermine le destin du paquet. Seules les chaînes intégrées (`input`, `output` et `forward`) ont des polices, car si un paquet atteint la fin d'une chaîne définie par l'utilisateur, le paquet revient sur la chaîne précédente.

La police peut être une des quatre premières destinations spéciales : `ACCEPT`, `DENY`, `REJECT` ou `MASQ`. `MASQ` est la seule destination valide pour la chaîne "forward".

Il est également important de noter qu'une destination `RETURN` dans une règle de l'une des chaînes intégrées est utile pour expliciter la destination de la chaîne lorsqu'un paquet correspond à la règle.

4.1.5 Opérations sur le camouflage

Il y a plusieurs paramètres que vous pouvez modifier pour le camouflage IP. Ils sont intégrés avec `ipchains` car il n'est pas évident d'écrire un outil séparé pour eux (bien que ceci devrait changer).

La commande du camouflage IP est "-M", et peut être combinée avec "-L" pour lister les connexions actuellement camouflées, ou avec "-S" pour configurer les paramètres du camouflage.

La commande "-L" peut être accompagnée par "-n" (montre des nombres à la place des noms de machines et de ports) ou "-v" (affiche les deltas dans les séquences de nombres pour la connexion camouflée, au cas où vous vous demanderiez).

La commande "-S" doit être suivie par trois valeurs de fin d'attente, toutes en secondes : pour les sessions TCP, pour les sessions TCP précédées d'un paquet FIN, et pour les paquets UDP. Si vous ne voulez pas changer l'une de ces valeurs, donnez-lui simplement une valeur de "0".

Les valeurs par défaut sont listées dans "/usr/src/linux/include/net/ip_masq.h", actuellement 15 minutes, 2 minutes et 5 minutes, respectivement.

La valeur changée le plus souvent est la première, pour le FTP (voir 5.2.3 (Cauchemars du FTP) plus bas).

Notez que les problèmes avec la mise en place de temps de fin d'attente sont listés dans 6.9 (Je n'arrive pas à configurer mes temps d'attente !).

4.1.6 Vérifier un paquet

Parfois, vous voulez savoir ce qui arrive lorsqu'un certain paquet entre dans votre machine, par exemple pour déboguer votre chaîne pare-feu. `ipchains` dispose de la commande "-C" pour autoriser celà, en utilisant exactement les mêmes routines que celles que le noyau utilise pour vérifier les vrais paquets.

Vous spécifiez sur quelle chaîne le paquet doit être testé en mettant son nom après l'argument "-C". Même si le noyau commence toujours par traverser les chaînes `input`, `output` ou `forward`, vous êtes autorisé à commencer en traversant n'importe quelle chaîne pour tester.

Les détails du "paquet" sont spécifiés en utilisant la même syntaxe que celle utilisée pour spécifier les règles pare-feu. En particulier, un protocole ("-p"), une adresse source ("-s"), une adresse de destination ("-d") et une interface ("-i") sont nécessaires. Si le protocole est TCP ou UDP, alors une source unique et une destination unique doivent être spécifiées, et un type et un code ICMP doivent être spécifiés pour le protocole ICMP (à moins que l'option "-f" soit spécifiée pour indiquer une règle de fragment, auquel cas ces options sont illégales).

Si le protocole est TCP (et que l'option "-f" n'est pas spécifiée), l'option "-y' doit être explicitée, afin d'indiquer que le paquet test doit être configuré avec le bit SYN.

Voici un exemple de test d'un paquet TCP SYN venant de 192.168.1.1, port 60000, et allant sur 192.168.1.2, port www, arrivant de l'interface eth0 et entrant dans la chaîne "input" (il s'agit d'une initialisation classique d'une connexion WWW) :

```
# ipchains -C input -p tcp -y -i eth0 -s 192.168.1.1 60000 -d 192.168.1.2 www
packet accepted
#
```

4.1.7 Voir ce qui arrive avec des règles multiples précisées en une seule fois

Parfois, une simple ligne de commande peut affecter de multiples règles. Ceci se fait par deux méthodes. Premièrement, si vous spécifiez un nom de machine qui correspond (en utilisant DNS) à de multiples adresses IP, `ipchains` agira comme si vous aviez tapé de multiples commandes avec chaque combinaison d'adresses.

Ainsi, si le nom de machine "www.foo.com" correspond à trois adresses IP, et si le nom de machine "www.bar.com" correspond à deux adresses IP, alors la commande "ipchains -A input -j reject -s www.bar.com -d www.foo.com" ajoutera six règles à la chaîne `input`.

L'autre méthode pour avoir `ipchains` réalisant de multiples actions est d'utiliser l'option bidirectionnelle ("-b"). Cette option fait agir `ipchains` comme si vous aviez tapé la commande deux fois, la deuxième fois avec les arguments "-s" et "-d" inversés. Ainsi, pour éviter la transmission soit de, soit vers 192.168.1.1, vous pourriez utiliser la commande :

```
# ipchains -b -A forward -j reject -s 192.168.1.1
#
```

Personnellement, je n'aime pas beaucoup l'option "-b" ; si vous préférez la convivialité, voyez 4.2.1 (Utiliser ipchains-save) plus bas.

L'option -b peut être utilisée avec les commandes insérer ("-I"), supprimer ("-D") (mais pas avec la variation qui prend un numéro de règle), ajouter ("-A") et vérifier ("-C").

Une autre option utile est "-v" (bruyant) qui imprime exactement ce que `ipchains` fait avec vos commandes. Ceci est utile si vous traitez avec des commandes qui peuvent affecter de multiples règles. Par exemple, nous devons ici vérifier le comportement de fragments entre 192.168.1.1 et 192.168.1.2.

```
# ipchains -v -b -C input -p tcp -f -s 192.168.1.1 -d 192.168.1.2 -i lo
  tcp opt   ---f- tos 0xFF 0x00  via lo    192.168.1.1  -> 192.168.1.2    * ->   *
packet accepted
  tcp opt   ---f- tos 0xFF 0x00  via lo    192.168.1.2  -> 192.168.1.1    * ->   *
packet accepted
#
```

4.2 Exemples utiles

J'ai une connexion intermittente en PPP (`-i ppp0`). Je récupère les news (`-p TCP -s news.virtual.net.au nntp`) et le courrier (`-p TCP -s mail.virtual.net.au pop-3`) à chaque fois que je me connecte. J'utilise la méthode ftp de Debian pour mettre ma machine à jour régulièrement (`-p TCP -y -s ftp.debian.org.au ftp-data`). Je visionne le web au travers du proxy de mon FAI (Fournisseur d'Accès Internet) lorsque je suis en ligne (`-p TCP -d proxy.virtual.net.au 8080`), mais je déteste les publicités de doubleclick.net des Archives de Dilbert (`-p TCP -y -d 199.95.207.0/24` et `-p TCP -y -d 199.95.208.0/24`).

