Cahiers de l'Admin

Le livre de PF

Packet Filter

Sécurité, filtrage et qualité de service

Peter N. M. Hansteen

Adapté par Maxime Derche

Valable pour OpenBSD 4.6, FreeBSD 7.2 et NetBSD 5

EYROLLES

Peter N. M. **Hansteen**

Cahiers de l'Admin

Le livre de PF

Packet Filter

Adapté par
Maxime **Derche**

EYROLLES

Collection « Accès libre »

Pour que l'informatique soit un outil, pas un ennemi !

Économie du logiciel libre.
F. Elie.
N°12463, à paraître 2009, 195 pages.

Freemind – *Boostez votre efficacité.*
X. Delengaigne, P. Mongin.
N°12448, 2009, 272 pages.

Spip 2 – *Premiers pas pour créer son site avec Spip 2.0.3.*
A.-L. Quatravaux, D. Quatravaux.
N°12502, 2009, 300 pages.

Réussir son site web avec XHTML et CSS.-
M. Nebra.
N°12307, 2e édition, 2008, 306 pages.

Réussir un site web d'association… avec des outils libres !
A.-L. Quatravaux et D. Quatravaux.
N°12000, 2e édition, 2007, 372 pages.

Réussir son site e-commerce avec osCommerce.
D. Mercer.
N°11932, 2007, 446 pages.

OpenERP – *Pour une gestion d'entreprise efficace et intégrée.*
F. Pinckaers, G. Gardiner.
N°12261, 2008, 276 pages.

PGP/GPG – *Assurer la confidentialité de ses mails et fichiers.*
M. Lucas, ad. par D. Garance , contrib. J.-M. Thomas.
N°12001, 2006, 248 pages.

Scenari – La chaîne éditoriale libre.
S. Crozat.
N°12150, 2007, 200 pages.

Mozilla Thunderbird – *Le mail sûr et sans spam.*
D. Garance, A.-L. et D. Quatravaux.
N°11609, 2005, 300 pages avec CD-Rom.

Firefox. *Retrouvez votre efficacité sur le Web !*
T. Trubacz, préface de T. Nitot.
N°11604, 2005, 250 pages.

Hackez votre Eee PC – *L'ultraportable efficace*.
C. Guelff.
N°12437, 2009, 306 pages.

Monter son serveur de mails Postfix sous Linux
M. Bäck *et al*., adapté par P. Tonnerre.
N°11931, 2006, 360 pages.

Ergonomie web – *Pour des sites web efficaces.*
A. Boucher.
N°12479, 2e édition 2009, 440 pages.

Joomla et VirtueMart – *Réussir sa boutique en ligne.*
V. Isaksen, avec la contribution de T. Tardif.
N°12381, 2008, 306 pages.

La 3D libre avec Blender.
O. Saraja.
N°12385, 3e édition, 2008, 456 pages avec DVD-Rom.

Dessiner ses plans avec QCad – *Le DAO pour tous*.
A. Pascual
N°12397, 2009, 278 pages.

Inkscape efficace
C. Gémy
N°12425, 2009, 280 pages.

Ubuntu efficace.
L. Dricot.
N°12362, 3e édition, à paraître 2009.

Gimp 2.6 – *Débuter en retouche photo et graphisme libre*.
D. Robert.
N°12480, 4e édition, 2009, 350 pages.

Gimp 2.4 efficace – *Dessin et retouche photo.*
C. Gémy.
N°12152, 2008, 402 pages avec CD-Rom.

Dotclear 2 – *Créer et administrer son blog.*
A. Caillau.
N°12407, 2008, 242 pages.

Chez le même éditeur

Ouvrages sur Linux et autres systèmes et logiciels libres

R. Hertzog, R. Mas. – **Debian Lenny.** *Gnu/Linux.* – N°12443, 2009, 442 pages avec DVD-Rom.
BSD, 2e édition (coll. *Cahiers de l'Admin*). E. Dreyfus. – N°11463, 2004, 300 pages.
Linux Administration. J.-F. Bouchaudy, G. Goubet. – N°12074, 2007, 800 pages.
Sécuriser un réseau Linux. B. Boutherin, B. Delaunay. – N°11960, 3e édition, 2007, 250 pages.
Mémento Unix/Linux. I. Hurbain, avec la contribution d'E. Dreyfus. – N°11954, 2006, 14 pages.
Debian. *Administration et configuration avancées.* M. Krafft, adapté par R. Hertzog et R. Mas. – N°11904, 2006, 674 pages.

Ouvrages sur la sécurité et l'administration

Sécurité informatique. *Méthode à l'usage des DSI, RSSI et administrateurs.* L. Bloch, C. Wolfhugel.N°12525, 2e édition 2009, 350 pages.
Management de la sécurité de l'information. *Implémentation ISO 27001*. A. Fernandez-Toro. – N°12218, 2008, 350 pages.
Management de la continuité d'activité. E. Besluau. – N°12346, 2008, 256 pages.
ITIL pour un service informatique optimal. C. Dumont. – N°12102, 2e édition, 2007, 378 pages.
Tableaux de bord de la sécurité réseaux. C. Llorens, L. Levier, D. Valois. – N°11973, 2e édition, 2006, 560 pages.
Admin'sys. *Gérer son temps…* T. Limoncelli, adapté par S. Blondeel. – N°11957, 2006, 274 pages.
SSL VPN. *Accès web et extranets sécurisés.* J. Steinberg, T. Speed, adapté par B. Sonntag. – N°11933, 2006, 220 pages.
Sécuriser enfin son PC. *Windows XP et Windows Vista.* P. Legand. – N°12005, 2007, 500 pages.
Sécurité PHP 5 et MySQL. D. Séguy, P. Gamache. – N°12114, 2007, 240 pages.
Mémento VMware Server. *Virtualisation de serveurs*. F. Manzano. N°12320, 2008, 14 pages.
Mémento Cisco. *IOS – Configuration générale.* R. Bergoin, C. Bourg. N°12347, 2008, 14 pages.

Peter N. M. **Hansteen**

Cahiers de l'Admin

Le livre de PF

Packet Filter

Adapté par
Maxime **Derche**

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris Cedex 05
www.editions-eyrolles.com

Remerciements à Christophe Addinquy, Claude Aubry, Laurent Bossavit, Antoine Contal, Elisabeth Ducarre, Marc Dumonte, David Gageot, Jean-Claude Grosjean, Marie-Pia Ignace, Freddy Mallet, Régis Médina, Pascal Pratmarty, Alain Pujol, Jean Tabaka, Dominic Williams.

Traduction autorisée de l'ouvrage en langue anglaise intitulé «The book of PF» de Peter N. M. Hansteen – ISBN 978-1-59327-165-7, publié par No Starch Press. Adapté de l'anglais par Maxime Derche.

À Gene Scharmann,
qui m'a pendant toutes ces années poussé en direction des logiciels libres.

Préface

Le filtre de paquets d'OpenBSD, Packet Filter (ou PF), jouit d'un grand succès et d'une grande attention depuis sa toute première publication dans OpenBSD 3.0, fin 2001. Packet Filter – dont l'histoire est détaillée plus loin dans ce livre – est né du besoin des développeurs et utilisateurs d'OpenBSD. Depuis sa première version, il a grandement évolué : il est devenu l'outil libre le plus puissant pour jouer le rôle de pare-feu, mais aussi pour équilibrer la charge et gérer le trafic réseau. Combiné à CARP et à `pfsync`, il permet aux administrateurs système de protéger leurs services des attaques, de les fiabiliser grâce à de la redondance, et permet la montée en charge en recourant à des grappes de serveurs gérées via `hoststated`.

Certes, je me suis impliqué dans le développement de Packet Filter ; mais j'en suis d'abord et avant tout un très grand utilisateur. J'emploie cet outil d'une part pour la sécurité – afin de gérer les menaces tant internes qu'externes – et d'autre part pour la fiabilité – pour faire tourner de larges pans d'infrastructures critiques de manière redondante et avec prise en compte de la montée en charge. Cela représente des économies pour mon employeur (l'Université d'Alberta, dont je dirige l'équipe d'administration système), tant en termes de temps d'indisponibilité qu'en termes de matériel et de logiciel. PF vous apportera les mêmes choses.

La complexité est un mal nécessaire quand on travaille avec de telles fonctionnalités. Pour quelqu'un qui connaît bien TCP/IP et OpenBSD, la documentation système de PF est plutôt importante et elle se suffit à elle-même. Mais, malgré les nombreux exemples qu'on y trouve, elle ne peut couvrir en détail tout ce qu'on peut faire avec PF (et les outils qui lui sont associés) sans risquer de devenir verbeuse au point d'être inutile pour les personnes expérimentées, qui en ont besoin en tant que référence.

Ce livre comble ce manque. Si vous êtes nouveau venu, il peut bien sûr vous aider à vous mettre à OpenBSD et à PF. Si vous êtes un utilisateur plus expérimenté, cet ouvrage vous montrera des exemples plus complexes, qui servent dans des cas dépas-

sant ceux de la moyenne. Depuis plusieurs années, Peter N. M. Hansteen est une excellente référence pour les personnes qui apprennent à utiliser PF au-delà de son simple rôle de pare-feu. Ce livre prolonge sa vocation à partager ses connaissances avec autrui.

Les pare-feux sont désormais omniprésents : beaucoup en utilisent un, voire plusieurs. Mais ce livre ne se contente pas de décrire la construction d'un pare-feu ; il enseigne aussi les techniques permettant de manipuler le trafic réseau – dont la compréhension est absolument indispensable à tout administrateur système et réseau. Il est aisé de construire ou d'acheter un pare-feu simple ; en revanche un pare-feu que vous pouvez modifier et gérer par vous-même est plus complexe.

Ce livre vous guidera dans cet apprentissage. Vous comprendrez non seulement comment construire un pare-feu, mais aussi comment PF fonctionne et comment exploiter sa puissance. L'ouvrage que vous tenez entre les mains est un investissement pour bien démarrer, sans faux départs ni temps perdu à bidouiller.

Bob Beck,
Directeur de la Fondation OpenBSD
http://www.openbsdfoundation.org
Edmonton, Alberta, Canada

Table des matières

Avant-propos

Le but de ce livre est d'expliquer comment construire un réseau adapté à des besoins identifiés. Pour le mettre en place, nous allons nous plonger dans des sujets relatifs aux pare-feux et aux fonctions qui leur sont associées, en commençant par un peu de théorie accompagnée d'exemples de filtrage et de gestion du trafic réseau.

Nous supposons que vous possédez une connaissance basique ou intermédiaire des concepts de réseau TCP/IP et d'administration Unix.

Toutes les informations de ce livre sont à prendre en compte en gardant à l'esprit l'avertissement qui suit : comme dans de nombreux autres domaines, ce qui est présenté peut être mis en pratique *de plusieurs manières différentes*. En outre, comme pour tout ouvrage traitant d'un logiciel, des changements sont susceptibles de survenir entre l'impression du livre et sa lecture.

Les informations de cet ouvrage sont aussi à jour et correctes que possible, et se réfèrent à OpenBSD version 4.2, FreeBSD 7.0 et NetBSD 4.0, incluant tous les correctifs disponibles un peu avant fin septembre 2007.

Ce livre est le descendant direct d'un didacticiel sur PF relativement populaire. Ce didacticiel est également la source de l'avertissement qui suit. Si vous assistez un jour à l'une de mes conférences, vous y aurez droit aussi :

Avertissement

Ceci n'est pas un HOWTO.

Ce document n'est pas conçu pour être utilisé comme un livre de recettes prémâchées que l'on peut copier-coller.

Pour que cette idée rentre bien, répétez après moi :

Le Serment de l'Administrateur Réseau

```
Ceci est mon réseau.

C'est le mien, ou précisément, celui de mon employeur ;
il est sous ma responsabilité et j'y tiens de tout mon cœur.
Beaucoup d'autres réseaux lui ressemblent,
mais aucun n'est comme lui.

Je jure solennellement
que je ne ferai aucun copier-coller d'un HOWTO sans réfléchir.
```

Ce qu'il faut bien comprendre, c'est que les règles et les configurations que je vous présente fonctionnent (je les ai testées et elles sont, d'une manière ou d'une autre, apparentées à ce qui a été mis en production) ; elles peuvent cependant très bien se révéler simplistes et il est presque certain qu'elles ne seront pas vraiment correctes pour *votre* réseau. Gardez à l'esprit que ce livre est conçu pour vous montrer des choses utiles et pour vous inspirer à réaliser de bonnes choses.

Faites l'effort de comprendre votre réseau et ce que vous devez faire pour l'améliorer.

Proscrivez les copier-coller à l'aveuglette, ni à partir de ce document, ni à partir d'aucun autre.

À propos du livre et remerciements

Le livre est conçu pour se suffire à lui-même. Il doit vous permettre de travailler sur vos machines sans que vous ayez trop fréquemment à fouiller dans les pages de manuel ou à vous référer aux ressources indiquées en annexe A.

Au départ, ce manuscrit était le texte d'une conférence présentée devant un groupe d'utilisateurs. La première présentation de cette conférence a eu lieu lors d'une rencontre du Groupe d'Utilisateurs de [BSD et] Linux de Bergen (Bergen Linux User Group, BLUG), le 25 janvier 2005. Après en avoir vu la traduction anglaise améliorée, Greg Lehey a suggéré que je la travaille encore un peu plus et que je la présente en tant que session didactique d'une demi-journée pour la conférence AUUG 2005. Après une série de révisions de ce didacticiel, j'ai finalement commencé au début de l'année 2007 à travailler sur ce qui allait devenir ce livre.

Les deux paragraphes suivants proviennent du manuscrit du didacticiel et s'appliquent toujours à ce livre :

> Ce manuscrit est une version améliorée d'un manuscrit préparé pour une conférence annoncée en ces termes (traduits du norvégien puis de l'anglais) :
> « Cette conférence traite des pare-feux et des fonctions qui leurs sont associées, avec des exemples réels prenant appui sur le PF (Packet Filter) du projet OpenBSD. PF permet de mettre en place des pare-feux et offre des fonctions telles que la traduction d'adresses réseau (NAT), le contrôle du trafic et la gestion de la bande passante, le tout au sein d'un même système, souple et accessible aux administrateurs système. Peter espère que la conférence vous donnera quelques idées à propos de la manière dont vous pouvez contrôler votre trafic réseau selon vos souhaits – empêcher certaines choses d'entrer dans votre réseau, diriger le trafic vers des hôtes précis et, bien entendu, embêter les spammeurs. »

Certaines parties du didacticiel (en tous cas, toutes celles vraiment utiles) se retrouvent dans ce livre sous une forme ou une autre. Au cours du processus de transformation du didacticiel en livre, un certain nombre de personnes m'ont proposé leurs remarques et leurs suggestions.

Les personnes ayant soulevé des points intéressants et utiles sur les premières versions de ce manuscrit sont Eystein Roll Aarseth, David Snyder, Peter Postma, Henrik Kramshøj, Vegard Engen, Greg Lehey, Ian Darwin, Daniel Hartmeier, Mark Uemura, Hallvor Engen et probablement quelques autres encore qui resteront perdus dans les archives de mes messages électroniques, jusqu'à ce que je puisse les en sortir par un `grep`.

J'aimerais remercier les organisations suivantes pour m'avoir gentiment apporté leur soutien : la NUUG Foundation pour m'avoir offert un voyage et avoir en partie financé mon apparition à la conférence AUUG 2005 ; les organisations respectives des événements AUUG, UKUUG, SANE, BSDCan et AsiaBSDCon, pour m'avoir invité à leurs conférences ; et enfin la FreeBSD Foundation pour avoir sponsorisé mes voyages à BSDCan 2006 et EuroBSDCon 2006.

Enfin, au cours du processus de transformation du manuscrit en livre, plusieurs personnes ont été d'une aide inappréciable pour l'améliorer. Je suis redevable envers Bill Pollock et Adam Wright pour leur développement éditorial précieux ; j'aimerais remercier Henning Brauer pour son excellente relecture technique ; un grand merci à Eystein Roll Aarseth, Jakob Breivik Grimstveit, Hallvor Engen, Christer Solskogen et Jeff Martin pour leurs suggestions et remarques intéressantes sur diverses parties du manuscrit ; et, enfin, un très grand merci à Megan Dunchak et Linda Recktenwald pour leurs efforts qui ont permis au livre d'atteindre son état final. Je suis particulièrement redevable envers Dru Lavigne pour avoir fait les présentations qui ont conduit à l'écriture de ce livre, qui ne serait sinon resté qu'un didacticiel en ligne et un support de conférence occasionnel.

J'adresse en dernier lieu (et non le moindre) mes remerciements à ma chère épouse, Birthe, et à ma fille, Nora, pour tout leur amour et leur soutien avant et pendant le processus d'écriture du livre. Cela n'aurait pas été possible sans vous.

Mais assez parlé, passons sans plus attendre au plat de résistance.

Si vous venez d'autre part

Si vous lisez ceci parce que vous envisagez de migrer votre configuration vers PF en partant d'un quelconque autre système, cette section est faite pour vous. Certaines des questions les plus courantes sont traitées ici, sous la forme de questions-réponses ou FAQ.

PF a l'air vraiment cool. Puis-je le faire tourner sur ma machine Linux ?

En un mot : non. Sur la liste de diffusion de PF, nous avons vu passer au fil des années des annonces de personnes prétendant avoir démarré un projet de portage de PF pour Linux ; mais, à l'heure où j'écris ces lignes (fin 2007), personne n'a affirmé avoir mené à terme un tel projet. La principale raison en est probablement que PF est un composant intégré au cœur de la pile réseau d'OpenBSD. Même après une décennie de développement parallèle, le code d'OpenBSD partage toujours suffisamment de fondements avec les autres BSD pour permettre le portage ; en revanche, porter PF sur un système non-BSD exigerait de réécrire une grande partie de son code, en plus du travail d'intégration nécessaire du côté du système cible.

> **CULTURE** **Signification du sigle BSD**
>
> Si le sigle *BSD* ne vous dit rien, voici une courte explication. Il signifie *Berkeley Software Distribution* (soit *Distribution Logicielle de Berkeley*) et se réfère à l'origine à un ensemble de logiciels développés pour le système d'exploitation Unix, par le personnel et les étudiants de l'Université de Californie, à Berkeley. Au fil du temps, l'ensemble s'est étendu pour devenir un système d'exploitation complet, qui devint à son tour l'ancêtre d'une famille de systèmes comprenant OpenBSD, FreeBSD, NetBSD, DragonFly BSD et même, dans une certaine mesure, le Mac OS X d'Apple.
> Pour une explication compréhensible de ce qu'est vraiment BSD, voir l'article « Explaining BSD » de Greg Lehey et, bien entendu, les sites web des projets cités.
> ▸ http://www.freebsd.org/doc/en/articles/explaining-bsd

Pour utiliser PF, vous devez installer et faire tourner un système BSD tel qu'OpenBSD, FreeBSD, NetBSD ou DragonFly BSD. Ce sont tous de bons systèmes d'exploitation, mais mon favori est OpenBSD, principalement parce que c'est dans ce système d'exploitation que se fait l'essentiel (si ce n'est la totalité) du déve-

loppement de PF ; par ailleurs, je trouve que l'approche très directe de ce système et de ses développeurs est plutôt rafraîchissante.

De temps en temps, les implémentations de PF des autres systèmes permettent de découvrir des bugs et / ou d'apporter des améliorations mineures, qui sont alors intégrées à la base de code principale de PF ; cependant, OpenBSD contient toujours la version de PF la plus avancée. Certaines des fonctionnalités décrites dans ce livre ne sont disponibles que dans les versions les plus récentes d'OpenBSD ; les autres BSD tendent à porter la version de PF la plus récente depuis OpenBSD vers leur propre base de code, à temps pour leur propre version suivante.

Si vous pensez faire tourner PF sous FreeBSD, NetBSD, DragonFly BSD ou d'autres systèmes, vous devriez jeter un œil aux notes de version de votre système pour savoir quelle est la version de PF qui y est intégrée.

Les utilisateurs de Linux souhaitant apprendre les rouages de la configuration d'un réseau sous BSD peuvent se référer à la section « Je connais Linux, mais j'ai besoin d'apprendre les bases sur BSD. Quelques pointeurs ? » ci-dessous.

Je connais Linux, mais j'ai besoin d'apprendre les bases sur BSD. Quelques pointeurs ?

Les différences et les points communs entre Linux et BSD peuvent être un vaste sujet si vous cherchez en profondeur mais, si vous connaissez relativement bien les bases, alors il ne vous faudra pas longtemps pour vous familiariser avec le mode de pensée des systèmes BSD. Dans le reste de ce livre, nous supposerons que vous connaissez les bases de la configuration réseau sous BSD. Donc, si vous connaissez mieux Linux que BSD, voici quelques remarques à propos de la configuration réseau d'un système BSD :

- Linux et BSD utilisent des conventions différentes pour nommer les interfaces réseau. À la différence de la convention de nommage de Linux, les interfaces réseau BSD ne sont pas étiquetées `eth0` et ainsi de suite. Les interfaces sont nommées à partir du nom du pilote qui les contrôle, ainsi que d'un numéro de séquence. Par exemple, les anciennes cartes 3Com utilisant le pilote `ep` apparaissent sous les noms `ep0`, `ep1`, et ainsi de suite, alors que que les cartes Gigabit Intel prendront quant à elles probablement les noms `em0,em1`, etc. Certaines cartes SMC prendront les noms `sn0`, et ainsi de suite. C'est, de fait, plutôt logique et vous verrez que vous vous habituerez facilement à ce système.
- La configuration est centrée sur `/etc/rc.conf`. En général, les systèmes BSD sont organisés pour lire la configuration depuis le fichier `/etc/rc.conf`, qui est consulté au démarrage par le script `/etc/rc`. OpenBSD recommande d'utiliser `/etc/rc.conf.local` pour les modifications locales, car `rc.conf` contient les

valeurs par défaut ; par contre, FreeBSD utilise `/etc/defaults/rc.conf` pour stocker les paramètres par défaut, faisant de `/etc/rc.conf` le bon endroit où apporter ses modifications locales. De plus, OpenBSD utilise un fichier de configuration par interface, nommé `hostname.<if>`, où vous remplacez `<if>` par le nom de l'interface.

- Et enfin, pour apprendre à vous servir de PF, vous devrez vous concentrer sur le fichier `/etc/pf.conf`, qui sera, dans une large mesure, votre création.

Si vous avez besoin d'une introduction plus importante et plus fournie sur le BSD de votre choix, allez voir la documentation du système d'exploitation, y compris les FAQ et les guides, sur le site web du projet. Vous pouvez également trouver en annexe A des suggestions de lectures plus avancées.

Pouvez-vous recommander un outil à interface graphique pour gérer mon jeu de règles PF ?

Ce livre est principalement destiné aux utilisateurs qui éditent eux-mêmes leurs jeux de règles, dans leur éditeur de texte favori[1]. Les jeux de règles donnés en exemple dans ce livre sont tellement simples que vous ne tireriez aucun bénéfice des options de visualisation offertes par les outils à interface graphique.

Un argument revient fréquemment : les fichiers de configuration de PF seraient en général tellement lisibles qu'on ne ressentirait que très rarement le besoin d'un outil de visualisation. Il existe néanmoins plusieurs outils à interface graphique qui permettent d'éditer et / ou de générer des configurations pour PF. Nous pouvons notamment citer *pfsense*, un système basé sur FreeBSD intégrant une interface graphique d'édition de règles.

Je vous recommande de commencer par voir dans ce livre ce qui s'applique à votre situation ; vous déterminerez ensuite si vous avez besoin d'un outil à interface graphique pour vous sentir à l'aise dans la conduite et la maintenance des systèmes que vous construisez.

Existe-t-il un outil que je pourrais utiliser pour convertir ma configuration AutreProduit® en une configuration PF ?

La meilleure stratégie pour migrer des configurations réseau d'un produit vers un autre (y compris des configurations de pare-feux) est sans conteste de revenir aux

1. Je ne m'étendrai pas sur les détails de l'éditeur que j'utilise. Si l'information vous intéresse vraiment, elle est facile à trouver même sans me contacter.

spécifications et politiques de configuration de votre réseau ou de votre pare-feu, afin de mettre en place ces politiques au moyen du nouvel outil.

Il y a plusieurs raisons à cela. Les autres produits auront inévitablement un jeu de fonctionnalités légèrement différent et la configuration existante, que vous avez créée pour AutreProduit®, utilise probablement d'autres approches pour répondre à des problèmes spécifiques, approches qui ne correspondent pas forcément (voire pas du tout) aux fonctionnalités de PF et des outils qui lui sont associés. Créer un ensemble de documents contenant une spécification complète et en toutes lettres de ce que doit faire votre configuration vous donne un autre avantage : il devient possible de vérifier si la configuration utilisée répond effectivement aux objectifs fixés à l'étape de conception.

Une politique documentée et maintenue à jour au fil de l'évolution de vos besoins vous simplifiera la vie. (Un bon point de départ consiste à insérer des commentaires dans votre fichier de configuration, afin d'expliquer le but de vos règles.) Dans certains cadres professionnels, on peut même exiger formellement une politique rédigée en toutes lettres.

L'idée de rechercher un moyen d'automatiser la conversion est tout à fait compréhensible et peut-être même que l'on s'y attend de la part d'un administrateur système. Je vous invite fortement à y résister, pour ne procéder à votre conversion qu'après avoir réévalué vos besoins techniques et opérationnels et (de préférence) après avoir créé ou mis à jour une spécification ou une politique formelle.

> Certains outils à interface graphique servant de panneau d'administration prétendent pouvoir sortir des fichiers de configuration pour plusieurs produits et donc pouvoir être utilisés comme outil de conversion. Toutefois, cela ajoute par là même une autre couche d'abstraction entre vous et votre jeu de règles ; vous dépendez donc des auteurs de l'outil, ce qui vous empêche de bien saisir le fonctionnement de PF. Encore une fois, je recommande de lire au moins les parties intéressantes de ce livre avant même de réfléchir au bien-fondé d'une conversion automatisée.

Où puis-je en apprendre plus ?

Il existe plusieurs sources d'informations de qualité au sujet de PF et des systèmes sur lesquels il fonctionne. Vous en avez déjà trouvé une : ce livre. Vous trouverez des références vers d'autres ressources, en ligne et sur papier, en annexe A.

Si vous disposez d'un système BSD où PF est installé, consultez les pages de manuel en ligne (c'est-à-dire les *pages man*) pour obtenir des informations sur la version exacte du logiciel que vous utilisez. Sauf indication expresse, les informations de ce livre s'appuient sur ce à quoi ressemble le monde vu de la ligne de commande d'un système OpenBSD 4.2.

Petit encouragement : un haïku PF

Si vous n'êtes pas encore convaincu (ou que vous lisez quand même), un petit encouragement serait peut-être le bienvenu. Au fil des années, PF a fait l'objet d'un certain nombre de paroles et d'écrits – parfois stupides, parfois merveilleux et parfois complètement étranges.

Le poème cité ci-dessous est une bonne indication du niveau d'inspiration que PF suscite parfois chez ses utilisateurs. Le poème fit son apparition sur la liste de diffusion de PF, dans un fil de discussion qui avait démarré par un message intitulé « Y a-t-il des choses que PF ne puisse pas faire ? » (« Things PF can't do? »), en mai 2004. L'auteur du message était une personne sans grande expérience des pare-feux et qui trouvait logiquement difficile d'atteindre la configuration désirée.

Cela mena, bien entendu, à quelques discussions, où plusieurs participants furent d'avis que, si PF était difficile pour les débutants, les alternatives n'étaient certainement pas meilleures de beaucoup. Le fil de discussion se termina par le haïku suivant, composé par Jason Dixon. Je le donne intact, accompagné des commentaires de Jason[1] :

> Comparé à iptables, PF est comme ce haïku :
>
> *A breath of fresh air,* Un souffle d'air frais,
> *floating on white rose petals,* Flottant sur de blancs pétales,
> *eating strawberries.* En mangeant des fraises.
>
> Et voilà que je m'emporte :
>
> *Hartmeier codes now,* Hartmeier développe,
> *Henning knows not why it fails,* Henning ne comprend pas
> *fails only for n00b.* Pourquoi les nuls n'y arrivent pas.
>
> *Tables load my lists,* Des tables chargent mes listes,
> *tarpit for the asshole spammer,* Punition pour les spammers.
> *death to his mail store.* Mort à leur commerce !
>
> *CARP due to Cisco,* CARP vient de Cisco,
> *redundant blessed packets,* Paquets redondants bénis,
> *licensed free for me.* Sous licence libre.

Certains des concepts que Jason mentionne ici peuvent ne pas vous sembler familiers mais, si vous poursuivez votre lecture, cela s'éclairera peu à peu.

Je vais à présent arrêter de déblatérer et enchaîner sur le premier chapitre, qui vous présentera certains des concepts les plus importants dans le domaine des réseaux.

1. Jason Dixon, sur la liste de diffusion de PF, le 20 mai 2004.
   Voir http://marc.info/?l=openbsd-pf&m=108507584013046&w=2.

Qu'est-ce que PF ? 1

Vous êtes ici parce que vous avez entendu parler d'un produit réseau nommé PF et il est fort probable que vous lisiez ce livre parce ce que vous souhaitez savoir de quoi il retourne. Il est vraisemblablement utile de commencer par passer un peu de temps sur l'historique du projet, afin de bien remettre les choses dans leur contexte.

CULTURE Retour sur l'histoire de Packet Filter

Le sous-système Packet Filter d'OpenBSD, auquel la plupart des personnes se réfèrent en utilisant simplement la forme abrégée PF, fut initialement écrit au cours d'une période de développement extrêmement rapide, pendant l'été et l'automne 2001 (enfin, dans l'hémisphère Nord), par Daniel Hartmeier et d'autres développeurs d'OpenBSD. Le résultat fut lancé en tant que partie intégrante d'OpenBSD 3.0 en décembre 2001, activé par défaut dans le système de base.

Le nouveau sous-système de pare-feu logiciel d'OpenBSD devint subitement nécessaire quand Darren Reed annonça au monde qu'IPFilter, à l'époque encore assez intimement intégré à OpenBSD, n'était finalement plus distribué sous licence BSD. La nouvelle licence en était presque une copie mot à mot, à laquelle il ne manquait que le droit d'apporter des modifications au code et de distribuer le résultat. Ainsi la version OpenBSD d'IPFilter contenait un grand nombre de modifications et d'adaptations qui, du coup, n'étaient plus autorisées. IPFilter fut donc retiré de l'arbre des sources d'OpenBSD le 29 mai 2001 et, pendant quelques semaines, OpenBSD-current ne comporta aucun logiciel de pare-feu.

Heureusement, en Suisse, Daniel Hartmeier se livrait déjà à quelques petites expériences de hacking dans le code réseau du noyau. Il commença par injecter une petite fonction de sa fabrication dans la pile réseau, puis à faire passer les paquets à travers. Quelque temps plus tard, il se mit à envisager le filtrage. C'est alors que la crise de la licence intervint.

