Linux Bridge+Firewall Mini-HOWTO version 1.2.0

Peter Breuer (*ptb@it.uc3m.es*)
Adaptation française par Etienne BERNARD (*eb@via.ecp.fr*) 19 Décembre 1997

Contents

1 Introduction

Vous devriez lire l'original *Bridging mini-HOWTO* `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/mini/Bridge` (NdT : ou *en version française* `ftp://ftp.lip6.fr/pub/linux/french/docs/mini/Bridge`) par Chris Cole pour une vision différente sur le sujet. L'adresse email de Chris Cole est *chris@polymer.uakron.edu* . La version de cet HOWTO, à partir duquel ce document est construit est la version 1.03, daté du 23 août 1996.

2 Quoi, et pourquoi (et comment ?)

2.1 Quoi

Un pont est un élément qui connecte intelligement des brins grâce à deux cartes ethernet. Un *firewall* est un élément isolant intelligent.

2.2 Pourquoi

Si vous avez de nombreux ordinateur, vous pouvez désirer installer un pont :

1. pour économiser le prix d'un nouveau *hub* lorsqu'il se trouve que vous avez une carte ethernet libre ;
2. pour éviter d'avoir à apprendre l'*IP-forwarding* et d'autres trucs alors que vous *avez* deux cartes dans votre ordinateur ;
3. pour éviter des travaux de maintenance pour d'éventuels changements futurs !

Le terme "nombreux ordinateurs" peut même représenter seulement trois ordinateurs, si ceux-ci font du routage ou du pontage ou qu'ils changent de place dans la pièce de temps en temps ! Vous pouvez même vouloir un pont pour vous amuser à trouver à quoi cela sert. Je voulais un pont pour la raison 2 (2).

Si vous êtes intéressé par le point 1 (1), vous êtes peu dans votre cas. Lisez le *NET-2-HOWTO* `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/NET-2-HOWTO` et le *Serial-HOWTO* `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/Serial-HOWTO` pour de meilleurs astuces.

Vous désirez un *firewall* si :

1. vous essayez de protéger votre réseau des accès extérieur, ou
2. vous désirez interdire l'accès au monde extérieur aux machines de votre réseau.

Bizarrement, j'avais besoin du point 2 (2) ici aussi. La politique de mon université pour le moment est de ne pas jouer le rôle de fournisseur d'accès à Internet pour les *undergraduates*.

2.3 Comment

J'ai commencé par du pontage entre deux cartes réseau sur une machine jouant le rôle de *firewall*, et j'ai fini par lancer le *firewall* sans avoir coupé le pont. Cela a l'air de fonctionner, et c'est beaucoup plus flexible que chaque configuration isolée. Je peux arrêter le *firewall* et continuer à faire fonctionner le pont ou arrêter le pont lorsque je veux être plus prudent.

Je suppose que la partie "pont" du noyau se trouve juste au-dessus de la couche physique et que la partie *firewall* se trouve dans une couche réseau supérieure, afin que les parties de pontage et de *firewalling* agissent en fait comme si elles étaient connectées en "série" et non pas en "parallèle" (aie !), selon le schéma suivant :

```
-> Pont-entrant -> Firewall-entrant -> Noyau -> Firewall-sortant -> Pont-sortant ->
```

Il n'y a pas d'autre façon d'expliquer comment une machine peut être en même temps “conducteur” et “isolant”. Il existe quelques embuches, mais j'en parlerai plus tard. Schématiquement, vous devez router les paquets que vous voulez filtrer. De toute façon, cela a l'air de fonctionner parfaitement pour moi, et voici comment...

3 PONT

3.1 Logiciel

Récupérez l' *utilitaire de configuration du pont* `ftp://shadow.cabi.net/pub/Linux/BRCFG.tgz` depuis la page personnelle d'Alan Cox. C'est la même référence que dans le document de Chris. Je n'ai pas compris que c'était un URL *ftp* en non un URL *http*...

3.2 Lecture préliminaires

Lisez le *Multiple Ethernet HOWTO* `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/mini/Multiple-Ethernet` pour obtenir des conseils pour faire reconnaître et pour configurer plus d'une carte réseau.