J'autorise les gens à essayer le ftp sur ma machine lorsque je suis en ligne (`-p TCP -d $LOCALIP ftp`), mais je n'autorise personne de l'extérieur à prétendre avoir une adresse IP sur mon réseau interne (`-s 192.168.1.0/24`). Ceci est communément appelé IP spoofing, et il y a un meilleur moyen de se protéger dans les noyaux 2.1.x et suivants : voir 5.7 (Comment mettre en place la protection contre l'IP spoof ?).

Cette configuration est relativement simple, car il n'y a pour l'instant aucune autre machine sur mon réseau interne.

Je ne veux pas que des processus locaux (ie. Netscape, lynx, etc.) se connectent à doubleclick.net :

```
# ipchains -A output -d 199.95.207.0/24 -j REJECT
# ipchains -A output -d 199.95.208.0/24 -j REJECT
#
```

Maintenant je désire changer les priorités des divers paquets sortants (il n'y a pas vraiment d'intérêt à le faire pour les paquets entrants). Puisqu'il y a un certain nombre de ces règles, il y a intérêt à les mettre ensemble dans une seule chaîne, nommée `ppp-out`.

```
# ipchains -N ppp-out
# ipchains -A output -i ppp0 -j ppp-out
#
```

Délai minimal pour le trafic web et telnet :

```
# ipchains -A ppp-out -p TCP -d proxy.virtual.net.au 8080 -t 0x01 0x10
# ipchains -A ppp-out -p TCP -d 0.0.0.0 telnet -t 0x01 0x10
#
```

Coût faible pour les données ftp, nntp et pop-3 :

```
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 ftp-data -t 0x01 0x02
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 nntp -t 0x01 0x02
# ipchains -A ppp-out -p TCP -d 0.0.0.0/0 pop-3 -t 0x01 0x02
#
```

Il y a quelques restrictions sur les paquets venant de l'interface ppp0 ; créons une chaîne nommée "ppp-in" :

```
# ipchains -N ppp-in
# ipchains -A input -i ppp0 -j ppp-in
#
```

Maintenant, aucun des paquets venant de `ppp0` ne doit prétendre avoir une adresse source de la forme 192.168.1.*, donc nous les enregistrons et les interdisons :

```
# ipchains -A ppp-in -s 192.168.1.0/24 -l -j DENY
#
```

J'autorise l'entrée des paquets UDP pour le DNS (je fais tourner un serveur de nom cache qui renvoie toutes les demandes sur 203.29.16.1, donc je m'attend à des réponses DNS venant uniquement de là), l'entrée du ftp, et le retour des données ftp (ftp-data) uniquement (ce qui doit être uniquement entre un port strictement supérieur à 1023, et pas sur les ports X11 autour de 6000).

```
# ipchains -A ppp-in -p UDP -s 203.29.16.1 -d $LOCALIP dns -j ACCEPT
# ipchains -A ppp-in -p TCP -s 0.0.0.0/0 ftp-data -d $LOCALIP 1024:5999 -j ACCEPT
# ipchains -A ppp-in -p TCP -s 0.0.0.0/0 ftp-data -d $LOCALIP 6010: -j ACCEPT
# ipchains -A ppp-in -p TCP -d $LOCALIP ftp -j ACCEPT
#
```

Enfin, les paquets local vers local sont OK :

```
# ipchains -A input -i lo -j ACCEPT
#
```

Maintenant, ma police par défaut sur la chaîne `input` est `DENY`, ce qui fait que tout le reste disparaît :

```
# ipchains -P input DENY
#
```

NOTE : Je ne configurerais pas mes chaînes dans cet ordre, car des paquets pourraient passer durant la configuration. Le moyen le plus sûr est généralement de configurer la police à DENY tout d'abord, et ensuite d'insérer les règles. Bien sûr, si vos règles ont besoin d'effectuer des demandes DNS pour résoudre des noms de machines, vous pourriez avoir des problèmes.

4.2.1 Utiliser ipchains-save

Configurer des chaînes pare-feu de la manière exacte dont vous le voulez, et se rappeller ensuite des commandes que vous avez utilisé pour le faire la fois suivante est insupportable.

Donc, `ipchains-save` est un script qui lit votre configuration actuelle des chaînes et la sauve dans un fichier. Pour le moment, je conserverais le suspens en ce qui concerne l'utilité de `ipchains-restore`.

`ipchains-save` peut sauver une chaîne seule, ou toutes les chaînes (si aucun nom de chaîne n'a été spécifié). La seule option actuellement autorisée est le "-v" qui affiche les règles (vers stderr) lorsqu'elles sont sauvées. La police de la chaîne est aussi sauvée pour les chaînes `input`, `output` et `forward`.

```
# ipchains-save > my_firewall
Saving `input'.
Saving `output'.
Saving `forward'.
Saving `ppp-in'.
Saving `ppp-out'.
#
```

4.2.2 Utiliser ipchains-restore

`ipchains-restore` remet les chaînes que vous avez sauvé avec `ipchains-save`. Il peut prendre deux options : "-v" qui décrit chaque règle lorsqu'elle est ajoutée, et "-f" qui force le nettoyage des chaînes définies par l'utilisateur si elles existent, comme décrit plus bas.

Si une chaîne définie par l'utilisateur est trouvée à l'entrée, `ipchains-restore` vérifie que cette chaîne existe déjà. Si elle existe, alors vous serez interrogé pour savoir si la chaîne doit être nettoyée (suppression de toutes les règles) ou si la restauration de la chaîne doit être sautée. Si vous spécifiez "-f" sur la ligne de commande, vous ne serez pas interrogé ; la chaîne sera nettoyée.

Par exemple :

```
# ipchains-restore < my_firewall
Restoring ‘input’.
Restoring ‘output’.
Restoring ‘forward’.
Restoring ‘ppp-in’.
Chain ‘ppp-in’ already exists. Skip or flush? [S/f]? s
Skipping ‘ppp-in’.
Restoring ‘ppp-out’.
Chain ‘ppp-out’ already exists. Skip or flush? [S/f]? f
Flushing ‘ppp-out’.
#
```

5 Divers

Cette section contient toutes les informations et les questions fréquemment posées que je ne pouvais faire entrer dans la structure ci-dessus.

5.1 Comment organiser vos règles pare-feu

Cette question nécessite un peu de réflexion. Vous pouvez tenter de les organiser pour optimiser la vitesse (minimiser le nombre de vérifications de règles pour la plupart des paquets) ou pour augmenter la maniabilité.

Si vous avez une connexion intermittente, disons une connexion PPP, vous pouvez configurer la première règle de la chaîne d'entrée pour être '-i ppp0 -j DENY' au lancement, puis avoir quelque chose comme ceci dans votre script `ip-up` :

```
# Re-crée la chaîne "ppp-in"
ipchains-restore -f < ppp-in.firewall

# Remplace la règle DENY par un saut vers la chaîne se chargeant du ppp
ipchains -R input 1 -i ppp0 -j ppp-in
```

Votre script `ip-down` pourrait ressembler à ça :

```
ipchains -R input 1 -i ppp0 -j DENY
```

5.2 Ce qu'il ne faut pas filtrer

Il y a un certain nombre de choses auxquelles vous devez faire attention avant de commencer à filtrer quelque chose que vous n'auriez pas voulu filtrer.

