The OpenBSD Packet Filter HOWTO

Wouter Coene

version 20011007, engendrée le 6 novembre 2001
disponible sur `http://www.inebriated.demon.nl/pf-howto/`

Table des matières

1 Introduction

OpenBSD Packet Filter (OpenBSD `PF`) est le paquetage de firewall stateful intégré dans le noyau OpenBSD depuis la version 3.0 d'OpenBSD. Ce document décrit la mise en œuvre et l'administration des règles de PF ainsi que les tables de correspondance NAT.

1.1 Public visé

L'audience visée par ce document est celle des administrateurs systèmes et réseau, disposant d'une connaissance préliminaire des bases de fonctionnement des réseaux et protocoles sous-jacents. La connaissance d'autres systèmes de pare-feu, bien que non requise, peut aider dans la compréhension des sujets les plus complexes.

1.2 Copyright et licence

The OpenBSD Packet Filter HOWTO version 20011007
Copyright (C) Wouter Coene, 2001.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met :

– Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
– Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
– The name of the author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS DOCUMENT IS PROVIDED BY THE AUTHOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES ; LOSS OF USE, DATA, OR PROFITS ; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFT-

WARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Vous pouvez me contacter à wouter@inebriated.demon.nl si vous avez une quelconque question (en anglais, SVP).

Vous pouvez me contacter à olem@IDEALX.org si vous avez une quelconque question sur la traduction française de ce document (français, SVP).

2 Les bases de la protection avec pare-feu

Un pare-feu est supposé protéger un réseau d'attaques potentielles provenant d'autres réseaux. Il agit en inspectant les paquets qui transitent entre les réseaux, et en imposant des restrictions quant aux types, sources, destinations et contenus de ces paquets.

Cette section explique comment mettre en œuvre un pare-feu en utilisant PF. Pour les besoins de cette explication, nous utiliserons un réseau d'entreprise et son pare-feu comme exemple. Le réseau consiste en un réseau `TCP/IP` ayant l'adresse réseau `260.250.1.0/24`. Un serveur web d'entreprise est placé sur le nœud `260.250.1.3`. Un pare-feu avec deux interfaces réseau, `xl0` et `xl1` relie le réseau d'entreprise à l'Internet. Le réseau d'entreprise est connecté à l'interface `xl1`, et le lien Internet est connecté à `xl0`.

2.1 Jeux de règles basiques

Le composant de pare-feu de PF utilise un jeu de règles qui décrivent les actions à entreprendre pour certains paquets. Ces règles sont lues depuis un fichier puis chargées dans le noyau OpenBSD en utilisant la commande `pfctl` (pour plus d'information sur le chargement d'un jeu de règles, consultez la section 2.11).

Un tel fichier de règles pourrait être composé de l'exemple suivant :

```
block in all
pass  in all
```

Analysons ce qui se passerait dans ce cas. La première règle indique à PF de bloquer tous les paquets. À la différence d'autres pare-feux, PF ne stoppe pas son analyse de configuration quand il trouve une occurence qui correspond, à la place, il note le fait qu'il prévoit de bloquer le paquet, puis il continue l'analyse de sa configuration.

La règle suivante indique à PF de laisser passer tous les paquets, quelles que soient leurs destinations. Encore une fois, PF note le fait qu'il prévoit de laisser passer le paquet, et poursuit son analyse de configuration.

Comme il n'y a pas de règle suivante, PF commence à regarder ce qu'il avait prévu de faire avec son paquet. Dans notre cas, la dernière règle prévoyait de laisser passer le paquet, c'est donc ce que PF fera.

Bien, tout ceci ne semble pas très utile, nous allons essayer quelque chose de plus intéressant. Laissons l'accès ouvert vers le serveur Web d'entreprise qui

fonctionne sur le nœud `260.250.1.3`[1]. Vous pourriez essayer quelque chose dans le genre de :

```
block in all
pass  in from any to 260.250.1.3/32
```

Encore une fois, la première règle dit à PF de noter qu'il bloquera le paquet tant qu'une autre règle ne lui dira pas autre chose.

