

IpTables par l'exemple

Arnaud de Bermingham

IpTables par l'exemple

Contact : duracell@chez.apinc.point.org
révision par Jice <jice@chez.lea-linux.point.org>
révision par Fred <fred@chez.lea-linux.point.org>

Introduction

Cet article présente de façon pratique la mise en place d'un firewall / proxy sur une machine Linux tournant avec un noyau 2.4. Pour des informations plus théoriques sur les firewall/proxies, vous pouvez lire l'[article firewall](#).

* Présentation d'IpTables

IpTables est une solution complète de firewall (noyau 2.4) remplaçant *ipchains* (noyau 2.2) tournant sous le système GNU/Linux. IpTables permet de faire du firewalling *stateful* (à états), de la translation de port et d'adresse, du filtrage au niveau 2 et beaucoup d'autres choses que nous n'allons pas aborder comme le "Mangle" ou modification des paquets à la volée (atmosphère).

IpTables est fiable et dispose de très nombreuses options qui permettent de faire du filtrage très fin.

* Licence de cet article

Cette documentation est sous licence LDP

* Licence de NetFilter

NetFilter est sous licence libre GPL, i.e. gratuit et modifiable du moment que les modifications et améliorations apportées soit rendues publiques.

1/ Installation

1.1/ Prérequis

IpTables est installé en standard sur de nombreuses distributions Linux récentes. En particulier, il est installé sur Linux RedHat 7.1 et sur la plupart des distributions basées sur un kernel 2.4.x.

IpTables a besoin d'un **kernel de génération 2.4** compilé avec des options spéciales. Ceci ne pose pas de problèmes avec les noyaux 2.4 génériques des principales distributions basées sur cette génération de kernel.

1.2/ Options de compilation du kernel

Si vous désirez re-compiler votre kernel, il faut spécifier les options nécessaires au fonctionnement d'iptables.

Les options suivantes doivent être activées en module (M) ou dans le kernel (Y) :

```
CONFIG_PACKET
CONFIG_NETFILTER

CONFIG_IP_NF_CONNTRACK
CONFIG_IP_NF_FTP
CONFIG_IP_NF_IRC
CONFIG_IP_NF_IPTABLES
CONFIG_IP_NF_FILTER
CONFIG_IP_NF_NAT
CONFIG_IP_NF_MATCH_STATE
CONFIG_IP_NF_TARGET_LOG
CONFIG_IP_NF_MATCH_LIMIT
CONFIG_IP_NF_TARGET_MASQUERADE
```

et éventuellement :

```
CONFIG_IP_NF_COMPAT_IPCHAINS pour garder la compatibilité avec ipchains.
CONFIG_IP_NF_COMPAT_IPFWADM pour garder la compatibilité avec ipfwadm.
CONFIG_IP_NF_TARGET_REDIRECT indispensable, pour les proxies transparents par exemple.
CONFIG_IP_NF_MATCH_MAC permet de matcher avec les adresses MAC.
```

Ne pas oublier le support réseau et TCP/IP et compiler le kernel comme d'habitude :
make dep && make clean && make bzImage && make modules && make modules_install

1.3/ Installation

Sur une RedHat 7.1, récupérer le package `netfilter` et l'installer comme d'habitude avec `rpm -Uvh netfilter-x.y.z.rpm`.
Sur les versions inférieures à la 7.1, il faut compiler un kernel 2.4 car iptables ne supporte pas les kernels 2.2.x. La compilation de iptables est complexe. Le mieux est de lire le fichier `INSTALL` du package des sources.

1.4/ Chargement des modules

Dans le cas où les options iptables du kernel ont été compilées en modules, il est nécessaire de charger ces modules avant de pouvoir utiliser iptables :

```
# modprobe ip_tables
```

selon les besoins, on peut éventuellement charger les modules suivants :

```
# modprobe ip_nat_ftp
# modprobe ip_nat_irc
# modprobe iptable_filter
# modprobe iptable_mangle
# modprobe iptable_nat
```

Si on a besoin de pouvoir forwarder* les paquets IP (dans la plupart des cas), il sera nécessaire d'exécuter cette commande :

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

afin de l'indiquer au noyau.