5.2.1 Les paquets ICMP

Les paquets ICMP sont utilisés (entre autres choses) pour indiquer des problèmes aux autres protocoles (comme TCP et UDP). Les paquets "destination-unreachable" (destination non accessible) en particulier. Le bloquage de ces paquets signifie que vous n'obtiendrez jamais les erreurs "Host unreachable" ou "No route to host" ; toute connexion attendra une réponse qui ne viendra jamais. C'est irritant, mais rarement fatal.

Un problème plus inquiétant est le rôle des paquets ICMP dans la découverte MTU. Toutes les bonnes implémentations de TCP (y compris celle de Linux) utilisent la recherche MTU pour tenter de trouver quel est le plus grand paquet qui peut atteindre une destination sans être fragmenté (la fragmentation diminue les performances, principalement lorsque des fragments occasionnels sont perdus). La recherche MTU fonctionne en envoyant des paquets avec le bit "Don't Fragment" mis, et en envoyant ensuite des paquets plus petits s'il reçoit un paquet ICMP indiquant "Fragmentation needed but DF set" ('fragmentation-needed'). C'est un paquet de type "destination-unreachable", et s'il n'est jamais reçu, l'hôte local ne réduira pas le MTU, et les performances seront abyssales ou inexistantes.

Notez qu'il est commun de bloquer tous les messages ICMP de redirection (type 5) ; ils peuvent être utilisés pour manipuler le routage (bien que les piles IP bien conçues disposent de gardes-fou), et sont donc souvent considérés comme quelques peu risqués.

5.2.2 Connexions TCP au DNS (serveur de nom)

Si vous tentez de bloquer toutes les connexions TCP sortantes, rappellez vous que le DNS n'utilise pas toujours UDP ; si la réponse du serveur dépasse les 512 octets, le client utilise une connexion TCP (allant toujours sur le port numéro 53) pour obtenir les données.

Ceci peut être un piège car le DNS fonctionnera "en gros" si vous interdisez de tels transferts TCP ; vous pouvez expérimenter des délais longs et étranges, et d'autres problèmes liés au DNS si vous le faites.

Si les demandes de votre DNS sont toujours dirigées vers les mêmes sources externes (soit directement en utilisant la ligne `nameserver` dans `/etc/resolv.conf` ou en utilisant un serveur de noms cache en mode de redirection), alors vous n'aurez besoin d'autoriser que les connexions du port `domain` sur ce serveur de nom à partir du port local `domain` (si vous utilisez un serveur de nom cache) ou d'un port élevé (> 1023) si vous utilisez `/etc/resolv.conf`.

5.2.3 Cauchemars du FTP

L'autre problème classique du filtrage de paquets est celui posé par le FTP. Le FTP a deux **modes** ; le mode traditionnel est appelé **mode actif** et le plus récent est appelé **mode passif**. Les navigateurs web utilisent souvent le mode passif par défaut, mais les programmes de ftp en ligne de commmande utilisent en général par défaut le mode actif.

En mode actif, lorsque l'hôte distant désire envoyer un fichier (ou même les résultats d'une commande `ls` ou `dir`), il essaye d'ouvrir une connexion TCP sur la machine locale. Celà signifie que vous ne pouvez filtrez ces connexions TCP sans supprimer le FTP actif.

Si vous avez comme option l'utilisation du mode passif, alors tout va bien ; le mode passif fait passer les connexions de données du client au serveur, même pour les données arrivantes. Autrement, il est recommendé de n'autoriser que les connexions TCP vers les ports supérieurs à 1024 et de les interdire entre 6000 et 6010 (6000 est utilisé par X-Windows).

5.3 Filtrer le ping de la mort (Ping of Death)

Les machines Linux sont maintenant immunisées du fameux **Ping of Death**, qui implique l'envoi de paquets ICMP illégalement grands qui font déborder les buffers de la pile TCP du récepteur et causent de gros dégâts.

Si vous voulez protéger des machines qui peuvent être vulnérables, vos pouvez simplement bloquer les fragments ICMP. Les paquets ICMP normaux ne sont pas assez gros pour nécessiter la fragmentation, et vous ne casserez rien à part les gros pings. J'ai entendu parler (non confirmé) de rapports comme quoi quelques systèmes ont seulement besoin du dernier fragment d'un paquet ICMP déformé pour les corrompre, donc bloquer seulement le premier fragment n'est pas recommandé.

Même si tous les programmes exploitant cette erreur que j'ai vu utilisent l'ICMP, il n'y a pas de raisons qu'un fragment TCP ou UDP (ou d'un protocole inconnu) ne puisse être utilisé pour cette attaque, donc le bloquage des fragments ICMP est seulement une solution temporaire.

5.4 Filtrer teardrop et bonk

Teardrop et Bonk sont deux attaques (principalement contre les machines sous Microsoft Windows NT) qui reposent sur des fragments superposés. Avoir votre routeur Linux défragmenter, ou interdisant tous les fragments vers vos machines vulnérables sont d'autres options.

5.5 Filtrer les bombes à fragments

Quelques piles TCP moins fiables sont connues pour avoir des problèmes à gérer de larges ensembles de fragments de paquets lorsqu'elles ne reçoivent pas tous les fragments. Linux n'a pas ce problème. Vous pouvez filtrer les fragments (ce qui peut casser les utilisations légitimes) ou compiler votre noyau avec l'option "IP: always defragment" mise sur "Y" (seulement si votre machine Linux est la seule route possible pour ces paquets).

5.6 Changer les règles pare-feu

Il y a quelques problèmes de temps qui sont impliqués dans la modification des règles pare-feu. Si vous n'y faites pas attention, vous pouvez laisser entrer des paquets lorsque vous avez fait la moitié de vos changements. Une approche simpliste serait de faire la suite :

```
# ipchains -I input 1 -j DENY
# ipchains -I output 1 -j DENY
# ipchains -I forward 1 -j DENY

... Mise en place des changements ...

# ipchains -D input 1
# ipchains -D output 1
# ipchains -D forward 1
#
```

Ceci supprime tous les paquets pour la durée des changements.

Si vos changements sont restreints à une chaîne simple, vous pouvez créer une nouvelle chaîne avec les nouvelles règles, et ensuite remplacer ("-R") la règle qui pointait sur la vieille chaîne par une qui pointe sur la nouvelle chaîne ; ensuite, vous pouvez supprimer la vieille chaîne. Le remplacement se fera à vitesse atomique.

5.7 Comment mettre en place la protection contre l'IP spoof ?

L'IP spoofing est une technique dans laquelle un hôte envoie des paquets qui prétendent venir d'un autre hôte. Puisque le filtrage des paquets prend ses décisions sur la base de cette adresse source, l'IP spoofing est utilisé pour abuser les filtres de paquets. Elle est également utilisée pour cacher l'identité d'un attaquant utilisant les techniques SYN, Teardrop, Ping of Death et autres dérivés (ne vous inquiétez si vous ne savez pas ce que c'est).