La règle suivante indique à PF de laisser passer les paquets dont la destination est indiquée comme étant `260.250.1.3`, où le '`/32`' indique à PF qu'il doit vérifier la correspondance de l'adresse sur l'intégralité des 32 bits.

Mais vous pourriez vous poser la question du trafic de retour vers les clients. C'est en fait aussi simple, il suffit d'autoriser le trafic depuis `260.250.1.3` vers toutes les destinations :

```
block in all
pass  in from any to 260.250.1.3/32
pass  in from 260.250.1.3/32 to any
```

2.2 Jeux de règles avancées

Supposons qu'il soit décidé qu'un autre site Web dusse fonctionner sur le serveur Web. Ce nouveau site contenant de l'information confidentielle à l'entreprise, il ne devrait pas être accessible depuis l'exterieur. Il est décidé que ce site Web, au contraire du site publique, ne fonctionnera que sur un port `TCP` non standard, le port `8000`.

Maintenant, en plus de filtrer les paquets sur leurs adresses de destination, vous devez également les filtrer sur les ports `TCP`, afin d'être assuré que personne ne puisse se connecter au port `8000` depuis l'extérieur pour consulter de l'information confidentielle :

```
block in all
pass  in proto tcp from any to 260.250.1.3/32 port = 80
pass  in proto tcp from 260.250.1.3/32 port = 80 to any
```

Comme vous pouvez le voir, nous ne filtrons pas seulement sur le numéro de port, nous indiquons également à PF de n'autoriser que les paquets du protocole `TCP` à passer à travers le firewall vers le serveur Web d'entreprise.

[1] un exemple complètement inventé

2.3 Conservation d'état : keep state

Le composant pare-feu de PF est capable de se rappeler des sessions TCP, UDP ou ICMP qui sont ouvertes, et peut filtrer des paquets en rapport avec cette table d'état. Ce mécanisme est appellé « keep state » (conservation d'état).

Quand PF voit un paquet qui correspond à une règle qui lui indique de conserver l'état (keep state), il crée alors une nouvelle entrée dans la table d'état en se basant sur les informations contenues dans le paquet. Les paquets suivants de la même session ne passeront ainsi plus par la phase de correspondance avec la configuration, et seront gérés de la façon indiquée dans la règle originale (celle qui indiquait le keep state).

La conservation d'état sur les connexions TCP implique une vérification scrupuleuse des numéros de séquence des paquets en les comparant avec la table d'état, et rejetant tous les paquets qui ne correspondent pas à la connexion visée, réduisant ainsi les chances d'exploitation des failles de piles TCP peu avancées des nœuds placés derrière le pare-feu.

Supposons qu'il soit décidé que la navigation sur le Web soit possible depuis notre réseau d'entreprise. Cela implique que le pare-feu laisse passer les connexions TCP sur le port 80 ainsi que les requêtes DNS sur le port 53.

En utilisant le mécanisme de conservation d'état de PF pour cela, vous pouvez rapidement définir des règles simples sans ouvrir le réseau entier aux attaques dans le même temps. Pendant que nous y sommes, nous utiliserons également le mécanisme de conservation d'état pour l'accès au serveur web, ce qui du coup, résultera en un contrôle plus fin sur les règles d'accès à ce serveur :

```
block in all
pass  in proto udp from 260.250.1.0/24 to any port = 53 keep state
pass  in proto tcp from 260.250.1.0/24 to any port = 80 keep state
pass  in proto tcp from any to 260.250.1.3/32 port = 80 keep state
```

La troisième règle permet aux nœuds du réseau d'entreprise de faire des connexions vers les nœuds externes en utilisant le port HTTP, en indiquant à PF de créer une entrée dans sa table d'état pour ces connexions. La dernière règle indique à PF de faire de même avec les connexions faites depuis les nœuds externes vers le serveur web d'entreprise, rendant ainsi la règle de gestion du trafic de retour que nous utilisions alors inutile.