Le premier commit CVS du code de PF arriva le dimanche 24 juin 2001 à 19:48:58 UTC. Suivirent quelques mois d'intense activité, et la version de PF publiée avec OpenBSD 3.0 contenait une implémentation de filtrage de paquets plutôt complète, intégrant la traduction d'adresses réseau (NAT).

UN PEU D'HISTOIRE

Il est intéressant de remarquer que l'épisode du copyright d'IPFilter incita l'équipe OpenBSD à procéder à un audit des licences de la totalité de l'arborescence des sources, afin d'éviter à l'avenir des situations similaires. Dans les mois qui suivirent, d'autres problèmes potentiels furent découverts et résolus, entraînant la suppression de certains problèmes de licence potentiels risquant d'attirer des ennuis à toute personne impliquée dans le développement de logiciels libres. Theo de Raadt résuma tout cet effort dans un message à la liste de diffusion *openbsd-misc* le 20 février 2003 ; ce message est disponible sur les archives de listes de diffusion MARC (et ailleurs) :

- http://marc.info/?l=openbsd-misc&m=104570938124454&w=2.

De ce point de vue, Daniel Hartmeier et les autres développeurs de PF firent bon usage de leur expérience avec le code d'IPFilter. Daniel présenta à USENIX 2002 un article comportant des tests de performance : il montrait que OpenBSD 3.1, Packet Filter avait des performances supérieures ou égales, en condition de stress, à celles d'IPFilter sous OpenBSD 3.1, ou iptables sous Linux.

Les tests lancés sur le PF original d'OpenBSD 3.0 montrèrent aussi que le code avait gagné en efficacité entre les versions 3.0 et 3.1. Voir l'article détaillé sur le site web de Daniel Hartmeier ;

- http://www.benzedrine.cx/pf-paper.html

Tout cela eut lieu il y a plusieurs années et, comme le reste du monde, OpenBSD et PF ont dû s'adapter à l'incroyable vitesse d'évolution des réseaux. Il n'y a pas eu de tests comparables récents, mais au vu de mes expériences en production (et celle d'autres utilisateurs), la surcharge due au filtrage réalisé par PF est pratiquement négligeable.

À seule fin d'information et pour montrer que du vieux matériel bas de gamme peut toujours servir, la machine qui joue le rôle de passerelle entre le réseau de mon travail et le reste du monde est un Pentium III 450 MHz avec 384 Mo de RAM. Lors de ma dernière vérification, la machine n'avait jamais été sollicitée à plus de 4% de sa puissance (96% d'inactivité selon la commande `top`).

Le code de PF a naturellement suscité de l'intérêt dans le reste de la famille BSD : PF est disponible dans le système de base d'OpenBSD comme filtre de paquets par défaut, et le projet FreeBSD l'a graduellement adopté comme l'un des trois systèmes de filtrage intégrés par défaut sous forme de paquetage à partir de sa version 5.3. PF a également été importé dans NetBSD et DragonFly BSD.

AVERTISSEMENT Version de PF traitée

Dans ce livre, nous nous concentrerons principalement sur la version la plus récente de PF, disponible dans OpenBSD 4.2. Lorsque cela s'avérera nécessaire, nous ferons des remarques sur les différences significatives existant entre cette version et celles qui sont intégrées dans les autres systèmes.

> **Remarque** **Packet Filter sous Windows ?**
>
> Il y a même un pare-feu personnel pour Microsoft Windows qui prétend être basé sur PF. Ce produit, appelé *Core Force*, dépasse le cadre de ce livre mais, si vous êtes intéressé, vous pourrez trouver plus d'informations sur le site web de Core Security.
>
> ▸ http://www.coresecurity.com

Définitions préliminaires : filtre de paquets et pare-feu

Il est temps de définir certains termes et concepts déjà employés plus tôt.

Packet Filter est un logiciel de *filtre de paquets*, c'est-à-dire un logiciel qui inspecte les paquets réseau au niveau du port et du protocole, avant de décider ce qu'il faut en faire. Packet Filter opère en majeure partie dans l'espace noyau, au sein du code réseau.

Il fonctionne dans un monde peuplé de *paquets*, de *protocoles*, de *connexions*, de *ports* et de *services*. Dans ce monde, il existe aussi des *interfaces*, des *adresses sources* et des *adresses de destination*, ainsi que quelques autres caractéristiques relatives aux paquets et aux connexions.

À partir des informations de provenance et de destination d'un paquet, de protocole ou de connexion dont il dépend, du port d'où il vient et de celui auquel il est destiné, PF peut décider s'il doit le laisser passer et, le cas échéant, de déterminer où le diriger.

Le filtrage peut aussi se faire à partir du *contenu* du paquet – c'est ce qu'on appelle habituellement le *filtrage de niveau applicatif* – ; mais ce n'est pas ce que fait PF.

> Nous reviendrons plus tard sur certains cas où PF déléguera ce genre de tâches à d'autres logiciels mais, pour le moment, commençons par les bases.

Nous avons déjà mentionné le concept de *pare-feu* (ou *firewall*). En effet la fonctionnalité la plus importante de PF et des logiciels similaires est leur capacité à identifier et bloquer le trafic (ou les utilisateurs) qu'il n'est pas souhaitable de voir pénétrer le réseau local. Depuis quelque temps, ce terme de *pare-feu* est devenu galvaudé et je dois admettre que je n'en suis pas un grand fan. Mais, comme la notion de filtre de paquets est intiment liée à celle de pare-feu dans les esprits, j'utiliserai le terme de *pare-feu* aux endroits où son usage sera approprié.

Traduction d'adresses réseau (NAT)

Autre concept dont nous parlerons beaucoup, celui des adresses d'*entrée* et de *sortie*, et d'adresses *routables* et *non routables*. En fait, ce concept n'est pas directement lié aux pare-feux ou au filtrage de paquets mais, vu la configuration des réseaux de nos jours, il est nécessaire d'en parler.

Soyons clairs à ce propos : la *NAT* (*Network Address Translation*, soit traduction d'adresses réseau) *ne joue pas le rôle d'un pare-feu*. Il s'agit là d'un malentendu courant et si vous poursuivez la lecture de ce livre, vous réaliserez à la fois pourquoi les personnes les moins bien informées tendent à amalgamer NAT et pare-feu et pourquoi cela n'a, en fait, aucun sens. Mais d'abord, revenons-en aux comment et aux pourquoi.

Pourquoi Internet vit-il sur quelques mensonges bénins ?

La terminologie de l'adressage que nous tenons aujourd'hui plus ou moins pour acquise est une relique du début des années 1990. À cette époque, la commercialisation d'Internet venait tout juste de commencer et quelqu'un se mit à calculer le nombre d'ordinateurs qui se connecteraient à Internet si celle-ci se poursuivait. Les chiffres étaient stupéfiants.

Quand les protocoles d'Internet furent rédigés, les ordinateurs étaient des machines grandes et coûteuses, qui servaient habituellement un grand nombre de personnes à la fois, chacune utilisant un terminal plus ou moins évolué. Certains de ces ordinateurs tournaient sous Unix ou même sous l'une des premières versions de BSD, tandis que les autres fonctionnaient généralement avec le système propriétaire du constructeur (même si, à l'époque, il n'était pas très difficile pour un universitaire ou un entrepreneur technique d'avoir accès au code source de ces systèmes propriétaires). Certains utilisateurs produisaient même des correctifs, qui étaient ensuite intégrés dans les versions suivantes du système du constructeur.

Seuls les instituts de recherche, les universités et quelques entreprises sous contrat avec le Pentagone étaient autorisées à se connecter au réseau des réseaux qu'on finirait par appeler Internet. L'idée était essentiellement qu'une adresse de 32 bits, soit quatre octets, suffirait pour très longtemps. Avec un espace d'adressage de 32 bits, il était possible de gérer des millions de machines.

Bien des années plus tard, au début des années 1990, Internet n'était plus un simple projet scientifique financé par le Ministère de la Défense des États-Unis d'Amérique. Le temps de l'expérimentation était révolu ; les théories initiales sur un réseau décentralisé et résistant aux attaques militaires s'étaient révélées non seulement possibles mais également faisables. Le monde avait suffisamment changé pour que la commercialisation d'Internet puisse commencer.

Des FAI commerciaux commencèrent à proposer à leurs clients l'accès à Internet et des millions de petites machines peu coûteuses voulurent subitement s'y connecter. Les nouveaux utilisateurs passaient principalement par des lignes téléphoniques, mais il en arrivait beaucoup et leur nombre était en augmentation constante. Le développement montra tous les signes de continuation et même d'accélération. Cela signifiait que les personnes intelligentes qui avaient bâti le réseau avaient encore du travail à faire.

Une prise en charge native d'IPv6 dans PF

Ces personnes intelligentes prirent le problème à la racine et commencèrent à travailler sur une solution basée sur un espace d'adressage plus grand – appelée depuis *IP version 6* ou, en abrégé, IPv6 – qui utilise des adresses de 128 bits. IPv6 est la solution à long terme, conçue à la fois pour remplacer et pour interopérer (dans la mesure du possible) de manière transparente avec les réseaux existants. Bien que certains systèmes propriétaires aient mis du temps avant de prendre part au challenge IPv6, les BSD bénéficient de la gestion intégrée d'IPv6 grâce au code provenant du projet KAME, qui fait partie de la pile réseau de base depuis des années. PF possède donc une prise en charge native d'IPv6 depuis le tout début.

> **PROJET Kame**
>
> Pour citer la page web principale du projet à l'adresse http://www.kame.net, « Le projet KAME était un effort conjoint de six entreprises Japonaises en vue de fournir une pile libre pour IPv6, IPsec et Mobile IPv6 à destination des variantes de BSD. » Les principales activités de recherche et développement furent considérées achevées en mars 2006. Maintenant que les parties importantes ont été incorporées aux systèmes visés, la seule activité reste la maintenance.

Migrer le monde entier vers un autre type d'adressage réseau était censé prendre quelques années, et on décida qu'une solution transitoire était nécessaire.

> En fait, la migration est loin d'être achevée, en partie à cause de décisions de conception d'IPv6 qui restent controversées à cause de possibles implications en matière de sécurité. Les choses empirèrent légèrement quand des problèmes de sécurité plutôt sérieux furent découverts début 2007, d'autant plus qu'ils sont les conséquences directes de la conception d'IPv6. L'impact de ces problèmes sur l'adoption d'IPv6 n'est pas encore clair au moment où j'écris ces lignes.

En attendant : la solution temporaire appelée NAT

La solution temporaire, qui se révéla moins provisoire et plus populaire que ses inventeurs ne l'avaient probablement prévu, se compose de deux parties.

L'une est un mécanisme conçu pour présenter au reste du monde des « mensonges bénins », en permettant aux passerelles réseau de réécrire les adresses des paquets. L'autre est la désignation de rangs d'adresses, qui n'étaient pas assignées et qui ne seraient utilisées qu'au sein des réseaux ne communiquant pas directement avec Internet. De cette manière, plusieurs machines différentes, placées à des endroits différents, pourraient avoir la même adresse locale mais, parce que l'adresse serait traduite avant que le trafic ne sorte sur Internet, il n'y aurait pas de collision.

Si le trafic généré par ces adresses non routables arrivait sur Internet, les routeurs qui y seraient confrontés auraient une raison valide de bloquer le paquet. Après tout, avec une adresse source appartenant aux rangs privés, les paquets ne devraient jamais être visibles à l'extérieur, dans la partie publique d'Internet.

Nous appelons ce système *traduction d'adresses réseau* (*Network Address Translation*, *NAT*), parfois également désigné par le terme de mascarade d'IP (*IP masquerade*) ou une de ses variantes.

> **NORME RFC 1631 et 1918 pour la NAT**
>
> Les deux RFC qui définissent le pourquoi et le comment de tout ceci datent respectivement de mai 1994 pour la RFC 1631, « Le traducteur d'adresse réseau IP (NAT) »), et de février 1996 pour la RFC 1918, « Allocation d'adresses pour les Internets privés ». Voir l'annexe A pour de plus amples informations et d'autres références.

Il peut y avoir plusieurs raisons pour utiliser les adresses définies par la RFC 1918 mais, traditionnellement et historiquement, la raison principale en était l'indisponibilité ou l'aspect peu pratique des adresses officielles. La RFC 1918 fournissait la dernière pièce du puzzle pour la solution transitoire censée pallier le manque d'adresses IP, avec l'allocation de rangs d'adresses IP spécifiques aux réseaux internes, permettant aux administrateurs réseau de s'y retrouver. En mettant en place le mécanisme de NAT au niveau de la passerelle, la crise semblait résolue. La clé était la traduction, qui permet aux paquets de circuler librement, avec un minimum de restrictions.

> Je continue à contredire les personnes qui croient qu'on ne peut filtrer des paquets sans NAT ou que la NAT fournit toute la sécurité réseau dont on pourrait avoir besoin. Ni l'une ni l'autre de ces assertions ne sont vraies, comme vous le verrez si vous poursuivez votre lecture ou consultez d'autres ouvrages sérieux de littérature réseau. Voir l'annexe A pour de plus amples informations et d'autres références.

PF aujourd'hui

Jusqu'ici, nous avons couvert les généralités. Plusieurs années se sont écoulées depuis 2001 et le code de PF est passé par plusieurs révisions. Certaines de ces révisions ont introduit de nouvelles fonctionnalités majeures, tandis que d'autres ont servi à stabiliser ou à optimiser PF. Sous sa forme actuelle, celle d'OpenBSD 4.2, PF est un filtre de paquets stable et mature, capable de nombreuses choses si vous le souhaitez.

- PF classe les paquets en se basant sur la famille d'adresses, le protocole, le port ou l'ensemble de ports de destination ou source, le type de paquet et l'adresse source ou de destination. Il peut même classer les paquets relativement à une interface ou à un groupe d'interfaces spécifiques ; il peut également, avec un degré de précision raisonnable, classer le trafic en se basant sur le système d'exploitation source, ainsi qu'un certain nombre d'autres paramètres.
- PF peut également diriger le trafic vers des destinations autres que celle désignée par l'expéditeur – par exemple vers une autre machine, vers un programme pour traitement ultérieur ou vers un démon écoutant sur un port, local ou non. (Aux chapitres 3 et 6, nous verrons quelques exemples de configurations où de tels programmes extérieurs fournissent des services spécifiques et interagissent avec PF de plusieurs manières intéressantes.)
- Avant que PF ne soit écrit, OpenBSD intégrait le code ALTQ pour gérer l'équilibrage de charge et le calibrage du trafic. ALTQ a fini par être intégré à PF, principalement pour des raisons pratiques.
- Même si la NAT n'est pas une partie requise dans un filtre de paquets, des raisons pratiques font qu'il est intéressant de gérer la logique de réécriture d'adresses à proximité. PF intègre donc également la logique de NAT.
- Résultat de plusieurs années de développement basé sur les véritables besoins des administrateurs réseau, cet ensemble puissant d'outils est à votre disposition ; il ne vous reste plus qu'à le configurer via un unique fichier de configuration aisément lisible, */etc/pf.conf*.

Votre système est sans doute livré avec un `pf.conf` contenant déjà des suggestions en commentaires, afin de vous aider à configurer votre PF, ainsi qu'avec des exemples dans les répertoires de documentation tels que `/usr/share/pf/`. Ces exemples sont utiles en tant que référence, mais nous ne les utiliserons pas directement dans ce livre. Au contraire : nous construirons dans les chapitres à venir un fichier `pf.conf` à partir de rien, en suivant une approche incrémentale et didactique.

Première configuration simple

2

Dans ce chapitre, nous allons créer une configuration très simple avec PF. Nous commencerons par voir la plus petite configuration possible : une machine unique, configurée pour communiquer avec un réseau unique. Ce réseau pourrait très bien être Internet.

Vos deux outils principaux pour la configuration de PF sont votre éditeur de texte favori et l'outil d'administration en ligne de commande `pfctl`. D'ordinaire, pour l'administration quotidienne, vous éditerez votre jeu de règles dans le fichier `/etc/pf.conf`, puis vous chargerez vos modifications en utilisant `pfctl`. L'application `pfctl` peut également accomplir un certain nombre d'autres tâches et elle possède un *grand* nombre d'options. Nous explorerons certaines de ces options dans les quelques chapitres à venir.

Au cas où vous vous poseriez la question, il existe des interfaces web pour les tâches d'administration de PF, mais elles ne font pas partie du système de base. Les développeurs de PF ne sont pas hostiles à ce genre de solutions mais ils n'ont, à ce jour, jamais vu d'interface graphique de configuration de PF qui surpasse clairement l'édition de `pf.conf` et l'utilisation de `pfctl` en ligne de commande.

Utilisez sudo !

J'ai tendance à utiliser `sudo` quand j'ai besoin de faire quelque chose qui exige des privilèges. `sudo` figure dans le système de base d'OpenBSD et vous le trouverez facilement ailleurs, sous forme de port ou de paquetage. Si vous n'utilisez pas encore `sudo`, vous devriez vous y mettre. Évitez de vous tirer une balle dans le pied simplement parce que vous avez oublié que vous étiez `root` dans une fenêtre de terminal.

Sous OpenBSD : première configuration simple

Si vous souhaitez activer PF au démarrage, vous devez indiquer au système `rc` de démarrer le service. Dans OpenBSD, cela se fait en éditant ou en créant le fichier `/etc/rc.conf.local` et en ajoutant cette ligne magique (quoique simple) :

```
pf=YES        # activer PF
```

De plus, il est possible de spécifier le fichier où PF trouvera ses règles :

```
pf_rules=/etc/pf.conf # spécifiez le fichier contenant vos règles
```

Au démarrage suivant, PF sera activé. Vous pourrez le vérifier en recherchant le message `PF enabled` dans la console.

> **CONSEIL Conserver les paramètres par défaut**
>
> Cela dit, placer votre configuration dans un autre fichier que le */etc/pf.conf* par défaut ne vaut probablement pas le coup. L'utilisation du paramètre par défaut vous permet ici de bénéficier d'un certain nombre de fonctionnalités automatiques bien pratiques, comme la sauvegarde de votre configuration dans */var/backup* toutes les nuits.

Le fichier `/etc/pf.conf` qui apparaît après une installation normale d'OpenBSD, de FreeBSD, de NetBSD ou de tout autre système embarquant PF, contient un certain nombre de suggestions utiles, mais elles sont toutes placées en commentaires.

Vous n'avez en vérité pas besoin de redémarrer votre machine pour activer PF. Vous pouvez le faire aussi facilement en utilisant `pfctl`. Et comme personne ne veut redémarrer sans une bonne raison, tapez cette commande pour activer PF sur un système en cours de fonctionnement :

```
$ sudo pfctl -e
```

Toutefois, comme nous n'avons pour le moment pas de jeu de règles, PF ne fait rien du tout.

Il est sans doute intéressant de remarquer que, si vous redémarrez votre machine à ce moment, le script `rc` d'OpenBSD activera un jeu de règles par défaut, qui est en fait chargé avant l'activation d'une quelconque interface réseau. Ce jeu de règles par défaut est conçu comme une mesure de sécurité au cas où votre passerelle démarrerait avec une configuration invalide. Il vous permet de vous connecter et de corriger la (ou les) erreur(s) de syntaxe qui a (ou ont) empêché votre jeu de règles d'être chargé.

Le jeu de règles par défaut autorise un petit nombre de services de base : `ssh` depuis n'importe où, résolution de noms basique et montage NFS.

Les lignes de commande de pfctl

Pour des raisons pratiques, `pfctl` est capable de gérer plusieurs opérations en une seule ligne de commande. Pour activer PF et charger le jeu de règles, exécutez cette commande :

```
sudo pfctl -ef /etc/pf.conf
```

Mieux vaut répéter que, si vous redémarrez votre machine à ce moment, le script `rc` d'OpenBSD activera un jeu de règles par défaut, qui est en fait chargé avant que *toute* interface réseau soit activée.

Certaines des premières versions des portages de PF négligeaient d'apporter le jeu de règles par défaut. Cela suscita des discussions sur les listes de diffusion de ces projets mais, au moment de la sortie de ce livre, ils devraient tous s'être alignés sur un jeu de règles par défaut qui soit sensé.

Sous FreeBSD : première configuration simple

Les programmes bien écrits voyagent bien ; les utilisateurs de FreeBSD vous diront que, d'où qu'il vienne, tout bon code trouve sa place, tôt ou tard, dans FreeBSD. PF n'a pas fait exception et, depuis FreeBSD 5.2.1 et la fin de la série 4.x, PF et les outils qui lui sont liés font partie du système.

Sous FreeBSD, vous avez en fait besoin d'un peu plus de magie dans votre fichier `/etc/rc.conf`, mais ce n'est toujours qu'un ensemble de commandes simples. Il y a cependant quelques différences entre FreeBSD 4.x d'une part et FreeBSD 5.x et supérieur d'autre part, notamment en ce qui concerne PF. Prenez pour référence le *FreeBSD Handbook*, et tout spécialement le chapitre concernant PF, que l'on trouve à l'adresse http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-pf.html, afin de voir quelles sont les informations qui s'appliquent à votre cas. Le code du PF figurant dans FreeBSD 7.0 est équivalent au code du PF d'OpenBSD 4.1. Si vous allez voir dans votre fichier `/etc/defaults/rc.conf`, vous verrez que les valeurs par défaut des réglages concernant PF sont les suivantes :

```
pf_enable="NO"                  # YES pour activer packet filter (pf)
pf_rules="/etc/pf.conf"         # fichier de règles de PF
pf_program="/sbin/pfctl"        # là où est le programme pfctl
pf_flags=""                     # autres drapeaux pfctl
```

```
pflog_enable="NO"            # YES pour activer le journal de PF
pflog_logfile="/var/log/pflog"   # lieu où stocker le journal de pflogd
pflog_program="/sbin/pflogd"     # là où est le programme pflogd
pflog_flags=""               # autres drapeaux pour pflogd
pfsync_enable="NO"           # Expose l'état de pf pour synchronisation
                             # avec d'autres hôtes
pfsync_syncdev=""            # Interface par laquelle passe pfsync
pfsync_ifconfig=""           # Autres options d'ifconfig(8) pour pfsync
```

Les seules informations que vous devez vraiment ajouter à votre configuration sont :

```
pf_enable="YES"           # Activer PF (charger le module si nécessaire)
pflog_enable="YES"         # Démarrer pflogd(8)
```

Sous FreeBSD, PF est par défaut compilé en tant que module noyau. Cela signifie que vous pouvez démarrer PF par la commande :

```
$ sudo kldload pf
```

suivie de :

```
$ sudo pfctl -e
```

La commande `pfctl -e` devrait produire la sortie suivante :

```
No ALTQ support in kernel
ALTQ related functions disabled
pf enabled
```

En supposant que vous avez inséré les lignes appropriées dans votre fichier `/etc/rc.conf`, vous pourriez également utiliser le script `rcPF` pour l'administrer. Utilisez pour activer PF :

```
$ sudo /etc/rc.d/pf start
```

ou pour le désactiver :

```
$ sudo /etc/rc.d/pf stop
```

Le script `rc` PF propose de même quelques options supplémentaires. Mais remarquons qu'à ce stade, nous n'avons toujours pas de jeu de règles. Ici encore, comme nous n'avons pas pris la peine d'écrire un véritable jeu de règles, PF ne fait rien du tout – il fait juste passer les paquets.

Sous NetBSD : première configuration simple

Sous NetBSD 2.0 et supérieur, PF est disponible en tant que module noyau pouvant être installé via les paquetages (`security/pflkm`) ou compilé au sein d'une configuration de noyau statique. Depuis NetBSD 3.0, PF fait partie du système de base.

Si vous souhaitez activer PF dans la configuration de votre noyau (au lieu de charger le module noyau), ajoutez les lignes suivantes à la configuration de votre noyau :

```
pseudo-device pf            # PF packet filter
pseudo-device pflog         # PF log interface
```

Dans `/etc/rc.conf` vous avez besoin de faire figurer les lignes suivantes pour activer respectivement les modules noyau, PF et l'interface de journalisation de PF :

```
lkm="YES"                   # do load kernel modules
pf=YES
pflogd=YES
```

Si vous avez installé le module, chargez-le par la commande :

```
$ sudo modload /usr/lkm/pf.o
```

suivie de :

```
$ sudo pfctl -e
```

pour activer PF.

Vous pouvez tout aussi bien activer PF en lançant les scripts `rc` :

```
$ sudo /etc/rc.d/pf start
```

et lancer la journalisation :

```
$ sudo /etc/rc.d/pflogd start
```

Pour activer automatiquement le module au démarrage, ajoutez la ligne suivante à `/etc/lkm.conf` :

```
/usr/lkm/pf.o - - - - AFTERMOUNT
```

Si tout est encore correct à cet instant, alors vous êtes prêt à créer votre premier jeu de règles PF.

Premier jeu de règles – machine seule et autonome

Il s'agit de la plus simple configuration possible : une machine seule, qui ne fera tourner aucun service, et qui ne communiquera qu'avec un seul réseau (qui peut être Internet).

Pour le moment, nous utiliserons un fichier `/etc/pf.conf` qui ressemblera à ceci :

```
block in all
pass out all keep state
```

Ce jeu de règles refuse tout trafic entrant, autorise le trafic qui vient de la machine même et retient les informations concernant l'état de nos connexions. C'est ainsi que sont évaluées les règles dans les configurations de PF : elles sont lues de haut en bas et c'est la *dernière* règle de votre jeu correspondant au paquet ou à la connexion qui est appliquée. C'est tout ce que vous devez savoir à ce propos pour l'instant. Nous reviendrons sur l'ordre d'évaluation plus tard, quand notre jeu de règles aura grandi par rapport à celui-ci.

La partie `keep state` (*conserver l'état*) de la règle indique à PF que, lorsqu'une connexion correspond à cette règle, nous souhaitons laisser passer le trafic de retour correspondant à cette connexion, dans l'autre sens. Pour ce faire, nous conservons les informations relatives à la connexion en utilisant une entrée dans la *table d'états*. Ces informations comprennent divers compteurs et numéros de séquence qui sont normalement plutôt utiles. Nous pouvons ordonner à PF d'agir de différentes manières sur les informations d'état mais, dans un cas aussi simple que celui-ci, le but principal est de laisser passer le trafic de retour de la connexion que nous avons initiée, afin qu'il puisse arriver jusqu'à nous.

Remarquons que, depuis OpenBSD 4.1, les règles `pass` conservent par défaut les informations d'état et le jeu de règles équivalent (suivant le nouveau style) est donc encore plus simple :

```
# jeu de règles minimal,
# OpenBSD 4.1 et supérieur conserve l'état par défaut
block in all
pass out all
```

En fait, vous pourriez même laisser tomber le mot-clé `all` si vous le vouliez. Les autres BSD adopteront probablement le nouveau comportement par défaut sous peu et nous nous tiendrons au nouveau style de règle dans le reste de ce livre, en prenant garde de rappeler tout cela de temps en temps, au cas où vous utiliseriez un système utilisant l'ancien style.

En fait, le nouveau comportement par défaut correspond à `keep state flags S/SA`, ce qui assure que seuls les paquets SYN initiaux dans l'établissement d'une connexion peuvent créer un état, éliminant du coup des scénarios d'erreurs compliqués. Si vous souhaitez filtrer sans prendre en compte l'état, vous pouvez spécifier `no state` pour les règles où vous ne voulez pas enregistrer ou conserver les informations d'état. Sous FreeBSD, le code de PF équivalent à celui d'OpenBSD 4.1 a été intégré à la version 7.

Il va presque sans dire que laisser passer tout le trafic généré par un hôte spécifique implique une grande confiance dans la fiabilité de cet hôte. Il ne faut accepter ceci que s'il s'agit d'une machine à laquelle vous savez que vous pouvez faire confiance.

Si vous êtes prêt à utiliser le jeu de règles, chargez-le par la commande :

```
$ sudo pfctl -ef /etc/pf.conf
```

Le jeu de règles devrait se charger sans message d'erreur ni avertissement. Sur tous les systèmes (sauf les plus lents), l'invite `$` devrait nous être renvoyée immédiatement.

Tester le jeu de règles

Même avec un jeu de règles très simple comme celui-ci, vous pouvez procéder à des tests afin de voir s'il fonctionne de la manière attendue ou pas.
Tester pour voir si votre configuration est conforme à vos attentes est toujours une bonne idée et un test adéquat deviendra encore plus essentiel une fois que vous en serez arrivé à des configurations plus compliquées. Une bonne pratique consiste à rédiger un cas de test pour chaque modification apportée au jeu de règles et à compléter avec les résultats que vous espérer observer ; plus tôt vous en prendrez l'habitude et mieux ce sera.
Pour le jeu de règles que nous avons ici, vous pourriez tester quelque chose de basique comme la résolution de noms en regardant la sortie de `$ host eyrolles.com`, qui doit retourner des informations telles que l'adresse IP de l'hôte `eyrolles.com` et les noms d'hôtes des échangeurs de messagerie électronique du domaine. Si vous avez un accès SSH distant sur un autre système, vérifiez que vous pouvez vous y connecter et y lancer des commandes. Vous pourriez également surfer sur le Web (sous OpenBSD, `lynx` fait partie du système de base).
Tous les services auxquels vous tentez d'accéder depuis votre système devraient fonctionner ; tout service auquel vous tenteriez d'accéder sur votre machine, depuis n'importe où ailleurs, devrait vous valoir un message de refus de connexion.

Un peu plus strict, avec des listes et des macros

Le premier jeu de règles était un exemple très élémentaire et, même si nous aurions pu l'utiliser pour illustrer quelques bases du fonctionnement des réseaux et du filtrage de paquets, il est probablement trop simpliste pour être utilisé en pratique. Pour une configuration légèrement plus structurée et complète, nous pouvons construire un exemple un peu plus réaliste. Ce jeu de règles sera cependant toujours basé sur le même système seul et autosuffisant, toujours connecté à un réseau unique.

Dans cette configuration, nous commencerons par tout interdire, puis nous n'autoriserons que les fonctionnalités dont nous sommes sûrs d'avoir besoin. Cela nous donne l'opportunité de présenter deux des fonctionnalités qui font de PF un outil si merveilleux : les listes et les macros.