* forwarder

il s'agit de faire passer des paquets IP d'une interface réseau vers une autre. Par exemple, un paquet qui arrive de l'internet via un modem ou une carte réseau sera *redirigé* (ou *forwardé*) vers la carte réseau par laquelle le firewall est attaché au réseau local.

Nota Bene : tous les modprobe semblent inutiles car le kernel les charge automatiquement si l'on se sert de l'une des fonctionnalités d'iptables.

2/ Présentation

2.1/ Les tables

- **Table NAT** (Network Address Translation) : Table utilisée pour la translation d'adresse ou la translation de port. Il a 2 types de chaînes[#] : *PREROUTING* qui permet de spécifier "à l'arrivée du firewall" et la chaîne *POSTROUTING* qui permet de spécifier "à la sortie du firewall". Il existe 3 targets (ou cibles) : *DNAT*^{*}, *SNAT*^{*} et *MASQUERADE*^{*}.
- **Table FILTER** : C'est la table par défaut lorsque l'on en spécifie pas. Cette table contient toutes les règles de filtrage, il existe 3 types de chaînes : *FORWARD* pour les paquets passant par le firewall, *INPUT* pour les paquets entrant et *OUTPUT* pour les paquets sortants. Les cibles disponibles sont : *ACCEPT*, *DENY*, *DROP*, *REJECT* °.
- **Table Mangle** : C'est la table qui contient les règles pour la modification de paquets. Elle est peu utilisée et ne sera pas décrite dans cet article.

A noter : Les règles sont matchées dans l'ordre, par défaut la table *FILTER* est vide et donc accepte tout. Aucune règle de translation d'adresse n'est présente par défaut.

chaîne

une chaîne est une suite de règles, qui sont prises dans l'ordre ; dès qu'une règle matche un paquet, elle est déclenchée, et la suite de la chaîne est ignorée.

* SNAT

permet de modifier l'adresse source du paquet.

* DNAT

permet de modifier l'adresse destination du paquet.

* MASQUERADE

Une gateway transforme les paquets sortants passant par elle pour donner l'illusion qu'ils sortent de la gateway elle-même par un port alloué dynamiquement ; lorsque la gateway reçoit une réponse sur ce port, elle utilise une table de correspondance entre le port et les machines du réseau privé qu'elle gère pour lui faire suivre le paquet.

° policy ACCEPT

permet d'accepter un paquet grâce à la règle vérifiée.

° policy DROP

Rejet d'un paquet sans message d'erreur si la règle est vérifiée ("non ! j'en veux pas mais je dis rien à l'expéditeur").

° policy REJECT

Rejet avec un retour de paquet d'erreur à l'expéditeur si la règle est vérifiée ("un paquet recommandé de La Poste refusé par son destinataire").

2.2/ Les commandes

IpTables n'est pas livré avec une interface graphique ; les commandes et les règles sont passées en ligne de commande. Le mieux est d'écrire des scripts (à rajouter dans `/etc/rc.d/init.d`) qui permettent d'appliquer toutes les règles d'un seul coup, dès le démarrage du Linux.

2.2.1/ Commandes principales

-A --append : Ajoute la règle à la fin de la chaîne spécifiée

Exemple :

```
# iptables -A INPUT ...
```

-D --delete : Permet de supprimer une chaîne. On peut la matcher de 2 manières, soit en spécifiant le numéro de la chaîne à supprimer, soit en spécifiant la règle à retirer.

Exemples :

```
# iptables -D INPUT --dport 80 -j DROP
```

```
# iptables -D INPUT 1
```

-R --replace : Permet contrairement à --delete de remplacer la chaîne spécifiée.

Exemple :

```
# iptables -R INPUT 1 -s 192.168.0.1 -j DROP
```

-I --insert : Permet d'ajouter une chaîne dans un endroit spécifié de la chaîne.