Utiliser le mécanisme de conservation d'état peut faire penser que l'on va surcharger le pare-feu, ce qui réduirait le trafic. Cependant, la manipulation de la table d'état par PF est bien plus rapide que l'analyse de configuration.

Un jeu de règles composé de 50 règles nécessite 50 comparaisons, alors qu'une table d'état avec 50 000 entrées ne nécessite qu'environ 16 comparaisons, en raison de sa structure hiérarchique binaire. Ce constat, combiné au potentiel de sécurité qu'amène la rédaction de règles plus claires, fait qu'il vaut mieux utiliser le mécanisme de conservation d'état, même pour les tâches les plus simples qui auraient pourtant été faites simplement sans lui.

2.4 Échappement du jeu de règles : le mot-clé `quick`

Parfois, on a besoin de stopper l'analyse du jeu de règles dès qu'une condition est rencontrée pour un certain paquet sur une certaine règle. Pour cela, PF utilise le mot clef `quick`. Une règle qui dispose de l'option `quick` provoquera l'arrêt de l'analyse du jeu de règle pour le paquet qui correspond.

Ce fonctionnement est particulièrement utilisé pour protéger votre réseau contre les paquets forgés (spoofing), comme le montre l'exemple suivant :

```
block in all
block in quick from 10.0.0.0/8 to any
block in quick from 172.16.0.0/12 to any
block in quick from 192.168.0.0/16 to any
block in quick from 255.255.255.255/32 to any
pass  in all
```

Ce jeu de règles indique à PF de rejeter immédiatement les paquets dont l'origine est `10.0.0.0/8`, `172.16.0.0/12`, `192.168.0.0/16` et `255.255.255.255/32`. Les autres paquets sont acceptés.

L'utilisation de cet échappement procure des gains en performances significatifs quand il est utilisé sur des règles qui gèrent un gros trafic.

2.5 Correspondance avec des interfaces réseau

Il est également possible de faire correspondre une règle à une interface réseau. Adaptons notre exemple d'anti-spoofing ci-dessus, en gardant à l'esprit la stucture de notre réseau d'entreprise :

```
block in all
block in quick on xl0 from 10.0.0.0/8 to any
block in quick on xl0 from 172.16.0.0/12 to any
block in quick on xl0 from 192.168.0.0/16 to any
block in quick on xl0 from 255.255.255.255/32 to any
pass  in all
```

2.6 Correspondance avec des drapeaux TCP

Pour être en mesure de bloquer des paquets invalides, vous pouvez indiquer à PF de filtrer le trafic sur des drapeaux TCP (*flags*) en utilisant le mot clef `flags`, suivi d'une liste de drapeaux à prendre en compte (séparés par un backslash), et d'un optionel masque de drapeaux.

PF, pour chaque paquet TCP, commencera par ignorer tous les drapeaux sauf ceux spécifiés dans le masque, puis jugera de la correspondance avec les drapeaux indiqués. Ainsi, spécifier '`flags S/SA`' indique à PF de commencer par ignorer tous les drapeaux sauf SYN et ACK, puis de vérifier si le drapeau SYN est présent.

Les drapeaux suivants sont reconnus :

F : FIN, pour mettre fin à une connexion
S : SYN, pour entamer une connexion
R : RST, pour mettre à zéro (reset) une connexion
P : PSH, pour s'assurer que toutes les données sont arrivées
A : ACK, pour acquitter l'arrivée d'un paquet
U : URG, indiquant que ce paquet est urgent

À titre d'exemple, un paquet demandant une nouvelle connexion ne contient que le drapeau SYN, et un paquet accusant acceptation d'une connexion comporte les drapeaux SYN et ACK. Un paquet indiquant le refus d'une connexion comporte les drapeaux ACK et RST.

L'utilisation d'un combinaison invalide de drapeaux TCP est une méthode répendue pour scanner les ports ouverts d'un nœud de façon discrète. En utilisant le mot-clé `flags`, vous pouvez défendre votre système contre ces scans discrets, et forcer les scanners de ports à utiliser des méthodes qui sont plus facilement détectables.