> **CONSEIL Tout interdire d'abord**
>
> Vous vous demandez peut-être pourquoi j'applique une politique de refus par défaut à ce jeu de règles. La réponse courte est que cela vous donne un meilleur contrôle, au prix d'un peu de réflexion. Toute l'idée du filtrage de paquets réside dans le fait de prendre le contrôle, pas de jouer au chat et à la souris avec les mauvais garçons. Marcus Ranum a écrit un article très amusant et très instructif à ce propos, intitulé « Les Six idées les plus stupides en matière de sécurité informatique », que je vous recommande chaudement. Résumé en français dans un livre sur la sécurité informatique, il est d'une très bonne lecture.
>
> ▸ http://www.ranum.com/security/computer_security/editorials/dumb/index.html
>
> 🕮 Bloch, *Sécurité informatique – Méthode et principes*, Eyrolles 2009 (2e édition)

Nous allons modifier un peu notre `/etc/pf.conf`, en commençant par la règle :

```
block all
```

C'est un peu plus restrictif que le premier jeu de règles que nous avons utilisé. La nouvelle règle bloque tout le trafic dans les deux directions, entrant et sortant. C'est le bon comportement à adopter par défaut et il faudra vous y habituer. Dans tous les jeux de règles complets que nous développerons dans les quelques chapitres à venir, c'est la règle de base pour le filtrage. Les règles placées juste après celle-ci permettront de laisser un peu d'air à votre trafic mais, avant d'en arriver à nos vraies règles, nous devons apporter quelques modifications supplémentaires en tête du fichier de configuration. Nous devons définir des macros que nous pourrons utiliser ultérieurement dans le jeu de règles :

```
tcp_services = "{ ssh, smtp, domain, www, pop3, auth, https, pop3s }"
udp_services = "{ domain }"
```

Nous présentons ici plusieurs éléments. Vous savez à présent à quoi ressemblent les macros et nous avons également montré que les macros peuvent être des listes. Vous vous attendiez peut-être aussi à voir des numéros de port mais, comme vous le voyez, PF comprend les règles qui utilisent des noms de services aussi bien que les règles qui utilisent des numéros de port. Les noms sont ceux qui apparaissent dans votre fichier `/etc/services`.

Cela donne des éléments à insérer dans nos règles, que nous éditons légèrement de manière à ce qu'elles ressemblent à ceci :

```
block all
pass out proto tcp to any port $tcp_services
pass proto udp to any port $udp_services
```

> Souvenez-vous d'ajouter `keep state` à ces règles si votre système est équipé d'une version de PF antérieure à celle d'OpenBSD 4.1.

Les chaînes `$tcp_services` et `$udp_services` sont des références à des macros. Les macros sont développées à leur place au chargement du jeu de règles, et le jeu de règles utilisé verra alors les listes complètes insérées aux endroits où les macros sont référencées. Les macros sont excellentes pour la lisibilité. Même dans un petit jeu de règles comme celui-ci, les règles sont plus simples à comprendre et à maintenir que si nous avions directement intégré la liste complète ou les numéros de port.

C'est là que vous devez prendre l'habitude de toujours chercher les parties de votre règle qui pourraient raisonnablement faire l'objet d'une macro, afin d'en améliorer la lisibilité. Au fur et à mesure que votre jeu de règles s'agrandira, vous ne pourrez que vous réjouir de de l'avoir fait.

Vous vous dites peut-être qu'UDP est sans état et sans connexion ; toutefois, PF crée et maintient des données équivalentes aux informations d'état pour le trafic UDP, de manière à assurer que le trafic UDP de retour est autorisé à son arrivée. Un exemple courant démontrant l'utilité des informations d'état d'UDP est la gestion de la résolution de noms. Quand vous demandez à un serveur de noms de résoudre un nom de domaine en une adresse IP ou une adresse IP en un nom d'hôte, il est assez raisonnable de supposer que vous souhaitez recevoir la réponse. La mémorisation des informations d'état ou de leur équivalent fonctionnel concernant votre trafic UDP rend cela possible.

Comme nous avons modifié notre fichier `pf.conf`, nous devons charger les nouvelles règles :

```
$ sudo pfctl -f /etc/pf.conf
```

S'il n'y a pas d'erreur de syntaxe, `pfctl` ne renverra aucun message au cours du chargement des règles.

Il est intéressant de remarquer que vous pouvez utiliser le drapeau `-v` pour que `pfctl` se montre plus volubile :

```
$ sudo pfctl -vf /etc/pf.conf
```

Mais dans ce cas précis, cela ne fera probablement aucune différence. S'il n'y a pas d'erreur dans votre jeu de règles, `pfctl` ne produira aucune sortie avant de vous renvoyer l'invite de commande.

Si vous avez beaucoup modifié votre jeu de règles, il se peut que vous vouliez vérifier les règles avant de tenter de les charger. La commande permettant cela est `pfctl -nf /etc/pf.conf`. L'option `-n` indique que les règles ne doivent être qu'interprétées et non chargées. Cela vous permet de corriger toutes les erreurs. Si `pfctl` trouve une erreur de syntaxe dans votre jeu de règles, il se terminera avec un message d'erreur mettant en avant le numéro de la ligne où se trouve l'erreur.

Tester le jeu de règles

Une fois que `pfctl` arrive à charger votre jeu de règles sans erreur, c'est le moment de retourner à vos tests afin de voir si les règles que vous avez rédigées se comportent comme prévu.
Le test de résolution de nom par une commande telle que `$ host nostarch.com` devrait toujours fonctionner et produire le même résultat que dans l'exemple précédent. Mais, à ce stade, il est probable que votre système ait gardé en cache votre dernière requête DNS : vous devez donc, pour ce test, choisir un domaine auquel vous n'avez pas accédé récemment (par exemple le domaine d'un parti politique pour lequel vous ne pensez *pas* voter). Vous devriez également être en mesure de surfer sur le Web et d'utiliser des services liés à la messagerie électronique. Mais toute tentative d'accéder à des services autres que `ssh`, `smtp`, `domain`, `www`, `pop3`, `auth`, `https` et `pop3s` sur un quelconque système distant devrait échouer. De même que pour la première batterie de tests, votre propre système devrait aussi refuser toute connexion ne correspondant à aucune entrée de la table d'états. Cela signifie que seul le trafic de retour correspondant aux connexions initiées par votre machine est autorisé à entrer.

Dans tous les cas, et à moins que vous n'ayez vidangé (*flush*) vos règles (en utilisant `pfctl -F` suivi d'un mot-clé de spécification) avant d'essayer de charger vos nouvelles règles à partir de votre fichier de configuration, le dernier jeu de règles valide sera toujours en usage et ce, jusqu'à ce que soit vous désactiviez PF, soit vous chargiez un nouveau jeu de règles. C'est ainsi que fonctionne le chargement d'un jeu de règles : `pfctl` vérifie la syntaxe, puis charge les règles entièrement et enfin les applique. Quand un jeu de règles valide est chargé, il n'y a pas d'état intermédiaire dans lequel

un jeu partiel ou vide serait chargé. Cela signifie que la vidange du jeu de règles est rarement une bonne idée, puisqu'elle met effectivement votre filtre de paquets en mode `pass all` (tout laisser passer).

Les statistiques de pfctl

Les tests que vous venez d'effectuer ont montré que PF fonctionne et que, fort heureusement, vos règles se comportent comme prévu. Mais dans d'autres circonstances vous voudrez peut-être, d'une part, vous assurer que PF est bien lancé et, d'autre part, visualiser les statistiques de son activité. En plus d'activer ou de désactiver PF et de charger les jeux de règles, le programme `pfctl` offre de nombreux types d'affichage d'informations. Pour accéder à ces fonctionnalités, utilisez `pfctl -s` en ajoutant le type d'informations que vous souhaitez afficher.

L'exemple suivant vient de ma passerelle personnelle, pendant que je travaillais à l'écriture de ce livre :

```
$ sudo pfctl -s info
Status: Enabled for 17 days 00:24:58          Debug: Urgent

Interface Stats for ep0              IPv4              IPv6
  Bytes In                     9257508558                 0
  Bytes Out                     551145119               352
  Packets In
    Passed                        7004355                 0
    Blocked                         18975                 0
  Packets Out
    Passed                        5222502                 3
    Blocked                            65                 2

State Table                         Total              Rate
  current entries                      15
  searches                        19620603            13.3/s
  inserts                           173104             0.1/s
  removals                          173089             0.1/s
Counters
  match                             196723             0.1/s
  bad-offset                             0             0.0/s
  fragment                              22             0.0/s
  short                                  0             0.0/s
  normalize                              0             0.0/s
  memory                                 0             0.0/s
  bad-timestamp                          0             0.0/s
  congestion                             0             0.0/s
```

```
ip-option                    28            0.0/s
proto-cksum                 325            0.0/s
state-mismatch              983            0.0/s
state-insert                  0            0.0/s
state-limit                   0            0.0/s
src-limit                    26            0.0/s
synproxy                      0            0.0/s
```

Ici, la première ligne indique que PF est activé et qu'il tourne depuis un peu plus de deux semaines, ce qui remonte à la dernière mise à jour de mon système vers le dernier snapshot en date (oui, à cette époque ma passerelle personnelle tournait sous OpenBSD-current). Les informations sont globalement alignées sur les statistiques que vous pourriez trouver sur la passerelle d'un petit réseau qui ne fonctionnerait qu'en IPv4.

EXPLICATION DE TEXTE

Ne vous alarmez pas des trois paquets passés et des deux bloqués dans la colonne IPv6. IPv6 est intégré par défaut dans OpenBSD et, au moment de la configuration de l'interface réseau, la pile TCP/IP envoie des requêtes de sollicitation du voisinage IPv6 pour le lien adresse locale par défaut. Dans une configuration réseau IPv4 exclusive, seuls les tous premiers paquets passent réellement et, au moment où notre jeu de règles PF est complètement chargé, les deux derniers paquets IPv6 sont bloqués par notre règle de blocage par défaut, `block all`.

Vous voudrez peut-être à présent passer un peu de temps à explorer `pfctl` et votre configuration. Comme nous l'avons déjà signalé auparavant, `pfctl` est un programme puissant qui propose un grand nombre d'options. Ainsi, `pfctl -s all` fournit des informations très détaillées sur le PF en cours d'exécution. Essayez cette commande et voyez par vous-même. Tant que vous y êtes, jetez un œil aux autres options de `pfctl`. La commande `man 8 pfctl` vous donnera toutes les informations.

COMMUNAUTÉ Traduction française des pages de manuel d'openBSD

Le traducteur de ce livre héberge sur son site web un projet de traduction des pages de manuel d'OpenBSD et les pages de manuel relatives à PF ont déjà commencé à être traduites. N'hésitez pas à consulter http://www.mouet-mouet.net/doku.php?id=maxime:openbsd:manpages-fr pour voir les pages traduites et, pourquoi pas, participer au projet !

Avec le dernier jeu de règles, vous disposez d'une machine unique qui devrait pouvoir communiquer plutôt bien avec d'autres machines connectées à Internet. Ce jeu de règles vraiment élémentaire sert de point d'entrée pour prendre le contrôle du trafic sur votre réseau. La notion de contrôle est l'un des thèmes sous-jacents les plus

importants de ce livre. De bien des manières, le reste de cet ouvrage évoquera comment étendre votre jeu de règles dans le but de répondre aux besoins de votre réseau, tout en conservant fermement le contrôle de ce qui circule dans vos réseaux.

Mais certaines choses manquent à l'appel. Vous voudrez probablement ajouter des règles pour laisser passer, au moins, du trafic ICMP et UDP – ne serait-ce que pour vos propres besoins de dépannage.

En allant plus loin, vous devriez commencer à penser aux services réseau qui ont un impact sur votre sécurité, comme `ftp`. Même s'il existe des options plus modernes et plus sécurisées, vous serez vraisemblablement amené à gérer ce type de service. Vous n'avez peut-être pas à le faire dans l'immédiat, sur le système personnel autosuffisant, mais, quand vous en serez à mettre en place une passerelle pour un réseau impliquant plus d'utilisateurs et davantage d'ordinateurs, ces conséquences gagneront en importance.

L'utilisation intelligente du filtrage de paquets pour gérer des services qui exigent de la sécurité est un thème récurrent de ce livre.

Les choses sérieuses 3

Dans les chapitres précédents, nous avons présenté une configuration simple, pour un filtrage de paquets simple, dans le cas d'une machine unique. Dans ce chapitre nous continuerons avec cette configuration simple, mais nous allons avancer vers un terrain plus conventionnel : la passerelle filtrante. Bien que la plupart des sujets traités dans ce chapitre soient potentiellement utiles dans le cas d'une machine unique, nous nous focaliserons à présent sur la mise en place d'une passerelle qui gérera les services réseau courants.

Une passerelle simple, avec NAT si besoin

Nous allons commencer par construire ce que vous associez probablement au terme de *pare-feu* : une machine qui agit comme une passerelle pour au moins une autre machine. En plus de transmettre les paquets entre ses divers réseaux, la mission de cette machine sera d'améliorer le rapport signal / bruit du trafic réseau qu'elle gère. C'est là qu'entre en jeu notre configuration PF.

Mais avant de nous plonger dans les détails techniques, nous devons creuser un peu la théorie. Suivez-moi, cela vous évitera les problèmes que j'ai trop souvent observés sur les listes de diffusions et les groupes de nouvelles.

Les passerelles et le piège de in, out et on

Avec la configuration à machine unique, la vie est relativement simple. Le trafic que vous créez doit passer en direction du reste du monde ou être bloqué par vos règles de filtrage, et vous décidez de ce que vous souhaitez laisser entrer en provenance d'ailleurs.

Quand vous montez une passerelle, vos perspectives ne sont plus les mêmes. Vous passez de l'état d'esprit *Moi contre le réseau extérieur* à la posture *Je décide de ce qui transite vers tous les réseaux auxquels je suis connecté ou en provenance d'eux*. La machine possède au moins deux interfaces réseau (voire plus), chacune étant connectée à un réseau différent.

Il est alors raisonnable de penser que, si vous voulez que le trafic passe du réseau connecté à `re1` aux hôtes connectés à `re0`, vous aurez besoin d'une règle telle que :

```
pass in proto tcp on re1 from re1:network to re0:network port $ports
keep state
```

qui gardera également une trace de l'état des connexions.

> En fait, même si la partie `keep state` est redondante avec le comportement par défaut si vous travaillez sous OpenBSD 4.1 ou équivalent, il n'est généralement pas nécessaire de retirer cette spécification de vos règles lorsque vous mettez à jour votre jeu de règles depuis une version précédente. Pour faciliter la transition, les exemples de ce livre feront cette distinction quand cela sera nécessaire.

Mais l'une des erreurs les plus courantes et suscitant le plus de plaintes en matière de configuration de pare-feu consiste à ne pas se rendre compte que le mot-clé `to` ne garantit pas, en lui-même, que le paquet sera autorisé à passer au niveau de toutes les interfaces.

La règle que nous avons écrite ne laisse passer (`in`) que le trafic destiné (`to`) au niveau (`on`) de l'interface nommée dans la règle.

Pour permettre au paquet d'avancer un peu plus loin et d'atteindre le réseau suivant, vous avez besoin d'une règle correspondante énonçant par exemple :

```
pass out proto tcp on re0 from re1:network to re0:network port $ports
keep state
```

Cette règle fonctionnera, mais n'atteindra pas forcément votre but. En fait, cette règle ne laisse passer que les paquets destinés au réseau directement connecté à `re0`, rien de plus.

Si vous avez de bonnes raisons pour nécessiter des règles aussi spécifiques dans votre jeu, alors vous savez que vous en avez besoin et vous en connaissez les raisons. Une fois que vous aurez atteint la fin du livre (ou peut-être un peu avant), vous devriez être capable de savoir si de telles règles sont nécessaires. Mais, pour la passerelle simple que nous construisons dans ce chapitre, vous voulez probablement écrire des règles qui ne soient pas spécifiques à une interface. En fait, il est des cas où il n'est

même pas réellement utile de spécifier une direction. Ce que vous voulez probablement utiliser, c'est une règle qui dit :

```
pass proto tcp from re1:network to any port $ports keep state
```

afin d'autoriser votre accès local à Internet et de laisser le travail de détective aux codes `antispoof` et `scrub`. (Ils sont tous deux plutôt bons et nous y reviendrons plus tard.) Pour le moment, nous acceptons juste le fait que, pour une configuration simple, les règles sur les extrémités `in` et `out` des interfaces ne valent pas la peine d'être employées.

Pour un administrateur système surchargé, un jeu de règles lisible est un jeu de règles sain.

Dans le reste de ce livre, et sauf exceptions, nous garderons les règles aussi simples que possible à des fins de lisibilité. Dans certains cas, nous devrons spécifier la direction et l'interface : nous y reviendrons plus loin dans le livre.

Rappel sur ce qu'est votre réseau local…

Nous avons présenté un peu plus haut dans ce chapitre la notation `interface:réseau`. Il s'agit d'un beau raccourci, mais vous pouvez rendre votre jeu de règles encore plus lisible et facile à maintenir en améliorant un peu la macro.

Par exemple, vous pourriez définir une macro `$localnet` qui représenterait le réseau directement attaché à votre interface interne (`re1:network` dans les exemples précédents).

Vous pourriez tout aussi bien changer la définition de `$localnet` en notation adresse IP/masque de sous-réseau pour nommer un réseau, comme `192.168.100.0/24` pour un sous-réseau d'adresses IPv4 privées, ou encore `fec0:dead:beef::/64` pour un rang IPv6.

Si votre réseau l'exige, vous pourriez même définir votre `$localnet` comme une liste de réseaux. Quels que soient vos besoins, une bonne définition de `$localnet` et une règle `pass` calquée sur ce modèle :

```
pass proto { tcp, udp } from $localnet to any port $ports
```

peuvent vous éviter quelques migraines. Nous nous tiendrons à cette convention à partir d'ici.

Mise en place

Nous supposons que la machine a gagné une nouvelle carte réseau ou que vous avez tout du moins relié votre réseau local à un (ou plusieurs) réseau(x) supplémentaire(s), que ce soit via Ethernet, PPP ou autre.

Dans notre contexte, il n'est pas très intéressant de voir en détail le processus de configuration des interfaces. Par contre, nous devons savoir que les interfaces sont activées et qu'elles fonctionnent.

Pour la discussion et les exemples qui suivent, la seule différence entre une configuration PPP et une configuration Ethernet sera le nom des interfaces, et nous ferons notre possible pour nous débarrasser des noms de ces interfaces aussi vite que possible.

En premier lieu, nous devons activer la configuration passerelle afin que la machine transfère vers d'autres réseaux le trafic qu'elle reçoit sur une interface, via une autre interface. Nous le ferons pour commencer sur la ligne de commande grâce au programme `sysctl` ; en IPv4 traditionnel, on procède ainsi :

```
# sysctl net.inet.ip.forwarding=1
```

Si vous avez besoin de transférer le trafic IPv6, la commande `sysctl` est :

```
# sysctl net.inet6.ip6.forwarding=1
```

C'est bien pour le moment mais, pour que cela fonctionne toujours après un redémarrage de l'ordinateur, vous devez entrer ces réglages dans les fichiers de configuration concernés.

Sous OpenBSD et NetBSD, on procède en éditant `/etc/sysctl.conf`. Ces lignes figurent en commentaires dans le fichier `sysctl.conf` par défaut d'OpenBSD et on les active en supprimant le dièse (`#`) qui figure en début de ligne, pour obtenir ceci :

```
net.inet.ip.forwarding=1
net.inet6.ip6.forwarding=1
```

Sous NetBSD, vous devez ajouter ces lignes au fichier si elles n'y figurent pas encore. Éditer `/etc/sysctl.conf` fonctionnera également sous FreeBSD mais, si l'on veut respecter les conventions de ce système, alors il faut ajouter les lignes suivantes au fichier `/etc/rc.conf` :

```
gateway_enable="YES"      #pour ipv4
ipv6_gateway_enable="YES" #pour ipv6
```

L'effet est identique : le script `rc` de FreeBSD fixe les deux valeurs via la commande `sysctl`. Mais il y a une plus grande partie de la configuration de FreeBSD qui se trouve centralisée dans le fichier `rc.conf`.

Voici venu le moment de vérifier : les interfaces que vous comptez utiliser sont-elles toutes activées ? Utilisez `ifconfig -a` ou `ifconfig nom_interface` pour le savoir.

Sur l'un de mes systèmes, la sortie de `ifconfig -a` ressemble à ceci :

```
peter@delilah:~$ ifconfig -a
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33224
        groups: lo
        inet 127.0.0.1 netmask 0xff000000
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x5
xl0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:60:97:83:4a:01
        groups: egress
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 194.54.107.18 netmask 0xfffffff8 broadcast 194.54.107.23
        inet6 fe80::260:97ff:fe83:4a01%xl0 prefixlen 64 scopeid 0x1
fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:30:05:03:fc:41
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 194.54.103.65 netmask 0xffffffc0 broadcast 194.54.103.127
        inet6 fe80::230:5ff:fe03:fc41%fxp0 prefixlen 64 scopeid 0x2
pflog0: flags=141<UP,RUNNING,PROMISC> mtu 33224
enc0: flags=0<> mtu 1536
```

Votre configuration est probablement un peu différente. Ici, les interfaces physiques de mon système sont `xl0` et `fxp0`, alors que les interfaces logiques `lo0` (l'interface de bouclage, *loopback*), `enc0` (l'interface d'encapsulation pour IPsec) et `pflog0` (le périphérique de journalisation de PF) existent très certainement sur votre système aussi.

Si vous utilisez une connexion de type *dial-up* (modem téléphonique), vous utilisez probablement une variante quelconque de PPP pour vous connecter à Internet et votre interface externe est, en fait, le pseudo-périphérique `tun0`. Si votre connexion passe par une sorte de connexion de type *broadband* (haut-débit) comme l'ADSL, alors vous disposez peut-être d'une interface Ethernet pour communiquer avec. Mais si vous faite partie du grand nombre d'utilisateurs d'ADSL qui utilisent PPP par Ethernet (PPPoE), alors l'interface externe correcte sera l'un des deux pseudo-périphériques `tun0` ou `pppoe0` (selon que vous utilisez le pppoe(8) de l'espace utilisateur ou le pppoe(4) du noyau), et non l'interface Ethernet physique.

Selon votre configuration, vous devrez peut-être procéder à d'autres réglages spécifiques à un périphérique. Tout ce que je peux dire, c'est qu'il vous faut le faire pour que nous puissions avancer vers le niveau TCP/IP et nous occuper de notre configuration de filtrage de paquets.

Si vous prévoyez toujours d'autoriser tout trafic initié par les machines internes, votre `/etc/pf.conf` pourrait ressembler à ceci :

```
ext_if = "re0"     # macro for external interface
                   # - use tun0 or pppoe0 for PPPoE
int_if = "re1"     # macro for internal interface
localnet = $int_if:network
# ext_if IP address could be dynamic, hence ($ext_if)
nat on $ext_if from $localnet to any -> ($ext_if)
block all
pass from { lo0, $localnet } to any keep state
```

Remarquez l'utilisation de macros pour assigner des noms logiques aux interfaces réseau. Ici, on utilise des cartes RealTek Gigabit Ethernet, mais c'est la dernière fois que cela aura un intérêt pour nous. Dans des cas aussi simples que celui-ci, nous ne gagnons pas beaucoup à utiliser des macros mais, dès que le jeu de règles se mettra à grossir, vous apprendrez à apprécier la lisibilité qu'elles apportent.

Remarquez également la règle `nat`. C'est là que nous gérons la traduction d'adresses réseau, depuis l'adresse non routable de votre réseau local vers l'unique adresse officielle qui vous a été assignée. Si votre réseau utilise des adresses routables officielles, vous pouvez abandonner cette ligne pour votre configuration.

Les parenthèses qui entourent la dernière partie de la règle `nat`, `($ext_if)`, sont là pour compenser la possibilité que l'adresse IP de l'interface externe soit assignée dynamiquement. Ce détail assure que votre trafic réseau ne sera pas interrompu en cas de changement d'adresse IP externe.

D'un autre côté, ce jeu de règles autorise probablement davantage de trafic que ce que vous voulez réellement laisser sortir de votre réseau. Sur l'un des réseaux où j'ai travaillé, la macro équivalente était :

```
client_out = "{ ftp-data, ftp, ssh, domain, pop3, auth, nntp, http,
 ➥ https, 446, cvspserver, 2628, 5999, 8000, 8080 }"
```

avec la règle :

```
pass inet proto tcp from $localnet to any port $client_out
```

Cela peut sembler une sélection de ports un peu curieuse, mais c'est exactement ce dont avaient besoin les personnes qui travaillaient sur ce réseau à cette époque. Certains des ports donnés sous forme numérique étaient exigés par des applications précises. Vos besoins sont probablement différents dans le détail, mais cela devrait couvrir au moins certains des services les plus utiles.

Il y a également quelques règles `pass` supplémentaires. Nous reviendrons bientôt sur les plus intéressantes d'entre elles. Voici une règle `pass` utile à ceux qui veulent avoir la possibilité d'administrer leurs machines à partir d'un autre emplacement :

```
pass in inet proto tcp from any to any port ssh
```

ou, si vous préférez :

```
pass in inet proto tcp from any to $ext_if port ssh
```

La partie `from any` est effectivement très permissive. Elle vous permet de vous connecter depuis n'importe où, ce qui est intéressant si vous voyagez beaucoup et que vous avez besoin d'un accès SSH depuis des endroits inconnus partout dans le monde. Si vous n'êtes pas aussi mobile que cela (mettons que vous n'avez pas pris goût aux conférences organisées aux antipodes ou que vous voulez vraiment que vos collègues se débrouillent par eux-mêmes quand vous êtes en vacances), alors vous voudrez peut-être resserrer la partie `from` afin qu'elle ne comprenne que les endroits d'où vous et les autres administrateurs êtes susceptibles de vous connecter, pour des raisons professionnelles légitimes.

De toute façon, ce jeu de règles très basique n'est pas encore complet. La prochaine étape consiste à faire fonctionner le service de noms pour nos clients. Nous commençons par une autre macro au début de notre jeu de règles :

```
udp_services = "{ domain, ntp }"
```

Ajoutons à cela une règle qui laisse passer ce trafic au travers du pare-feu :

```
pass quick inet proto { tcp, udp } to any port $udp_services
```

Remarquez le mot-clé `quick` dans cette règle. Nous avons commencé à écrire des jeux de règles composés de plusieurs règles et le temps est venu de revoir les relations et les interactions qui existent entre ces règles.

Nous avons effleuré ce sujet dans les chapitres précédents, mais répéter ce point ne fera pas de mal : les règles sont évaluées de haut en bas, dans l'ordre où elles sont

écrites dans le fichier de configuration. Pour chaque paquet ou connexion évalués par PF, c'est *la dernière règle qui correspond* qui est appliquée.

Le mot-clé `quick` permet de s'affranchir de cet ordre. Lorsqu'un paquet correspond à une règle `quick`, il est traité selon cette règle. Le paquet n'est pas comparé aux règles suivantes, même si certaines peuvent lui correspondre. Au fur et à mesure que votre jeu de règles grandira, vous trouverez cela pratique, par exemple quand vous aurez besoin d'isoler quelques exceptions des règles générales.

Cette règle `quick` gère également le protocole NTP (*Network Time Protocol*), qui est utilisé pour la synchronisation des horloges. Le point commun entre le service de noms et les protocoles de synchronisation d'horloge est qu'ils peuvent, dans certaines circonstances, communiquer alternativement en TCP et en UDP.

Tester votre jeu de règles

Vous n'avez peut-être pas encore pris l'habitude de formaliser une batterie de tests pour votre jeu de règles, mais vous avez pourtant toutes les raisons de le faire afin de déterminer si votre configuration fonctionne comme prévu.

Les même tests que pour le tout premier exemple du livre s'appliquent encore, sauf que vous devez tester depuis les autres hôtes de votre réseau en plus de votre passerelle filtrante. Pour chacun des services que vous avez spécifiés dans vos règles `pass`, vérifiez que les machines de votre réseau local reçoivent les résultats attendus. Depuis toutes les machines de votre réseau local, une commande comme `$ host eyrolles.com` doit renvoyer exactement le même résultat que lors du test effectué sur la machine unique, et le trafic correspondant aux services spécifiés doit passer.

> **Humour**
>
> Sauf si les informations ont changé. Bien entendu, nul n'est à l'abri d'une plaisanterie car certains administrateurs système sont des blagueurs invétérés… mais, la plupart du temps, les modifications de zones DNS sont dues à de véritables besoins concernant une organisation ou un réseau en particulier.

Vous pouvez pensez que cela n'est pas nécessaire, mais vérifier que le jeu de règles fonctionne comme prévu ne fait pas de mal, même depuis l'extérieur de votre passerelle. Si vous avez suivi ce chapitre à la lettre, il ne devrait pas être possible de contacter les machines de votre réseau local depuis l'extérieur.

Pourquoi seulement des adresses IP et pas de noms d'hôtes ou de noms de domaines ?

En regardant les exemples donnés jusqu'ici, vous avez probablement remarqué que les jeux de règles contiennent tous des macros qui se développent en adresses IP ou en rangs d'adresses IP, mais jamais en noms d'hôtes ou de domaines. Vous vous demandez probablement pourquoi. Après tout, nous avons vu que PF vous permet d'utiliser des noms de services dans vos règles, alors pourquoi pas des noms d'hôtes ou des noms de domaines ?
La réponse est que vous pouvez certes utiliser des noms de domaines ou d'hôtes dans vos règles, mais le jeu de règles ne serait alors valide qu'une fois le service de noms actif et accessible. Or dans la configuration par défaut, PF est chargé avant l'activation de tout service réseau. Cela signifie que, si vous voulez utiliser des noms de domaines ou d'hôtes dans votre configuration PF, vous devez modifier la séquence de démarrage du système (notamment en éditant `/etc/rc.local`) afin de ne charger le jeu de règles dépendant du service de noms qu'une fois celui-ci disponible.

Le triste cas de FTP

La courte liste des ports TCP les plus courants, vue plus haut, contient entre autres FTP, protocole de transfert de fichiers classique. FTP est une relique du début d'Internet, époque où les expérimentations étaient nombreuses et où la sécurité n'était pas encore une réelle préoccupation. FTP date en fait d'avant TCP/IP, et il est possible de suivre la trace du développement du protocole au travers de plus de 50 RFC.

RFC RFC 114 du protocole FTP : 1971

La toute première RFC décrivant le protocole FTP est la RFC 114, datée du 10 avril 1971. Le passage à TCP/IP se produisit avec FTP version 5, comme indiqué dans les RFC 765 et 775, datant respectivement de juin et décembre 1980.

Après plus de trente ans, FTP est à la fois une vieille épine dans le pied et un enfant à problèmes, surtout pour les administrateurs qui tentent de combiner FTP et pare-feu. FTP est un protocole ancien et bizarre, présentant beaucoup d'inconvénients. Les principales charges qui pèsent contre lui sont :

- les mots de passe sont transférés en clair ;
- le protocole demande au moins deux connexions TCP (contrôle et données) sur des ports distincts ;
- lorsqu'une session est établie, les données sont communiquées via des ports tirés au hasard.

Tous ces points constituent des défis en termes de sécurité, avant même d'envisager les faiblesses potentielles des logiciels clients ou serveurs. Tous les vieux barbus des réseaux vous le diront : ces problèmes sont connus pour surgir au pire moment possible.

Quel que soit le cas de figure, il existe des options plus modernes et plus sécurisées pour le transfert de fichiers, telles que `sftp` ou `scp`, qui gèrent toutes deux l'authentification et le transfert des données via des connexions chiffrées. Les professionnels compétents du domaine des TIC devraient préférer des formes de transfert de fichiers autres que FTP.

Sans préjuger de notre professionnalisme ou de nos préférences, il y a toujours des moments où nous sommes confrontés à des problèmes que nous préférerions ne pas avoir à gérer. Dans le cas qui nous occupe, à savoir laisser passer du trafic FTP au travers d'un pare-feu, nous pouvons toujours rediriger le trafic vers un petit programme écrit tout spécialement pour cela. Le bon point pour nous est que la gestion de FTP nous offre notre première chance d'étudier les redirections.