Le meilleur moyen de se protéger de l'IP spoofing est la vérification de l'adresse source, et il est réalisé par le code de routage, et non par le pare-feu et autres.
Cherchez un fichier nommé `/proc/sys/net/ipv4/conf/all/rp_filter`. S'il existe, alors l'activation de la vérification de l'adresse source à chaque lancement est la bonne solution pour vous. Pour se faire, insérez les lignes suivantes quelque part dans vos scripts d'initialisation, avant l'initialisation des interfaces réseau :

```
# This is the best method: turn on Source Address Verification and get
# spoof protection on all current and future interfaces.
if [ -e /proc/sys/net/ipv4/conf/all/rp_filter ]; then
  echo -n "Setting up IP spoofing protection..."
  for f in /proc/sys/net/ipv4/conf/*/rp_filter; do
      echo 1 > $f
  done
  echo "done."
else
  echo PROBLEMS SETTING UP IP SPOOFING PROTECTION.  BE WORRIED.
  echo "CONTROL-D will exit from this shell and continue system startup."
  echo
  # Start a single user shell on the console
  /sbin/sulogin $CONSOLE
fi
```

Si vous ne pouvez faire ceci, vous pouvez insérer manuellement les règles pour protéger chaque interface. Ceci nécessite la connaissance de chaque interface. La série des noyaux 2.1 et supérieurs rejette automatiquement les paquets qui prétendent venir des adresses 127.* (réservées pour l'interface de loopback, `lo`).

Par exemple, disons que vous avez trois interfaces, `eth0`, `eth1` et `ppp0`. Nous pouvons utiliser `ifconfig` pour nous donner les adresses et les masques de réseau de chaque interface. Disons que `eth0` est rattachée à un réseau 192.168.1.0 avec le masque de sous-réseau 255.255.255.0, `eth1` est raccordée à un réseau 10.0.0.0 avec le masque de sous-réseau 255.0.0.0, et `ppp0` connectée à l'Internet (où toutes les adresses sauf les adresses IP privées réservées sont autorisées), nous insérerions les règles suivantes :

```
# ipchains -A input -i eth0 -s ! 192.168.1.0/255.255.255.0 -j DENY
# ipchains -A input -i ! eth0 -s 192.168.1.0/255.255.255.0 -j DENY
# ipchains -A input -i eth1 -s ! 10.0.0.0/255.0.0.0 -j DENY
# ipchains -A input -i ! eth1 -s 10.0.0.0/255.0.0.0 -j DENY
#
```

Cette approche n'est pas aussi bonne que l'approche par vérification de l'adresse source, parce que si votre réseau change, vous devrez changer vos règles pare-feu pour être à jour.

Si vous utilisez un noyau de série 2.0, vous pouvez également vouloir protéger l'interface loopback, en utilisant une règle telle que :

```
# ipchains -A input -i ! lo -s 127.0.0.0/255.0.0.0 -j DENY
#
```

5.8 Projets avancés

Il y a une bibliothèque dans l'espace utilisateur que j'ai écrit et qui est inclue avec la distribution source, nommée "libfw". Elle utilise la capacité de IP Chains 1.3 et suivants pour copier un paquet vers l'espace utilisateur (via l'option du noyau IP_FIREWALL_NETLINK).

La valeur "mark" peut être utilisée pour spécifier les paramètres de Qualité de Service pour les paquets, ou pour spécifier comment les paquets doivent être renvoyés. Je ne l'ai cependant jamais utilisée, mais si vous voulez écrire sur ce sujet, contactez moi.

Des choses comme le **stateful inspection** (je préfère le terme "pare-feu dynamique") peuvent être implémentées dans l'espace utilisateur en utilisant cette bibliothèque. D'autres idées géniales peuvent être le contrôle des paquets sur une base "par utilisateur" en faisant une demande sur un démon résidant dans l'espace utilisateur. Ceci devrait être relativement simple.

5.8.1 SPF : Stateful Packet Filtering

ftp://ftp.interlinx.bc.ca/pub/spf `ftp://ftp.interlinx.bc.ca/pub/spf` est le site du projet SPF de Brian Murrell, qui permet le suivi de connexion dans l'espace utilisateur. Ceci ajoute une dose significative de sécurité pour les sites à faible bande passante.

Il y a peu de documentation pour le moment, mais voici un message que Brian a envoyé à la liste de diffusion en répondant à quelques questions :

```
> Je présume qu'il fait exactement ce que je veux~: installer une règle de
> "retour" temporaire pour laisser entrer des paquets en réponse à une
> requête extérieure.

Yup, c'est exactement ce à quoi il sert. Plus il comprendra de protocoles,
plus il obtiendra de règles de retour exactes. Pour l'instant il supporte
(de mémoire, excusez les erreurs ou les omissions) le FTP (à la fois actif et
passif, intérieur et extérieur), un peu de RealAudio, de traceroute, d'ICMP et
d'un ICQ basique (transmission des serveurs ICQ, connexions TCP directes, mais
hélas la seconde connexion directe TCP pour des trucs comme le transfert de
fichiers, etc., ne sont pas encore présentes)

> S'agit-il d'un remplaçant pour ipchains ou d'un ajout~?

C'est un ajout. Pensez à ipchains comme étant le moteur pour autoriser et
empêcher les paquets de traverser une machine Linux. SPF est le pilote, gérant
et surveillant constamment le traffic et disant à ipchains comment changer ses
polices pour refléter les changements dans les schémas du traffic.
```

5.8.2 Modification des données ftp par Michael Hasenstein

Michael Hasenstein de SuSE a codé un correctif pour le noyau qui ajoute le suivi des connexions ftp à ipchains. Celui-ci peut être trouvé sur *http://www.csn.tu-chemnitz.de/~mha/patch.ftp-data-2.gz* `http://www.csn.tu-chemnitz.de/~mha/patch.ftp-data-2.gz`

5.9 Extensions futures

Les codes de pare-feu et de NAT sont en cours de remise à jour pour le 2.3. Les plans et les discussions sont disponibles sur l'archive de netdev, et sur la liste ipchains-dev. Ces extensions doivent nettoyer de nombreux problèmes d'utilisation (réellement, la mise en place du pare-feu et le camouflage ne devrait pas être *aussi dur*, et devrait autoriser une croissance pour un pare-feu beaucoup plus flexible).

6 Problèmes classiques

6.1 ipchains -L est gelé !

Vous bloquez probablement les demandes DNS ; il finira probablement par stopper. Essayez d'utiliser l'option "-n" (numérique) d'ipchains, qui supprimera la recherche des noms.