En reprenant notre exemple précédent dans ce HOWTO, nous voulons maintenant que seuls les paquets TCP disposant uniquement du drapeau SYN parmi ceux qui disposent des drapeaux SYN et ACK, soient ceuxpris en compte dans notre table d'état :

```
block in all
pass  in proto udp from 260.250.1.0/24 to any port = 53 keep state
pass  in proto tcp from 260.250.1.0/24 to any port = 80 \
                                    flags S/SA keep state
pass  in proto tcp from any to 260.250.1.3/32 port = 80 \
                                    flags S/SA keep state
```

Ce jeu de règles empêchera les méthodes de scanning discrètes, mentionnées ci-dessus, de passer à travers notre pare-feu.

2.7 Lots

Il est possible de spécifier un lot de nœuds, au lieu de devoir spécifier une seule adresse (source ou destination). Pour cela, il suffit de mettre les adresses entre crochets, en les séparant par des virgules.

Ainsi, si votre ancien jeu de règles ressemblait au suivant :

```
block in quick on xl0 from 10.0.0.0/8 to any
block in quick on xl0 from 172.16.0.0/12 to any
block in quick on xl0 from 192.168.0.0/16 to any
block in quick on xl0 from 255.255.255.255/32 to any
```

Vous pouvez le remplacer par la seule règle suivante :

```
block in quick on xl0 from { 10.0.0.0/8, 172.16.0.0/12, \
              192.168.0.0/16, 255.255.255.255/32 } to any
```

Ce traitement par lots est également possible pour des interfaces, protocoles et ports. Le programme `pfctl` découpera ce type de règles en une règle par objet contenue dans le lot, ainsi PF peut optimiser votre jeu de règles avec les techniques décrites dans la section 2.9. De plus, l'utilisation de lots améliore grandement la lisibilité lors de l'emploi d'un grand nombre de nœuds, interfaces, protocoles ou ports.

2.8 Expansion de variables

PF supporte également l'expansion de variables, sur le modèle du shell. Les variables sont définies en leur affectant une valeur, et expansées en préfixant leur nom par le signe dollar ('$') :

```
webserver=260.250.1.3/32
pass  in from any to $webserver port = 80 keep state
```

2.9 Optimisation d'un jeu de règles : skip steps

À la différence d'IPFilter, OpenBSD PF ne dispose pas du mot-clé `group`. Les développeurs de PF ont préférés un schéma nommé *skip steps* où les jeux de règles sont optimisés automatiquement.

Imaginons que votre jeu de règles ressemble au suivant :

```
block in quick on xl0 from 10.0.0.0/8 to any
```

```
block in quick on xl0 from 172.16.0.0/12 to any
block in quick on xl0 from 192.168.0.0/16 to any
block in quick on xl0 from 255.255.255.255/32 to any
```

Pour chaque paquet, ce jeu de règles est évalué du début à la fin. Imaginons qu'un paquet soit reçu sur l'interface `xl1`. La première règle est évaluée, mais ne correspond pas. Maintenant, comme les autres règles correspondent également à l'interface `xl0`, PF peut sans problème sauter ces règles.

Quand vous chargez un jeu de règles, les paramètres suivants sont comparés entre les règles successives (dans l'ordre suivant) :

1. interface
2. protocole
3. adresse source
4. port source
5. adresse de destination
6. port de destination

Pour chaque règle, PF calcule automatiquement un « skip step » pour chacun de ces paramètres, qui indiqueront à PF combien de règles ont la même valeur pour le paramètre.

Si un paquet reçu sur l'interface `xl1` est comparé à notre jeu de règles en exemple, PF détectera que l'interface du paquet ne correspond pas à la première règle, et comme les trois règles suivantes mentionnent également l'interface `xl0`, il les sautera.

Ainsi, si vous désirez optimiser votre jeu de règles pour une meilleure performance, vous devriez trier vos règles en commençant par l'interface, puis par protocoles, puis par adresses et ports source, et finalement par adresses et ports de destination.