Le moyen le plus simple pour gérer FTP consiste à faire en sorte que PF redirige le trafic de ce service vers une application externe qui agit comme un *proxy* (mandataire ou intermédiaire) pour ce service. Le proxy interagit ensuite avec votre filtre de paquets au travers d'une interface clairement définie.

Suivant votre configuration et le système d'exploitation que vous utilisez comme plate-forme pour votre pare-feu, vous avez trois ou quatre choix possibles.

Nous les présenterons toutes, dans un ordre plus ou moins chronologique (en suivant leur âge).

Le proxy FTP originel de PF, qui est à présent de l'ordre de l'historique, est décrit dans la section ci-après.

Nous passerons ensuite à deux solutions intermédiaires et plus récentes (développées par Camiel Dobbelaar) dans la section suivante « FTP, PF, et les adresses routables : ftpsesame, pftpx et ftp-proxy ».

Enfin, nous verrons le proxy FTP moderne qui fut introduit dans OpenBSD 3.9, présenté à la section « Le FTP nouveau : ftp-proxy ».

FTP au travers d'une NAT : ftp-proxy

En novembre 2005, l'ancien ftp-proxy (`/usr/libexec/ftp-proxy`) fut remplacé dans OpenBSD-current par le nouveau ftp-proxy, qui réside dans le répertoire `/usr/sbin/`. C'est le logiciel qui est inclus dans les versions d'OpenBSD postérieures à la 3.9.

L'ancien programme ftp-proxy, qui fait partie du système de base des systèmes embarquant des versions de PF correspondant à OpenBSD 3.8 ou une version antérieure, est habituellement appelé via le super-serveur `inetd` au moyen d'une entrée du fichier `/etc/inetd.conf`.

> **VERSION Seulement pour le PF d'OpenBSD 3.8**
>
> Cette section est valable seulement pour les systèmes où PF est équivalent à celui d'OpenBSD 3.8. Elle deviendra obsolète dès que le dernier portage de PF sera passé aux nouvelles versions. Si vous utilisez une version moderne de PF, sautez directement à la section « Le FTP nouveau style : ftp-proxy » pour des informations à jour.

Vous devrez peut-être activer le service `inetd` en ajoutant la ligne suivante :

```
inetd_enable="YES"
```

à votre `rc.conf`, et en ajustant au besoin d'autres paramètres de configuration qui lui sont liés.

La ligne citée ici spécifie que ftp-proxy tourne en mode NAT sur l'interface de bouclage (`loopback`) :

```
127.0.0.1:8021 stream tcp nowait root /usr/libexec/ftp-proxy ftp-proxy -n
```

Cette ligne figure peut-être déjà dans `inetd.conf`, éventuellement commentée par un caractère `#` en début de ligne.

Sous FreeBSD, il existe déjà une ligne appropriée, commentée, avec une syntaxe légèrement différente. Pour activer ftp-proxy, il suffit de la décommenter.

Pour activer votre modification, redémarrez `inetd`.

Sous FreeBSD, NetBSD et les autres BSD basés sur rcNG, cela se fait par la commande suivante ou un équivalent (consulter `man 8 inetd` en cas de doute) :

```
$ sudo /etc/rc.d/inetd restart
```

Passons à présent à la redirection en elle-même. Les règles de redirection (`rdr`) et de NAT (`nat`) font partie de la même classe de règles. D'autres règles peuvent s'y référer directement et les règles de filtrage peuvent dépendre d'elles. Logiquement, les règles `rdr` et `nat` doivent être définies *avant* les règles de filtrage.

Dans notre fichier `/etc/pf.conf`, nous insérons notre règle `rdr` immédiatement après la règle `nat` :

```
rdr pass on $int_if proto tcp from any to any port ftp -> 127.0.0.1 port 8021
```

Le trafic redirigé doit, de plus, être autorisé à passer. Cela se fait par :

```
pass in on $ext_if inet proto tcp from port ftp-data to ($ext_if)
 ➥ user proxy flags S/SA keep state
```

Sauvegardez votre fichier `pf.conf`, puis chargez les nouvelles règles avec la commande :

```
$ sudo pfctl -f /etc/pf.conf
```

Dès ce moment, vos utilisateurs pourront utiliser FTP – avant même que vous leur ayez annoncé l'ouverture du service.

Cet exemple suppose que vous utilisiez de la NAT sur une passerelle, avec des adresses non routables du côté intérieur. Cette configuration couvre les bases et devrait coopérer correctement avec un grand nombre de serveurs et de clients FTP. En pratique, vous aurez peut-être à compenser quelques couacs d'un côté ou de l'autre et vous devrez pour cela regarder du côté des diverses options qu'offre le proxy FTP.

Allez voir la page de manuel de ftp-proxy (`man ftp-proxy`), vous y trouverez des exemples de méthodes permettant de restreindre les rangs de ports des connexions de données, ainsi que des variations sur l'interaction avec d'autres applications et services.

FTP, PF et les adresses routables : ftpsesame, pftpx et ftp-proxy

Pour les cas où le réseau local utilise des adresses officielles et routables à l'intérieur du pare-feu, des utilisateurs ont rapporté avoir eu des difficultés à faire fonctionner le ftp-proxy antérieur à celui d'OpenBSD 3.9. À cette époque, je me suis moi-même penché sur le sujet. J'ai passé vraiment beaucoup de temps dessus mais c'est un ami Néerlandais, Camiel Dobbelaar, qui a fini par proposer une solution concrète à ce problème. Sa solution était un démon, appelé ftpsesame.

Les réseaux locaux utilisant des adresses officielles à l'intérieur d'un pare-feu sont apparemment suffisamment rares pour que j'évite d'en parler davantage ici. Si vous faites tourner un système d'exploitation dont le code de PF date d'avant OpenBSD 3.9, gardez à l'esprit le nom de ftpsesame.

Sous FreeBSD, ftpsesame est disponible via le système de ports : `ftp/ftpsesame`. Vous pouvez tout aussi bien télécharger ftpsesame sur le site web de Sentia à l'adresse http://www.sentia.org/projects/ftpsesame/.

Une fois installé et configuré, ftpsesame s'accroche à votre jeu de règles grâce à une *ancre* (`anchor`), c'est-à-dire un sous-ensemble nommé du jeu de règles. La documentation se compose d'une page de manuel, avec des exemples que vous pouvez probablement vous contenter de copier-coller.

ftpsesame n'a jamais pu entrer dans le système de base et Camiel en est venu à écrire une nouvelle solution au même problème. Le nouveau programme, appelé au départ pftpx, est disponible à l'adresse http://www.sentia.org/downloads/pftpx-0.8.tar.gz et via le système de ports de FreeBSD en tant que `ftp/pftpx`. Le paquetage pftpx est livré avec une page de manuel bien écrite et plutôt complète pour vous aider à démarrer.

Une version un peu plus développée, renommée ftp-proxy pour prendre la succession de l'ancien programme du même nom, fait désormais partie d'OpenBSD depuis la version 3.9 ; il s'agit en effet de `/usr/sbin/ftp-proxy`. Ce nouveau ftp-proxy est décrit dans la section suivante.

Le FTP nouveau : ftp-proxy

Si vous travaillez avec une version de PF basée sur celle d'OpenBSD 3.9 ou une version ultérieure, c'est la version de ftp-proxy que vous utilisez.

> Ceci s'applique aux versions d'OpenBSD postérieures à la 3.9 et aux systèmes équivalents.

Le nouveau ftp-proxy interagit avec votre jeu de règles via un système d'*ancres*, où le proxy insère et efface les règles qu'il construit pour gérer votre trafic FTP. Comme pour son prédécesseur, la configuration de ftp-proxy se fait principalement par des copier-coller de la page de manuel.

Si vous mettez à jour votre ftp-proxy à partir d'une version précédente, retirez la ligne `ftp-proxy` de votre fichier `inetd.conf` avant de redémarrer `inetd` ou de le désactiver (au cas où il ne serait plus utile).

Ensuite, activez ftp-proxy en ajoutant la ligne suivante à votre fichier `/etc/rc.conf.local` ou `/etc/rc.conf` (selon le système que vous utilisez) :

```
ftpproxy_flags=""
```

Vous pouvez, si vous le voulez, démarrer manuellement le proxy en lançant `/usr/sbin/ftp-proxy`.

Passons au fichier `pf.conf`. Vous avez besoin de définir deux ancres dans la section NAT :

```
nat-anchor "ftp-proxy/*"
rdr-anchor "ftp-proxy/*"
```

Les deux sont nécessaires, même si votre configuration n'utilise pas de NAT. Si vous migrez depuis une version précédente de ftp-proxy, votre jeu de règles contient probablement déjà la section appropriée. Sinon, ajoutez-la :

```
rdr pass on $int_if proto tcp from any to any port ftp -> 127.0.0.1
 ➥ port 8021
```

Descendez au niveau des règles de filtrage et ajoutez une ancre pour le proxy :

```
anchor "ftp-proxy/*"
```

Enfin, ajoutez une règle `pass` afin de laisser passer les paquets du proxy vers le reste du monde :

```
pass out proto tcp from $proxy to any port 21 keep state
```

où `$proxy` se développe en l'adresse à laquelle le proxy est liée.

Cet exemple couvre une situation simple, où des clients doivent contacter des serveurs FTP situés ailleurs. Cette configuration devrait fonctionner correctement avec la plupart des combinaisons de clients et de serveurs FTP. En pratique, vous aurez peut-être à compenser quelques couacs d'un côté ou de l'autre et vous devrez, pour cela, regarder du côté des diverses options qu'offre le proxy FTP.

Si vous cherchez comment faire tourner un serveur FTP protégé par PF et ftp-proxy, vous pourriez lancer un autre ftp-proxy en mode inverse (au moyen de l'option `-R`, pour *reverse*), sur un autre port, avec d'autres redirections et d'autres règles `pass` qui lui soient propres.

Faciliter le dépannage de votre réseau

Faciliter le dépannage de votre réseau est un sujet potentiellement vaste. Dans la plupart des cas, la possibilité de déboguer ou de dépanner facilement un réseau TCP/IP dépend de la manière dont vous traitez le protocole Internet conçu spécialement à cet effet, c'est-à-dire ICMP (*Internet Control Message Protocol*).

ICMP est le protocole servant à l'envoi et à la réception de *messages de contrôle* entre les hôtes et les passerelles, principalement dans le but de fournir un retour à l'expéditeur sur les éventuelles mauvaises conditions de route entre lui et l'hôte ciblé.

Il y a beaucoup de trafic ICMP et celui-ci passe normalement en tâche de fond pendant que vous surfez sur le Web, lisez vos messages électroniques ou transférez des fichiers. Les routeurs (vous êtes au courant que vous êtes en train d'en construire un, n'est-ce-pas ?) utilisent ICMP pour négocier la taille des paquets et d'autres paramètres de transmission, selon un processus que l'on appelle souvent *découverte de la MTU du chemin* (*path MTU discovery*).

Certains administrateurs système vous ont peut-être déjà décrit ICMP comme étant le mal ou, si leur compréhension est un peu plus profonde, comme étant un mal nécessaire. La raison en est purement historique. Il y a quelques années, on a découvert que les piles réseau de plusieurs systèmes d'exploitation contenaient du code qui pouvait provoquer le crash d'une machine si on lui envoyait une requête ICMP suffisamment grande.

L'une des entreprises les plus touchées fut Microsoft et vous trouverez beaucoup de lecture sur le bug *ping of death* en utilisant votre moteur de recherche préféré. Mais tout ceci remonte à la seconde moitié des années 1990 et tous les systèmes d'exploitation modernes ont largement assaini leur code réseau depuis. C'est, tout du moins, ce qu'on nous pousse à croire. L'une des toutes premières parades consistait à bloquer purement et simplement les requêtes ICMP ECHO (*ping*), voire tout le trafic ICMP. De nos jours, ces jeux de règles existent depuis environ dix ans et certaines personnes les laissent toujours en vigueur parce qu'elles ont toujours peur. Il n'y a cependant plus ou presque de raison de s'inquiéter du trafic ICMP destructeur, mais nous couvrirons dans les sections suivantes les manières de gérer le trafic ICMP qui transite en provenance de votre réseau ou vers lui.

Alors, nous laissons tout passer ?

La question évidente devient : si ICMP est aussi bon et utile, ne devrions-nous pas tout laisser passer, toujours ? La réponse est : cela dépend.

Laisser le trafic passer sans condition facilite bien entendu le débogage, mais cela facilite également pour quiconque la tâche d'extraction d'informations relatives à votre réseau. Cela signifie qu'une règle comme :

```
pass inet proto icmp from any to any
```

pourrait ne pas être optimale si vous souhaitez masquer le fonctionnement interne de votre réseau. Pour être honnête, il faut également signaler que certains trafics ICMP vont lourdement peser sur vos règles `keep state`.

La solution de facilité : au final, c'est moi qui décide

La solution la plus simple pourrait très bien être de laisser passer tout le trafic ICMP émanant de votre réseau local et de stopper, au niveau de votre passerelle, les sondes venant d'ailleurs :

```
pass inet proto icmp icmp-type $icmp_types from $localnet to any keep state
pass inet proto icmp icmp-type $icmp_types from any to $ext_if keep state
```

Stopper les sondes au niveau de la passerelle peut, dans tous les cas, se révéler un choix attrayant mais voyons quelques autres possibilités qui vous montreront un peu la flexibilité de PF.

Laisser passer les ping

Le jeu de règles que nous avons développé jusqu'ici a un gros inconvénient : les commandes de dépannage classiques telles que `ping` et `traceroute` ne fonctionneront pas. Peu de vos utilisateurs s'en soucieront et, dans la mesure où c'est la commande `ping` qui a au départ effrayé les gens au point qu'ils en sont venus à filtrer ou bloquer le trafic ICMP, il y a apparemment des gens qui se sentent mieux sans. Si vous faites partie de l'audience que je cible, alors vous serez heureux d'avoir la possibilité d'utiliser ces outils de dépannage – et ils rempliront en effet ce rôle si nous ajoutons quelques petits détails à notre jeu de règles. La commande `ping` utilise ICMP ; afin que notre jeu de règles reste clair, nous commençons par définir une autre macro :

```
icmp_types = "echoreq"
```

et nous ajoutons une règle qui utilise cette définition :

```
pass inet proto icmp all icmp-type $icmp_types keep state
```

Si d'autres types de paquets ICMP doivent pouvoir passer, vous pouvez développer `$icmp_types` en une liste contenant les types de paquets que vous souhaitez autoriser.

Aider traceroute

La commande `traceroute` est une autre commande bien utile quand vos utilisateurs prétendent qu'Internet ne fonctionne pas. Par défaut, le `traceroute` Unix utilise des connexions UDP selon un ensemble de formules basées sur la destination. La règle suivante fonctionne avec les commandes `traceroute` de toutes les variantes d'Unix auxquelles j'ai eu accès, y compris GNU/Linux :

```
# autorise la sortie du rang par défaut de traceroute(8) :
# "base+nhops*nqueries-1" (33434+64*3-1)
pass out on $ext_if inet proto udp from any to any port 33433 >< 33626 keep state
```

Voilà qui donne l'occasion de voir à quoi ressemblent les rangs de ports, utiles dans certains cas.

L'expérience montre que les implémentations de `traceroute` des autres systèmes d'exploitation fonctionnent à peu près de la même manière. À l'exception notable de Microsoft Windows. Sur cette plate-forme, le programme `TRACERT.EXE` utilise des ICMP ECHO. Donc, si vous voulez que les traceroute de Windows fonctionnent, vous n'avez besoin que de la première règle de la section précédente, que vous utilisiez pour laisser passer les `ping`. On peut aussi très bien demander au programme `traceroute` Unix d'utiliser d'autres protocoles et il se comportement exactement comme son homologue de chez Microsoft si vous utilisez son option `-I` en ligne de commande. Vous pouvez lire la page de manuel de `traceroute` (ou son code source) pour avoir tous les détails.

Cette solution est extraite d'un message d'*openbsd-misc*. Je trouve que cette liste de diffusion (et les archives web de ce genre de listes, munies d'une fonction de recherche, comme par exemple http://marc.info) est une bonne source d'informations concernant OpenBSD et PF.

Découverte de la MTU du chemin (path MTU discovery)

La dernière chose dont il me reste à vous parler à propos du dépannage est la *découverte de la MTU du chemin*. Les protocoles Internet sont conçus pour être indépendants du matériel et l'une des conséquences de ce choix est qu'il n'est pas toujours possible de prédire de façon fiable la taille de paquet optimale pour une connexion donnée. La principale contrainte qui pèse sur la taille des paquets est appelée *Unité Maximale d'Émission* (*Maximum Transmission Unit*, soit *MTU*). Elle fixe la limite haute de la taille des paquets pour une interface. La commande `ifconfig` vous montrera la MTU de vos interfaces réseau.

Les implémentations modernes de TCP/IP s'attendent à être capables de déterminer la taille de paquet adaptée à une connexion grâce à un processus qui consiste simplement à envoyer des paquets de tailles variables en activant le drapeau « ne pas fragmenter », attendant en retour un paquet ICMP « type 3, code 4 » quand la limite haute est atteinte. Pour le moment, vous n'avez pas à vous plonger dans la lecture des RFC. Le type 3 signifie *impossible d'atteindre la destination* et le code 4 est le raccourci pour *il y a besoin d'une fragmentation, mais le drapeau « ne pas fragmenter » est activé*. Cela signifie que, si les connexions à des réseaux ayant une MTU différent de celle du vôtre vous semblent en deçà de l'optimum, vous pourriez essayer de modifier légèrement votre liste de types ICMP afin de laisser passer les paquets *destination unreacheable* :

```
icmp_types = "{ echoreq, unreach }"
```

Comme vous le voyez, cela signifie que nous n'avons pas besoin de modifier la règle `pass` :

```
pass inet proto icmp all icmp-type $icmp_types keep state
```

Je vais maintenant vous dévoiler un petit secret : dans la plupart des cas, ces règles ne sont pas nécessaires à la découverte de la MTU du chemin, mais elles ne font pas de mal non plus. Mais même si le comportement `keep state` par défaut s'occupe de la plupart du trafic ICMP dont vous avez besoin, PF vous laisse filtrer sur la base de toutes les variations de types et de codes ICMP. Si vous voulez plus de détails, les types et codes possibles sont documentés dans les pages de manuel `icmp(4)` et `icmp(6)`. Pour les généralités, lisez les RFC.

Les principales RFC Internet décrivant ICMP et certaines techniques qui lui sont associées sont les RFC 792, 950, 1191, 1256, 2521 et 2765, tandis que les mises à jour de ICMP pour IPv6 se trouvent dans les RFC 1885, 2463 et 2466. Ces documents sont disponibles à divers endroits sur Internet, comme par exemple http://www.ietf.org et http://www.faqs.org, et probablement aussi via votre système de gestion des paquetages.

Les tables vous simplifient la vie

Vous trouvez peut-être à présent que ce système de création de règles est horriblement statique et rigide. Il existe, après tout, des types de données susceptibles d'être filtrés et redirigés à un moment donné, mais qui ne méritent pas d'être placés dans un fichier de configuration ! C'est exact et PF dispose également de mécanismes pour gérer ces situations.

Les tables en sont un exemple. Elle sont utilisées pour les listes d'adresses IP qui peuvent être manipulées sans avoir à recharger le jeu de règles entier, mais également quand on veut des recherches rapides.

Les noms de tables apparaissent toujours entre un signe inférieur et un signe supérieur (< et >), comme ceci :

```
table <clients> persist { 192.168.2.0/24, !192.168.2.5 }
```

Ici, le réseau `192.168.1.1/24` fait partie de la table, avec une exception. L'adresse `192.168.2.5` est exclue en utilisant l'opérateur `!` (NON logique). Le mot-clé `persist` assure que la table en elle-même existera même si aucune règle ne s'y réfère pour le moment. Il est intéressant de remarquer que l'on peut également charger des tables depuis des fichiers. Chaque élément doit figurer sur une ligne à part, comme dans le fichier `/etc/clients` :

```
192.168.2.0/24
!192.168.2.5
```

qui est à son tour utilisé pour initialiser la table dans `/etc/pf.conf` :

```
table <clients> persist file "/etc/clients"
```

Vous pouvez donc, par exemple, modifier l'une de nos règles précédentes de la manière suivante :

```
pass inet proto tcp from <clients> to any port $client_out
```

afin de gérer le trafic sortant de vos ordinateurs clients. Avec tout cela, vous pouvez manipuler en direct le contenu de la table, comme ceci :

```
$ sudo pfctl -t clients -T add 192.168.1/16
```

Remarquez que cela ne modifie que la copie en mémoire de la table, ce qui veut dire que que la modification ne survivra ni à une panne de courant ni à un redémarrage, à moins que vous ne vous arrangiez pour les stocker.

Vous pouvez choisir de maintenir la copie sur disque de la table en utilisant une tâche `cron` qui écrit le contenu de la table dans un fichier sur le disque, et ceci à intervalle régulier, en utilisant une commande telle que :

```
$ sudo pfctl -t clients -T show >/etc/clients
```

Ou bien, vous pouvez éditer le fichier `/etc/clients` et remplacer le contenu de la table en mémoire par les données du fichier :

```
$ sudo pfctl -t clients -T replace -f /etc/clients
```

Pour les opérations que vous effectuerez fréquemment (insérer ou effacer des éléments, remplacer le contenu de la table, etc.), vous finirez tôt ou tard par écrire un script shell.

Un exemple courant et très facile à implémenter consiste à renforcer les restrictions d'accès au réseau via des tâches `cron` qui remplacent le contenu des tables d'adresses sources des règles `pass` et ce, à des horaires spécifiques. Sur certains réseaux, vous pourriez même avoir besoin de règles d'accès différentes selon les jours de la semaine. La seule vraie limite tient à vos besoins propres et à votre créativité.

Nous reviendrons vite à des exemples pratiques d'utilisation des tables, notamment quelques programmes qui interagissent avec les tables de manière intéressante.

> L'un des programmes qui interagit avec les tables PF est le démon DHCP dhcpd. Sous OpenBSD, consultez la page de manuel de dhcpd et tout particulièrement les drapeaux `-A` et `-L`.

Réseau sans fil, facile 4

Il est plutôt tentant d'affirmer que, sous BSD, et particulièrement sous OpenBSD, il n'est pas besoin de simplifier la mise en place d'un réseau sans fil, parce ce que c'est déjà une opération simple. Mettre en place un réseau sans fil n'est pas très différent de la construction d'un réseau filaire, mais il y a quelques problèmes qui se présentent puisque nous travaillons avec des ondes radio et non des câbles. Nous prendrons un peu de temps pour passer brièvement en revue certains de ces problèmes, avant de passer aux étapes pratiques nécessaires à la création d'une configuration utilisable.

Une fois que nous aurons couvert les bases de la mise en place d'un réseau sans fil, nous discuterons des options permettant de rendre ce réseau plus intéressant et plus difficile à casser.

Quelques généralités sur la norme IEE 802.11

En principe, la mise en place d'une interface réseau, quel que soit son type, se fait en deux étapes : d'une part, établir un lien et, d'autre part, configurer l'interface pour le trafic TCP/IP.

Dans le cas des interfaces câblées de type Ethernet, l'établissement du lien consiste habituellement à brancher un câble et à voir s'allumer la diode indicatrice de lien. Mais certaines interfaces exigent d'autres étapes. Le réseau via des connexions téléphoniques par exemple, requiert des étapes en rapport avec la téléphonie, par exemple la composition d'un numéro de téléphone pour obtenir un signal porteur.

Dans le cas des réseaux sans fil de type IEEE 802.11, l'obtention du signal porteur implique quelques étapes au plus bas niveau. Vous devez d'abord sélectionner le bon canal dans le spectre des fréquences assignées. Une fois que vous avez le signal, vous

devez régler les quelques paramètres d'identification réseau du niveau lien. Enfin, si la station à laquelle vous voulez vous lier utilise une quelconque forme de chiffrement de niveau lien, vous devez procéder aux réglages adéquats et, probablement, négocier des paramètres supplémentaires.

Heureusement, dans les systèmes BSD, toute la configuration des périphériques réseau sans fil se fait via diverses options de la même commande, `ifconfig`, exactement comme pour n'importe quelle autre interface réseau.

> **VERSION ifconfig et wicontrol/ancontrol**
>
> Sur certains systèmes, des programmes anciens et spécifiques à un périphérique tels que `wicontrol` et `ancontrol` sont toujours présents, mais ils sont pour la plupart dépréciés et en cours de remplacement par des fonctionnalités de `ifconfig`. Sous OpenBSD, le regroupement au sein de `ifconfig` est terminé.

Cependant, et puisque nous présentons ici les réseaux sans fil, nous devons passer en revue la sécurité des divers niveaux de la pile réseau depuis cette nouvelle perspective.

Il y a globalement trois types de mécanismes de sécurité IEE 802.11. Ils sont simples et populaires et nous en parlerons rapidement dans les sections suivantes.

> Pour un tour d'horizon plus complet des problématiques de sécurité dans les réseaux sans fil, voir, par exemple, les articles et présentations du Professeur Kjell Jørgen Hole sur son site web (http://www.kjhole.com et http://www.kjhole.com/Standards/WiFi/WiFiDownloads.html). Pour les développements récents dans le domaine du Wi-Fi, les Wi-Fi Net News à l'adresse http://wifinetnews.com/archives/cat_security.html et « The Unofficial 802.11 Security Web Page » à l'adresse http://www.drizzle.com/~aboba/IEEE sont fortement recommandés.

Filtrage d'adresses MAC

Pour faire bref : nous ne filtrerons pas les adresses MAC avec PF.

Un certain nombre de points d'accès grand public proposent le filtrage des adresses MAC mais, contrairement aux idées reçues, cela ne renforce pas vraiment la sécurité. Le marketing a bien fonctionné parce que la plupart des clients ne sont pas au courant du fait qu'il est possible de modifier l'adresse MAC d'à peu près tous les adaptateurs réseau du marché actuel.

Si vous souhaitez vraiment essayer le filtrage d'adresses MAC, allez voir du côté de la fonctionnalité `bridge(4)` et des fonctionnalités de filtrage MAC offertes par `brconfig(8)` sous OpenBSD. Nous parlerons au prochain chapitre des ponts et des meilleurs moyens de les utiliser dans le cadre du filtrage de paquets.

> HOWTO **Changer l'adresse MAC**
>
> Une rapide recherche dans les pages de manuel vous indiquera que la commande pour changer l'adresse MAC de l'interface `rum0` est simplement `ifconfig rum0 lladdr 00:ba:ad:f0:0d:11`, cette dernière série de chiffres et de lettres étant la nouvelle adresse MAC.

WEP

L'une des conséquences de l'utilisation d'ondes radio au lieu de câbles pour déplacer des données est qu'il est comparativement plus facile pour un intrus de capturer vos données au passage. Les concepteurs de la famille de standards de réseaux sans fil 802.11 semblaient au courant de cet état de faits et ils proposèrent une solution, qui fut commercialisée sous le nom de *Wired Equivalent Privacy* (*WEP*).

Malheureusement, durant le processus de conception, ils ne se documentèrent pas sur les recherches récentes en matière de chiffrement, pas plus qu'ils ne consultèrent de chercheur spécialiste de ce domaine. C'est donc sans grande surprise que l'on constate que les professionnels du chiffrement considèrent comme un vulgaire bricolage la méthode de chiffrement de niveau lien proposée. Et c'est sans surprise non plus que l'on apprît que le chiffrement WEP avait été cassé (par ingénierie inverse), quelques mois à peine après la commercialisation des premiers produits.

Pour diverses raisons, ce système reste largement supporté et utilisé, même si n'importe qui peut télécharger des outils libres et gratuits pour déchiffrer du trafic WEP en quelques minutes. Tous les équipements IEEE 802.11 gèrent au moins WEP, et un nombre surprenant de produits offre également le filtrage d'adresses MAC.

Vous devriez considérer que le système WEP n'apporte qu'une sécurité marginale à un réseau sans fil où rien n'est chiffré. Vous ne pouvez en fait compter que sur l'effet dissuasif, face à un attaquant paresseux et peu compétent.

WPA

Il arriva vite aux oreilles des concepteurs de WEP que leur système avait été facilement cassé et ils planchèrent donc sur version révisée et légèrement plus cohérente, qu'ils appelèrent *Wi-Fi Protected Access* ou *WPA*.

WPA a l'air meilleur que WEP, tout du moins sur le papier, mais ses spécifications sont beaucoup trop compliquées pour que ses implémentations se répandent bien. WPA a de plus été critiqué pour des problèmes de conception ainsi que pour des bugs. Si l'on ajoute à cela les problèmes habituels d'accès à la documentation et au matériel, la prise en charge de WPA dans les logiciels libres est très variable. Si la spécification de votre projet inclut WPA, lisez avec attention la documentation de votre système d'exploitation et celle de votre pilote de périphérique.

Il va sans dire que vous aurez besoin de mesures de sécurité supplémentaires, telles que le chiffrement SSH ou SSL, afin de maintenir un certain niveau de confidentialité pour vos flux de données.

Choisir le bon matériel selon la tâche demandée

Le choix du bon matériel n'est pas forcément un problème. Sous BSD, vous n'avez besoin que d'une simple commande :

```
$ apropos wireless
```

pour obtenir la liste des pages de manuel comportant le mot *wireless* (*sans fil*) dans leur titre.

Documentation

Il est également possible de consulter les pages de manuel sur le Web. Rendez-vous sur le site d'OpenBSD ou sur les sites web des autres projets de la famille BSD ; ils offrent des moteurs de recherche pour pages de manuels, basés sur des mots-clés.

- http://www.openbsd.org

Même sur un système fraîchement installé, cela vous donnera la liste complète de tous les pilotes de périphériques réseau sans fil disponibles dans le système d'exploitation. L'étape suivante consiste à lire les pages de manuel de ces différents pilotes et de comparer les listes de périphériques compatibles au matériel disponible dans les systèmes que vous pensez utiliser. Prenez bien le temps de réfléchir à vos besoins. Pour les tests, des *dongles* USB bas de gamme type `rum` ou `ural` fonctionneront. Quand vous serez prêt à construire une infrastructure permanente, vous envisagerez du matériel plus haut de gamme. La lecture de l'annexe B vous sera peut-être utile.

Mise en place d'un réseau sans fil simple

Pour commencer à construire notre premier réseau sans fil, mieux vaut prendre comme point de départ la configuration de passerelle du chapitre précédent. Dans la conception de notre réseau, il est probable que le réseau sans fil ne sera pas directement rattaché à Internet et qu'il passera par une passerelle quelconque. Il est donc intéressant de réutiliser la passerelle dont nous disposons, en l'adaptant un peu afin qu'elle puisse jouer le rôle de point d'accès sans fil, sur la base de modifications

mineures que nous présenterons dans les paragraphes à venir. Après tout, c'est plus pratique que de démarrer une nouvelle configuration à partir de zéro.