6.2 Le camouflage/redirection ne fonctionne pas !

Vérifiez si la redirection de paquets est activée (dans les noyaux récent, elle est désactivée par défaut ce qui signifie que les paquets ne tentent jamais de traverser la chaîne "forward"). Vous pouvez changer ce défaut (en tant que super-utilisateur) en tapant

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
#
```

Si ceci marche pour vous, vous pouvez le mettre quelque part dans vos scripts de lancement, de manière à ce que la redirection soit activée à chaque fois ; vous devrez toutefois configurer votre pare-feu avant le lancement de cette commande, autrement il peut y avoir passage de paquets.

6.3 -j REDIR ne marche pas !

Vous devez autoriser la retransmission des paquets (voir plus haut) pour que celle-ci fonctionne ; sinon le code de routage supprime le paquet. Ainsi, si vous utilisez juste la redirection sans avoir de transmission, vous devez y faire attention.

Notez que REDIR (dans la chaîne d'entrée) n'affecte pas les connexions d'un processus local.

6.4 Les interfaces joker ne fonctionnent pas !

Il y avait une erreur dans les versions 2.1.102 et 2.1.103 du noyau (et dans quelques anciens correctifs que j'avais sorti) qui faisait planter les commandes ipchains utilisant une interface joker (comme `-i ppp+`).

Cette erreur a été corrigée dans les noyaux récents, et dans le correctif 2.0.34 du site web. Vous pouvez aussi le corriger à la main dans les sources du noyau en changeant la ligne 63 de include/linux/ip_fw.h :

```
#define IP_FW_F_MASK    0x002F  /* All possible flag bits mask   */
```

Ceci doit en fait être "0x003F". Corrigez et recompilez le noyau.

6.5 TOS ne fonctionne pas !

C'est de ma faute : la configuration du champ Type of Service ne change pas le Type of Service dans les noyaux 2.1.102 à 2.1.111. Ce problème a été corrigé dans le 2.1.112.

6.6 ipautofw et ipportfw ne fonctionnent pas !

Pour les 2.0.x, c'est vrai ; je n'ai pas le temps de créer et de maintenir un correctif de taille gigantesque pour ipchains et ipautofw/ipportfw.

Pour les 2.1.x, récupérez ipmasqadm de Juan Ciarlante sur

```
<url url="http://juanjox.linuxhq.com/"
        name="http://juanjox.linuxhq.com/">
```

et utilisez-le exactement de la manière dont vous auriez utilisé `ipautofw` ou `ipportfw`, à part qu'à la place de `ipportfw` vous devez taper `ipmasqadm portfw`, et à la place de `ipautofw` vous devez taper `ipmasqadm autofw`.

6.7 XosView ne marche pas !

Mettez à jour à la version 1.6.0 ou supérieure, qui ne nécessite pas de règle pare-feu pour les noyaux 2.1.x. Il semblerait être à nouveau cassé dans le 1.6.1 ; veuillez voir l'auteur (ce n'est pas de ma faute !).

6.8 Erreur de segmentation avec -j REDIRECT !

C'était une erreur dans ipchains version 1.3.3. Veuillez mettre à jour.

6.9 Je ne peux pas modifier les temps d'attente du camouflage !

C'est vrai (pour les noyaux 2.1.x) jusqu'au et incluant le 2.1.112. Ceci est pour le moment vigoureusement traqué, et au moment où vous lirez ce document il se pourrait que ce soit résolu. Ma page web contiendra un correctif lorsqu'il sera disponible.

6.10 Je veux firewaller IPX !

Ainsi qu'un certain nombre d'autres personnes, il semble. Mon code couvre seulement IP, malheureusement. Du bon côté, tout est là pour pouvoir firewaller IPX ! Vous devrez juste écrire le code ; je vous aiderai joyeusement là où ce sera possible.

7 Un exemple sérieux

Cet exemple est extrait du tutorial donné par Michael Neuling et moi-même en mars 1999 lors du Linux-World ; ce n'est pas le seul moyen de régler le problème donné, mais c'est probablement le plus simple. J'espère que vous le jugerez informatif.

7.1 L'arrangement

- Un réseau interne camouflé (sous divers systèmes d'exploitation), que nous appellerons "BON" ;
- des serveurs exposés dans un réseau séparé (nommé "ZDM" pour Zone Démilitarisée) ;
- une connexion PPP à l'Internet (nommé "MAUVAIS").

```
Réseau externe (MAUVAIS)
        |
        |
    ppp0|
 ---------------
 | 192.84.219.1|             Réseau des serveurs (ZDM)
 |             |eth0
 |             |-----------------------------------------------
 |             |192.84.219.250 |             |              |
 |             |               |             |              |
 |192.168.1.250|               |             |              |
 ---------------          --------       -------        -------
        | eth1            | SMTP |       | DNS |        | WWW |
        |                 --------       -------        -------
        |              192.84.219.128  192.84.219.129  192.84.218.130
        |
Réseau Interne (BON)
```

7.2 Buts

Sur la machine filtrant les paquets :

PING tout réseau

Très utile si la machine est hors-service.

TRACEROUTE tout réseau

Une fois de plus, utile pour les diagnostics.

Accès au DNS

Pour rendre ping et DNS plus utile.

À l'intérieur de la Zone Démilitarisée :

Serveur mail

- SMTP vers l'extérieur
- Accepte le SMTP de l'intérieur et de l'extérieur
- Accepte le POP3 de l'intérieur

Serveur de nom

- Envoi de requêtes DNS vers l'extérieur
- Accepte les requêtes DNS de l'intérieur, l'extérieur, et du filtre de paquets

Serveur web

- Accepte les requêtes HTTP de l'intérieur et de l'extérieur
- Accès rsync de l'intérieur

En interne :

Autorise WWW, ftp, traceroute et ssh vers l'extérieur

Ce sont des services standards à autoriser : on autorise parfois les machines internes à tout faire, mais ici nous serons plus restrictifs.

Autorise le SMTP vers le serveur mail

Bien entendu, nous voulons qu'il leur soit possible d'envoyer du courrier vers l'extérieur.

Autorise le POP3 vers le serveur mail

C'est ainsi qu'ils lisent leur courrier.

Autorise les requêtes DNS vers le serveur de nom

Ils doivent pouvoir rechercher des noms externes pour le web, le ftp, traceroute ou ssh.

Autorise rsync vers le serveur web

C'est ainsi qu'ils synchronisent le serveur web externe avec l'interne.

Autorise les requêtes WWW vers le serveur web

Bien entendu, nous voulons qu'ils puissent se connecteur au serveur web externe.

Autorise le ping vers le filtre de paquets

Il est courtois de l'autoriser ; ils peuvent ainsi tester si le pare-feu est coupé (ainsi nous ne serons pas tenus responsables si un site extérieur est coupé).

7.3 Avant le filtrage des paquets

- Protection anti-spoofing

  Puisque nous n'avons pas de routage asymétrique, nous pouvons simplement mettre en marche l'anti-spoofing pour toutes les interfaces.

  ```
  # for f in /proc/sys/net/ipv4/conf/*/rp_filter; do echo 1 > $f; done
  #
  ```

- Mettre en place des règles de filtrage en interdiction (DENY) totale :

  Nous autorisons tout de même le traffic local, mais nous interdisons tout le reste.

  ```
  # ipchains -A input -i ! lo -j DENY
  # ipchains -A output -i ! lo -j DENY
  # ipchains -A forward -j DENY
  #
  ```

- Configurer les interfaces

  Ceci est généralement réalisé par les scripts de lancement. Faîtes bien attention à ce que les règles ci-dessus soient insérées avant que les interfaces ne soient configurées, afin de prévenir le passage de paquets avant l'insertion des règles.

- Insertion des modules de camouflage par protocole Nous devons insérer le module de camouflage du FTP, ainsi le ftp passif et actif fonctionnera 'uniquement' du réseau interne.