2.10 Faisons le point

Faisons le point en consolidant ce que nous avons appris sur PF, en utilisant notre exemple de réseau d'entreprise. Le jeu de règles suivant est le résultat de nos cogitations :

```
# positionnons quelques variables
externe=xl0
interne=xl1
entreprise=260.250.1.0/24
```

```
serveurweb=260.250.1.3/32
usurpees={ 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, \
                                    255.255.255.255/32 }

# bloquer par défaut
block in all

# permettre la navigation sur le web à partir du réseau d'entreprise
pass  in quick on $interne proto udp from $entreprise to any \
                                          port = 53 keep state
pass  in quick on $interne proto tcp from $entreprise to any \
                                port = 80 flags S/SA keep state

# dégager les paquets usurpateurs
block in quick on $externe from $usurpees to any

# permet l'accès vers le serveur web d'entreprise depuis Internet
pass  in quick on $externe proto tcp from any to $serveurweb \
                                port = 80 flags S/SA keep state
```

Vous avez ainsi votre pare-feu en place. Bien sûr, ce n'est pas la seule chose à faire pour protéger un réseau d'entreprise, mais pour votre première ligne de défense, PF fera très bien le travail.

2.11 Charger votre jeu de règles

Une fois que vous serez satisfait de votre jeu de règles, vous pourrez le sauvegarder sous le nom de `/etc/pf.conf` et redémarrer votre machine, ou exécuter :

```
pfctl -R /etc/pf.conf
```

3 Ponts Filtrants

Un pont, ou répéteur, est un périphérique réseau qui connecte deux (ou plus) segments réseaux ensemble. On le retrouve communément sous la forme d'une simple boite qui ne fait que répéter tous les paquets provenant d'un segement réseau vers le ou les autres segments réseaux. Un serveur OpenBSD peut être utilisé pour assurer cette fonction, ce qui permet d'utiliser PF pour filtrer au passage le trafic entre les différents segments réseaux.

Cette section explique comment mettre en œuvre un pont filtrant en utilisant OpenBSD PF. Les réseaux utilisés dans les exemples suivants sont ceux d'un réseau d'université, le réseau `IPv4` d'adresse `10.0.0.0/8`, consistant en un nombre assez large de postes de travail pour les étudiants, d'un serveur Web sur `10.0.0.1/32` et d'un serveur de shell sur `10.0.0.2/32`.

Pour des raisons de sécurité, l'équipe technique de l'université souhaite séparer les deux segments réseaux, et filtrer le trafic entre-eux, en minimisant les changements de la topologie réseau existante.

La décision est prise d'utiliser un serveur OpenBSD, avec le segment réseau des postes étudiants sur l'interface `xl0`, et le segment des serveurs connecté sur l'interface `xl1`.

3.1 Deux directions

Les paquets passant à travers le pont passent à travers PF deux fois : ils viennent par une interface, et sortent par l'autre. Ainsi, notre jeu de règles doit permettre le trafic entrant et sortant, nous permettant de commencer avec les règles suivantes :

```
pass in  on xl0 any
pass out on xl0 any
pass in  on xl1 any
pass out on xl1 any
```

Ces règles permettront à tout trafic recçu sur `xl0` et `xl1` d'entrer et sortir du pont, pour permettre au dit trafic d'être expédié sur le bon segment réseau.

3.2 Filtrage à conservation d'état (statefull filtering)

L'implémentaton OpenBSD Packet Filter dispose d'une fonctionnalité très pratique appellée conservation d'état (state-keeping), décrite dans la section

2.3, qui sera utilisée dans notre exemple pour améliorer la sécurité de notre segment réseau pour serveurs.

Cependant, il faut garder à l'esprit une chose quand on utilise du filtrage à conservation d'état : les entrées dans la table d'état sont indéxées par une clef qui repose sur les adresses sources et destination, et ports `TCP`, où l'ordre de ces paires et important. Si un paquet sortant depuis `A` vers `B` crée une entrée dans la table d'état, PF passera sur les paquets sortants depuis `A` vers `B`, et les paquets entrants depuis `B` vers `A`. Il continuera cependant à bloquer les paquets sortants depuis `B` vers `A`, et les paquets entrants depuis `A` vers `B`, ce qui est le cas parfaitement désiré pour les applications n'utilisant pas la technologie de pont.