> Nous sommes actuellement en train de construire une infrastructure et nous allons commencer par mettre en place le point d'accès. Si vous préférez voir d'abord la partie client, rendez-vous plus bas à la section « La partie client » pour revenir ensuite à la présente section.

Comme nous l'avons déjà mentionné, la première étape consiste à s'assurer que la carte dont vous disposez est prise en charge et à vérifier que le pilote charge et initialise la carte correctement. Au démarrage, les messages du système défilent à la console, mais ils sont également stockés dans le fichier `/var/run/dmesg.boot`. Vous pouvez visualiser ce fichier ou utiliser la sortie de la commande `dmesg`. Une carte PCI correctement activée vous donnera une sortie comparable à celle-ci :

```
ath0 at pci1 dev 4 function 0 "Atheros AR5212" rev 0x01: irq 11
ath0: AR5212 5.6 phy 4.1 rf5111 1.7 rf2111 2.3, ETSI1W, address
00:0d:88:c8:a7:c4
```

Si l'interface que vous voulez configurer peut être branchée à chaud, comme un périphérique USB ou PCCARD, vous pouvez visualiser les messages du noyau dans le fichier `/var/log/messages`. Lancez par exemple `tail -f` sur le fichier avant de brancher le périphérique.

Ensuite, configurez l'interface afin d'activer le lien, ce qui permettra enfin de configurer le système pour TCP/IP. Vous pouvez le faire depuis la ligne de commande, comme ceci :

```
$ sudo ifconfig ath0 up mediaopt hostap mode 11b chan 11 nwid unwiredbsd
nwkey 0x1deadbeef9
```

Cette commande réalise plusieurs opérations en même temps. Elle configure l'interface `ath0`, l'active (par le paramètre `up`) et spécifie qu'il s'agit du point d'accès d'un réseau sans fil (`mediaopt hostap`) ; elle active ensuite explicitement le mode `11b`, le canal `11`, et elle utilise enfin le paramètre `nwid` pour fixer le nom du réseau à `unwiredbsd` ; la clé WEP (`nwkey`) est définie par la chaîne hexadécimale `0x1deadbeef9`.

Utilisons `ifconfig` pour vérifier que la commande à bien configuré l'interface selon nos souhaits :

```
$ ifconfig ath0
ath0: flags=8823<UP,BROADCAST,NOTRAILERS,SIMPLEX,MULTICAST> mtu 1500
lladdr 00:11:95:ca:e6:59
groups: wlan
media: IEEE802.11 autoselect mode 11b hostap
status: no network
ieee80211: nwid unwiredbsd chan 11 bssid 00:11:95:ca:e6:59 nwkey <not displayed>
inet6 fe80::211:95ff:feca:e659%ath0 prefixlen 64 tentative scopeid 0x5
```

Remarquez le contenu des lignes `media:` et `ieee802.11:`. Elles devraient correspondre à ce que vous avez entré dans la ligne de commande d'`ifconfig`. Une fois opérationnelle la partie lien de votre réseau sans fil, vous pouvez passer à l'étape suivante et assigner une adresse IP à l'interface :

```
$ sudo ifconfig ath0 10.50.90.1
```

Sous OpenBSD, ces deux étapes peuvent se faire d'un coup en créant un fichier `/etc/hostname.ath0`, au contenu similaire à ceci :

```
up mediaopt hostap mode 11b chan 11 nwid unwiredbsd nwkey 0x1deadbeef9
inet 10.50.90.1
```

et en lançant `/etc/netstart ath0` (vous devez être root pour cela) ou en attendant patiemment le prochain redémarrage.

Remarquez que la configuration se fait en deux lignes. La première génère une commande `ifconfig` qui met en place l'interface avec les paramètres correspondant au réseau sans fil physique et la seconde, qui établit l'adresse IP, n'est exécutée qu'après la fin de la première. Comme il s'agit là de notre point d'accès, nous fixons explicitement le canal et nous activons un chiffrement WEP (que nous avons décrit plus haut comme faible) en réglant le paramètre `nwkey`.

Sous FreeBSD et NetBSD, vous pouvez normalement combiner tous les paramètres en une seule ligne `rc.conf` :

```
ifconfig_ath0="mediaopt hostap mode 11b chan 11 nwid unwiredbsd nwkey
0x1deadbeef inet 10.50.90.1"
```

Cependant, avec certaines combinaisons de matériel, on ne peut pas fixer en même temps les options de niveau lien et l'adresse IP. Si votre configuration ne fonctionne pas en une seule ligne, vous devrez placer les deux lignes dans votre fichier `/etc/`

`start_if.ath0` (en remplaçant bien entendu `ath0` par le nom de votre interface, si besoin est).

> Assurez-vous d'avoir bien lu la dernière mise à jour de la page de manuel d'`ifconfig` pour être au courant des autres options qui seraient plus appropriées à votre configuration.

Le jeu de règles PF du point d'accès

Une fois les interfaces configurées, il est temps de commencer à configurer le point d'accès en passerelle filtrante.

Vous pouvez commencer par copier la configuration passerelle de base du chapitre 3. Activez la passerelle via les entrées adéquates dans le fichier `sysctl.conf` ou `rc.conf` ; copiez ensuite nos règles dans le fichier `pf.conf`. Selon ce que vous avez jugé utile dans le dernier chapitre, le fichier `pf.conf` devrait peu ou prou ressembler à ceci :

```
ext_if = "re0" # macro pour l'interface externe
               # - utilisez tun0 ou pppoe0 for PPPoE
int_if = "re1" # macro pour l'interface interne
localnet = $int_if:network
client_out = "{ ssh, domain, pop3, auth, nntp, http,
                ➥  https, cvspserver, 2628, 5999, 8000, 8080 }"
udp_services = "{ domain, ntp }"
icmp_types = "{ echoreq, unreach }"
# l'adresse de ext_if peut être dynamique, d'où le ($ext_if)
nat on $ext_if from $localnet to any -> ($ext_if)
block all
pass quick inet proto { tcp, udp } from $localnet to any port $udp_services
pass log inet proto icmp all icmp-type $icmp_types
pass inet proto tcp from $localnet to any port $client_out
```

La seule modification qui soit strictement nécessaire pour que votre point d'accès fonctionne consiste à faire correspondre la définition de la macro `int_if` à l'interface sans fil. Dans notre exemple, cela signifie que la ligne devrait maintenant être

```
int_if = "ath0" # macro pour l'interface interne
```

Vous voudrez très certainement aussi mettre en place `dhcpd` pour servir des adresses et d'autres informations réseau importantes aux clients, une fois qu'ils se sont associés à votre point d'accès. La mise en place de `dhcpd` est relativement simple si vous lisez les pages de manuel.

C'est tout ce qu'il y a à dire. Cette configuration vous donne un point d'accès BSD fonctionnel, avec un minimum de sécurité via le chiffrement WEP (même s'il s'agit plus d'une affiche « INTERDICTION D'ENTRER » qu'autre chose). Si vous devez gérer FTP, vous pouvez copier la configuration ftp-proxy du chapitre 3 et apporter des modifications similaires sur le reste du jeu de règles.

Si votre point d'accès possède trois interfaces ou plus

Si votre conception de réseau ordonne que votre point d'accès soit également la passerelle d'un réseau filaire ou même de plusieurs réseaux sans fil, vous devez procéder à de légères modifications dans votre jeu de règles. Au lieu de vous contenter de modifier la macro `int_if`, ajoutez plutôt une nouvelle définition pour l'interface sans fil, comme par exemple :

```
air_if = "ath0"
```

Dans une configuration de passerelle sans fil, vos interfaces sans fil seront probablement sur des sous-réseaux dédiés et il peut donc se révéler utile que chacun d'entre eux dispose de sa propre règle `nat` :

```
nat on $ext_if from $air_if:network to any -> ($ext_if) static-port
```

Selon votre politique, vous voudrez peut-être aussi ajuster la définition de `$localnet` ou, tout du moins, inclure `$air_if` dans vos règles `pass` aux endroits adéquats. Et, une fois encore, si vous devez gérer FTP, une redirection séparée pour le réseau sans fil doit être mise en place vers ftp-proxy.

Gérer IPsec et les solutions VPN

Les détails sur la mise en place d'un Réseau Privé Virtuel (*Virtual Private Network* ou VPN) dépassent du cadre de ce chapitre, que ce soit par l'utilisation des outils IPsec intégrés, d'OpenSSH ou de tout autre outil. Cependant, au vu du faible niveau de sécurité des réseaux sans fil en général, vous voudrez probablement mettre en place des sécurités supplémentaires. Une configuration VPN peut se révéler utile dans votre cas, voire essentielle.

Vous disposez globalement de trois catégories de choix.

SSH

Si votre VPN est basé sur des tunnels SSH, le jeu de règles de base contient tout le filtrage dont vous avez besoin. Aux yeux de votre filtre de paquets, le trafic qui passe dans votre tunnel sera impossible à distinguer du reste du trafic SSH.

IPsec avec échange de clés par UDP (IKE/ISAKMP)

Plusieurs variantes d'IPsec dépendent énormément (à un niveau critique) de l'échange de clés via `proto udp port 500`, et `proto {tcp, udp} port 4500` pour la traversée de NAT (*NAT Traversal*, NAT-T). Vous devez laisser passer ce trafic afin de permettre l'établissement des flux. Certaines implémentations ont également besoin que le trafic du protocole ESP (protocole 50) puisse passer entre les hôtes : `pass proto esp from $source to $target`.

Filtrage sur l'interface d'encapsulation d'IPsec

Avec une bonne configuration IPsec, vous pouvez faire en sorte que PF filtre directement sur l'interface `enc0` : `pass on enc0 proto ipencap from $source to $target keep state (if-bound)`.

Vous trouverez en annexe A des références vers de la littérature intéressante sur le sujet.

La partie client

Tant que tous vos clients sont des systèmes BSD, alors la configuration est extrêmement simple. Pour se connecter au point d'accès que nous venons de mettre en place, vos clients OpenBSD ont simplement besoin d'un fichier de configuration `hostname.<if>` comme celui-ci :

```
up media autoselect mode 11b chan 11 nwid unwiredbsd nwkey 0x1deadbeef9
dhcp
```

Essayez d'abord depuis la ligne de commande, avec :

```
$ sudo ifconfig ath0 up mode 11b chan 11 nwid unwiredbsd nwkey
0x1deadbeef9
```

suivie de :

```
$ sudo dhclient ath0
```

La commande `ifconfig` devrait se terminer sans rien renvoyer, tandis que la commande `dhclient` devrait afficher un résumé de son dialogue avec le serveur DHCP :

```
DHCPREQUEST on ath0 to 255.255.255.255 port 67
DHCPREQUEST on ath0 to 255.255.255.255 port 67
DHCPACK from 10.50.90.1
bound to 10.50.90.11 -- renewal in 1800 seconds.
```

Encore une fois, sous FreeBSD, ces lignes sont à placer dans votre fichier `/etc/start_if.ath0` (il faut remplacer `ath0` par le nom de l'interface, si besoin).

authpf : le gardien de votre réseau sans fil

Comme toujours, il existe d'autres moyens pour configurer la sécurité de votre réseau sans fil. Quelle que soit la faiblesse de la protection qu'offre le chiffrement WEP, les professionnels de la sécurité tendent à s'accorder sur le fait qu'elle est tout de même suffisante pour qu'un éventuel assaillant se dise que vous n'entendez pas le laisser utiliser vos ressource réseau sans rien faire.

La configuration que nous avons construite dans la section précédente « Mise en place d'un réseau sans fil simple » est fonctionnelle. Elle permettra à tous les clients correctement configurés de se connecter, mais cela peut se révéler être un problème en soi, dans la mesure où cette configuration n'embarque pas de véritable solution vous permettant de décider qui peut ou non utiliser votre réseau.

Comme nous l'avons déjà signalé, le filtrage d'adresses MAC n'est pas vraiment une défense solide contre les attaquants. Il est trop facile de changer d'adresse MAC. Les développeurs d'OpenBSD ont choisi une approche radicalement différente à ce problème en introduisant authpf dans OpenBSD version 3.1. Au lieu de conditionner l'accès à un identifiant matériel tel que l'adresse MAC de la carte réseau, les développeurs d'OpenBSD ont décidé que les mécanismes d'authentification de l'*utilisateur* déjà en place sont robustes et hautement flexibles, et sont donc plus appropriés pour cette tâche. L'outil `authpf` est un shell utilisateur que permet au système de charger des règles PF selon l'utilisateur, ce qui permet effectivement de décider qui fait quoi.

Pour utiliser `authpf`, vous créez des utilisateurs en leur assignant le programme authpf comme shell. Pour obtenir l'accès au réseau, les utilisateurs se connectent à la passerelle par SSH. Une fois l'utilisateur authentifié, authpf charge les règles que vous avez définies pour lui ou la classe d'utilisateurs à laquelle il appartient.

Ces règles, qui s'appliquent à l'adresse IP d'où l'utilisateur s'est connecté, restent chargées et activées tant que l'utilisateur reste connecté par SSH. Dès qu'il se décon-

necte, les règles sont déchargées et, dans la plupart des cas, tout trafic non SSH provenant de l'adresse IP de l'utilisateur est refusé. Avec une bonne configuration, seul le trafic initié par les utilisateurs authentifiés sera autorisé à passer.

Remarquons que, sous OpenBSD, `authpf` est l'une des classes de login offertes par défaut, comme vous le constaterez la prochaine fois que vous créerez un utilisateur au moyen du programme `adduser`.

Pour les systèmes où la classe de login authpf n'est pas disponible par défaut, vous pouvez ajouter les lignes suivantes à votre fichier `login.conf` :

```
authpf:\
     :welcome=/etc/motd.authpf:\
     :shell=/usr/sbin/authpf:\
     :tc=default:
```

Les sections suivantes contiennent quelques exemples qui peuvent correspondre ou pas à votre situation, mais j'espère qu'ils vous donneront des idées que vous pourrez utiliser.

Une passerelle authentifiante élémentaire

La mise en place d'une passerelle authentifiante avec authpf implique la création et la maintenance de quelques fichiers en plus de votre `pf.conf`, notamment `authpf.rules` ; les autres fichiers sont plutôt statiques et vous ne passerez plus beaucoup de temps dessus une fois que vous les aurez créés.

Commencez par créer un `/etc/authpf/authpf.conf` vide. Il doit exister pour qu'authpf fonctionne, mais il n'a en réalité pas besoin de contenir quoi que ce soit : créer un fichier vide via `touch` est donc suffisant.

Voici les modifications à apporter à `/etc/pf.conf`. Nous devons avant tout créer les macros des interfaces :

```
ext_if = "re0"
int_if = "ath0"
```

authpf demande en outre une table, qu'il peuplera avec les adresses IP des utilisateurs authentifiés :

```
table <authpf_users> persist
```

Les règles `nat`, si vous en avez besoin, pourraient très bien aller dans `authpf.rules`, mais il n'est pas gênant de les conserver dans le fichier `pf.conf` pour une configuration aussi simple que celle-ci :

```
nat on $ext_if from $localnet to any -> ($ext_if)
```

Nous créons ensuite les ancres pour authpf, où les règles de `authpf.rules` sont chargées une fois l'utilisateur authentifié :

```
nat-anchor "authpf/*"
rdr-anchor "authpf/*"
binat-anchor "authpf/*"
anchor "authpf/*"
```

Cela nous amène à la fin des modifications nécessaires dans le fichier `pf.conf` pour qu'authpf puisse fonctionner.

En ce qui concerne le filtrage, nous partons d'un `block all` par défaut, puis nous ajoutons les règles `pass` dont nous avons besoin. Et, pour le moment, nous avons seulement besoin que le trafic SSH puisse passer sur le réseau local :

```
pass quick on $int_if inet proto { tcp, udp } to $int_if port ssh
```

Pour ce qui sort à partir de là, c'est à vous de décider. Voulez-vous permettre à vos clients de bénéficier du service de résolution de noms avant d'être authentifiés ? Si c'est le cas, placez également les règles `pass` pour le service `domain` en `tcp` et en `udp` dans votre `pf.conf`.

Pour une configuration simple et relativement égalitaire, vous pourriez inclure le reste de notre jeu de règles basique, en modifiant toutefois les règles `pass` de manière à autoriser le trafic provenant des adresses de la table `<authpf_users>` au lieu des adresses de votre réseau local :

```
pass quick inet proto { tcp, udp } from <authpf_users> to any port
$udp_services
pass inet proto tcp from <authpf_users> to any port $client_out
```

Pour une configuration plus différenciée, vous pourriez mettre le reste de votre jeu de règles dans `/etc/authpf/authpf.rules` ou placer les règles concernant chacun des utilisateurs particuliers dans un fichier `authpf.rules` spécifique, placé dans le dossier de chacun des utilisateurs sous `/etc/authpf/users/`. Si vos utilisateurs ont toujours besoin d'une protection, votre `/etc/authpf/authpf.rules` pourrait contenir des règles du type :

```
client_out = "{ ssh, domain, pop3, auth, nntp, http, https }"
udp_services = "{ domain, ntp }"
pass quick inet proto { tcp, udp } from $user_ip to any port $udp_services
pass inet proto tcp from $user_ip to any port $client_out
```

La macro `$user_ip` est construite au sein de authpf et se développe en l'adresse IP depuis laquelle l'utilisateur s'est authentifié. Ces règles s'appliquent à tout utilisateur authentifié auprès de votre passerelle.

Un ajout intéressant, et relativement facile à mettre en œuvre, consiste à insérer des règles spécifiques conçues pour les utilisateurs ayant des besoins différents du reste de la population. Si un fichier `authpf.rules` existe dans le répertoire propre à l'utilisateur dans le dossier `/etc/authpf/users/`, les règles de ce fichier seront chargées pour cet utilisateur.

Cela signifie que votre naïf utilisateur Peter, sous Windows, qui n'a besoin que de surfer sur le Web et d'avoir accès à un service qui fonctionne sur un port élevé d'une machine spécifique, pourrait se voir comblé par un fichier `/etc/authpf/users/peter/authpf.rules` comme celui-ci :

```
client_out = "{ domain, http, https }"
pass inet from $user_ip to 192.168.103.84 port 9000
pass quick inet proto { tcp, udp } from $user_ip to any port $client_out
```

Par contre, Christina, collègue de Peter, utilise OpenBSD et sait généralement ce qu'elle fait, même si elle génère parfois du trafic en provenance ou en direction de ports bizarres. Vous pourriez lui donner un peu de liberté en mettant la ligne suivante dans `/etc/authpf/users/christina/authpf.rules` :

```
pass from $user_ip os = "OpenBSD" to any
```

Cela signifie que Christina peut faire à peu près tout ce qu'elle veut par TCP, tant qu'elle s'authentifie depuis ses machines OpenBSD.

Grand ouvert en apparence, mais complètement fermé en réalité

Dans certaines situations, il peut être intéressant de configurer votre réseau de manière ouverte et non chiffrée au niveau lien, tout en renforçant les restrictions via `authpf`. L'exemple suivant est très similaires aux zones Wi-Fi que vous pouvez rencontrer dans les aéroports et autres espaces publics : n'importe qui peut s'associer au point d'accès et obtenir une adresse IP, mais toute tentative d'accès au Web sera redi-

rigée vers une page spécifique jusqu'à ce que l'utilisateur se soit authentifié d'une manière ou d'une autre[1].

Le fichier `pf.conf` suivant repose encore sur notre jeu de règles de base, avec deux ajouts importants : une macro et une redirection.

```
ext_if = "re0"
int_if = "ath0"
auth_web="192.168.27.20"
dhcp_services = "{ bootps, bootpc }" # serveur et client DHCP
table <authpf_users> persist
rdr pass on $int_if proto tcp from ! <authpf_users> to any port
 ➥ http -> $auth_web
nat on $ext_if from $localnet to any -> ($ext_if)
nat-anchor "authpf/*"
rdr-anchor "authpf/*"
binat-anchor "authpf/*"
anchor "authpf/*"
pass quick on $int_if inet proto { tcp, udp } to $int_if port dhcp_services
pass quick inet proto { tcp, udp } from $int_if:network to any port domain
pass quick on $int_if inet proto { tcp, udp } to $int_if port ssh
```

La macro `$auth_web` et la redirection garantissent que tout le trafic web provenant d'adresses absentes de la table `<authpf_users>` mène tous les utilisateurs non authentifiés vers une adresse spécifique.

Mettez en place à cette adresse un serveur web correspondant à vos besoins. Cela pourrait être n'importe quoi, d'une simple page donnant les coordonnées de la personne à contacter pour accéder au réseau, jusqu'à un système qui accepte les cartes de crédit et gère la création d'un compte utilisateur.

Il est intéressant de noter que la résolution de noms fonctionnera dans cette configuration, mais toutes les tentatives de surf sur le Web arriveront à l'adresse `$auth_web`. Une fois l'utilisateur authentifié, vous pouvez ajouter des règles générales et / ou spécifiques aux fichiers `authpf.rules` selon votre situation.

1. Merci à Vegard Engen pour l'idée et pour m'avoir montré sa configuration, dont l'esprit est ici conservé même si les détails diffèrent.

Des réseaux plus grands ou plus complexes

5

Dans ce chapitre, nous allons poursuivre la construction sur les fondations posées dans les chapitres précédents, afin de répondre aux véritables défis que posent les réseaux de plus grande envergure ou ceux hébergeant des applications ou des utilisateurs relativement exigeants. Les exemples de configuration de ce chapitre présupposent que votre filtre de paquets gère les services que vous faites fonctionner sur votre réseau local. Nous étudierons tout cela dans une perspective Unix, en nous concentrant sur SSH, la messagerie électronique et les services web. Nous donnerons quelques références sur la manière de gérer les autres services.

Quand les utilisateurs ont des besoins spécifiques sur votre réseau : le filtrage de services

Le temps passe et les besoins évoluent. Évoluer peut signifier que votre organisation (et son réseau) grandit ou que vous avez décidé de migrer des points critiques de l'infrastructure professionnelle vers BSD et PF.

Ce chapitre traite de ce que vous devrez accomplir pour combiner filtrage de paquets et services accessibles depuis l'extérieur de votre réseau local. Le nombre et la nature des complications que cela implique sur votre jeu de règles dépendent de la conception de votre réseau – et, dans une certaine mesure, du nombre d'adresses routables dont vous disposez.

Les pages suivantes seront consacrées aux bases du filtrage de paquets dans les réseaux dotés de services accessibles depuis l'extérieur. Nous commencerons par créer une base, qui prendra la forme de configurations à adresses routables et officielles.

Nous aborderons ensuite des situations à une seule adresse routable, avec des solutions basées sur PF pour rendre le service utilisable malgré les restrictions.

Un serveur web et un serveur de messagerie électronique à l'intérieur – adresses routables

Quel est le niveau de complexité de votre réseau ? De quel niveau de complexité a-t-il besoin ?

Commençons par un scénario de base, où les clients de l'exemple donné au chapitre 3 accueillent trois nouveaux voisins : un serveur web, un serveur de messagerie électronique et un serveur de fichiers. Dans ce scénario, nous utilisons des adresses routables et officielles, car cela nous simplifie un peu la vie. L'utilisation d'adresses routables a également d'autres avantages : elles nous permettent de faire tourner le service de noms de domaines (DNS) de notre domaine d'exemple (`example.com`) sur deux des nouvelles machines, l'une en tant que maître et l'autre en tant qu'esclave.

> En ce qui concerne le DNS, il est toujours intéressant d'avoir au moins un serveur esclave situé quelque part en dehors de votre réseau (en fait, certains domaines du niveau le plus haut ne vous permettront pas d'enregistrer un domaine si vous ne répondez pas à ce critère). Il peut également être judicieux de s'arranger pour de disposer d'un serveur mail de secours, hébergé ailleurs. Gardez cela à l'esprit quand vous construisez votre réseau. Cependant, cela n'affecte en rien la manière d'écrire notre jeu de règles PF.

Jusqu'ici, la disposition physique du réseau est restée plutôt simple. Nous plaçons les nouveaux serveurs dans le même réseau local que les clients ; ces serveurs peuvent être dans une pièce à part, mais ils doivent être situés sur le même segment de réseau ou le même switch que les clients. Conceptuellement, le réseau est similaire à celui de la figure 5-1.

Une fois arrêtés les paramètres de base du réseau, nous pouvons nous atteler à mettre en place un bon jeu de règles afin de gérer les services nécessaires. Là encore, nous partons du jeu de règles de base et nous y ajoutons quelques macros à des fins de lisibilité.

Les macros requises découlent assez naturellement des spécifications. Il nous en faut pour :

- notre serveur web (`webserver = "192.0.2.227"`) et les services qu'il propose (`webports = "{ http, https }"`) ;
- le serveur de messagerie (`emailserver = "192.0.2.225"`) et les services qu'il propose (`email = "{ smtp, pop3, imap, imap3, imaps, pop3s }"`) ;
- les serveurs de noms (`nameservers = "{ 192.0.2.221, 192.0.2.223 }"`).

Nous supposons que le serveur de fichiers n'a pas à être accessible depuis le monde extérieur (à moins que nous ayons opté pour un service visible depuis l'extérieur, comme un serveur de noms esclave ayant autorité pour notre domaine).

> **CULTURE RFC 3330 pour préciser les adresses IP réservées aux exemples**
>
> Le réseau `example.com` vit dans le bloc `192.0.2.0/24`, qui est défini dans la RFC3330 comme étant réservé aux exemples et à la documentation. Nous utilisons ce rang d'adresses principalement pour nous démarquer des exemples NAT présentés ailleurs dans le livre, qui utilisent des adresses issues de l'espace d'adressage « privé » défini dans la RFC 1918.

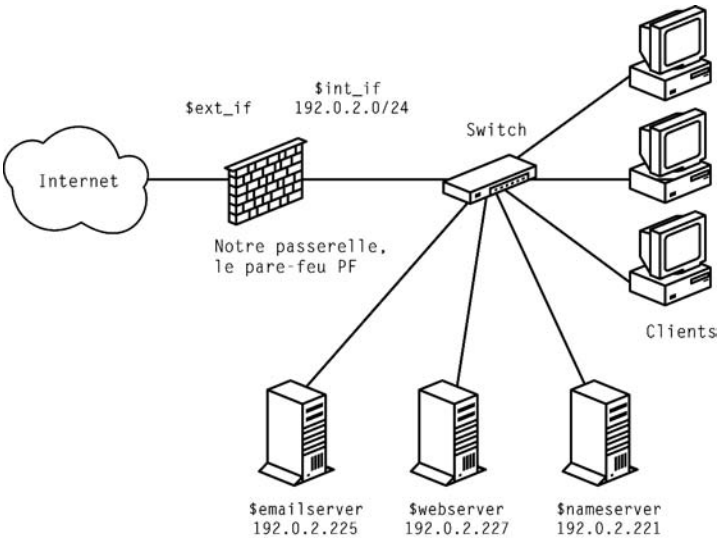


Figure 5–1
Un réseau élémentaire, avec des clients et des serveurs à l'intérieur.

Nos macros définies, nous ajoutons les règles `pass`. Commençons par le serveur web, que nous rendons accessible au monde entier par :

```
pass proto tcp from any to $webserver port $webports synproxy state
```

Remarquez l'option `synproxy state`. À l'établissement d'une connexion, PF laisse normalement aux deux extrémités le soin de régler elles-mêmes leurs paramètres, en laissant simplement passer les paquets s'ils correspondent à une règle `pass`. En activant `synproxy`, PF gère lui-même l'étape d'initialisation de la connexion et ne laisse la main aux extrémités qu'une fois la connexion correctement établie. Le fait que PF agisse comme un intermédiaire (ou proxy) pour la triple poignée de mains (*three-way handshake*) TCP aide à se prémunir des attaques de type *SYN flood* et autres malveillances pouvant conduire à saturer les ressources côté serveur. Ceci dit, l'option

`synproxy` consomme un peu plus de ressources que le comportement `keep state` par défaut et elle peut alourdir la charge du pare-feu de façon non négligeable.

De la même manière, nous permettons au monde entier d'entrer en contact avec le serveur mail :

```
pass proto tcp from any to $emailserver port $email synproxy state
```

Il est intéressant de remarquer que cette règle permet à tout client, quel que soit son emplacement, d'accéder au serveur (y compris aux protocoles de récupération de messages électroniques n'utilisant pas de chiffrement) exactement comme les clients du réseau local. C'est plutôt courant, mais il peut être judicieux d'étudier d'autres possibilités à la construction d'un nouveau réseau.

Pour être utile, le serveur mail doit également être capable d'envoyer des e-mails aux hôtes extérieurs au réseau local :

```
pass log proto tcp from $emailserver to any port smtp synproxy state
```

En gardant à l'esprit que le jeu de règles s'ouvre par une règle `block all`, cela signifie que seul le serveur de messagerie est autorisé à initier du trafic SMTP depuis le réseau local vers le reste du monde. Tout autre hôte du réseau devant envoyer des messages électroniques vers l'extérieur ou en recevoir doit obligatoirement passer par ce serveur mail. Cela peut être une solution adaptée pour, par exemple, empêcher dans la mesure du possible qu'un zombie d'envoi de spam ayant infecté une de vos machines puisse réellement nuire à autrui.

Enfin, les serveurs de noms doivent être accessibles aux clients extérieurs à notre réseau qui cherchent des informations concernant `example.com` ou tout autre domaine sur lequel nous avons autorité :

```
pass inet proto { tcp, udp } from any to $nameservers port domain
```

Une fois intégrés tous les services devant être accessibles depuis le monde extérieur, notre jeu de règles ressemble à peu près à ceci :

```
ext_if = "ep0" # macro pour l'interface externe
               # - utilisez tun0ou pppoe0 pour PPPoE
int_if = "ep1" # macro pour l'interface externe
localnet = $int_if:network
webserver = "192.0.2.227"
webports = "{ http, https }"
emailserver = "192.0.2.225"
```

```
email = "{ smtp, pop3, imap, imap3, imaps, pop3s }"
nameservers = "{ 192.0.2.221, 192.0.2.223 }"
client_out = "{ ssh, domain, pop3, auth, nntp, http,\
                https, cvspserver, 2628, 5999, 8000, 8080 }"
udp_services = "{ domain, ntp }"
icmp_types = "{ echoreq, unreach }"
block all
pass quick inet proto { tcp, udp } from $localnet to any port
$udp_services
pass log inet proto icmp all icmp-type $icmp_types
pass inet proto tcp from $localnet to any port $client_out
pass inet proto { tcp, udp } from any to $nameservers port domain
pass proto tcp from any to $webserver port $webports synproxy state
pass log proto tcp from any to $emailserver port $email synproxy state
pass log proto tcp from $emailserver to any port smtp synproxy state
```

C'est toujours une configuration plutôt simple mais qui, malheureusement, présente un problème de sécurité potentiel. La conception de ce réseau fait que les serveurs accessibles au monde entier se situent tous *dans le même réseau local* que les clients et il faudrait restreindre tous les services internes aux seuls clients locaux. En principe, cela signifie qu'il suffirait à un attaquant de compromettre un seul hôte du réseau local pour pouvoir accéder à toutes les ressources (l'attaquant serait alors sur un pied d'égalité avec tous les utilisateurs locaux). Selon le niveau de sécurité et de protection de chaque machine et ressource face aux accès non autorisés, cela peut être un ennui mineur ou un problème majeur.