Exemple :

```
# iptables -I INPUT 1 --dport 80 -j ACCEPT
```

-L --list : Permet d'afficher les règles.

Exemples :

```
# iptables -L # Affiche toutes les règles des chaînes de FILTER
```

```
# iptables -L INPUT # Affiche toutes les règles de INPUT (FILTER)
```

-F --flush : Permet de vider toutes les règles d'une chaîne.

Exemple :

```
# iptables -F INPUT
```

-N --new-chain : Permet de créer une nouvelle chaîne.

Exemple :

```
# iptables -N LOG_DROP
```

-X --delete-chain : Permet d'effacer une chaîne.

Exemple :

```
# iptables -X LOG_DROP
```

-P --policy : Permet de spécifier au kernel la target par défaut d'une chaîne DENY, ACCEPT, REJECT, DROP ...

Exemple :

```
# iptables -P INPUT DROP
```

2.2.2/ Commandes pour matcher

Remarques :

Le "!" peut être utilisé pour certaines commandes afin de spécifier le contraire (on peut le traduire par "sauf"). Par exemple une commande qui doit refuser tout trafic TCP sauf ce qui provient de l'adresse IP 10.42.42.42 sera traduite par la commande suivante :

Exemple :

```
# iptables -A INPUT -p tcp --source ! 10.42.42.42 -j DENY
```

Les adresses IP peuvent optionnellement être spécifiées avec le masque associé sous la forme [adresse ip]/[masque].

-p --protocol : Spécifier un protocole : tcp, udp, icmp, all (tous)

Exemple :

```
# iptables -A INPUT -p icmp -j DENY
```

-s --source : Spécifier une adresse source à matcher

Exemple :

```
# iptables -A INPUT -p tcp -s 192.168.42.42 -j ACCEPT
```

-d --destination : Spécifier une adresse destination

Exemple :

```
# iptables -A FORWARD -p tcp -d 10.1.0.1 -j ACCEPT
```

-i --in-interface : Spécifier une interface d'entrée

Exemple :

```
# iptables -A INPUT -p icmp -i eth0 -j DENY
```

-o --out-interface : Spécifier une interface de sortie

Exemple :

```
# iptables -A OUTPUT -p icmp -o eth0 -j DENY
```

-f --fragment : Paquet fragmenté

Exemple :

```
# iptables -A INPUT -p icmp -f -j DENY
```

--sport --source-port : Spécifier le port source ou une plage de ports, fonctionne aussi en udp, -m multiport permet de spécifier plusieurs ports à matcher.

Exemples :

```
# iptables -A INPUT -p tcp --sport 80 -j ACCEPT
# iptables -A INPUT -p udp --sport 80 -j DROP
# iptables -A OUTPUT -p tcp -m multiport --sport 3128,21,1000 -j DROP
# iptables -A OUTPUT -p tcp --sport 1024:2042 -j ACCEPT
```

--dport --destination-port : Spécifier le port destination ou une plage de ports, fonctionne aussi en udp, -m multiport permet de spécifier plusieurs ports à matcher.

Exemples :

```
# iptables -A INPUT -p tcp --dport 110 -j DENY
# iptables -A INPUT -p udp --dport 110 -j DENY
# iptables -A INPUT -p tcp -m multiport --dport 110,4242,119 -j DROP
# iptables -A INPUT -p tcp --sport 4925:4633 -j ACCEPT
```

--tcp-flags : Spécifier un flag tcp à matcher : SYN ACK FIN RST URG PSH ALL NONE

Exemple :

```
# iptables -A INPUT -p tcp --dport 42 --tcp-flags SYN,ACK -j ACCEPT
```

--icmp-type : Spécifier un type de paquet icmp à matcher

Exemple :

```
# iptables -A INPUT -p icmp --icmp-type 8 -j DROP
```

--mac-source : Spécifier l'adresse MAC à matcher

Exemple :

```
# iptables -A INPUT --mac-source 42.42.AA.42.42.AA -j DROP
```

--state : Permet de spécifier l'état du paquet à matcher parmi les états suivants :