  ```
  # insmod ip_masq_ftp
  #
  ```

7.4 Filtrage de paquets pour les paquets traversants

Avec le camouflage, il vaut mieux filtrer la chaîne de retransmission.

Cassez la chaîne de retransmission en plusieurs chaînes utilisateurs dépendant des interfaces sources/destination ; ceci ramène le problème à des problèmes plus gérables.

```
ipchains -N bon-zdm
ipchains -N mauvais-zdm
ipchains -N bon-mauvais
ipchains -N zdm-bon
ipchains -N zdm-mauvais
ipchains -N mauvais-bon
```

ACCEPTer les codes standards d'erreur ICMP est un fait classique, nous lui créons donc une chaîne.

```
ipchains -N icmp-acc
```

7.4.1 Configurer les sauts de la chaîne de transmission

Malheureusement, nous connaissons seulement (dans la chaîne de transmission) quelle est l'interface externe. Ainsi, pour se représenter de quelle interface vient le paquet, nous utilisons l'adresse source (l'anti-spoofing évite les problèmes liés aux adresses).

Notez que nous enregistrons tout ce qui ne vérifie aucune de ces règles (cependant, ceci ne devrait jamais arriver).

```
ipchains -A forward -s 192.168.1.0/24 -i eth0 -j bon-zdm
ipchains -A forward -s 192.168.1.0/24 -i ppp0 -j bon-mauvais
ipchains -A forward -s 192.84.219.0/24 -i ppp0 -j zdm-mauvais
ipchains -A forward -s 192.84.219.0/24 -i eth1 -j zdm-bon
ipchains -A forward -i eth0 -j mauvais-zdm
ipchains -A forward -i eth1 -j mauvais-bon
ipchains -A forward -j DENY -l
```

7.4.2 Définir la chaîne icmp-acc

Les paquets correspondant à l'une des erreurs ICMP sont acceptés, sinon le contrôle les rendra à la chaîne appellante.

```
ipchains -A icmp-acc -p icmp --icmp-type destination-unreachable -j ACCEPT
ipchains -A icmp-acc -p icmp --icmp-type source-quench -j ACCEPT
ipchains -A icmp-acc -p icmp --icmp-type time-exceeded -j ACCEPT
ipchains -A icmp-acc -p icmp --icmp-type parameter-problem -j ACCEPT
```

7.4.3 Bon (interne) vers ZDM (serveurs)

Restrictions internes :

- Autorise WWW, ftp, traceroute, ssh vers l'extérieur
- **Autorise le SMTP vers le serveur mail**

- **Autorise le POP3 vers le serveur mail**
- **Autorise les requêtes DNS vers le serveur de nom**
- **Autorise le rsync vers le serveur web**
- **Autorise le WWW vers le serveur web**
- Autorise le ping vers le filtre de paquets

On pourrait utiliser le camouflage du réseau interne vers la ZDM, mais ici nous ne le ferons pas. Puisque personne du réseau interne ne devrait tenter de choses démoniaques, nous enregistrons les paquets qui sont interdits.

```
ipchains -A bon-zdm -p tcp -d 192.84.219.128 smtp -j ACCEPT
ipchains -A bon-zdm -p tcp -d 192.84.219.128 pop-3 -j ACCEPT
ipchains -A bon-zdm -p udp -d 192.84.219.129 domain -j ACCEPT
ipchains -A bon-zdm -p tcp -d 192.84.219.129 domain -j ACCEPT
ipchains -A bon-zdm -p tcp -d 192.84.218.130 www -j ACCEPT
ipchains -A bon-zdm -p tcp -d 192.84.218.130 rsync -j ACCEPT
ipchains -A bon-zdm -p icmp -j icmp-acc
ipchains -A bon-zdm -j DENY -l
```

7.4.4 Mauvais (extérieur) vers ZDM (serveurs)

- Restrictions de la ZDM :
  - Serveur mail :
    * **SMTP vers l'extérieur**
    * **Accepte le SMTP venant** de l'intérieur et **extérieur**
    * Accepte le POP3 de l'intérieur
  - Serveur de noms :
    * **Envoi de requêtes DNS vers l'extérieur**
    * **Accepte le DNS venant** de l'intérieur, **extérieur** et du filtre de paquets
  - Serveur web :
    * **Accepte les requêtes HTTP venant de** l'intérieur et de **l'extérieur**
    * Accès rsync de l'intérieur
- Les trucs à autoriser du réseau extérieur vers la ZDM
  - N'enregistre pas les violations, elles peuvent arriver.

```
ipchains -A mauvais-zdm -p tcp -d 192.84.219.128 smtp -j ACCEPT
ipchains -A mauvais-zdm -p udp -d 192.84.219.129 domain -j ACCEPT
ipchains -A mauvais-zdm -p tcp -d 192.84.219.129 domain -j ACCEPT
ipchains -A mauvais-zdm -p tcp -d 192.84.218.130 www -j ACCEPT
ipchains -A mauvais-zdm -p icmp -j icmp-acc
ipchains -A mauvais-zdm -j DENY
```

7.4.5 Bon (intérieur) vers Mauvais (extérieur).

- Restrictions internes :
  - **Autorise le WWW, ftp, traceroute, ssh vers l'extérieur**
  - Autorise SMTP vers le serveur mail
  - Autorise POP-3 vers le serveur mail
  - Autorise DNS vers le serveur de nom
  - Autorise rsync vers le serveur web
  - Autorise WWW vers le serveur web
  - Autorise ping vers le filtre de paquets
- Un grand nombre de gens autorisent tout venant de l'intérieur vers les réseaux extérieurs, puis ajoutent des restrictions. Nous sommes fascistes.
  - Enregistre les violations.
  - Le ftp passif est géré par le module de camouflage.

```
ipchains -A bon-mauvais -p tcp --dport www -j MASQ
ipchains -A bon-mauvais -p tcp --dport ssh -j MASQ
ipchains -A bon-mauvais -p udp --dport 33434:33500 -j MASQ
ipchains -A bon-mauvais -p tcp --dport ftp --j MASQ
ipchains -A bon-mauvais -p icmp --icmp-type ping -j MASQ
ipchains -A bon-mauvais -j REJECT -l
```

7.4.6 ZDM vers Bon (intérieur)

- Restrictions internes :
  - Autorise WWW, ftp, traceroute, ssh vers l'extérieur
  - **Autorise SMTP vers le serveur mail**
  - **Autorise POP3 vers le serveur mail**
  - **Autorise DNS vers le serveur de noms**
  - **Autorise rsync vers le serveur web**
  - **Autorise WWW vers le serveur web**
  - Autorise ping vers le filtre de paquets
- Si nous camouflions le réseau intérieur de la ZDM, nous refuserions simplement les paquets venant d'un autre moyen. Tel quel, il autorise uniquement les paquets qui peuvent provenir d'une connexion pré-établie.