Dans la section suivante, nous étudierons quelques possibilités permettant d'isoler les services devant interagir avec le monde entier depuis le réseau local.

Un degré de séparation physique : présentation de la DMZ

À la section précédente, nous avons montré que l'on peut mettre en place des services sur un réseau local et les rendre disponibles à l'extérieur de manière sélective au travers d'un jeu de règles PF réfléchi. Cependant, il est possible d'exercer un contrôle plus fin sur les accès au réseau local et sur les services qui doivent être visibles de l'extérieur, en introduisant un degré de séparation physique.

La séparation logique et physique est assez simple à concrétiser. En déplaçant vers un réseau à part, attaché à une interface dédiée sur la passerelle, les machines hébergeant les services publics, on obtient en fait un réseau qui ne fait partie ni du réseau local ni de la partie publique d'Internet. Conceptuellement, le réseau séparé ressemble à celui de la figure 5-2.

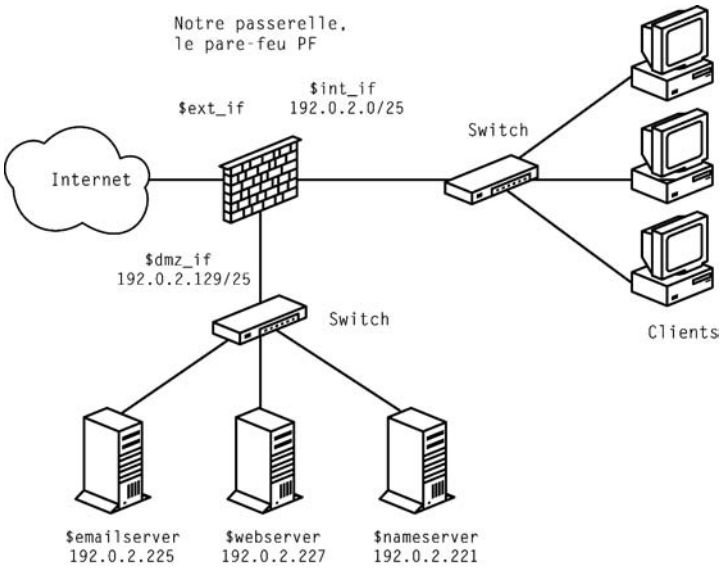


Figure 5–2
Un réseau où les serveurs sont dans la DMZ.

On peut voir ce petit réseau comme une zone de calme relatif entre les territoires de plusieurs factions hostiles. C'est donc sans surprise que, il y a quelques années, quelqu'un a adopté la formule *DeMilitarized Zone* (DMZ, pour zone démilitarisée) pour décrire ce type de configuration. Le terme est resté.

Pour l'allocation des adresses, on peut soit assigner au nouveau réseau DMZ une part adéquate de l'espace d'adressage officiel, soit déplacer cette partie du réseau (qui n'a pas besoin d'adresses publiquement accessibles et routables) dans un environnement NAT. Ces deux façons de faire conduisent à filtrer au moins une interface de plus. En fait, comme nous le verrons plus tard, en cas de pénurie avérée d'adresses officielles, il est également possible d'héberger une configuration DMZ dans un environnement entièrement DMZ.

Les ajustements à apporter au jeu de règles PF sont mineurs. Si nécessaire, on peut modifier la configuration de toutes les interfaces. La même logique de base doit s'appliquer au jeu de règles, mais les définitions des macros (`webserver`, `emailserver`, `nameservers` et peut-être d'autres) peuvent nécessiter quelques ajustements pour refléter la nouvelle disposition du réseau.

Dans notre exemple, nous pourrions choisir de segmenter la partie de notre rang d'adresses où nous avons déjà placé nos serveurs ; si nous laissons un peu de marge pour d'éventuels agrandissements futurs, nous pouvons mettre en place la nouvelle `dmz_if` sur un sous-réseau `/25` portant l'adresse et le masque de sous-réseau `192.0.2.129/255.255.255.128`. Avec cette configuration,il n'est pas vraiment néces-

saire de toucher au jeu de règles pour que le filtrage de paquets fonctionne après la création d'une DMZ physiquement séparée.

Que cette réflexion trouve son origine dans une certaine paresse ou dans un souci d'organisation, elle souligne néanmoins l'importance d'une politique d'allocation réfléchie des adresses.

Il peut se révéler utile de renforcer le jeu de règles en éditant les règles `pass`, afin de n'autoriser le trafic en direction ou en provenance des serveurs que sur les interfaces correspondant réellement aux services :

```
pass in on $ext_if proto { tcp, udp } from any to $nameservers port domain
pass in on $int_if proto { tcp, udp } from $localnet to $nameservers port domain
pass out on $dmz_if proto { tcp, udp } from any to $nameservers port domain
pass in on $ext_if proto tcp from any to $webserver port $webports
pass in on $int_if proto tcp from $localnet to $webserver port $webports
pass out on $dmz_if proto tcp from any to $webserver port $webports
pass in log on $ext_if proto tcp from any to $emailserver port smtp
pass in log on $int_if proto tcp from $localnet to $emailserver port $email
pass out log on $dmz_if proto tcp from any to $emailserver port smtp
pass in on $dmz_if from $emailserver to any port smtp
pass out log on $ext_if proto tcp from $emailserver to any port smtp
```

On peut choisir de lier à une interface particulière les autres règles `pass` relatives au réseau local, mais elles continueront à fonctionner si elles restent en l'état.

Répartir la charge : redirection vers une grappe d'adresses

Au fil du temps, un ou plusieurs des services accessibles au monde entier peuvent devenir plus sophistiqués, plus gourmands en ressources, ou tout simplement susciter tellement de trafic qu'un seul serveur ne suffit plus.

Il y a de nombreuses manières de répartir la charge d'un service entre plusieurs machines, y compris en affinant les réglages du service en lui-même. Les détails de la gestion d'un serveur web dépassent le cadre de cet ouvrage mais, en ce qui concerne la répartition de charge au niveau réseau, PF propose des fonctionnalités élémentaires pour rediriger le trafic vers un ensemble de plusieurs adresses ou *grappe d'adresses* (*address pools*).

Pour le serveur web de notre exemple, nous disposons déjà de la macro de l'adresse publique (`webserver = "192.0.2.227"`) correspondant au nom d'hôte auquel vos utilisateurs ont associé un marque-page (par exemple `www.example.com`).

Pour assurer la répartition de la charge, on met en place autant de serveurs identiques (ou, du moins, équivalents) que nécessaire puis on modifie légèrement les règles pour

introduire la redirection. Commençons par ajouter la macro qui décrit la grappe de serveurs web :

```
webpool = "{ 192.0.2.214, 192.0.2.215, 192.0.2.216, 192.0.2.217 }"
```

Puis on spécifie la redirection (on peut même, une fois que tout est en place, retirer le serveur web d'origine).

```
rdr on $ext_if from any to $webserver port $webports -> $webpool \
round-robin sticky-address
```

L'option `round-robin` signifie que PF répartit la charge entre les machines de la grappe en parcourant successivement et cycliquement toutes les adresses de la grappe. L'option `sticky-address` garantit que toute nouvelle connexion sera systématiquement redirigée vers la même machine que la connexion d'origine.

L'option `sticky-address` peut se révéler essentielle si le service dépend de paramètres spécifiques à un client ou à une session, qui pourraient être perdus si le client est redirigé vers le service équivalent d'un hôte différent. Dans les autres contextes, où même la répartition de la charge n'est pas une exigence absolue, la sélection aléatoire (`random`) de l'adresse de redirection peut être adéquate :

```
rdr on $ext_if from any to $webserver port $webports -> $webpool random
```

Il est intéressant de noter que même les organisations disposant d'un grand nombre d'adresses routables officielles ont choisi d'introduire la traduction d'adresses entre, d'une part, leurs grappes de serveurs à charge répartie et, d'autre part, Internet. Cette technique fonctionne tout aussi bien dans les diverses configurations basées sur la NAT, mais la traduction d'adresses offre des possibilités et des défis supplémentaires.

La répartition de charge dans les règles de l'art avec hoststated

Après avoir assuré pendant quelque temps la répartition de charge avec une redirection `round-robin`, vous remarquerez peut-être que celle-ci ne s'adapte pas automatiquement aux conditions extérieures. Par exemple, imaginons qu'un ou plusieurs des hôtes appartenant à votre grappe de redirection viennent à tomber. À moins d'avoir pris quelques précautions, le trafic sera toujours redirigé vers les adresses IP de la liste et ce, même si l'hôte visé est injoignable ou incapable de gérer les requêtes.

Une solution de supervision est clairement nécessaire ici et, heureusement, le système de base d'OpenBSD en fournit une : le démon de vérification d'état des hôtes

`hoststated`. `hoststated` interagit avec votre configuration PF et permet de retirer de votre grappe les hôtes non fonctionnels.

Introduire `hoststated` dans votre configuration peut cependant nécessiter des modifications mineures à votre jeu de règles. `hoststated` fonctionne en termes de *services* et il est capable d'ajouter ou de retirer les adresses IP des hôtes à destination ou en provenance de *tables* PF. Le démon interagit avec votre jeu de règles au travers d'une ancre de redirection dédiée, nommée `hoststated`. Pour illustrer une amélioration possible du fonctionnement de notre exemple grâce à `hoststated`, nous allons revoir un peu le jeu de règles qui gère la répartition de charge.

En tête de votre fichier `pf.conf`, ajoutez la ligne suivante à la section NAT :

```
rdr-anchor "hoststated/*"
```

Dans le jeu de règles de répartition de charge, nous avions défini notre grappe de serveurs web :

```
webpool = "{ 192.0.2.214, 192.0.2.215, 192.0.2.216, 192.0.2.217 }"
```

ainsi que cette redirection :

```
rdr on $ext_if from any to $webserver port $webports -> $webpool \
round-robin sticky-address
```

Pour améliorer un peu le fonctionnement de cette configuration, nous supprimons la redirection et nous laissons à `hoststated` le soin de la gérer par le biais de l'ancre de redirection associée. La cible de redirection est donc transformée en une table créée et maintenue par `hoststated`. Ne retirez cependant pas la règle `pass` : votre jeu de règles en a toujours besoin pour laisser le trafic s'écouler vers les adresses IP des tables de `hoststated`.

NOTE DU TRADUCTEUR

Le paragraphe ci-dessus est la traduction de l'errata publié sur le site web de No Starch Press :

- http://www.nostarch.com/pf_updates.htm

et surtout :

- http://www.bsdly.net/bookofpf/errata-01-hoststated-rdr.txt

Après avoir modifié le fichier `pf.conf`, passons au fichier de configuration de `hoststated`, `hoststated.conf`. La syntaxe de ce fichier de configuration est assez

similaire à celle de `pf.conf`, afin d'en faciliter la lecture et la compréhension. Commençons par ajouter les définitions des macros que nous utiliserons plus tard :

```
web1="192.0.2.214"
web2="192.0.2.215"
web3="192.0.2.216"
web4="192.0.2.217"
webserver="192.0.2.227"
sorry_server="192.0.2.200"
```

Elles correspondent toutes à des définitions qui pourraient figurer dans un fichier `pf.conf`. L'intervalle de vérification par défaut de `hoststated` est de 10 secondes : un hôte est donc indisponible au plus 10 secondes avant d'être mis hors-ligne. Soyons prudents et fixons cet intervalle à 5 secondes, afin de minimiser de temps d'indisponibilité :

```
interval 5     # vérifie les hôtes toutes les 5 secondes
```

Créons à présent une table, nommée `webpool`, qui utilise la plupart des macros :

```
table webpool {
        check http "/status.html" code 200
        timeout 300
        real port 80
        host $web1
        host $web2
        host $web3
        host $web4
}
```

Cette table définit les hôtes membres mais elle impose également à `hoststated` de vérifier la disponibilité d'un hôte. Il lui demande pour cela le fichier `status.html` par le protocole HTTP et s'attend à un code de retour égal à `200` (c'est-à-dire le code renvoyé par un serveur web quand un client lui demande un fichier effectivement disponible).

Jusqu'ici, pas de grande surprise, n'est-ce pas ? `hoststated` se chargera d'exclure les hôtes de la table en cas d'indisponibilité. Mais que se passe-t-il si tous les hôtes de la table `webpool` tombent ? Les développeurs ont heureusement prévu ce cas et introduit le concept de *tables de secours* (*backup tables*) pour les services. C'est la dernière partie de la définition du service `www`, avec la table `sorry` dans le rôle de table de secours :

```
table sorry {
        check icmp
        real port 80
        host $sorry_server
}
service www {
        virtual ip $webserver port 80
        table webpool
        backup table sorry
}
```

Les hôtes de la table `sorry` sont ceux qui prennent la relève si jamais la table `webpool` devient vide. Cela signifie que l'on doit configurer un service capable d'émettre un message « Désolé, nous sommes hors-service. » au cas où tous les hôtes de la grappe de serveurs web tombent.

Si `hoststated` doit être activé au démarrage, on ajoute la ligne :

```
hoststated_flags=""    # pour l'usage normal : ""
```

au fichier `rc.conf.local`. Notez toutefois que la plupart de vos interactions avec `hoststated` se feront au travers du programme d'administration `hoststatectl`. En plus de la supervision des états, `hoststatectl` permet de recharger la configuration de `hoststated` et d'activer ou de désactiver des hôtes, des tables et des services individuels. On peut même visualiser les états de manière interactive, comme ceci :

```
$ sudo hoststatectl show summary
Type      Id    Name                Avlblty  Status
service    0    www                          active
table      0    webpool                      active (2 hosts up)
host       3    192.0.2.217           0.00%  down
host       2    192.0.2.216         100.00%  up
host       1    192.0.2.215           0.00%  down
host       0    192.0.2.214         100.00%  up
table      1    sorry                        active (1 hosts up)
host       4    192.0.2.200         100.00%  up
```

Dans cet exemple, la grappe de serveurs web est en piteux état : seuls deux des quatre hôtes fonctionnent effectivement. Heureusement, la table de secours fonctionne toujours. Ici, toutes les tables sont actives, avec au moins un hôte en fonctionnement. Quand une table n'a plus aucun membre, la colonne `Status` indique `empty` (*vide*).

Si l'on demande à `hoststatectl` des informations sur les hôtes, on obtient des données centrées sur les hôtes :

```
$ sudo hoststatectl show hosts
Type      Id    Name                 Avlblty Status
service    0    www                          active
table      0    webpool                      active (2 hosts up)
host       3    192.0.2.217            0.00% down
                total: 0/6 checks
host       2    192.0.2.216          100.00% up
                total: 0/6 checks
host       1    192.0.2.215            0.00% down
                total: 0/6 checks
host       0    192.0.2.214          100.00% up
                total: 6/6 checks
table      1    sorry                        active (1 hosts up)
host       4    192.0.2.200          100.00% up
                total: 6/6 checks
```

Si vous devez sortir un hôte de la grappe (pour maintenance ou une quelconque opération gourmande en temps de calcul), vous pouvez utiliser `hoststatectl` pour le désactiver par la commande suivante :

```
$ sudo hoststatectl host disable 192.0.2.214
```

Dans la plupart des cas, cette commande renverra `command succeeded` pour indiquer que l'opération s'est terminée avec succès. Une fois la maintenance terminée et la machine remise en ligne, vous pouvez la réintégrer à la grappe de `hoststated` par cette commande :

```
$ sudo hoststatectl host enable 192.0.2.214
```

Vous devriez presque immédiatement voir apparaître une nouvelle fois le message `command succeeded`.

En plus de la répartition de charge élémentaire que nous avons montrée ici, `hoststated` a été étendu dans les version récentes d'OpenBSD afin d'offrir un certain nombre de fonctionnalités supplémentaires, qui en font un outil attrayant dans d'autres problématiques complexes. Il peut désormais se comporter en proxy de niveau 7 et gérer des fonctions de relais pour HTTP et HTTPS : cela comprend la gestion du protocole avec ajout et réécriture de l'en-tête et de l'URL, et même la gestion des sessions et des cookies.

La gestion du protocole doit être taillée sur mesure pour votre application. Voici un relais HTTPS simple pour répartir la charge du trafic web chiffré à destination du serveur web :

```
protocol httpssl {
      protocol http
           header append "$REMOTE_ADDR" to "X-Forwarded-For"
          header append "$SERVER_ADDR:$SERVER_PORT" to "X-Forwarded-By"
           header change "Keep-Alive" to "$TIMEOUT"
           url hash "sessid"
           cookie hash "sessid"
           path filter "*command=*" from "/cgi-bin/index.cgi"

           ssl { sslv2, ciphers "MEDIUM:HIGH" }
           tcp { nodelay, sack, socket buffer 65536, backlog 128 }
}
```

Le gestionnaire de protocole ci-dessus définit un ensemble d'opérations simples sur les en-têtes HTTP, ainsi que les paramètres SSL et les paramètres spécifiques à TCP afin d'optimiser la gestion des connexions. Les options `header` opèrent sur les en-têtes de protocole et insèrent les valeurs des variables, soit en les ajoutant aux en-têtes existant (`append`), soit en écrasant le contenu par une nouvelle valeur (`change`). Les hachages `url` et `cookie` sont utilisés par le répartiteur de charge pour sélectionner l'hôte de la grappe cible vers lequel transférer la requête. `path filter` (*filtre de chemin*) indique que toute requête `GET` (y compris la première chaîne citée en tant que sous-chaîne de la seconde) doit être refusée. Les options `ssl` définissent que seuls les algorithmes de chiffrement SSL version 2 sont acceptés, avec des clés de longueur moyenne à haute (c'est-à-dire au minimum de 128 bits).

> Voir la page de manuel d'OpenSSL pour de plus amples informations concernant les options liées aux algorithmes de chiffrement.

Enfin, les options `tcp` indiquent que le drapeau ToS doit être fixé à `nodelay` et que l'on doit utiliser la méthode d'acquittement sélectif (*selective acknowledgment*, RFC 2018) ; elles fixent en outre la taille du tampon du socket ainsi que le nombre de connexions simultanées dont le répartiteur de charge conserve la trace.

La définition de relais utilisant le gestionnaire de protocole suit le même modèle que la définition du service www que nous avons posée précédemment :

```
relay wwwssl {
        # Tourne en tant qu'accélérateur SSL
        listen on $webserver port 443 ssl
        protocol httpssl
        table webhosts loadbalance
}
```

Il est cependant probable que vos applications web SSL disposeront d'un jeu de paramètres légèrement différent.

Enfin, `hoststated` peut être configuré de manière à prendre en charge CARP pour gérer la tolérance aux pannes des hôtes utilisant `hoststated` dans votre réseau (voir « Redondance et tolérance aux pannes : CARP et pfsync »). Il faut pour cela fixer le compteur de dégradation (*demotion counter*) de CARP pour les groupes d'interfaces spécifiés ou les groupes du démarrage ou de l'extinction. Comme toutes les briques du système OpenBSD, `hoststated` est livré avec sa documentation, sous forme de pages de manuel. Pour les options que nous n'avons pas couvertes ici (il y en a quelques-unes), je recommande une étude approfondie des pages de manuel de `hoststated`, `hoststated.conf` et `hoststatectl`, ainsi que l'expérimentation pour parvenir à une configuration optimale.

Un serveur web et un serveur de messagerie électronique à l'intérieur – version NAT

Revenons un peu sur nos pas jusqu'au scénario de base, dans lequel les clients de l'exemple du chapitre 3 accueillent trois nouveaux voisins : un serveur mail, un serveur web et un serveur de fichiers. Cette fois-ci, les adresses visibles de l'extérieur sont soit indisponibles, soit trop chères, et faire tourner plusieurs autres services sur une machine qui est à la base un pare-feu n'est pas une option intéressante.

Cela signifie que nous sommes réduits à faire de la traduction d'adresses sur la passerelle. Heureusement, les mécanismes de redirection de PF simplifient la gestion des serveurs à l'intérieur d'une passerelle NAT. Les spécifications du réseau sont les mêmes que dans le cas `example.com` que nous venons de traiter : nous devons faire tourner un serveur web qui sert des données en clair (HTTP) et chiffrées (HTTPS), ainsi qu'un serveur mail qui envoie et reçoit des messages électroniques tout en permettant aux clients internes et externes d'utiliser un certain nombre de protocoles d'envoi et de réception :

```
webserver = "192.168.2.7"
webports = "{ http, https }"
emailserver = "192.168.2.5"
email = "{ smtp, pop3, imap, imap3, imaps, pop3s }"

rdr on $ext_if proto tcp from any to $ext_if port $webports -> $webserver
rdr on $ext_if proto tcp from any to $ext_if port $email -> $emailserver

pass proto tcp from any to $webserver port $webports synproxy state
pass proto tcp from any to $emailserver port $email synproxy state
pass proto tcp from $emailserver to any port smtp synproxy state
```

Nous utilisons ici aussi l'option `synproxy` : cela signifie que PF devra gérer l'établissement des connexions (la triple poignée de mains TCP) au nom de notre serveur ou de notre client, avant de passer la main aux applications qui tournent de chaque côté. Pour rappel, ce mécanisme offre un certain degré de protection contre les attaques basées sur les paquets SYN.

DMZ avec NAT

Avec une configuration entièrement NAT, la réserve d'adresses disponibles pour allocation à la DMZ sera probablement plus importante que dans notre exemple précédent de la section « Un degré de séparation physique : présentation de la DMZ », mais les mêmes principes s'appliquent. Lors du transfert d'un serveur vers un réseau physiquement à part, il faut vérifier l'exactitude des définitions de macros dans le jeu de règles et, au besoin, ajuster les valeurs.

Comme dans le cas des adresses routables, il peut être utile de renforcer le jeu de règles en éditant les règles `pass`, de manière à ce que le trafic à destination et en provenance des serveurs ne soit autorisé à emprunter que les interfaces réellement pertinentes pour les services concernés :

```
pass in on $ext_if proto { tcp, udp } from any to $nameservers port domain
pass in on $int_if proto { tcp, udp } from $localnet to $nameservers port domain
pass out on $dmz_if proto { tcp, udp } from any to $nameservers port domain
pass in on $ext_if proto tcp from any to $webserver port $webports
pass in on $int_if proto tcp from $localnet to $webserver port $webports
pass out on $dmz_if proto tcp from any to $webserver port $webports
pass in log on $ext_if proto tcp from any to $emailserver port smtp
pass in log on $int_if proto tcp from $localnet to $emailserver port $email
pass out log on $dmz_if proto tcp from any to $emailserver port smtp
pass in on $dmz_if from $emailserver to any port smtp
pass out log on $ext_if proto tcp from $emailserver to any port smtp
```

S'il est intéressant pour vous de rédiger d'autres règles `pass` référençant l'interface de votre réseau local, vous pouvez l'envisagez mais, si vous les laissez intactes, elles continueront à fonctionner.

Redirection pour répartition de charge

Les règles de répartition de charge basées sur de la redirection, vues plus haut à la section « Répartir la charge : redirection vers une grappe d'adresses », fonctionnent tout aussi bien en régime NAT, où l'adresse publique est l'interface externe de la passerelle et les adresses de redirection appartiennent à un rang privé.

La principale différence entre le cas des adresses routables et la version NAT est qu'après avoir ajouté la définition de `webpool` :

```
webpool = "{ 192.168.2.7, 192.168.2.8, 192.168.2.9, 192.168.2.10 }"
```

on doit éditer la redirection existante, qui devient :

```
rdr on $ext_if from any to $webserver port $webports -> $webpool \
     round-robin sticky-address
```

À partir de là, la DMZ à NAT se comporte comme celle à adresses officielles et routables.

Retour sur le réseau unique utilisant la traduction d'adresses

Cela peut vous surprendre, mais il est des cas où mettre en place un petit réseau se révèle plus difficile que travailler avec un grand réseau.

Revenons à la situation où les serveurs se trouvent sur le même réseau physique que les clients. Le configuration NAT de base de la section « Un serveur web et un serveur de messagerie électronique à l'intérieur – version NAT » fonctionne très bien, mais jusqu'à un certain point seulement. En fait, tout fonctionne parfaitement tant que l'on cherche uniquement à permettre au trafic provenant de l'extérieur d'atteindre vos serveurs.

Afin de vous éviter de jongler entre les pages, voici la configuration de base :

```
webserver = "192.168.2.7"
webports = "{ http, https }"
emailserver = "192.168.2.5"
email = "{ smtp, pop3, imap, imap3, imaps, pop3s }"

nat on $ext_if from $localnet to any -> ($ext_if)

rdr on $ext_if proto tcp from any to $ext_if port $webports -> $webserver
rdr on $ext_if proto tcp from any to $ext_if port $email -> $emailserver

pass proto tcp from any to $webserver port $webports synproxy state
pass proto tcp from any to $emailserver port $email synproxy state
pass proto tcp from $emailserver to any port smtp synproxy state
```

Si vous tentez d'accéder aux services liés à l'adresse officielle depuis les hôtes de votre réseau interne, vous réaliserez rapidement que les requêtes provenant du réseau local et destinées aux services redirigés n'atteignent jamais l'interface externe. C'est parce que tout le travail de redirection et de traduction se fait sur celle-ci.

La passerelle reçoit les paquets provenant du réseau local sur l'interface externe, avec l'adresse de cette interface pour adresse de destination. La passerelle reconnaît l'adresse comme étant l'une des siennes et tente de gérer la requête comme si elle était destinée à l'un de ses propres services locaux. Par conséquent, les redirections ne fonctionnent pas très bien pour le trafic intérieur.

Heureusement, il existe plusieurs solutions à ce problème, qui est tellement courant que le *Guide de l'Utilisateur PF* y propose quatre solutions différentes (dont le transfert des serveurs vers une DMZ, déjà évoqué). Dans ce livre consacré à PF, nous nous concentrerons sur une solution basée sur PF, qui consiste à traiter le réseau local comme un cas particulier de nos règles de redirection et de NAT.

> Voir le chapitre « Redirection et réflexion » dans le *Guide de l'Utilisateur PF.*
> ▸ http://www.openbsd.org/faq/pf/fr/rdr.html#reflect

Nous devons intercepter les paquets réseau provenant du réseau local et les gérer de manière adéquate, en nous assurant que le trafic de retour est bien dirigé vers l'hôte ayant initié la connexion.

Cela signifie que, pour que les redirections fonctionnent comme prévu depuis le réseau local, nous devons ajouter des règles de redirection spécifiques. Ces règles seront le pendant des règles qui gèrent les requêtes provenant de l'extérieur :

```
rdr on $int_if proto tcp from $localnet to $ext_if port $webports -> $webserver
rdr on $int_if proto tcp from $localnet to $ext_if port $email -> $emailserver
no nat on $int_if proto tcp from $int_if to $localnet
nat on $int_if proto tcp from $localnet to $webserver port $webports -> $int_if
nat on $int_if proto tcp from $localnet to $emailserver port $email -> $int_if
```

De cette manière, nous « plions » le système de redirection et la logique de traduction d'adresses pour la faire correspondre à nos attentes et nous n'avons pas besoin de toucher aux règles `pass`.

Filtrer des groupes d'interfaces

Un réseau est peut-être composé de plusieurs sous-réseaux n'ayant pas forcément besoin d'interagir avec le réseau local, sauf pour des services courants tels que la messagerie électronique, le Web, les fichiers et l'impression. La gestion du trafic à destination et en provenance de tels sous-réseaux dépend de la conception du réseau. Une bonne approche consiste à traiter chaque réseau de moindre privilège comme un réseau à part, attaché à sa propre interface d'une passerelle filtrante commune, et de lui adjoindre ensuite un jeu de règles autorisant seulement les interactions souhaitées, sachant que les réseaux voisins sont attachés à la passerelle principale.

Sur la passerelle en elle-même, il peut être intéressant de rassembler les interfaces logiquement similaires en *groupes d'interfaces* et d'appliquer les règles de filtrage au groupe plutôt qu'aux interfaces. Les groupes d'interfaces, tels qu'implémentés dans l'option `group` d'`ifconfig`, sont à l'origine apparus dans OpenBSD 3.6 et ont été adoptés dans FreeBSD 7.0.

Toutes les interfaces réseau en état de fonctionnement peuvent être configurées de manière à appartenir à un (ou plusieurs) groupe(s). Certaines interfaces appartiennent automatiquement à l'un des groupes par défaut. Par exemple, toutes les interfaces sans fil IEEE 802.11 appartiennent au groupe `wlan`, tandis que les interfaces associées à la route par défaut appartiennent au groupe `egress`. Heureusement, une interface peut être membre de plusieurs groupes à la fois et la commande `ifconfig` permet d'ajouter des interfaces à un groupe :

```
# ifconfig sis2 group untrusted
```

(ou l'équivalent dans le fichier `hostname.sis2` sous OpenBSD, ou la ligne `ifconfig_sis2=` dans le fichier `rc.conf` sous FreeBSD 7.0 ou supérieur).

Au niveau des règles de filtrage, on peut ensuite traiter le groupe d'interfaces à peu près de la même façon qu'une interface unique :

```
pass in on untrusted to any port $webports
pass out on egress to any port $webports
```

Il est intéressant de noter que filtrer sur les groupes d'interfaces permet d'écrire des jeux de règles quasiment indépendants du matériel. Tant que les fichiers `hostname.<if>` ou les lignes `ifconfig_<if>` affectent les interfaces aux bons groupes, les jeux de règles qui filtrent sur ces groupes d'interfaces seront totalement portables entre machines partageant la même configuration matérielle ou non.

Des macros comme groupes d'interfaces

Sur les systèmes où les groupes d'interfaces ne sont pas disponibles, on peut quand même arriver à un effet proche en étant créatif dans l'utilisation des macros, comme ce qui suit :

```
untrusted = "{ ath0 ath1 wi0 ep0 }"
egress = "sk0"
```

La puissance des étiquettes (tags)

Dans certains réseaux, on ne peut pas décider de l'endroit où un paquet devrait être autorisé à passer en se basant uniquement sur des critères simples tels que le sous-réseau ou le service. Le niveau de contrôle exigé par la politique de sécurité peut compliquer le jeu de règles ainsi que sa maintenance.

Heureusement, PF offre encore un autre mécanisme de classification et de filtrage : l'étiquetage des paquets. Étiqueter correctement les paquets consiste à placer un `tag` (*étiquette*) sur les paquets entrants correspondant à une règles `pass` donnée puis, au niveau d'une autres interface réseau, à laisser circuler les paquets en fonction de l'étiquette qu'ils portent.

On pourrait prendre pour exemple le point d'accès sans fil monté au u chapitre 4 : il est raisonnable de supposer qu'il injectera du trafic dans le réseau local, avec une adresse source apparente égale à l'adresse son interface `$ext_if`.