ESTABLISHED : paquet associé à une connexion déjà établie
 NEW : paquet demandant une nouvelle connexion
 INVALID : paquet associé à une connexion inconnue
 RELATED : Nouvelle connexion mais liée, idéal pour les connexions FTP

Exemples :

```
# iptables -A INPUT -i eth0 -p tcp --dport 80 -m state --state NEW,ESTABLISHED -j ACCEPT
# iptables -A OUTPUT -o eth0 -p tcp --sport 80 -m state --state ESTABLISHED -j ACCEPT
```

Spécificités NAT :

--to-destination : Utilisé en target pour le DNAT, permet de spécifier l'adresse de destination de la translation, on peut également spécifier un port s'il est différent du port source.

Exemples :

```
# iptables -t nat -A PREROUTING -d 42.12.42.12 -p tcp --dport 110 -j DNAT --to-destination 192.168.1.2:6110
# iptables -t nat -A PREROUTING -d ! 42.12.42.12 -p tcp --dport 80 -j DNAT --to-destination 192.168.2.1:3128
```

--to-source : Utilisé en target pour le SNAT, permet de spécifier l'adresse source de la translation.

Spécificités pour les LOGS :

--log-level : Level, niveau de log

Exemple : Cf. chapitre 3

--log-prefix : Permet de spécifier un préfixe pour les logs.

Exemple : Cf. chapitre 3

2.2.3/ Quelques exemples :

Les exemples qui suivent supposent que vous êtes reliés à internet par modem via l'interface ppp0 (mais en remplaçant ppp0 par eth0 – par exemple, on peut adapter les exemples pour d'autres type de liaisons) et que votre réseau local est 192.168.1.0/24 (classe C).

- Pour fixer les **politiques par défaut** (cad: ce qui se passe quand aucune règle ne correspond – ne matche pas), ici, on refuse tout (normal, on fait un firewall, oui ou non ?) :

```
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP
```

- Pour **logger** tout ce qu'on jette :

```
iptables -N LOG_DROP
iptables -A LOG_DROP -j LOG --log-prefix '[IPTABLES DROP] : '
iptables -A LOG_DROP -j DROP
```

Et ensuite, plutôt que de mettre `-j DROP`, il faut mettre `-j LOG_DROP` et les trois dernières règles doivent être :

```
iptables -A FORWARD -j LOG_DROP
iptables -A INPUT -j LOG_DROP
iptables -A OUTPUT -j LOG_DROP
```

- Pour **accepter** tout ce qui se passe sur l'interface `lo` (sinon ce n'est pas la peine d'activer le réseau !) :

```
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT
```

- Pour **accepter** tout ce qui se passe sur le **réseau local** `192.168.1.0` :

```
iptables -A INPUT -s 192.168.1.0/24 -j ACCEPT
iptables -A OUTPUT -d 192.168.1.0/24 -j ACCEPT
iptables -A FORWARD -s 192.168.1.0/24 -j ACCEPT
```

- Pour accepter les résolutions de nom (ie: le **dns**) :

```
iptables -A INPUT -i ppp0 --protocol udp --source-port 53 -j ACCEPT
iptables -A OUTPUT -o ppp0 --protocol udp --destination-port 53 -j ACCEPT
iptables -A INPUT -i ppp0 --protocol tcp --source-port 53 -j ACCEPT
iptables -A OUTPUT -o ppp0 --protocol tcp --destination-port 53 -j ACCEPT
```

- Pour accepter le trafic **web** (on veut surfer!) :

```
iptables -A INPUT -i ppp0 --protocol tcp --source-port 80 -m state --state ESTABLISHED -j LOG_ACCEPT
iptables -A OUTPUT -o ppp0 --protocol tcp --destination-port 80 -m state --state NEW, ESTABLISHED -j LOG_ACCEPT
```

La première ligne pour accepter ce qui entre sur notre interface `ppp0` sur le port 80 (le port `http`) si c'est une connexion déjà établie, la seconde pour accepter ce qui sort sur `ppp0` sur le port 80 si c'est une nouvelle connexion ou si c'est une connexion déjà établie.