En revanche, quand on utilise de la conservation d'état sur un pont, un paquet traverse PF deux fois : le paquet entrant sur une interface et le paquet sortant sur l'autre interface.

Il y a deux solutions à ce problème. La première est de créer deux entrées dans la table d'état, une pour chaque connexion, et d'utiliser deux règles avec l'option `keep state`. Cependant, une telle utilisation de la table d'état augmente la charge sur le serveur/pont, et n'est pas recommendé, car comme nous allons le voir, la deuxième option est simple et élégante :

Du point de vue de PF, les paquets tranversent le pont deux fois. Si vous observez une interface, vous y observerez exactement le même trafic que sur l'autre interface, à l'exception de la direction qui est inversée. Ainsi, nous pouvons ignorer une interface et appliquer le filtrage sur l'autre.

Nous voudrions conserver l'état des connexions sur les serveurs Web et shell, et comme nous faisons moins confiance au segment réseau des postes de travail qu'à celui des serveurs, nous filtrerons sur l'interface `xl0`, autorisant simplement tout trafic sur l'interface `xl1` :

```
web=10.0.0.1/32
shell=10.0.0.2/32

pass  in  quick on xl1 all
pass  out quick on xl1 all

block in  on xl0 all
block out on xl0 all

pass  in  quick on xl0 proto tcp from any to $web \
                                port = 80 keep state
pass  in  quick on xl0 proto tcp from any to $shell \
                        port = { 22, 23 } keep state
```

4 Astuces de pare-feu

Dans le but d'améliorer la sécurité du (des) nœud(s) qu'il est sensé protéger, OpenBSD PF dispose de quelques fonctionnalités uniques pour corriger les erreurs des piles `TCP/IP`. Ces fonctionnalités sont détaillées dans cette présente section.

4.1 Modulation d'état : state modulation

Pour assurer le transport des paquets `TCP`, le protocole `TCP` utilise des séquences numérotées, initialisées par un numéro déterminé aléatoirement (ISN) au début de la connexion, et qui s'incrémente pour chaque bit transmis. En revanche, de nombreuses implémentations répendues de `TCP` utilisent un générateur de nombre aléatoire douteux pour générer ces ISN[2], rendant plus facile pour une personne malicieuse, la contrefaçon de paquets provenant de tels systèmes.

C'est pour cela que les développeurs d'OpenBSD PF ont choisi d'ajouter la *modulation d'état (state modulation)*. En pratique, il s'agit de générer un numéro initial de séquence plus aléatoire pour les connexions correspondant à une règle de PF, et de translater les numéros de séquence des paquets passant le pare-feu, entre ceux du nœud original et ceux générés par le pare-feu, et vice-versa.

Cela peut être fait en ajoutant le mot-clé `modulate state` à une règle de PF, comme dans l'exemple suivant qui vise à protéger le réseau d'entreprise décrit dans le chapitre précédent :

```
pass  in quick on xl1 proto tcp from 260.250.1.0/24 to any \
                                    flags S/SA modulate state
```

L'option `modulate state` implique automatiquement l'usage de `keep state`.

4.2 Normalisation des paquets

Comme certaine piles `IP` n'implémentent pas correctement la défragmentation de paquets, OpenBSD PF met à disposition la directive `scrub`. Si une règle `scrub` correspond à un paquet, le composant de normalisation de PF s'assure que le paquet est défragmenté et vierge de toute anomalie avant de

[2]Pour plus d'information sur la génération des ISN, ainsi qu'une étude sur la génération des ISN dans les systèmes d'exploitation les plus répandus, consultez `http://razor.bindview.com/publish/papers/tcpseq.html`

l'envoyer à sa destination finale[3].