Dans ce scénario, on pourrait alors ajouter les lignes suivantes au jeu de règles d'une passerelle conduisant vers plusieurs points d'accès semblables :

```
wifi = "{ 10.0.0.115, 10.0.0.125, 10.0.0.135, 10.0.0.145 }"
pass in on $int_if from $wifi to $wifi_allowed port $wifi_ports tag wifigood
pass out on $ext_if tagged wifigood
```

où les définitions des macros `$wifi_allowed` et `$wifi_ports` correspondent aux exigences du site.

Puisque la complexité du jeu de règles s'accroît au gré des évolutions des besoins du réseau, l'usage de `tag` et de `tagged` dans les règles `pass` est intéressant : cela favorise la lisibilité et simplifie la maintenance du jeu de règles. Chose importante : les étiquettes sont *collées* aux paquets (*sticky*). Un paquet étiqueté par une règle correspondante peut très bien l'être également par toutes les autres règles correspondantes, pas seulement par la dernière. On peut, par exemple, coller plusieurs étiquettes sur le trafic entrant via un jeu de règles `pass` ; un autre jeu de règles `pass` est alors utilisé en complément pour déterminer la porte de sortie des paquets en fonction des étiquettes qu'ils portent.

> Comme on l'a vu au chapitre 3, sous OpenBSD 4.2, ftp-proxy a acquis la capacité d'étiqueter les paquets, ce qui simplifie son intégration dans les configurations complexes. Voir la page de manuel de ftp-proxy pour de plus amples détails.

Pare-feu ponté

Un *pont* (*bridge*) Ethernet se compose de deux interfaces ou plus, configurées pour transférer des trames Ethernet de manière transparente qui ne sont pas directement visibles par les couches supérieures telles que la pile TCP/IP. Dans un contexte de filtrage, la configuration d'un pont est souvent jugée intéressante parce qu'elle implique de réaliser le filtrage sur une machine qui ne possède pas d'adresse IP. Si la machine en question fonctionne sous OpenBSD ou un système similaire, elle peut toujours filtrer et rediriger le trafic.

L'avantage principal d'une telle configuration est qu'il devient plus difficile d'attaquer le pare-feu en lui-même. L'inconvénient est que toutes les tâches d'administration doivent se faire à la console du pare-feu, à moins de configurer une interface accessible depuis un réseau sécurisé ou une console série.

Il en découle également qu'un pont sans adresse IP ne peut pas servir de passerelle pour un réseau et ne peut héberger aucun service sur les interfaces pontées. En fait, on peut peut voir un pont comme un boîtier intelligent posé sur le câble, servant à filtrer et rediriger le trafic.

Quelques avertissements s'imposent quand on veut utiliser un pare-feu sous forme de pont :

Les interfaces sont placées en mode *promiscuous*, ce qui signifie qu'elles reçoivent absolument tous les paquets du réseau.

- Les ponts travaillent au niveau Ethernet et, par défaut, transfèrent tous les types de paquets, pas seulement TCP/IP.
- L'absence d'adresse IP pour les interfaces interdit d'exploiter les fonctionnalités de redondance les plus avancées (par exemple CARP).

La marche à suivre pour configurer un pont diffère légèrement selon le système d'exploitation. Les exemples à suivre sont très simples et ne couvrent pas tous les cas possibles, mais ils devraient être suffisants pour se lancer.

Configuration d'un pont simple sous OpenBSD

Le noyau *GENERIC* d'OpenBSD contient tout le code nécessaire à la configuration et au filtrage d'un pont. À moins d'avoir compilé un noyau modifié duquel a été retiré le code de pont, la configuration est plutôt directe.

Pour créer en ligne de commande un pont avec deux interfaces, il faut commencer par créer le périphérique pont. Par convention, le premier périphérique d'un type donné se voit attribuer le numéro de séquence 0 : nous créons donc le périphérique `bridge0` avec la commande suivante :

```
$ sudo ifconfig bridge0 create
```

Avant la commande `brconfig` à venir, on vérifie à l'aide d'`ifconfig` que les futures interfaces membres (ici `ep0` et `ep1`) sont actives mais ne possèdent pas d'adresse IP.

On configure ensuite le pont avec la commande :

```
$ sudo brconfig bridge0 add ep0 add ep1 blocknonip ep0 blocknonip ep1 up
```

Sous OpenBSD, la commande `brconfig` intègre du code de filtrage et nous choisissons dans cet exemple l'option `blocknonip` afin que chaque interface bloque tout le trafic non IP.

> Sous OpenBSD, la commande `brconfig` offre son propre jeu d'options de filtrage en plus des options de configuration. Les pages de manuel `bridge(4)` et `brconfig(8)` donnent de plus amples informations. Notez bien que, comme elle opère au niveau Ethernet, `brconfig` est en mesure de filtrer sur les adresses MAC. `brconfig` peut également étiqueter les paquets pour traitement ultérieur dans les règles PF, via le mot-clé `tagged`.

Pour rendre la configuration permanente, on crée ou on édite le fichier `/etc/hostname.ep0` (`ep0` pour notre exemple, à adapter au besoin au nom de l'interface) pour y entrer la ligne suivante :

```
up
```

On procède de même pour l'autre interface avec le fichier `/etc/hostname.ep1` :

```
up
```

Enfin, on entre la configuration du bridge dans `/etc/bridgename.bridge0` :

```
add ep0 add ep1 blocknonip ep0 blocknonip ep1 up
```

Cela signifie que le pont est activé et que l'on peut passer à la création des règles de filtrage PF.

Configuration d'un pont simple sous FreeBSD

Pour FreeBSD, la procédure est un peu plus compliquée. Pour pouvoir utiliser un pont, le noyau en cours de fonctionnement doit inclure (ou pouvoir charger) le module `if_bridge`. Les configurations par défaut du noyau construisent ce module et donc, dans les circonstances normales, on peut passer directement à la création de l'interface.

Pour compiler le périphérique pont dans le noyau, on ajoute la ligne :

```
device if_bridge
```

au fichier de configuration du noyau. On peut aussi charger le périphérique au démarrage en plaçant la ligne :

```
if_bridge_load="YES"
```

dans le fichier `/etc/loader.conf`.

On crée le périphérique pont en tapant la commande suivante :

```
$ sudo ifconfig bridge0 create
```

La création de l'interface `bridge0` génère un ensemble de `sysctl` liés aux ponts :

```
$ sudo sysctl net.link.bridge
net.link.bridge.ipfw: 0
net.link.bridge.pfil_member: 1
net.link.bridge.pfil_bridge: 1
net.link.bridge.ipfw_arp: 0
net.link.bridge.pfil_onlyip: 1
```

Mieux vaut vérifier que ces valeurs `sysctl` sont bien disponibles : leur existence confirme l'activation du pont. Sinon, on doit revenir en arrière pour trouver l'endroit et la cause du problème. Cependant, ces valeurs s'appliquent au filtrage sur l'interface pont et nous ne devons donc pas y toucher, car le filtrage de niveau IP est activé par défaut sur les interfaces membres (les extrémités du tuyau).

Avant la commande `ifconfig` à venir, on doit vérifier que les futures interfaces membres (ici `ep0` et `ep1`) sont actives mais ne possèdent pas d'adresse IP.

On configure ensuite le pont à l'aide de cette commande :

```
$ sudo ifconfig bridge0 addm ep0 addm ep1 up
```

Pour rendre cette configuration permanente, on ajoute les lignes suivantes au fichier `/etc/rc.conf` :

```
ifconfig_ep0="up"
ifconfig_ep1="up"
cloned_interfaces="bridge0"
ifconfig_bridge0="addm ep0 addm ep1 up"
```

Cela signifie que le pont est activé et que l'on peut passer à la création des règles de filtrage PF. Voir la page de manuel `if_bridge(4)` pour de plus amples informations spécifiques à FreeBSD.

Configuration d'un pont simple sous NetBSD

Sous NetBSD, la configuration par défaut du noyau n'intègre pas la prise en charge du filtrage de pont. On doit donc compiler un noyau modifié en ajoutant l'option :

```
options         BRIDGE_IPF      # bridge uses IP/IPv6 pfil hooks too
```

au fichier de configuration. Une fois créé le noyau intégrant le code de pont, la configuration se fait de manière assez directe.

Pour créer un pont avec deux interfaces en ligne de commande, on commence pas créer le périphérique `bridge0` :

```
$ sudo ifconfig bridge0 create
```

Avant la commande `brconfig` à venir, on doit vérifier à l'aide de la commande `ifconfig` que les futures interfaces membres (ici `ep0` et `ep1`) sont actives mais ne possèdent pas d'adresse IP.

On configure ensuite le pont à l'aide de la commande :

```
$ sudo brconfig bridge0 add ep0 add ep1 up
```

On active ensuite le filtrage sur l'interface `bridge0` :

```
$ sudo brconfig bridge0 ipf
```

Pour rendre la configuration permanente, on doit créer ou éditer le fichier `/etc/ifconfig.ep0` (`ep0` pour notre exemple, adapter le nom de l'interface si besoin) pour y ajouter la ligne suivante :

```
up
```

Pour l'autre interface, on procède de même avec le fichier `/etc/ifconfig.ep1` :

```
up
```

Enfin, on ajoute la configuration du pont dans `/etc/ifconfig.bridge0` :

```
create
!add ep0 add ep1 up
```

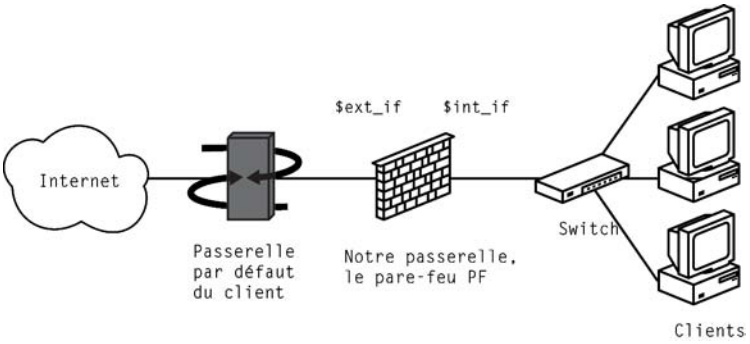
Cela signifie que le pont est activé et que l'on peut passer à la création des règles de filtrage PF.

Pour de plus amples informations, voir la documentation de NetBSD concernant PF.
- http://www.netbsd.org/Documentation/network/pf.html

Le jeu de règles du pont

Voici le fichier `pf.conf` de la version boîtier-sur-le-câble du jeu de règles de base avec lequel nous avons commencé ce chapitre. Le réseau change encore légèrement, pour ressembler à la figure 5.3.

Figure 5–3
Un réseau avec un pare-feu ponté.



Les machines du réseau local partagent une passerelle par défaut commune, qui n'est pas le pont, mais qui pourrait théoriquement se situer aussi bien à l'intérieur qu'àl'extérieur du pont.

```
ext_if = ep0
int_if = ep1
localnet= "192.0.2.0/24"
webserver = "192.0.2.227"
webports = "{ http, https }"
emailserver = "192.0.2.225"
email = "{ smtp, pop3, imap, imap3, imaps, pop3s }"
nameservers = "{ 192.0.2.221, 192.0.2.223 }"
client_out = "{ ssh, domain, pop3, auth, nntp, http, https, cvspserver, 2628, 5999,
8000, 8080 }"
udp_services = "{ domain, ntp }"
icmp_types = "{ echoreq, unreach }"
set skip on $int_if
block all
pass quick on $ext_if inet proto { tcp, udp } from $localnet to any port
$udp_services
pass log on $ext_if inet proto icmp all icmp-type $icmp_types
pass on $ext_if inet proto tcp from $localnet to any port $client_out
pass on $ext_if inet proto { tcp, udp } from any to $nameservers port domain
pass on $ext_if proto tcp from any to $webserver port $webports synproxy state
pass log on $ext_if proto tcp from any to $emailserver port $email synproxy state
pass log on $ext_if proto tcp from $emailserver to any port smtp synproxy state
```

Il est possible de monter des configurations significativement plus complexes. Souvenez-vous néanmoins que, même si les redirections fonctionneront, vous ne pourrez faire tourner aucun service sur les interfaces n'ayant pas d'adresse IP.

Gestion des adresses non routables venues d'ailleurs

Même avec une passerelle correctement configurée pour gérer le filtrage et l'éventuelle traduction d'adresses, il n'est pas rare de devoir compenser les mauvaises configurations des autres.

Laisser passer sur Internet du trafic comportant des adresses non routables est erreur tellement courante que cela en devient déprimant. Le trafic provenant d'adresses non routables est également utilisé dans diverses techniques d'attaque de déni de service (*Denial of Service*, DoS) : bloquer explicitement ce type de trafic à l'entrée de votre réseau est donc une bonne idée.

Voici l'une des solutions possibles ; pour bien faire, elle bloque toute tentative de connexion vers des des adresses non routables au travers de l'interface externe de la passerelle :

```
martiens = "{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, \
              10.0.0.0/8, 169.254.0.0/16, 192.0.2.0/24, \
              0.0.0.0/8, 240.0.0.0/4 }"

block in quick on $ext_if from $martiens to any
block out quick on $ext_if from any to $martiens
```

Ici, la macro `martiens` énumère les adresses RFC 1918 et quelques autres rangs interdits de circulation sur la partie publique d'Internet par diverses RFC. Le trafic en provenance et à destination de ces adresses est silencieusement abandonné au niveau de l'interface externe de la passerelle.

Les détails de mise en œuvre pour ce type de protection varieront, entre autres, selon la configuration du réseau. La conception de votre réseau pourrait, par exemple, exiger d'inclure ou d'exclure d'autres rangs que ceux-ci.

> Notez bien qu'au lieu d'une macro, on pourrait aisément utiliser une table pour représenter `martiens`.

Une défense pro-active : SSH et listes noires, grises et blanches

6

Au chapitre précédent, nous avons dépensé un temps et une énergie considérables à garantir la disponibilité des services que nous souhaitons proposer, même avec un filtrage strict. À présent que l'installation est fonctionnelle, vous remarquerez rapidement que certains services ont, malheureusement, tendance à attirer un peu plus l'attention que d'autres.

Dans ce chapitre, nous verrons comment exploiter les fonctionnalités intégrées de PF, telles que les tables et les options de conservation d'état, parfois en tandem avec des programmes de l'espace utilisateur, pour couper court à ce problème et construire un réseau plus fonctionnel.

Voici le scénario : nous avons mis en place un réseau avec filtrage de paquets pour répondre aux besoins du site. Pour rendre ce réseau réellement fonctionnel, nous l'avons doté de services devant être accessibles depuis l'extérieur : malheureusement, cette ouverture peut fort bien être exploitée à des fins malveillantes.

Deux services sont généralement disponibles sur un réseau : la connexion distante via le Protocole de Shell Sécurisé (*Secure Shell Protocol*, SSH) et SMTP (messagerie électronique). Tous deux sont des cibles tentantes pour les personnes malintentionnées. Dans la section qui suit, « Garder les méchants à distance », nous présenterons quelques solutions pour compliquer l'accès non autorisé via SSH, avant de nous pencher sur la protection de vos serveurs de courrier contre les spammeurs.

Garder les méchants à distance

Le service de Shell Sécurisé, qu'on appelle couramment SSH, est assez crucial pour les administrateurs Unix. C'est généralement l'interface principale avec une machine

et le fait qu'il soit souvent activé sur les systèmes *puissants* en a fait une cible de prédilection pour les *scripts kiddies*.

Si vous faites tourner un service de connexion par Shell Sécurisé sur une machine quelconque, je suis sûr que cet extrait des journaux d'authentification vous est familier :

```
Sep 26 03:12:34 skapet sshd[25771]: Failed password for root from 200.72.41.31 port
40992 ssh2
Sep 26 03:12:34 skapet sshd[5279]: Failed password for root from 200.72.41.31 port
40992 ssh2
Sep 26 03:12:35 skapet sshd[5279]: Received disconnect from 200.72.41.31: 11: Bye Bye
Sep 26 03:12:44 skapet sshd[29635]: Invalid user admin from 200.72.41.31
Sep 26 03:12:44 skapet sshd[24703]: input_userauth_request: invalid user admin
Sep 26 03:12:44 skapet sshd[24703]: Failed password for invalid user admin from
200.72.41.31
port 41484 ssh2
Sep 26 03:12:44 skapet sshd[29635]: Failed password for invalid user admin from
200.72.41.31
port 41484 ssh2
Sep 26 03:12:45 skapet sshd[24703]: Connection closed by 200.72.41.31
Sep 26 03:13:10 skapet sshd[11459]: Failed password for root from 200.72.41.31 port
43344 ssh2
Sep 26 03:13:10 skapet sshd[7635]: Failed password for root from 200.72.41.31 port
43344 ssh2
Sep 26 03:13:10 skapet sshd[11459]: Received disconnect from 200.72.41.31: 11: Bye
Bye
Sep 26 03:13:15 skapet sshd[31357]: Invalid user admin from 200.72.41.31
Sep 26 03:13:15 skapet sshd[10543]: input_userauth_request: invalid user admin
Sep 26 03:13:15 skapet sshd[10543]: Failed password for invalid user admin from
200.72.41.31
port 43811 ssh2
Sep 26 03:13:15 skapet sshd[31357]: Failed password for invalid user admin from
200.72.41.31
port 43811 ssh2
Sep 26 03:13:15 skapet sshd[10543]: Received disconnect from 200.72.41.31: 11: Bye
Bye
Sep 26 03:13:25 skapet sshd[6526]: Connection closed by 200.72.41.31
```

Je n'en copie pas l'intégralité car cela devient répétitif, mais c'est à cela que ressemble une attaque par force brute. Quelqu'un ou, plus probablement, un ordinateur piraté situé quelque part sur Internet, tente de trouver par force brute un nom d'utilisateur et un mot de passe donnant à l'attaquant un accès à votre système.

La réponse la plus simple consisterait à écrire une règle `pf.conf` qui bloquerait tous les accès. Ceci conduit toutefois à d'autres problèmes, par exemple pour autoriser l'accès aux personnes légitimes. Configurer `sshd` pour autoriser uniquement l'authentification par clé secrète pourrait être une solution, mais cela ne repousserait probablement pas

les tentatives incessantes des scripts kiddies. Déplacer le service vers un autre port est envisageable mais, là aussi, toute personne inondant de connexions le port 22 est tout à fait capable d'analyser tous vos ports jusqu'à trouver le bon.

Depuis OpenBSD 3.7 (ou FreeBSD 6.0), PF offre une solution légèrement plus élégante. Les règles `pass` peuvent être rédigées de manière à limiter les actions possibles pour les hôtes connectés.

Pour faire bonne mesure, on peut bannir les mal-élevés en les inscrivant dans une table d'adresses auxquelles on refuse tout ou partie des accès. On peut même choisir d'abandonner toutes les connexions existantes provenant des machines qui outrepassent leurs droits. Voici comment faire.

Commençons par installer la table, en ajoutant la ligne suivante dans la section des tables du fichier `/etc/pf.conf` :

```
table <bruteforce> persist
```

Établissons ensuite, assez tôt dans le jeu de règles, la règle qui bloque le trafic des brutes :

```
block quick from <bruteforce>
```

Ajoutons enfin la règle `pass` :

```
pass inet proto tcp from any to $localnet port $tcp_services \
     keep state (max-src-conn 100, max-src-conn-rate 15/5, \
         overload <bruteforce> flush global)
```

C'est assez similaire à ce que nous avons vu auparavant, n'est-ce pas ? En fait, la première partie est identique à la règle construite précédemment. Les éléments requérant une attention particulière se trouvent entre parenthèses : ce sont les *options de conservation d'état*. Elles servent à alléger la charge du réseau.

`max-src-conn` est le nombre de connexions simultanées autorisées depuis un hôte. Dans cet exemple, je l'ai fixé à `100`. Cependant, la valeur est à adapter au modèle de trafic du réseau.

`max-src-conn-rate` est le taux de nouvelles connexions autorisées depuis un hôte quelconque, ici `15` connexions par tranche de `5` secondes. Là encore, vous êtes seul juge de ce qui correspond à vos besoins.

`overload <bruteforce>` signifie que tout hôte qui dépasse ces limites voit son adresse ajoutée à la table `<bruteforce>`. Notre jeu de règles bloque tout le trafic provenant de ces adresses.

Avertissement

Un point important à remarquer : une fois qu'un hôte dépasse l'une de ces limites et se retrouve dans la liste des contrevenants, la règle ne s'applique plus au trafic provenant de cet hôte. Vous devez vous assurer que les contrevenants sont gérés au moins par une règle de blocage par défaut ou un procédé similaire.

`flush global` indique que, lorsqu'un hôte atteint la limite, la connexion de cet hôte est terminée (*vidangée*). La partie `global` indique que le `flush` s'applique également aux connexions correspondant aux autres règles `pass`.

L'effet est dramatique. Les attaquants se retrouvent généralement confrontés à un message `Fatal: timeout before authentication`, exactement ce que nous voulons.

Encore une fois, gardez à l'esprit que cette règle est essentiellement conçue comme un exemple et que les besoins de votre réseau peuvent fort bien être mieux traités par des règles (ou des combinaisons de règles) différentes.

Régler le nombres de connexions simultanées ou le taux de connexions à une valeur trop basse peut conduire à verrouiller le trafic légitime. Un tel scénario, avec des règles strictes sur le dépassement de la charge, comporte clairement un risque d'un auto-déni de service si la configuration comporte un grand nombre d'hôtes situés derrière une passerelle NAT commune et ayant des besoins de connexions externes légitimes.

Si, par exemple, on souhaite autoriser un grand nombre de connexions de manière générale, mais être un peu plus restrictif pour SSH, on peut ajouter quelques paramètres (suivant le modèle ci-dessous) à la règle sus-mentionnée et la placer tôt dans le jeu de règles :

```
pass quick proto { tcp, udp } from any to any port ssh \
     keep state (max-src-conn 15, max-src-conn-rate 5/3, \
        overload <bruteforce> flush global)
```

Pour trouver le jeu de paramètres le plus adapté à une situation donnée, consulter la page de manuel adéquate, ainsi que le Guide de l'Utilisateur PF (http://www.openbsd.org/faq/pf/fr/) – l'expérimentation peut aussi se révéler utile.

Il n'est pas obligatoire de bloquer tous ceux qui dépassent les bornes

Voici deux remarques importantes : d'une part, le mécanisme `overload` est une technique générale qui n'est donc pas limitée au service SSH ; d'autre part, bloquer tout le trafic des attaquants n'est pas toujours la solution.

Une règle `overload` peut servir à protéger un service de messagerie électronique ou un service web. Une table `overload` utilisée dans une règle permet d'assigner les

assaillants à une file d'attente dotée d'une bande passante minimale, comme nous le verrons au chapitre 7, dans la section « Diriger le trafic avec ALTQ ».

C'est également utile, dans le cas du Web, pour rediriger tout ou partie des requêtes HTTP vers une page web précise – un peu comme dans l'exemple d'`authpf` en fin de chapitre 4, section « Grand ouvert en apparence, mais complètement fermé en réalité ».

Nettoyer les tables avec pfctl

Pour le moment, nous avons des tables qui se remplissent grâce aux règles `overload` ; puisque notre passerelle doit rester en activité plusieurs mois d'affilée sans redémarrer, les tables grossiront adresse après adresse et occuperont un espace mémoire croissant.

Par ailleurs, il n'est pas rare de constater qu'une adresse IP bloquée à cause d'une attaque par force brute était en fait assignée dynamiquement, qu'elle est désormais assignée à un autre client du même FAI et que cette personne a des raisons légitimes de communiquer avec des hôtes du réseau.

Des situations de ce type nécessitent donc de pouvoir retirer de la table les entrées qui ne sont plus nécessaires. Dans OpenBSD 4.1, `pfctl` a acquis la capacité de faire *expirer* les entrées d'une table, en se basant sur le temps écoulé depuis la dernière remise à zéro de leurs statistiques.

Dans l'essentiel des cas, cela correspond au temps écoulé depuis la création de l'entrée dans la table. Le mot-clé est, sans surprise, `expire`, et l'âge de l'entrée dans la table est à préciser en secondes.

Par exemple, la commande :

```
# pfctl -t bruteforce -T expire 86400
```

retirera de la table `<bruteforce>` les entrées dont les statistiques ont été remises à zéro depuis plus de 86 400 secondes, soit 24 heures. Une entrée `crontab` peut être mise en place pour faire expirer les entrées des tables à intervalles réguliers, par exemple une fois par heure ou plusieurs fois par jour.

Le précurseur : expiretable

Avant que `pfctl` ne puisse faire expirer les entrées des tables, il existait un utilitaire prévu à cet effet : `expiretable`, écrit par Henrik Gustafsson. Ce programme remplit globalement la même fonction que la fonctionnalité `-T expire` de `pfctl` et il est surtout utile aux implémentations de PF basées sur OpenBSD 4.0 ou une version antérieure.

`expiretable` peut fonctionner en tant que démon qui retire les entrées de `<bruteforce>` datant de plus de 24 heures ; il faut alors ajouter au fichier `/etc/rc.local` une entrée du type :

```
/usr/local/sbin/expiretable -v -d -t 24h bruteforce
```

`expiretable` fut rapidement intégré à l'arbre des ports de FreeBSD et d'OpenBSD, respectivement en tant que `security/expiretable` et `sysutils/expiretable`.

Si `expiretable` n'est pas disponible via les paquetages de votre système, vous pouvez le télécharger depuis le site web de Henrik à l'adresse `http://expiretable.fnord.se`.

Embêter les spammeurs avec spamd

Un autre service qui requiert beaucoup d'attention est la messagerie électronique. Il s'agit de l'un des services Internet les plus anciens et personne ne voudrait avoir à s'en passer. Combinant SMTP et un protocole de récupération des messages, l'e-mail est l'un des services de base de n'importe quel réseau TCP/IP.

Au début des années 2000, l'Internet commercial a vu la montée du spam comme une menace envers SMTP. Diverses solutions de filtrage de contenu furent inventées, certaines *open source*. Même si un certain nombre d'entre elles fonctionnaient très bien sous BSD, l'équipe d'OpenBSD entreprit début 2003 d'élaborer sa propre solution de lutte contre le spam, nommée `spamd`. La première version de `spamd` fut diffusée comme partie intégrante d'OpenBSD 3.3 (lui-même publié le 1er mai 2003).

> En plus du démon anti-spam d'OpenBSD (basé sur l'idée de différer l'envoi effectif d'un message), le paquetage anti-spam SpamAssassin (basé sur du filtrage de contenu, voir http://spamassassin.apache.org) contient également un programme portant le nom `spamd`. Les deux programmes sont conçus pour aider à combattre le spam, mais ils représentent des approches très différentes du problème sous-jacent et ne sont pas directement interopérables. Ceci dit, s'ils sont correctement configurés sur un réseau, ils peuvent se révéler mutuellement complémentaires. Ils n'ont rien à voir entre eux et les développeurs ont bien veillé à installer leurs fichiers à des emplacements différents du système de fichiers ; il est donc possible, si le besoin s'en fait sentir, d'installer les deux programmes `spamd` sur le même système.

Le nouveau programme s'ancrait dans le filtre de paquets via un jeu de tables et de règles de redirection spécialement prévues. La conception de base est facile à comprendre et, en prenant appui sur ce que nous avons déjà appris de la construction

d'un jeu de règles PF, il ne devrait pas être difficile de comprendre les lignes `pf.conf` suivantes :

```
table <spamd> persist
table <spamd-white> persist
rdr pass on $ext_if inet proto tcp from <spamd> to \
         { $ext_if, $localnet } port smtp -> 127.0.0.1 port 8025
rdr pass on $ext_if inet proto tcp from !<spamd-white> to \
         { $ext_if, $localnet } port smtp -> 127.0.0.1 port 8025
```

Nous avons là deux tables portant des noms distincts. Pour le moment, contentons-nous de signaler cette distinction. Le point crucial est la redirection vers un démon (en écoute sur le port `8025`) de tout le trafic SMTP provenant des adresses qui figurent dans la première table et sont *absentes* de la seconde.

Souvenez-vous que vous n'êtes pas seul : la liste noire

L'élément principal qui sous-tend la conception originale de `spamd` est le grand nombre de messages envoyés par les spammeurs et la probabilité très faible que vous soyez le premier à recevoir l'un d'eux. De plus, l'envoi du spam intervient essentiellement depuis un nombre limité de réseaux tenus par des spammeurs et depuis un grand nombre de machines piratées. Les messages et les machines sont assez rapidement ajoutés aux listes noires, qui renseignent la première table de notre exemple.

Utilisation classique de spamd : les listes noires, le goudron et les plumes

En mode classique, `spamd` emploie une méthode appelée *tarpitting* (ce que l'on pourrait traduire par la formule « du goudron et des plumes »). Le démon présente sa bannière aux connexions SMTP provenant des adresses de sa liste noire, puis passe immédiatement dans un mode où il répond au trafic SMTP à la vitesse d'un octet à la fois : il fait ainsi perdre un maximum de temps à l'autre partie, alors que cela n'a quasiment aucun coût pour lui.

L'implémentation particulière des réponses SMTP à un octet est souvent appelée *shuttering* (obturation). Le *tarpitting-shuttering* basé sur des listes noires était le mode par défaut de `spamd` jusqu'à OpenBSD 4.0 compris.

La mise en place de `spamd` en mode traditionnel, c'est-à-dire en mode liste noire, se fait de manière plutôt directe. On commence par mettre en place les redirections et les définitions de tables dans le fichier `pf.conf`, puis on doit se pencher sur le fichier `spamd.conf`.

> Notez que, sous FreeBSD, `spamd` est un port, `mail/spamd/`. Si vous utilisez PF sous FreeBSD 5.x ou plus récent, vous devez installer le port et suivre les directives données par les messages du port avant de reprendre votre lecture.

Un fichier spamd.conf basique

Le fichier est, de base, bien documenté et la page de manuel donne des informations supplémentaires, mais nous allons récapituler ici l'essentiel.

> Sous OpenBSD 4.0 et les versions précédentes (et, par extension, les ports basés sur les versions antérieures à OpenBSD 4.1), `spamd.conf` se trouvait dans `/etc/`. Depuis OpenBSD 4.1, le fichier réside désormais dans le répertoire `/etc/mail/`.

Vers le début du fichier figure une ligne exempte de symbole `#` et contenant `all:\` : cette ligne définit les listes effectivement utilisées.

```
all:\
:uatraps:whitelist:
```

C'est là que l'on doit ajouter les listes noires (séparées par des symboles deux-points « `:` ») que l'on souhaite utiliser. Pour utiliser en complément des listes blanches (adresses à soustraire de la liste noire), on ajoute immédiatement après le nom de la liste noire concernée le nom de la liste blanche, par exemple `:listenoire:listeblanche:`.

Voici une définition de liste noire :

```
uatraps:\
        :black:\
        :msg="SPAM. Your address %A has sent spam within the last 24
hours":\
        :method=http:\
        :file=www.openbsd.org/spamd/traplist.gz
```

Juste après le nom de la liste figure le premier champ qui spécifie le type de liste, ici `black`. Le champ `msg` contient le message affiché, pendant le dialogue SMTP, aux expéditeurs placés en liste noire. Le champ `method` spécifie la manière dont le programme `spamd-setup` récupère les données de la liste, ici HTTP (`http`). Parmi les autres possibilités figurent FTP (`ftp`), l'utilisation d'un fichier résidant sur un système de fichiers monté ou l'exécution (`exec`) d'un programme externe. Enfin, le champ `file` précise le nom du fichier que `spamd` doit s'attendre à recevoir.