Vous pourrez trouver encore plus de détails sur le type de commandes magiques à passer au *prompt* se trouvent dans le *Boot Prompt HOWTO* `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/BootPrompt-HOWTO` .

Pour compléter vos lectures, lisez le *NET-2 HOWTO* `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/NET-2-HOWTO` . C'est un document plutôt long, et vous devrez y piocher les détails qui vous intéressent.

3.3 Configuration de lancement

Les lectures précédentes vont vous indiquer ce dont vous avez besoin pour préparer le noyau à reconnaître un deuxième périphérique ethernet lors du démarrage, en ajoutant la ligne suivante dans votre fichier `/etc/lilo.conf`, et en relançant `lilo` :

```
append = "ether=0,0,eth1"
```

Notez le "eth1". "eth0" représente la première carte. "eth1" est la seconde carte. Vous pouvez également ajouter les paramètres de démarrage à la ligne de commande que `lilo` vous offre. Pour trois cartes :

```
linux ether=0,0,eth1 ether=0,0,eth2
```

J'utilise `loadlin` pour lancer mon noyau Linux depuis DOS :

```
loadlin.exe c:\vmlinuz root=/dev/hda3 ro ether=0,0,eth1 ether=0,0,eth2
```

Notez que cette astuce oblige le noyau à détecter les cartes au démarrage. La détection ne sera pas faite si vous chargez les gestionnaires de périphérique ethernet en **module** (par sécurité, puisque l'ordre de détection ne peut être déterminé), donc si vous utilisez des modules, vous aurez à ajouter l'IRQ appropriée et le paramètre de port pour le gestionnaire de périphérique dans votre fichier `/etc/conf.modules`. Dans mon cas, j'ai les lignes :

```
alias eth0 3c509
alias eth1 de620
options 3c509 irq=5 io=0x210
options de620 irq=7 bnc=1
```

Vous pouvez savoir si vous utilisez les modules en utilisant “ps -aux” pour voir si `kerneld` est lancé, et en vérifiant qu'il y a des fichiers `.o` dans un sous-répertoire du répertoire `/lib/modules`. Utilisez le nom de répertoire que vous donne la commande `uname -r`. Si vous avez un `kerneld` lancé et/ou vous avez un fichier `foo.o`, éditez `/etc/conf.modules` et lisez avec soin la page de manuel de `depmod`.

Notez également que jusque récemment (noyau 2.0.25), le *driver* pour la carte **3c509** ne pouvait pas être utilisé pour plus d'une carte s'il était utilisé en module. J'ai vu un *patch* quelque part pour corriger cette limitation. Il devrait être inclus dans le noyau à l'heure où vous lisez ces lignes.

3.4 Configuration du noyau

Recompilez le noyau avec le *bridging* :

```
CONFIG_BRIDGE=y
```

J'ai également compilé mon noyau avec le *firewalling*, l'*IP-forwarding* et l'*IP-masquerading*. C'est seulement si vous désirez utiliser le *firewalling* également...

```
CONFIG_FIREWALL=y
CONFIG_NET_ALIAS=y
CONFIG_INET=y
CONFIG_IP_FORWARD=y
CONFIG_IP_MULTICAST=y
CONFIG_IP_FIREWALL=y
CONFIG_IP_FIREWALL_VERBOSE=y
CONFIG_IP_MASQUERADE=y
```

Vous aurez besoin en plus de la configuration réseau standard :

```
CONFIG_NET=y
```

et je ne pense pas que vous deviez vous préocupper des autres options réseau. Les options que je n'ai pas compilé dans le noyau sont sélectionnées en tant que modules afin que je puisse les ajouter éventuellement plus tard.

Installez le nouveau noyau, relancez `lilo` et redémarrez sur le nouveau noyau. Rien ne devrait avoir changé pour l'instant !

3.5 Adresses réseau

Chris dit qu'un pont ne doit pas avoir d'adresse IP mais ce n'est pas la configuration qui est présenté ici.