```
ipchains -A zdm-bon -p tcp ! -y -s 192.84.219.128 smtp -j ACCEPT
ipchains -A zdm-bon -p udp -s 192.84.219.129 domain -j ACCEPT
ipchains -A zdm-bon -p tcp ! -y -s 192.84.219.129 domain -j ACCEPT
ipchains -A zdm-bon -p tcp ! -y -s 192.84.218.130 www -j ACCEPT
ipchains -A zdm-bon -p tcp ! -y -s 192.84.218.130 rsync -j ACCEPT
ipchains -A zdm-bon -p icmp -j icmp-acc
ipchains -A zdm-mauvais -j DENY -l
```

7.4.7 ZDM vers Mauvais (extérieur)

- Restrictions de la ZDM :
  - Serveur mail
    - **SMTP vers l'extérieur**
    - **Accepte SMTP venant de** l'intérieur et de **l'extérieur**
    - Accepte POP3 venant de l'intérieur
  - Serveur de noms
    - **Envoi de requêtes DNS vers l'extérieur**
    - **Accepte le DNS de** l'intérieur, **l'extérieur** et du filtre de paquets
  - Serveur web
    - **Accepte HTTP venant de** l'intérieur et de **l'extérieur**
    - Accès rsync de l'intérieur
-
```
ipchains -A zdm-mauvais -p tcp -s 192.84.219.128 smtp -j ACCEPT
ipchains -A zdm-mauvais -p udp -s 192.84.219.129 domain -j ACCEPT
ipchains -A zdm-mauvais -p tcp -s 192.84.219.129 domain -j ACCEPT
ipchains -A zdm-mauvais -p tcp ! -y -s 192.84.218.130 www -j ACCEPT
ipchains -A zdm-mauvais -p icmp -j icmp-acc
ipchains -A zdm-mauvais -j DENY -l
```

7.4.8 Mauvais (extérieur) vers Bon (intérieur)

- Nous n'autorisons rien (de non camouflé) à passer du réseau extérieur vers le réseau intérieur.

```
ipchains -A mauvais-bon -j REJECT
```

7.4.9 Filtrage de paquets pour la machine Linux elle-même

- Si nous désirons utiliser le filtrage de paquets sur les paquets arrivant sur la machine elle-même, nous devons filtrer la chaîne d'entrée. Nous créons une chaîne pour chaque interface de destination :

```
ipchains -N mauvais-if
ipchains -N zdm-if
ipchains -N bon-if
```

- Créons les sauts vers elles :

```
ipchains -A input -d 192.84.219.1 -j mauvais-if
ipchains -A input -d 192.84.219.250 -j zdm-if
ipchains -A input -d 192.168.1.250 -j bon-if
```

Interface Mauvais (extérieur)

- Machine de filtrage des paquets :
  - **PING tous les réseaux**
  - **TRACEROUTE tous les réseaux**
  - Accès DNS
- L'interface extérieure reçoit aussi des réponses aux paquets camouflés, les erreurs ICMP leur correspondant et les réponses PING.

```
ipchains -A mauvais-if -i ! ppp0 -j DENY -l
ipchains -A mauvais-if -p TCP --dport 61000:65096 -j ACCEPT
ipchains -A mauvais-if -p UDP --dport 61000:65096 -j ACCEPT
ipchains -A mauvais-if -p ICMP --icmp-type pong -j ACCEPT
ipchains -A mauvais-if -j icmp-acc
ipchains -A mauvais-if -j DENY
```

Interface ZDM

- Restrictions du filtre de paquets :
  - **PING tous les réseaux**
  - **TRACEROUTE tous les réseaux**
  - **Accès DNS**
- L'interface ZDM reçoit les réponses DNS, ping et les erreurs ICMP

```
ipchains -A zdm-if -i ! eth0 -j DENY
ipchains -A zdm-if -p TCP ! -y -s 192.84.219.129 53 -j ACCEPT
ipchains -A zdm-if -p UDP -s 192.84.219.129 53 -j ACCEPT
ipchains -A zdm-if -p ICMP --icmp-type pong -j ACCEPT
ipchains -A zdm-if -j icmp-acc
ipchains -A zdm-if -j DENY -l
```

Interface Bon (intérieur)

- Restrictions du filtre de paquets
  - **PING tous les réseaux**
  - **TRACEROUTE tous les réseaux**
  - **Accès DNS**
- Restrictions intérieures :
  - Autorise WWW, ftp, traceroute, ssh vers l'extérieur
  - Autorise SMTP vers le serveur mail
  - Autorise POP3 vers le serveur mail
  - Autorise DNS vers le serveur de noms
  - Autorise rsync vers le serveur web
  - Autorise WWW vers le serveur web
  - **Autorise ping vers le filtre de paquets**
- L'interface intérieure reçoit les pings, les réponses ping et les erreurs ICMP.

```
ipchains -A bon-if -i ! eth1 -j DENY
ipchains -A bon-if -p ICMP --icmp-type ping -j ACCEPT
ipchains -A bon-if -p ICMP --icmp-type pong -j ACCEPT
ipchains -A bon-if -j icmp-acc
ipchains -A bon-if -j DENY -l
```

7.5 Finalement

- Supprime les règles de bloquage :

```
ipchains -D input 1
ipchains -D forward 1
ipchains -D output 1
```

8 Annexe : différences entre ipchains et ipfwadm

Quelques-uns de ces changements sont le résultat de changements du noyau, et quelques autres sont un résultat des différences entre `ipchains` et `ipfwadm`.

1. De nombreux arguments ont été modifiés : les majuscules indiquent dorénavant une commande, et les minuscules indiquent une option.
2. Les chaînes arbitraires sont supportées, ainsi même les chaînes intégrées ont des noms complets plutôt que des options (càd, "input" à la place de "-I").
3. L'option "-k" a sauté : utilisez "! -y".
4. L'option "-b" insère/ajoute/supprime actuellement deux règles, plutôt qu'une règle "bidirectionnelle" simple.
5. L'option "-b" peut être passée à "-C" pour effectuer deux vérifications (une dans chaque direction).
6. L'option "-x" de "-l" a été remplacée par "-v".
7. Les ports sources et destinations multiples ne sont plus supportés. Heureusement, la possibilité d'employer un intervalle de port aura le même effet.
8. Les interfaces peuvent seulement être spécifiées par leur nom (pas par leurs adresses). L'ancienne sémantique a été silencieusement changée dans la série des noyaux 2.1, de toute manière.
9. Les fragments sont examinés, mais pas automatiquement autorisés.
10. On s'est débarrassé des chaînes de comptage explicites.
11. On peut écrire des règles qui porteront uniquement sur certains protocoles IP.
12. L'ancien comportement de vérification de SYN et ACK (qui était auparavant ignoré pour les paquets non TCP) a changé ; l'option SYN n'est plus valide pour les règles non spécifiques au TCP.
13. Les compteurs sont maintenant de 64 bits sur les machines 32 bits, et non plus 32 bits.
14. L'inversion des options est dorénavant supportée.
15. Les codes ICMP sont maintenant supportés.
16. Les interfaces joker sont maintenant supportées.
17. Les manipulations de TOS sont maintenant vérifiées : l'ancien code du noyau les stoppait silencieusement pour vous en manipulant (illégalement) le bit TOS "Must Be Zero" ; ipchains retourne maintenant une erreur si vous essayez, de même que pour d'autres cas illégaux.

8.1 Guide de référence rapide

[Dans l'ensemble, les arguments des commandes sont en MAJUSCULES, et les options des arguments sont en minuscules]

Une chose à noter, le camouflage est spécifié par "-j MASQ" ; ceci est complètement différent de "-j ACCEPT", et n'est pas traité comme un effet de bord, au contraire d'`ipfwadm`.