Pour autoriser le **ssh**, il faut préciser le port 22; pour autoriser l'**irc**, le port 6667 (ou celui que vous utilisez pour vous connecter à votre serveur); pour le **smtp** (envoi d'emails), le port 25; pour le **pop3** (réception d'emails), le port 110; pour le **imap** (réception d'emails), les ports 143 et 220 (**imap3**) ; pour le **cvs**, le port 2401 ; pour le **https**, le port 443. De manière générale, le numéros de port se trouvent dans `/etc/services`.

- Pour le **ftp** c'est un peu plus complexe. D'abord, il faut charger le module : `ip_conntrack_ftp` (c'est lui qui suit – track en anglais – les connexions ftp) et, si vous *ratez* (en utilisant le masquering par exemple) vos connexions ftp vers d'autres postes le module : `ip_nat_ftp` :

```
modprobe ip_conntrack_ftp
# éventuellement : modprobe ip_nat_ftp
```

Ensuite, il faut taper les commandes suivantes :

```
iptables -A INPUT -i ppp0 -p tcp --sport 21 -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o ppp0 -p tcp --dport 21 -m state --state NEW, ESTABLISHED -j ACCEPT
```

Cela pour que la connexion puisse s'établir. Ensuite (et c'est là qu'on a besoin de `ip_conntrack_ftp`) :

```
iptables -A INPUT -i ppp0 -p tcp --sport 20 -m state --state ESTABLISHED,RELATED -j ACCEPT
iptables -A OUTPUT -o ppp0 -p tcp --dport 20 -m state --state ESTABLISHED -j ACCEPT
```

Pour que serveur puisse établir la connexion pour les données (en mode actif). Et enfin :

```
iptables -A INPUT -i ppp0 -p tcp --sport 1024:65535 --dport 1024:65535 -m state --state ESTABLISHED -j ACCEPT
iptables -A OUTPUT -o ppp0 -p tcp --sport 1024:65535 --dport 1024:65535 -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Pour que le serveur puisse établir la connexion pour les données (en mode passif). Ici aussi `ip_conntrack_ftp` est nécessaire.

- Pour **partager une connexion**, il faut que le forwarding soit activé dans le noyau (`echo 1 > /proc/sys/net/ipv4/ip_forward`), puis il faut autoriser iptable à faire le forwarding :

```
iptables -F FORWARD
iptables -A FORWARD -j ACCEPT
```

et enfin, cacher les machines forward-ées par le firewall :

```
iptables -A POSTROUTING -t nat -o ppp0 -j MASQUERADE
```

Sur chaque machine devant être cachée par le firewall (ou devant partager la connexion avec la machine qui est connectée à internet), il faut ajouter une route par défaut :

```
route add default gw 192.168.1.1
```

Si la machine connectée à internet a comme ip : 192.168.1.1. Il suffit avec une redhat/mandrake d'éditer :

```
/etc/sysconfig/network
```

et d'ajouter dedans :