Vous allez utiliser la machine pour vous connecter au réseau donc vous avez besoin d'une adresse et vous devez vous assurer que le device *loopback* configuré normalement afin que vos logiciels puisse communiquer avec ce à quoi ils s'attendent. Si *loopback* est désactivé, le *résolveur de noms* ou d'autres services ne fonctionneront pas. Voyez le NET-2-HOWTO, mais votre configuration standard devrait déjà avoir fait cela :

```
ifconfig lo 127.0.0.1
route add -net 127.0.0.0
```

Vous allez devoir donner des adresses à vos cartes réseau. J'ai changé le fichier `/etc/rc.d/rc.inet1` de ma slackware (3.x) pour configurer deux cartes et vous devrez juste regarder votre fichier de configuration du réseau et doubler ou tripler le nombre d'instructions s'y trouvant. Supposons que vous ayez déjà une adresse à

```
192.168.2.100
```

(cette adresse fait partie des adresses réservées pour des réseaux privés, mais ne faites pas attention, cela ne cassera rien si vous utilisez cette adresse par erreur) alors vous avez probablement une ligne ressemblant à

```
ifconfig eth0 192.168.2.100 netmask 255.255.255.0 metric 1
```

dans votre fichier de configuration. La première chose que vous allez probablement vouloir faire est couper l'espace des adresses atteintes par cette carte en deux afin de pouvoir éventuellement faire un pont ou filtrer entre les deux moitiés. Ajoutez donc une ligne qui réduit le masque de sous-réseau pour adresser un plus petit nombre de machines :

```
ifconfig eth0 netmask 255.255.255.128
```

Essayez cette configuration. Cela restreint la carte à l'espace des adresses entre .0 et .127.

A présent, vous pouvez configurer votre deuxième carte dans la deuxième moitié de l'espace des adresses locales. Assurez vous que personne n'utilise l'adresse que vous allez prendre. Pour des raisons de symétrie, j'utiliserai ici 228=128+100. N'importe quelle adresse conviendra, à condition qu'elle ne se trouve pas dans le masque de l'autre carte. Évitez les adresses spéciales comme .0, .1, .128, etc... à moins que vous sachiez ce que vous faites.

```
ifconfig eth1 192.168.2.228 netmask 255.255.255.128 metric 1
```

Cela restreint la deuxième carte aux adresses entre .128 et .255.

3.6 Routage réseau

C'est ici que les défauts de l'utilisation simultannée du pont et du firewall : vous ne pouvez pas filtrer des paquets qui ne sont pas routés. Pas de route, pas de firewall. Cette règle est vérifiée en tout cas dans les version 2.0.30 ou suivantes du noyau. Les filtres du firewall sont étroitement liés au code source de l'IP-Forwarding.

Cela ne signifie pas que vous ne pouvez pas utiliser le pont. Vous pouvez installer un pont entre deux cartes et filtrer à partir d'une troisième. Vous pouvez n'avoir que deux cartes et les faire filtrer une adresse IP externe, comme celle d'un routeur proche, à condition que le routeur relié à une seule carte.

En d'autres termes, puisque je veux utiliser la machine comme firewall, je dois contrôler avec précision la destination physique de certains paquets.

J'ai le petit réseau de machines sur un *hub* connecté à `eth0`, je configure donc un réseau de ce côté :

```
route add -net 192.168.2.128 netmask 255.255.255.128 dev eth0
```

Remplacez le 128 par un 0 pour un réseau de classe C entier. Ici, je ne le fais pas puisque j'ai juste divisé en deux l'espace d'adressage. Le "dev eth0" n'est pas nécessaire ici, puisque l'adresse de la carte fait partie de ce réseau, mais il peut être nécessaire de l'écrire chez vous. On pourrait désirer plus d'une carte prenant ce sous réseau en charge (127 machines sur un segment, bravo !) mais ces cartes utiliseraient le même masque de sous-réseau et seraient considérées comme une seule par la partie routage du noyau.