```
============================================================================
| ipfwadm     | ipchains               | Notes
----------------------------------------------------------------------------
| -A [both]   | -N acct                | Crée une chaîne "acct"
|             |& -I 1 input -j acct    | ayant des paquets entrants et
|             |& -I 1 output -j acct   | sortants qui la traversent.
|             |& acct                  |
----------------------------------------------------------------------------
| -A in       | input                  | Une règle sans destination
----------------------------------------------------------------------------
| -A out      | output                 | Une règle sans destination
----------------------------------------------------------------------------
| -F          | forward                | Utilise ça comme [chaîne].
----------------------------------------------------------------------------
| -I          | input                  | Utilise ça comme [chaîne].
----------------------------------------------------------------------------
| -O          | output                 | Utilise ça comme [chaîne].
----------------------------------------------------------------------------
| -M -l       | -M -L                  |
----------------------------------------------------------------------------
| -M -s       | -M -S                  |
----------------------------------------------------------------------------
| -a policy   | -A [chain] -j POLICY   | (voir -r et -m).
----------------------------------------------------------------------------
| -d policy   | -D [chain] -j POLICY   | (voir -r et -m).
----------------------------------------------------------------------------
| -i policy   | -I 1 [chain] -j POLICY| (voir -r et -m).
----------------------------------------------------------------------------
| -l          | -L                     |
----------------------------------------------------------------------------
| -z          | -Z                     |
----------------------------------------------------------------------------
| -f          | -F                     |
----------------------------------------------------------------------------
| -p          | -P                     |
----------------------------------------------------------------------------
| -c          | -C                     |
----------------------------------------------------------------------------
| -P          | -p                     |
----------------------------------------------------------------------------
| -S          | -s                     | Prend seulement un port ou un
|             |                        | intervalle, pas de multiples.
----------------------------------------------------------------------------
```

```
| -D            | -d                      | Prend seulement un port ou un
|               |                         | intervalle, pas de multiples.
----------------------------------------------------------------------------
| -V            | <none>                  | Utilise -i [nom].
----------------------------------------------------------------------------
| -W            | -i                      |
----------------------------------------------------------------------------
| -b            | -b                      | Dorénavant crée deux règles.
----------------------------------------------------------------------------
| -e            | -v                      |
----------------------------------------------------------------------------
| -k            | ! -y                    | Ne fonctionne pas à moins que
|               |                         | -p tcp ne soit également spécifié.
----------------------------------------------------------------------------
| -m            | -j MASQ                 |
----------------------------------------------------------------------------
| -n            | -n                      |
----------------------------------------------------------------------------
| -o            | -l                      |
----------------------------------------------------------------------------
| -r [redirpt]  | -j REDIRECT [redirpt]   |
----------------------------------------------------------------------------
| -t            | -t                      |
----------------------------------------------------------------------------
| -v            | -v                      |
----------------------------------------------------------------------------
| -x            | -x                      |
----------------------------------------------------------------------------
| -y            | -y                      | Ne fonctionne pas à moins que
|               |                         | -p tcp ne soit également spécifié.
----------------------------------------------------------------------------
```

8.2 Exemples de commandes ipfwadm traduites

Ancienne commande : ipfwadm -F -p deny

Nouvelle commande : ipchains -P forward DENY

Ancienne commande : ipfwadm -F -a m -S 192.168.0.0/24 -D 0.0.0.0/0

Nouvelle commande : ipchains -A forward -j MASQ -s 192.168.0.0/24 -d 0.0.0.0/0

Ancienne commande : ipfwadm -I -a accept -V 10.1.2.1 -S 10.0.0.0/8 -D 0.0.0.0/0

Nouvelle commande : ipchains -A input -j ACCEPT -i eth0 -s 10.0.0.0/8 -d 0.0.0.0/0

(Notez qu'il n'y a pas d'équivalent pour la spécification des interfaces par leur adresse : utilisez le nom de l'interface. Sur cette machine, 10.1.2.1 correspond à eth0).

9 Annexe : utiliser le script ipfwadm-wrapper

Le script shell `ipfwadm-wrapper` doit être un remplacement d'`ipfwadm` pour la compatibilité descendante avec ipfwadm 2.3a.

La seule option qu'il ne peut vraiment pas supporter est l'option "-V". Lorsqu'elle est utilisée, un avertissement est affiché. Si l'option "-W" est également utilisée, l'option "-V" est ignorée. Autrement, le script essaye de trouver le nom de l'interface associée à cette adresse, en utilisant `ifconfig`. Si ça ne marche pas (comme pour une interface désactivée), alors il sortira avec un message d'erreur.

Cet avertissement peut être supprimé soit en changeant le "-V" pour un "-W", ou en dirigeant la sortie standard du script vers /dev/null.

Si vous trouvez des erreurs dans ce script, ou une modification entre les effets du vrai ipfwadm et de ce script, *veuillez* me rapporter le problème : envoyez un courrier à ipchains@rustcorp.com avec le sujet "BUG-REPORT". Veuillez lister la version d'`ipfwadm` (`ipfwadm -h`), votre version d'`ipchains` (`ipchains -version`), la version du script d'emballage (`ipfwadm-wrapper -version`). Envoyez moi également la sortie de `ipchains-save`. Merci d'avance.

Le mélange d'`ipchains` avec le script `ipfwadm-wrapper` se fait à votre propre péril.

10 Annexe : remerciements

Un grand merci à Michael Neuling, qui a écrit la pré-version du code d'IP chains en travaillant pour moi. Des excuses publiques pour avoir rejeté son idée de cache des résultats, qu'Alan Cox a proposé plus tard et que j'ai finalement commencé à implémenter, ayant vu l'erreur de mon côté.

Merci à Alan Cox pour son support technique par email 24h/24, et pour ses encouragements.

Merci à tous les auteurs du code d'ipfw et d'ipfwadm, spécialement Jos Vos. Rester aux chevilles des géants et tout ça... Ceci s'applique également à Linux Torvalds et à tous les bricoleurs du noyau et de l'espace utilisateur.

Merci aux beta testeurs, chasseurs d'erreurs diligents, surtout Jordan Mendelson, Shaw Carruthers, Kevin Moule, Dr. Liviu Daia, Helmut Adams, Franck Sicard, Kevin Littlejohn, Matt Kemner, John D. Hardin, Alexey Kuznetsov, Leos Bitto, Jim Kunzman, Gerard Gerritsen, Serge Sivkov, Andrew Burgess, Steve Schmidtke, Richard Offer, Bernhard Weisshuhn, Larry Auton, Ambrose Li, Pavel Krauz, Steve Chadsey, Francesco Potorti' et Alain Knaff.