```
GATEWAY=192.168.1.1
```

puis de redémarrer le réseau :

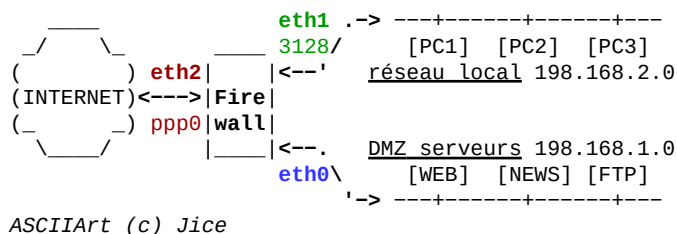
```
/etc/rc.d/init.d/network restart
```

3/ Application par l'exemple

Nous allons mettre en place un firewall / proxy.

Pour cet exemple, le firewall aura la connexion à Internet (interface eth2) et disposera de 2 pattes sur des réseaux privés (eth0 et eth1) :

- il fait office de proxy sur le port 3128 pour un réseau qui aura ainsi un accès internet
- et une DMZ* ("zone démilitarisée") sur laquelle il y a un ensemble de serveurs disponibles de l'extérieur dont un serveur web qui a pour adresse 192.168.1.2 écoutant sur le port 80 en TCP.



La classe d'adresse IP 192.168.2.0 correspond au réseau interne sur l'interface eth1.

La classe d'adresse IP 192.168.1.0 correspond à la DMZ sur l'interface eth0.

L'interface de la connexion Internet est ppp0 sur l'interface eth2.

* DMZ, ou zone démilitarisée

Sous-réseau dans lequel des serveurs accessibles depuis internet sont en adressage privé (classe d'adresse IP réservée comme 192.168.x.x) derrière un firewall.

3.1/ Le script init.d

Nous allons écrire un script qui permettra de charger automatiquement au démarrage de la machine ou sur demande les règles du firewall qui seront stockées dans le fichier /etc/firewall.sh.

Le script de démarrage sera nommé /etc/init.d/firewall. Bien sûr, on n'oubliera pas d'exécuter un `chmod +x` sur les 2 scripts que nous allons créer au long de ce chapitre.

Go !

Fichier de chargement /etc/init.d/firewall :

```
#!/bin/bash
#
# Lancement du script de Firewall
# Arnaud de Bermingham

. /etc/init.d/functions

RETVAL=0

# Fonction pour le lancement du firewall
start() {
    echo -n "Application des règles IpTables: "
    /etc/firewall.sh
    RETVAL=?
    [ -eq 0 ] && touch /var/lock/subsys/firewall
    echo
}

# Fonction pour arrêter le firewall (on flush)
stop() {
```

```
echo -n "Flush des règles IpTables: "  
/etc/flush_iptables.sh  
RETVAL=?  
[ -eq 0 ] && rm -f /var/lock/subsys/firewall  
echo  
}  
  
case "" in  
start)  
start  
;;  
stop)  
stop  
;;  
restart)  
stop  
start  
;;  
status)  
/sbin/iptables -L  
/sbin/iptables -t nat -L  
RETVAL=?  
;;  
*)  
echo "Usage: firewall {start|stop|restart|status}"  
RETVAL=1  
esac  
  
exit
```

C'est tout simple non ?

3.2/ Le script pour flusher (vider) les règles

Et maintenant : /etc/flush_iptables.sh.

```
#!/bin/sh  
#  
# Script pour vider les règles iptables  
# Arnaud de Bermingham  
# duracell@apinc.org  
  
#  
# On remet la police par défaut à ACCEPT  
#  
iptables -P INPUT ACCEPT  
iptables -P FORWARD ACCEPT  
iptables -P OUTPUT ACCEPT  
  
#  
# On remet les polices par défaut pour la table NAT  
#  
iptables -t nat -P PREROUTING ACCEPT  
iptables -t nat -P POSTROUTING ACCEPT  
iptables -t nat -P OUTPUT ACCEPT  
  
#  
# On vide (flush) toutes les règles existantes  
#  
iptables -F  
iptables -t nat -F  
  
#  
# Et enfin, on efface toutes les chaînes qui ne  
# sont pas à défaut dans la table filter et nat  
  
iptables -X  
iptables -t nat -X  
  
# Message de fin  
echo " [termine]"
```

Bon, on va enfin commencer les choses sérieuses : le script du firewall :)

3.3/ Les prérequis pour le script du firewall et création des tables pour les logs

Le script sera commenté au fur et à mesure, afin de décrire chaque étape.

```
#!/bin/sh

# script /etc/firewall.sh

# Firewall d'exemple a but pédagogique
# Arnaud de Bermingham
# duracell@apinc.org

# Activation du forwarding
# C'est pas pour faire joli, on aura des règles
# de forward et il faut bien que les paquets
# traversent la machine, donc on met ce fichier à 1

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

# Alors la, on va appliquer quelques astuces
# pour empêcher les attaques de type spoofing
# et bloquer les réponses ICMP du firewall,
# comme ça c'est très propre. Attention, le
# fait de bloquer le trafic ICMP sur
# le firewall bloque les pings.