Sur l'autre carte, j'ai une ligne qui passe directement à travers un gros routeur, auquel je fais confiance.

```
                                             client 129
         --                                        |    --
client 1   \    .0                   .128          |   /   net 1
client 2 --- Hub - eth0 - Kernel - eth1 - Hub - Router --- net 2
client 3 __/       .100             .228         .2 |   \__ net 3
                                                   |
                                             client 254
```

J'utilise une route fixe (c'est-à-dire "statique") depuis la carte vers ce routeur, puisque sinon il ferait partie du masque de sous-réseau de la première carte et le noyau se tromperait sur la manière d'envoyer les paquets au routeur. Je veux filtrer ces paquets et c'est une raison de plus de les router explicitement.

```
route add 192.168.2.2 dev eth1
```

Je n'en ai pas besoin, puisque je n'ai pas d'autres machines dans cette moitié de l'espace d'adressage, mais je déclare un réseau sur la seconde carte. La séparation de mes interfaces réseau en deux groupes grâce au routage me permettra éventuellement de faire du filtrage très précis, mais vous pouvez très bien vous en sortir avec beaucoup moins de routage que cela.

```
route add -net 192.168.2.128 netmask 255.255.255.128 dev eth1
```

J'ai également besoin d'envoyez tous les paquets non-locaux au monde et je dis donc au noyau de les envoyer au gros routeur :

```
route add default gw 192.168.2.2
```

3.7 configuration de la carte

Nous avions auparavant une configuration standard pour le réseau, mais comme nous faisons du *bridging*, nous devons écouter sur chaque carte les paquets qui ne nous sont pas destinés. Ceci doit aller dans le fichier de configuration réseau :

```
ifconfig promisc eth0
ifconfig promisc eth1
```

La page de manuel indique d'utiliser `allmulti=promisc`, mais cela ne fonctionnait pas pour moi.

3.8 Routage additionnel

J'ai remarqué une chose : j'ai dû passer la seconde carte dans un mode lui permettant aux questions du gros routeur à propos des machines que je cache sur mon réseau local.

```
ifconfig arp eth1
```

Pour faire bonne mesure, j'ai effectué cette opération pour l'autre carte aussi.

```
ifconfig arp eth0
```

3.9 Configuration du pont

Ajoutez la mise en route du pont dans votre fichier de configuration :

```
brcfg -enable
```

La configuration du pont mettra en route certains ports. Vous pouvez expérimenter l'allumage et l'extinction des ports un à la fois :

```
brcfg -port 0 -disable/-enable
brcfg -port 1 -disable/-enable
```

Vous pouvez obtenir un rapport sur l'état courant avec :

```
brcfg
```

sans aucun paramètres. Vous pourrez voir que le pont écoute, apprend, et effectue le *forwarding*. (Je ne comprends pas pourquoi le code répète la même adresse matérielle pour mes deux cartes, mais peu importe... le HOWTO de Chris affirme que c'est correct).

3.10 Essais

Si le réseau est encore en fonction, essayez votre script de configuration en vrai en arrêtant les deux cartes et en l'exécutant :

```
ifconfig eth0 down
ifconfig eth1 down
/etc/rc.d/rc.inet1
```

Avec un peu de chance, les divers systèmes tel **nfs**, **ypbind**, etc... ne s'en rendront pas compte. *N'essayez pas ceci si vous n'êtes pas derrière le clavier !*

Si vous désirez être plus prudent que cela, vous devrez arrêter le plus de démons possible, et démonter les répertoires NFS. Le pire qu'il puisse vous arriver est d'avoir à rebooter en mode *single-user* (le paramètre **"single"** de `lilo` ou `loadlin`), et de restaurer les fichiers à leur valeur d'avant les modifications.

3.11 Vérifications

Vérifiez qu'il existe un trafic différent sur chaque interface :

```
tcpdump -i eth0
```

```
(dans une fenêtre)
```

```
tcpdump -i eth1
```

```
(dans une autre fenêtre)
```

Vous devriez être habitué à l'utilisation de **tcpdump** pour trouver des évènements qui ne devraient pas se passer, ou qui existent mais ne devraient pas.