Normaliser l'intégralité du trafic d'un réseau demanderait une règle identique à la suivante :

```
scrub in all
```

L'utilisation de la directive `scrub` utilise de façon significative les ressources d'un serveur, ainsi elle devrait être limitée à la protection des implémentations des piles `TCP/IP` les plus faibles.

De plus, les options suivantes peuvent être utilisées avec la directive `scrub` :

no-df purge le bit `don't fragment` d'un paquet `IP` correspondant.

min-ttl ***number*** met en place un *time to live* minimum pour le paquet `IP` correspondant, rejetant les paquets qui ne correspondent pas à ce minimum.

[3] À l'époque de la rédaction de ce document, je n'ai pas une vision claire de l'interaction de la normalisation de paquet avec la conservation d'état. Est-ce que l'un des développeurs de PF pourrait en dire plus là-dessus ?

5 Migration depuis IPFilter

Le modèle d'ensembles de règles d'OpenBSD PF est dérivé de celui d'IPFilter. Cependant, il existe quelques petites différences qui sont présentées dans la présente section.

5.1 `head` and `group` n'existent plus

Les mots-clés `head` et `group`, utilisés par IPFilter pour grouper des règles, ne sont plus utilisés par OpenBSD PF. Si vous aviez l'habitude d'utiliser `head` et `group`, vous devrez manuellement réorganiser vos jeux de règles pour qu'ils puissent fonctionner avec OpenBSD PF.

OpenBSD PF dispose d'un mécanisme automatique d'optimisation des ensembles de règles, appellé « skip step ». Consultez la section 2.9 pour plus d'information.

6 Autres documentations

Il existe un bon nombre de sources d'informations disponibles sur Internet à propos d'OpenBSD PF et des pare-feux. La liste suivante propose quelques-uns des liens qui peuvent être intéressants :

`http://www.benzedrine.cx/pf.html` La page originale de ce qui est maintenant OpenBSD PF.

`http://www.obfuscation.org/ipf/` L'IPFilter HOWTO. Bien que le HOWTO que vous lisez maintenant tente d'être aussi complet que possible sur OpenBSD PF, il peut être intéressant de jeter un coup d'œil aux racines d'OpenBSD PF.

`http://www.linuxdoc.org/HOWTO/Firewall-HOWTO.html` Le Linux Firewall and Proxy HOWTO, qui couvre également le sujet de la mise en œuvre en espace utilisateur de mandataires comme Squid et SOCKS. Écrit pour Linux, il est également intéressant pour des utilisateurs d'OpenBSD. Une adaptation en français a été écrite par Bernard Choppy et se trouve disponible sur FREENIX (`http://www.freenix.org/unix/linux/HOWTO/Firewall-HOWTO.html`)

`http://www.openbsd.org/cgi-bin/man.cgi?query=pfctl&sektion=8&format=html` La page de manuel OpenBSD sur le programme `pfctl` .

`http://www.openbsd.org/cgi-bin/man.cgi?query=pf.conf&sektion=5&format=html` La page de manuel OpenBSD sur les ensembles de règles OpenBSD PF.

`http://www.openbsd.org/cgi-bin/man.cgi?query=pf&sektion=4&format=html` La page de manuel OpenBSD sur le périphérique `pf`. Intéressera principalement les programmeurs.

7 Remerciements

L'auteur tient à remercier les personnes suivantes pour l'aide qu'elles ont apportée sur des sujets difficiles, des clarifications sur le fonctionnement interne de OpenBSD PF, pour les corrections d'erreurs dans les version précédentes de ce document, et plus généralement pour les suggestions proposées, (par ordre alphabétique) :

– Mike Frantzen <frantzen@w4g.org>
– Markus Friedl <markus@openbsd.org>
– Artur Grabowski <art@blahonga.org>
– Daniel Hartmeier <daniel@benzedrine.cx>
– Erik Liden <erik@ipunplugged.com>
– Rod Whitworth <listener@witworx.com>
– Jim Zajkowski <jim@jimz.net>

Ont contribués à la traduction française :
– Olivier Lemaire <olivier.lemaire@IDEALX.com>
– Olivier Tharan <olivier.tharan@IDEALX.com>