# Je veux pas de spoofing

if [ -e /proc/sys/net/ipv4/conf/all/rp_filter ]
then
    for filtre in /proc/sys/net/ipv4/conf/*/rp_filter
    do
        echo 1 > $filtre
    done
fi

# pas de icmp

echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_all
echo 1 > /proc/sys/net/ipv4/icmp_echo_ignore_broadcasts

# On va utiliser iptables. Si on l'a compilé en module
# dans le kernel, il faut charger le module ip_tables.

modprobe ip_tables

# on va charger quelques modules supplémentaires pour
# gérer la translation d'adresse, l'IRC et le FTP
# Tu me fait 4 pompes. Chef oui chef !

modprobe ip_nat_ftp
modprobe ip_nat_irc
modprobe iptable_filter
modprobe iptable_nat

# Pour faire bien, on va vider toutes les règles
# avant d'appliquer les nouvelles règles de firewall

iptables -F
iptables -X

# On va rajouter 2 nouvelles chaînes.
# Ceci permettra d'ajouter des nouvelles cibles qui
# auront la possibilité de loguer ce qui se passe.

# La on logue et on refuse le paquet,
# on rajoute un préfixe pour pouvoir
# s'y retrouver dans les logs
iptables -N LOG_DROP
iptables -A LOG_DROP -j LOG \
    --log-prefix '[IPTABLES DROP] : '
iptables -A LOG_DROP -j DROP
```

```
# ici, on logue et on accepte le paquet,
# on rajoute un préfixe pour pouvoir
# s'y retrouver dans les logs
iptables -N LOG_ACCEPT
iptables -A LOG_ACCEPT -j LOG \
  --log-prefix '[IPTABLES ACCEPT] : '
iptables -A LOG_ACCEPT -j ACCEPT

# On veut faire un firewall efficace,
# donc la politique a appliquer est de tout
# refuser par défaut et rajouter une a une
# les règles que l'on autorise.
# Bien sur, on a RTFM un peu et on a vu que
# l'option -P permet de définir
# la cible par défaut

iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# Pour éviter les problèmes, on va tout accepter sur
# la machine en local (interface lo).
# Je déconseille de retirer cette règle car
# ça pose pas mal de problèmes et ça peut
# faire perdre la main sur la machine

iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# Bon, la partie initialisation et préparation est
# terminée, passons aux choses sérieuses

# Comme on l'a dit dans la présentation de
# l'architecture réseau, le firewall fait
# également office de proxy grâce par exemple
# à un squid installé dessus. On va donc
# accepter que le proxy ait une connexion
# internet directe. Tant qu'à faire, on va
# mettre des états pour que ça soit bien sécurisé

iptables -A OUTPUT -o ppp0 -m state \
  --state NEW,ESTABLISHED -p tcp --dport 80 -j ACCEPT
iptables -A INPUT -i ppp0 -m state \
  --state ESTABLISHED -p tcp --sport 80 -j ACCEPT

# Maintenant, on va faire en sorte que le
# proxy soit totalement transparent pour le LAN
# bénéficiant de la connexion internet.
# L'astuce consiste a rediriger toutes les
# requêtes ayant un port de destination 80
# vers le port 3128 du proxy, ici c'est le
# firewall (qui est sur le firewall et qui
# a l'adresse IP 192.168.2.1 ).
# Une règle de NAT suffira largement pour faire ça.

iptables -t nat -A PREROUTING -i eth1 -p tcp \
  --dport 80 -j DNAT --to-destination 192.168.2.1:3128