Par exemple, recherchez les paquets qui ont traversé le pont vers la seconde carte depuis le réseau interne. Ici, je cherche les paquets venant de la machine avec l'adresse .22 :

```
tcpdump -i eth1 -e host 192.168.2.22
```

A présent, envoyez un ping depuis l'hôte en .22 vers le routeur. Vous devriez voir le paquet affiché par tcpdump.

A présent, vous devriez avoir un pont, et qui possède également deux adresses réseau. Vérifiez que vous pouvez les `ping`er depuis l'extérieur de votre réseau local et depuis l'intérieur, que vous pouvez utiliser `telnet` et `ftp` depuis et vers l'intérieur du réseau.

4 FIREWALLING

4.1 Logiciel et lectures

Vous devriez lire le *Firewall-HOWTO* `ftp://sunsite.unc.edu/pub/Linux/docs/HOWTO/Firewall-HOWTO` .

Il vous indiquera où trouver `ipfwadm` si vous ne l'avez pas déjà. Vous pouvez également récupérer d'autres outils, mais seulement `ipfwadm` m'a été utile. C'est pratique et de bas niveau ! Vous pouvez voir exactement ce qu'il fait.

4.2 Vérifications préliminaires

Vous avez compilé l'IP-forwarding et le masquerading dans le noyau, et vous allez vérifier que le *firewall* est dans son état par défaut (il accepte) grâce à :

```
ipfwadm -I -l
ipfwadm -O -l
ipfwadm -F -l
```

Ce qui, dans l'ordre, "affiche les règles affectant la partie .."entrante ou sortante ou qui fait suivre (*masquerading*) ".. du *firewall*". L'option "-l" signifie "lister".

Si vous avez compilé l'IP accounting également :

```
ipfwadm -A -l
```

Vous devriez constater qu'aucune règle n'est définir et que l'action par défaut est d'accepter tous les paquets. Vous pouvez retourner à cet état à tout moment, avec :

```
ipfwadm -I -f
ipfwadm -O -f
ipfwadm -F -f
```

L'option "-f" signifie "flush" (en français, "vider"). Vous pourriez en avoir besoin.

4.3 Règle par défaut

Je veux interdire l'accès au monde entier depuis mon réseau interne, et rien d'autre, donc je vais donner comme dernière règle (comme règle par défaut) une règle indiquant d'ignorer les paquets venant du réseau interne et dirigés vers l'extérieur. Je place toutes les règles (dans cet ordre) dans le fichier `/etc/rc.d/rc.firewall` que j'execute depuis `/etc/rc.d/rc.local` au démarrage.

```
ipfwadm -I -a reject -S 192.168.2.0/255.255.255.128 -D 0.0.0.0/0.0.0.0
```

Le "-S" représente l'adresse source/masque de sous-réseau. L'option "-D" est pour l'adresse destination/masque de sous-réseau.

Ce format n'a plus le vent en poupe. `ipfwadm` est intelligent et connaît des abréviations courantes. Vérifier les pages de manuel.

Il peut être plus pratique de placer certaines ou toutes les règles sur la partie sortante du firewall en utilise "-O" au lieu de "-I", mais je supposerai que les règles sont conçues pour être utilisées sur la moitié entrante.

4.4 Accès par adresse

Avant la règle par défaut, je dois placer des règles qui servent d'exceptions pour cette interdiction générale des services extérieurs aux clients intérieurs.

Je veux traiter les adresses des machines derrière le firewall de manière spéciale. Je veux empêcher aux gens de se loguer sur la machine qui sert de firewall à moins qu'elles aient une permission spéciale, mais une fois connectées, elles devraient être capables de parler au monde extérieur.

```
ipfwadm -I -i accept -S 192.168.2.100/255.255.255.255 \
 -D 0.0.0.0/0.0.0.0
```

Je veux également que les clients internes soient capables de parler à la machine faisant *firewall*. Peut-être pourront-elles la persuader de les laisser sortir !

```
ipfwadm -I -i accept -S 192.168.2.0/255.255.255.128 \
 -D 192.168.2.100/255.255.255.255
```

Vérifiez que vous pouvez joindre les machines à l'intérieur du firewall depuis l'extérieur par `telnet`, mais que vous ne pouvez pas sortir. Vous pouvez faire le premier pas, mais les clients ne peuvent pas vous envoyer de messages. Essayez également `rlogin` et `ping` avec `tcpdump` lancé sur une carte ou l'autre. Vous devriez être capable de comprendre ce que vous lirez.