# Bon, c'est pas trop compliqué ! Maintenant
# on sait que l'on a un serveur web sur la DMZ
# (la machine d'adresse IP 192.168.1.2) sur
# le port 80. On souhaite que toutes les requêtes
# provenant d'internet arrivant sur l'adresse IP
# publique du serveur ( ici 42.42.42.42 ) soit
# redirigées sur le serveur web de la DMZ.
# Rien de bien compliqué. Dans l'exemple,
# on peut retirer le :80 de la target
# --to-destination

iptables -t nat -A PREROUTING -d 42.42.42.42 \
  -p tcp --dport 80 -j DNAT --to-destination 192.168.1.2:80

# C'est bien tout ça ! mais le problème c'est
# que les chaînes de la table FILTER sont toutes
```

```
# à DENY, donc tout ceci ne fait rien du tout.
# On va donc passer a la configuration du
# firewall proprement dit.

# On va quand même accepter les connexions ssh
# (port 22) provenant d'une machine (la votre en
# l'occurrence, on va dire 192.168.2.42) vers le
# firewall pour pouvoir modifier les règles
# facilement pour bien surveiller, on vas quand
# même loguer les connexions provenant de mon IP
# et à destination du ssh du firewall

iptables -A INPUT -i eth1 -s 192.168.2.42 -m state \
  --state NEW,ESTABLISHED -p tcp --dport 22 -j LOG_ACCEPT
iptables -A OUTPUT -o eth1 -d 192.168.2.42 -m state \
  --state ESTABLISHED -p tcp --sport 22 -j LOG_ACCEPT

# On veut que le LAN connecté à l'interface
# eth1 ait un accès complet à internet.
# La règle de NAT qui permettait d'avoir
# un proxy transparent sera automatiquement
# appliqué. L'interface correspondant
# à la connexion internet est ici ppp0

iptables -A FORWARD -i eth1 -o ppp0 -j ACCEPT
iptables -A FORWARD -o eth1 -i ppp0 -j ACCEPT

# Maintenant on donne le droit au LAN de
# consulter les pages web du serveur de la DMZ

iptables -A FORWARD -i eth1 -o eth0 -p tcp \
  --dport 80 --state NEW,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -p tcp \
  --sport 80 --state ESTABLISHED -j ACCEPT

# Maintenant il n'y a plus qu'à dire au firewall
# d'autoriser à transmettre des paquets TCP à
# destination du port 80 provenant de l'adresse
# IP publique (i.e. d'internet) vers le serveur
# web de la DMZ que nous avons naté précédemment.

iptables -A FORWARD -i ppp0 -o eth0 -p tcp \
  --destination-port 80 -m state --state NEW,ESTABLISHED -j ACCEPT
iptables -A FORWARD -o ppp0 -i eth0 -p tcp \
  --source-port 80 -m state --state ESTABLISHED -j ACCEPT

# Maintenant il ne reste plus grand chose à faire !

# Il faut permettre à l'ensemble du LAN de dialoguer
# sur internet avec la même adresse IP sinon, bien
# évidemment ça ne marchera pas (à moins que vous
# ayez 30 adresses ip !).
# Une petite règle de NAT avec un -j MASQUERADE
# suffira (masquerade = dialoguer avec l'adresse
# IP publique sur firewall)

iptables -t nat -A POSTROUTING \
  -s 192.168.2.0/24 -j MASQUERADE

# Il faut également que le serveur web de la DMZ
# soit masqueradé sinon, le serveur
# dialoguera sur internet avec son IP privée

iptables -t nat -A POSTROUTING \
  -s 192.168.1.0/24 -j MASQUERADE

# Toutes les règles qui n'ont pas passé les
# règles du firewall seront refusées et loguées...
# facile :

iptables -A FORWARD -j LOG_DROP
iptables -A INPUT -j LOG_DROP
iptables -A OUTPUT -j LOG_DROP
```

```
# Pour faire zoli  
echo " [Termine]"  
  
# c'est enfin fini
```

Et voilà ! le firewall de compèt' est prêt et fonctionnel.

Ceci était bien évidemment un exemple, vous pouvez dès à présent préparer votre propre firewall personnalisé !
Vous pouvez l'adapter à vos besoins, votre connexion vers internet (par ADSL par exemple), etc.