4.5 Accès par protocole

J'assouplis les règles protocole par protocole. Je veux que les `pings` depuis l'extérieur vers l'intérieur obtiennent une réponse, par exemple, donc j'ai ajouté la règle :

```
ipfwadm -I -i accept -P icmp -S 192.168.2.0/255.255.255.128 \
 -D 0.0.0.0/0.0.0.0
```

L'option "`-P icmp`" correspond au protocole tout entier.

Avant que j'installe un *proxy ftp*, j'autorise également les appels ftp sortants en relâchant les restrictions sur un port donné. Ceci vise les ports 20, 21 et 115 sur les machines externes :

```
ipfwadm -I -i accept -P tcp -S 192.168.2.0/255.255.255.128 \
 -D 0.0.0.0/0.0.0.0 20 21 115
```

Je n'ai pas pu faire fonctionner `sendmail` entre les clients locaux sans serveur de noms. Au lieu d'installer un serveur de nom directement sur le *firewall*, j'ai juste autorisé les requêtes TCP de résolution de nom visant le plus proche serveur de nom, et j'ai placé son adresse dans le fichier `/etc/resolv.conf` des clients. (`"nameserver 123.456.789.31"` sur une ligne séparée).

```
ipfwadm -I -i accept -P tcp -S 192.168.2.0/255.255.255.128 \
 -D 123.456.789.31/255.255.255.255 54
```

Vous pouvez trouver quel numéro de port et protocole requiert un service grâce à `tcpdump`. Utilisez ce service avec un `ftp` ou un `telnet` ou autre vers ou depuis une machine interne, et observez-le sur les ports d'entrée et de sortie du *firewall* avec `tcpdump` :

```
tcpdump -i eth1 -e host client04
```

par exemple. Le fichier `/etc/services/` est une importante source d'indices. Pour laisser ENTRER le `telnet` et le `ftp` depuis l'extérieur, vous devez autoriser un client local à envoyer des données vers l'extérieur sur un port donné. Je comprends pourquoi ceci est nécessaire pour le `ftp` (c'est le serveur qui établit la connexion de données), mais je me demande pourquoi `telnet` en a également besoin.

```
ipfwadm -I -i accept -P tcp -S 192.168.2.0/255.255.255.128 ftp telnet \
 -D 0.0.0.0/0.0.0.0
```

Il existe un problème particulier avec certains démons qui cherchent le nom d'hôte de la machine *firewall* afin de décider quelle est leur adresse réseau. J'ai eu des problèmes avec `rpc.yppasswdd`. Il insiste sur des informations *broadcast* qui indiquent qu'il se trouve en dehors du *firewall* (sur la seconde carte). Cela signifie que les clients à l'intérieur ne peuvent pas le contacter.

Au lieu de lancer l'IP aliasing ou de changer le code source du démon, j'ai traduit le nom vers l'adresse de la carte du côté intérieur sur les clients, dans leur fichier `/etc/hosts`.

4.6 Vérifications

Vous voudrez tester que vous pouvez toujours utiliser `telnet`, `rlogin` et `ping` depuis l'extérieur. Depuis l'intérieur, vous devriez être capable d'utiliser `ping` vers la sortie. Vous devriez également être capables d'utiliser le `telnet` vers la machine *firewall* depuis l'intérieur, et cette dernière manipulation devrait vous permettre d'accéder au reste.

Et voilà. A présent, vous voulez probablement apprendre **rpc**/**Y**ellow **P**ages et leur interaction avec le fichier de mots de passe. Le réseau derrière le firewall devrait vouloir empêcher aux utilisateurs non privilégiés de se logguer sur le *firewall*, et donc de sortir. C'est traité dans un autre HOWTO !