La définition d'une liste blanche (voir exemple ci-dessous) suit à peu près le même modèle, à l'exception du paramètre de message qui n'est pas nécessaire :

```
whitelist:\
        :white:\
        :method=file:\
        :file=/var/mail/whitelist.txt
```

> Choisissez vos sources de données avec précautions. Les listes noires suggérées dans le fichier `spamd.conf` par défaut peuvent exclure des pans entiers d'Internet, y compris des rangs d'adresses pouvant couvrir des pays entiers. Il va sans dire que, si votre site est censé échanger des messages électroniques légitimes avec l'un des pays en question, ces listes ne sont pas optimales. D'autres listes populaires sont connues pour lister des rangs `/16` entiers en tant que sources de spam. Il faut donc faire l'effort de lire scrupuleusement les détails de la politique de maintenance d'une liste noire avant de la mettre en production. C'est à vous de choisir les sources de données à utiliser ; vous n'êtes pas limité à celles proposées par défaut.

On ajoute au fichier `/etc/rc.conf` ou `/etc/rc.conf.local` les lignes et les paramètres de démarrage souhaités pour `spamd`. Par exemple, la ligne :

```
spamd_flags="-v" # for normal use: "" and see spamd-setup(8)
```

active `spamd` en mode liste noire, sous OpenBSD 4.0 et les versions antérieures. Le drapeau `-v` active la journalisation en mode « verbeux » (c'est-à-dire détaillé), qui est utile pour tracer l'activité de `spamd` à des fins de débogage.

> Si `spamd` doit tourner en mode liste noire pur, sans liste grise (décrite à la section suivante), sous OpenBSD 4.1 ou plus récent, on règle la variable `spamd_black` sur la valeur `YES` avant de le redémarrer pour désactiver le mode liste grise et activer le mode liste noire seule.

Une fois le fichier édité, on lance `spamd` avec les options souhaitées et on finalise la configuration en utilisant `spamd-setup`. Enfin, créer une une entrée `cron` qui appelle `spamd-setup` permet de mettre à jour les tables à des intervalles raisonnables.

Par défaut, `spamd` envoie ses entrées de journal dans les journaux généraux du système. Si celles-ci doivent être redirigées vers un fichier distinct (par exemple pour limiter le désordre dans les journaux système), il faut ajouter dans le fichier `syslog.conf` une entrée de ce type :

```
!!spamd
daemon.err;daemon.warn;daemon.info                    /var/log/spamd
```

Une fois que `spamd` fonctionne de manière satisfaisante, il est judicieux d'ajouter son fichier journal au système de rotation des journaux.

Après avoir lancé `spamd-setup` et rempli les tables, nous pouvons visualiser leur contenu en passant par `pfctl` ou d'autres applications. Pour modifier ou supprimer des entrées, il est conseillé d'employer l'utilitaire `spamdb` au lieu des fonctionnalités de gestion des tables de `pfctl` (nous en reparlerons plus tard).

Remarquez que, dans le fragment de `pf.conf` donné en exemple plus haut, les règles de redirection (`rdr`) sont également des règles `pass`. Si vos règles `rdr` n'incluent pas une partie `pass`, vous devez créer des règles `pass` qui laisseront passer le trafic vers les cibles de redirection. Vous devez également créer des règles afin de laisser circuler les e-mails légitimes. S'il existe déjà un service de messagerie électronique sur votre réseau, vous pouvez probablement utiliser vos anciennes règles `pass` SMTP.

Le mode liste noire pur n'existe plus que pour des raisons historiques. Un jeu de listes noires fiables et correctement maintenues tient occupées les machines connues pour envoyer du spam, ce qui est une bonne chose. Mais le véritable gain dans la lutte contre le spam vient des listes grises, qui constituent la base du fonctionnement du `spamd` moderne.

La liste grise : mon admin m'a interdit de parler aux étrangers

Le principe de liste grise (*greylisting*) consiste principalement à interpréter les standards SMTP actuels de manière plutôt stricte et à y ajouter un petit mensonge afin de se simplifier la vie.

Les spammeurs tendent à utiliser les équipements d'autrui pour envoyer leurs messages ; les logiciels qu'ils installent sans la permission du propriétaire légitime doivent être relativement légers, pour ne pas être détectés. L'expérience tend également à montrer que les spammeurs ne considèrent pas leurs messages comme importants. Si l'on considère tous ces éléments mis bout à bout, on en déduit que les logiciels de spam typiques ne sont probablement pas armés pour interpréter correctement les codes d'état SMTP.

Nous pouvons utiliser cela à notre avantage, comme l'a montré Evan Harris dans un article datant de 2003[1]. L'idée principale est que, sur une machine compromise, l'application d'envoi de spam n'essaie généralement qu'une seule fois de transmettre

1. L'article original de Harris, ainsi que d'autres documents et ressources utiles, sont disponibles à l'adresse http://www.greylisting.org. »

le message, sans chercher à vérifier le résultat ou le code de retour. Les véritables implémentations de SMTP interprètent les codes de retour SMTP et agissent en conséquence, et les véritables serveurs mail réessaient si la première tentative échoue pour cause d'erreur temporaire. La conception initiale et les premiers résultats de tests semblant prometteurs, un certain nombre d'implémentations de listes grises sont apparues dans les mois suivant la publication de l'article.

Même si les services Internet sont proposés sans garantie (ce que l'on décrit habituellement par la formule *best-effort services*, *service qui fait de son mieux*), un effort significatif de conception et de développement a été fourni pour rendre les services essentiels (tels que l'émission de messages électroniques SMTP) résistants aux pannes. En pratique, cela signifie qu'un service du type SMTP fait de son mieux pour transmettre des messages. C'est la raison qui justifie que l'on s'appuie sur le système de liste grise pour recevoir malgré tout les e-mails de la part des serveurs de messagerie électronique.

Le standard actuel pour l'émission est défini dans la RFC 2821. Dans la section 4.5.4.1 de ce document, titrée « Stratégie d'envoi », nous trouvons :

Dans un système typique, le programme qui compose un messager possède plusieurs méthodes pour demander une attention immédiate face à un nouveau message sortant, mais les messages ne pouvant être émis immédiatement DOIVENT être mis en file d'attente et l'expéditeur doit tenter de les renvoyer périodiquement.

et :

Quand une tentative a échoué, l'expéditeur DOIT respecter un certain délai avant de réessayer de contacter l'hôte de destination. En général, l'intervalle entre deux tentatives DEVRAIT être d'au moins 30 minutes ; cependant, des stratégies variables et plus sophistiquées sont gagnantes quand le client SMTP peut déterminer la raison de la non-remise.

La RFC 2821 avance ensuite en déclarant :

Il faut continuer à réessayer jusqu'à ce que le message soit transmis ou que l'expéditeur abandonne ; le temps au bout duquel on abandonne doit généralement être d'au moins 4 à 5 jours.

Pour résumer : le transfert d'un message électronique est un processus collaboratif qui fait de son mieux et la RFC déclare clairement que, si le site auquel vous essayez d'envoyer un message répond qu'il ne peut rien recevoir pour le moment, il est de votre *devoir* d'essayer plus tard, lui laissant ainsi une chance de se sortir de ses problèmes.

L'idée intelligente derrière le système de liste grise réside dans un petit mensonge bénin et pratique. Prétendre avoir un problème passager est en fait tout à fait équivalent à dire « *Mon admin m'a interdit de parler aux étrangers.* ». Les expéditeurs bien élevés, qui envoient des messages valides, reviendront plus tard ; en revanche, les

spammeurs n'ont aucun intérêt à attendre pour retenter leur chance, car cela augmenterait le coût d'expédition de leurs messages.

Voilà pourquoi l'idée des listes grises fonctionne toujours. Et, comme elle repose au fond sur le respect à la lettre des standards acceptés, les faux positifs sont vraiment rares.

> **RFC RFC 1123 et RFC 2821 pour les listes grises**
>
> Les RFC concernées sont principalement la RFC 1123 et la RFC 2821. Si vous choisissez de rejoindre le club des « bornés » qui respectent les standards à la lettre en utilisant des listes grises, vous devrez les lire, ne serait-ce que pour obtenir des informations générales sur le style de ces RFC. Souvenez-vous, le refus temporaire est en fait une fonctionnalité SMTP de tolérance aux pannes.

Le `spamd` d'OpenBSD gère les listes grises depuis OpenBSD 3.5, qui fut publié en mai 2004. Depuis OpenBSD 4.1, publié le 1er mai 2007, `spamd` tourne par défaut en mode liste grise.

Le plus incroyable à propos des listes grises, si on met de côté leur simplicité, c'est qu'elles fonctionnent toujours. Les spammeurs et les auteurs de *malwares* ont mis du temps à s'adapter. Nous verrons quelques exemples plus tard.

Mettre en place spamd en mode liste grise

Une fois que les règles nécessaires ont été mises en place dans le fichier `pf.conf`, la configuration de `spamd` pour le mode liste grise est assez triviale.

> **Sous FreeBSD, spamd a besoin de fdescfs**
>
> Remarquez que, pour utiliser `spamd` en mode liste grise sous FreeBSD, on doit disposer d'un descripteur de système de fichiers (voir `man 5 fdescfs`) monté dans `/dev/fd/`. Pour ce faire, on peut ajouter la ligne suivante au fichier `/etc/fstab` :
> `fdescfs /dev/fd fdescfs rw 0 0`
> Le code de `fdescfs` est disponible en tant que module noyau dans la configuration par défaut mais, si vous utilisez un noyau personnalisé, vous devrez peut-être vérifier que le module est disponible ou que le code y est compilé.

Commençons par placer les lignes concernant `spamd`, accompagnées des paramètres de démarrage, dans le fichier `/etc/rc.conf` ou `/etc/rc.conf.local`, par exemple :

```
spamd_flags="-v -G 2:4:864" # pour utilisation normale : "" et voir
spamd-setup(8)
spamd_grey=YES              # spamd en mode liste grise si YES
```

Encore une fois, la variable `spamd_grey` est superflue sous OpenBSD 4.1 et les versions plus récentes, puisque le mode liste grise est le mode par défaut.

> **RAPPEL Désactiver le mode liste grise**
>
> Comme nous l'avons mentionné plus haut dans la section « Un fichier spamd.conf basique », vous pouvez utiliser la variable `spamd_black` pour désactiver le mode liste grise.

Notez que vous pouvez personnaliser plusieurs paramètres liés au mode liste grise en utilisant la commande `spamd` : il vous suffit d'y ajouter l'option `-G`, suivie des paramètres, en ligne de commande. Cette liste d'éléments séparés par des symboles deux-points (`:`) se compose des valeurs `passtime`, `greyexp` et `whiteexp`. Ici, `passtime` décrit le nombre minimal de minutes que `spamd` considère comme intervalle raisonnable avant une nouvelle tentative. Par défaut, c'est 25 minutes, mais nous l'avons réduit à 2 minutes dans l'exemple. `greyexp` et `whiteexp` se mesurent en heures : `greyexp` est la durée de maintien d'une entrée dans la liste grise ; `whiteexp` est la durée de maintien d'une entrée en liste blanche. Les valeurs par défaut sont respectivement 4 et 864 heures (soit un peu plus d'un mois).

Suivre à la trace vos véritables connexions de messagerie : spamlogd

Rarement mentionné et peu documenté, `spamlogd` est l'un des plus importants programmes secondaires de `spamd`, chargé de mettre à jour la liste blanche. Comme son suffixe de démon le suggère, `spamlogd` travaille silencieusement et en retrait. Il enregistre les connexions journalisées en provenance et à destination des serveurs mail, afin de garder la liste blanche à jour. L'idée est de maintenir la fluidité des échanges vis-à-vis des hôtes avec lesquels vous communiquez régulièrement.

> Redémarrez `spamd` pour activer le mode liste grise. Si vous avez suivi jusqu'ici la progression naturelle, il est fort probable que `spamlogd` ait déjà été lancé automatiquement. Mais si votre configuration initiale n'incluait pas de liste grise, `spamlogd` n'est pas nécessairement lancé et vous verrez peut-être apparaître des symptômes étranges, comme les listes grises ou blanches qui ne se mettent pas à jour correctement. Dans des circonstances normales, vous ne devriez pas à avoir à démarrer `spamlogd` à la main. Redémarrer `spamd` après avoir activé le mode liste grise garantit que `spamlogd` est lui aussi chargé et disponible.

Pour pouvoir faire correctement son travail, spamlogd a besoin que soient journalisées les connexions en provenance et à destination de vos serveurs de messagerie électronique, comme dans le jeu de règles donné précédemment en exemple :

```
emailserver = "192.0.2.225"
pass log proto tcp from any to $emailserver port $email synproxy state
pass log proto tcp from $emailserver to any port smtp synproxy state
```

Sous OpenBSD 4.1 et versions ultérieures (et les systèmes équivalents), on peut créer plusieurs interfaces pflog et spécifier l'interface vers laquelle une règle doit journaliser. Si l'on souhaite séparer les données nécessaires à spamlogd du reste des journaux PF, on crée une interface pflog1 à part, en utilisant ifconfig pflog1 create ou en créant un fichier /etc/hostname.pflog1 contenant uniquement la ligne up. En changeant les lignes suivant ce modèle :

```
pass log (to pflog1) proto tcp from any to $emailserver port $email
pass log (to pflog1) proto tcp from $emailserver to any port smtp
```

en ajoutant -l pflog1 aux paramètres de démarrage de spamlogd, on sépare la journalisation liée à spamd du reste. Voir le chapitre 8 pour plus d'informations sur la journalisation.

Avec ces règles, spamlogd ajoutera à la liste blanche les adresses IP qui reçoivent les e-mails que vous envoyez. Cela ne garantit pas formellement que la réponse passera immédiatement mais, dans la plupart des configurations, cela permet d'accélérer significativement les choses.

Intervention manuelle avec spamdb

Vous pouvez être amené à visualiser ou modifier le contenu de vos listes blanches, grises ou noires. Ces enregistrements sont situés dans la base de données /var/db/spamdb et l'interface principale d'un administrateur pour gérer ces listes est spamdb.

Les premières versions de spamdb offraient simplement des options pour ajouter ou modifier des entrées de liste blanche à la base de données (spamdb -a nn.mm.nn.mm) et pour supprimer des entrées de liste blanche (spamdb -d nn.mm.nn.mm), ceci afin de compenser des défauts dans les listes noires utilisées ou dans les effets des algorithmes de liste grise.

spamdb a fait l'objet de développements intéressants en termes de fonctionnalités de piège par liste grise. Nous reviendrons d'ici peu sur le piège par liste grise et d'autres avancées récentes, mais voyons d'abord quelques rapports de terrain sur le comportement de spamd.

Quelques points importants sur l'usage quotidien de spamd

Que donne `spamd` en conditions réelles ? Les utilisateurs et administrateurs qui implémentent le système de liste grise sur leurs sites ont tendance à s'accorder sur le fait qu'il élimine la plupart de leurs spams. Nous commencerons par en étudier les effets au niveau des fichiers de journalisation, puis nous verrons quelques données.

Si l'on démarre `spamd` avec l'option de ligne de commande `-v` (journalisation détaillée), les journaux incluront davantage d'informations que les seules adresses IP. Avec la journalisation détaillée, un fichier journal ressemble à ceci :

```
Oct 2 19:53:21 delilah spamd[26905]: 65.210.185.131: connected (1/1),
lists: spews1
Oct 2 19:55:04 delilah spamd[26905]: 83.23.213.115: connected (2/1)
Oct 2 19:55:05 delilah spamd[26905]: (GREY) 83.23.213.115:
<gilbert@keyholes.net> ->
wkitp98zpu.fsf@datadok.no>
Oct 2 19:55:05 delilah spamd[26905]: 83.23.213.115: disconnected after 0
seconds.
Oct 2 19:55:05 delilah spamd[26905]: 83.23.213.115: connected (2/1)
Oct 2 19:55:06 delilah spamd[26905]: (GREY) 83.23.213.115:
<gilbert@keyholes.net> ->
<wkitp98zpu.fsf@datadok.no>
Oct 2 19:55:06 delilah spamd[26905]: 83.23.213.115: disconnected after 1
seconds.
Oct 2 19:57:07 delilah spamd[26905]: (BLACK) 65.210.185.131: <bounce-
3C7E40A4B3@branch15.summerbargainz.
com> -> <adm@dataped.no>
Oct 2 19:58:50 delilah spamd[26905]: 65.210.185.131: From: Auto
Insurance Savings
<noreply@branch15.summer-bargainz.com>
Oct 2 19:58:50 delilah spamd[26905]: 65.210.185.131: Subject: Start
SAVING MONEY on Auto
Insurance
Oct 2 19:58:50 delilah spamd[26905]: 65.210.185.131: To: adm@dataped.no
Oct 2 20:00:05 delilah spamd[26905]: 65.210.185.131: disconnected after
404 seconds. lists:
spews1
Oct 2 20:03:48 delilah spamd[26905]: 222.240.6.118: connected (1/0)
Oct 2 20:03:48 delilah spamd[26905]: 222.240.6.118: disconnected after 0
seconds.
Oct 2 20:06:51 delilah spamd[26905]: 24.71.110.10: connected (1/1),
lists: spews1
Oct 2 20:07:00 delilah spamd[26905]: 221.196.37.249: connected (2/1)
Oct 2 20:07:00 delilah spamd[26905]: 221.196.37.249: disconnected after
0 seconds.
Oct 2 20:07:12 delilah spamd[26905]: 24.71.110.10: disconnected after 21
seconds. lists: spews1
```

La première ligne correspond au début de la connexion d'une machine appartenant à la liste noire `spews1`. Les six lignes suivantes montrent l'enregistrement complet de deux tentatives de connexion réalisées par une autre machine, affichées toutes deux comme étant la deuxième connexion active. Cette seconde machine n'est pas encore en liste noire, elle est donc mise en liste grise.

> **Astuce**
>
> Remarquez l'adresse de destination plutôt curieuse (`wkitp98zpu.fsf@datadok.no`) du message que la machine en liste grise essaie d'expédier ici. Il y a une astuce bien utile que nous verrons plus bas à la section « Construire sa propre liste piège ».

Le `(GREY)` ou `(BLACK)` précédant les adresses indique le statut de liste grise ou noire, respectivement. La machine en liste noire engendre ensuite davantage d'activité et nous voyons un peu plus bas qu'elle abandonne, sans avoir accompli sa tâche, après 404 secondes (soit 6 minutes et 44 secondes). Les lignes restantes montrent quelques connexions très courtes, dont une provenant d'une machine déjà placée en liste noire. Cependant, la machine se déconnecte cette fois-ci trop rapidement pour que nous puissions voir le moindre drapeau `(BLACK)` au début du dialogue SMTP ; nous voyons toutefois une référence au nom de la liste (`spews1`) à la fin.

À en croire les données de divers sites web, il faut en moyenne 400 secondes pour qu'un expéditeur de spam naïf se retrouve en liste noire. Cela correspond également, à peu de choses près, au temps requis – à la vitesse d'un octet par seconde – pour achever le dialogue `MAIL TO:` avant que `spamd` ne rejette le message vers la file d'attente de l'expéditeur. Mais si vous cherchez bien dans vos journaux, vous trouverez probablement des exemples significativement plus longs. Voici un exemple provenant de la passerelle de mon lieu de travail, qui a pris plus de temps :

```
Dec 11 23:57:24 delilah spamd[32048]: 69.6.40.26: connected (1/1),
lists: spamhaus spews1
spews2
Dec 12 00:30:08 delilah spamd[32048]: 69.6.40.26: disconnected after
1964 seconds. lists:
spamhaus spews1 spews2
```

Cette machine précise apparaissait déjà sur plusieurs listes noires quand elle a tenté à 13 reprises d'expédier des messages, entre le 9 et le 12 décembre 2004. La dernière tentative a pris 32 minutes et 44 secondes, et l'expédition du message ne s'est pas achevée. La plupart des connexions sont cependant plus courtes que cela. Les expéditeurs de spam relativement intelligents laissent tomber la connexion dès les toutes premières secondes (comme la première entrée du journal donné en exemple).

D'autres abandonnent au bout d'environ 400 secondes, tandis qu'un tout petit nombre reste accroché pendant des heures[1].

De nos jours, la plupart des sites disposent d'une solution de filtrage de contenu afin de gérer le spam et les logiciels malveillants transportés par e-mail. Les sites qui complètent leur configuration par un `spamd` sur leur passerelle voient chuter de façon significative la charge des machines dédiées au filtrage de contenus.

Fondamentaux pour des listes grises efficaces (greytrapping)

Dans la première moitié de l'année 2005, c'est-à-dire au moment où démarrait le cycle de développement d'OpenBSD 3.8, les utilisateurs et les développeurs de `spamd` avaient accumulé suffisamment de données et d'expériences sur le comportement des spammeurs et leurs réactions face aux contre-mesures.

Nous savons déjà que les expéditeurs de spam utilisent rarement des outils entièrement conformes aux spécifications du protocole SMTP – c'est la raison pour laquelle le système de liste grise fonctionne. De plus, et comme nous l'avons déjà noté, non seulement les spammeurs envoient des messages en grand nombre, mais ils vérifient aussi très rarement la validité des adresses qu'ils fournissent à leurs machines piratées. Combinez ces deux éléments et vous verrez que, si une machine en liste grise essaie d'envoyer un message à une adresse invalide de votre domaine, il y a une forte probabilité que le message soit un spam ou un programme malveillant.

Le greytrapping, concrètement

Et c'est ainsi que `spamd` a dû apprendre le *greytrapping*, c'est-à-dire un piège basé sur une liste grise. Je trouve que l'implémentation du greytrapping dans `spamd` est plutôt élégante. Pour commencer, nous avons besoin d'un `spamd` en mode liste grise. L'autre composant crucial est une liste d'adresses qui, d'une part, appartiennent aux domaines gérés par le serveur mail et, d'autre part, qui ne recevront jamais le moindre message légitime. Il doit y en avoir au moins une et la limite haute est surtout définie par le nombre d'adresses que vous avez envie d'ajouter.

Ensuite, on utilise `spamdb` pour nourrir la liste et il ne reste plus qu'à profiter du spectacle. Au premier contact, un expéditeur qui tente d'envoyer un e-mail à une adresse

1. Le cas le plus extrême que nous ayons enregistré est resté 42 673 secondes, ce qui représente presque 12 heures. Voir l'annexe A pour des références vers d'autres publications et plus de données.

appartenant à votre liste de greytrapping se voit simplement ajouté à la liste grise, comme tous les autres avec qui nous n'avons encore jamais échangé de message.

Si la même machine effectue une nouvelle tentative, vers la même adresse invalide ou vers une autre adresse de la liste, alors le piège se déclenche. L'attaquant est placé dans une liste noire temporaire, nommée `spamd-greytrap`, pour une durée de 24 heures. Pour les 24 heures suivantes, tout trafic SMTP provenant de l'hôte pris au piège se verra répondre au rythme d'un octet à la fois.

Une durée de 24 heures est suffisamment courte pour ne pas causer d'interruption sérieuse de trafic légitime, puisque les véritables implémentations de SMTP continueront à tenter de transmettre le message pendant quelques jours au moins.

L'utilisation à grande échelle de cette technique montre qu'elle ne produit que très rarement, voire jamais, de faux positifs. Les machines qui continuent à spammer après 24 heures se verront rapidement passer à nouveau par le goudron et les plumes.

La liste-piège *ghosts of usenet postings past* de Bob Beck, est un excellent exemple de ce qu'on peut accomplir avec du greytrapping. Cette liste est générée automatiquement par des ordinateurs hébergeant `spamd`, à l'Université d'Alberta (Canada) ; elle contient rarement moins de 20 000 adresses IP. Le nombre d'hôtes varie grandement et il est monté jusqu'à environ 198 000. Au moment de l'écriture de ce livre (novembre 2007), la liste contenait environ 110 000 entrées. Bien que toujours officiellement en phase de test, elle a été rendue publique le 30 janvier 2006. À ma connaissance, la liste n'a jusqu'ici produit aucun faux positif et elle est disponible à l'adresse http://www.openbsd.org/spamd/traplist.gz pour utilisation dans votre fichier `spamd.conf`.

> **PRATIQUE** **Liste à jour**
>
> Cette liste est citée dans les fichiers d'exemple `spamd.conf` récents sous le nom de liste noire `uatraps`. En plus de cette liste, Bob recommande d'utiliser la liste `nixspam` de heise.de, qui figure également dans le fichier d'exemple `spamd.conf` et qui est générée à partir de diverses sources, avec une durée d'expiration de quatre jours. Le site web de Heise contient des informations détaillées à propos de cette liste.
>
> ▸ http://www.heise.de/ix/nixspam/dnsbl_en

Construire sa propre liste-piège

Pour construire une liste-piège depuis zéro, utilisez l'option `-T` de `spamdb`. Dans mon cas, l'adresse étrange que j'ai mentionnée plus haut était un candidat naturel :

```
$ sudo spamdb -T -a wkitp98zpu.fsf@datadok.no
```

SYNTAXE Chevrons et guillemets facultatifs pour spamdb sous OpenBSD 4.1

La commande lancée était en fait :

```
$ sudo spamdb -T -a "<wkitp98zpu.fsf@datadok.no>"
```

Sous OpenBSD 4.1 et ses versions plus récentes, `spamdb` n'exige plus ni les chevrons ni les guillemets, mais il les acceptera si vous les utilisez.

Cette adresse est complètement erronée. J'utilise le client de messagerie et de nouvelles GNUS et cela ressemble fort aux identifiants de message (*message-ID*) que génère le programme. Cet identifiant de message a probablement été extrait d'un groupe de nouvelles ou de la boîte de réception d'une quelconque victime d'un logiciel malveillant. Mais le spammeur a probablement supposé que cet identifiant restait utilisable, presque deux ans après. Comme vous le verrez, cette adresse de destination est recyclée.

```
Nov 6 09:50:25 delilah spamd[23576]: 210.214.12.57: connected (1/0)
Nov 6 09:50:32 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov 6 09:50:40 delilah spamd[23576]: (GREY) 210.214.12.57: <gilbert@keyholes.net> ->
<wkitp98zpu.fsf@datadok.no>
Nov 6 09:50:40 delilah spamd[23576]: 210.214.12.57: disconnected after 15 seconds.
Nov 6 09:50:42 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov 6 09:50:45 delilah spamd[23576]: (GREY) 210.214.12.57: <bounce-
3C7E40A4B3@branch15.summerbargainz.
com> -> <adm@dataped.no>
Nov 6 09:50:45 delilah spamd[23576]: 210.214.12.57: disconnected after 13 seconds.
Nov 6 09:50:50 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov 6 09:51:00 delilah spamd[23576]: (GREY) 210.214.12.57: <gilbert@keyholes.net> ->
<wkitp98zpu.fsf@datadok.no>
Nov 6 09:51:00 delilah spamd[23576]: 210.214.12.57: disconnected after 18 seconds.
Nov 6 09:51:02 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov 6 09:51:02 delilah spamd[23576]: 210.214.12.57: disconnected after 12 seconds.
Nov 6 09:51:02 delilah spamd[23576]: 210.214.12.57: connected (2/0)
Nov 6 09:51:18 delilah spamd[23576]: (GREY) 210.214.12.57: <gilbert@keyholes.net> ->
<wkitp98zpu.fsf@datadok.no>
Nov 6 09:51:18 delilah spamd[23576]: 210.214.12.57: disconnected after 16 seconds.
Nov 6 09:51:18 delilah spamd[23576]: (GREY) 210.214.12.57: <bounce-
3C7E40A4B3@branch15.summerbargainz.
com> -> <adm@dataped.no>
Nov 6 09:51:18 delilah spamd[23576]: 210.214.12.57: disconnected after 16 seconds.
Nov 6 09:51:20 delilah spamd[23576]: 210.214.12.57: connected (1/1), lists: spamd-greytrap
Nov 6 09:51:23 delilah spamd[23576]: 210.214.12.57: connected (2/2), lists: spamd-greytrap
Nov 6 09:55:33 delilah spamd[23576]: (BLACK) 210.214.12.57: <gilbert@keyholes.net> ->
<wkitp98zpu.fsf@datadok.no>
Nov 6 09:55:34 delilah spamd[23576]: (BLACK) 210.214.12.57: <bounce-
3C7E40A4B3@branch15.summer-bargainz.com> -> <adm@dataped.no>
```

Ce fragment de journal montre que la machine du spammeur est mise en liste grise au premier contact, puis essaie maladroitement d'expédier un message à l'adresse ajoutée à ma liste-piège, avant de terminer dans la liste noire de `spamd-greytrap` après quelques minutes. Nous savons désormais ce qu'il fera pendant la prochaine vingtaine d'heures.

En corollaire, il semblerait que, même si le spammeur change de machine pour envoyer ses messages, les adresses `From:` et `To:` restent les mêmes. Le fait qu'il essaie toujours d'émettre vers une adresse qui n'a jamais pu recevoir de message indique fortement que ce spammeur ne vérifie pas souvent ses listes. Au moment où vous lirez ces lignes, il serait intéressant de voir si le domaine de l'expéditeur est toujours enregistré. À l'heure où j'écris ces lignes, il ne reçoit aucun message électronique et il est clairement marqué comme étant à vendre.

Supprimer et gérer les entrées piégées

`spamdb` propose quelques autres options qu'il est bon de connaître. L'option `-T`, combinée à `-d`, vous permet de supprimer des adresses e-mail de la liste-piège, tandis que l'option `-t` (en minuscules), combinée à `-a` ou `-d`, vous permet d'ajouter ou supprimer des adresses IP piégées dans la base de données.

L'export de vos listes d'adresses piégées ne demande que de combiner `spamdb`, `grep` et un peu d'imagination sur une une seule ligne.

Synchroniser plusieurs listes grises spamd

À partir d'OpenBSD 4.1, `spamd` est capable de synchroniser les bases de données de listes grises entre un nombre quelconque de passerelles de listes grises qui coopèrent entre elles.

La mise en œuvre intervient via un ensemble d'options de ligne de commande `spamd` : l'option `-Y` spécifie une *cible de synchronisation*, c'est-à-dire les adresses IP des autres passerelles `spamd` que vous souhaitez informer des mises à jour de vos listes grises. Par ailleurs, l'option `-y` spécifie un *récepteur de synchronisation*, qui est l'adresse ou l'interface où cette instance de `spamd` est prête à recevoir les mises à jour de liste grise de la part des autres hôtes.

La commande de démarrage de notre passerelle `spamd` principale, `mainoffice-gw.example.com`, peut se voir ajouter les options suivantes pour établir respectivement une cible de synchronisation et un récepteur de synchronisation :

```
-Y minorbranch-gw.example.com -y mainoffice-gw.example.com
```

Inversement, `minorbranch-gw.example.com`, qui se trouve dans un autre bureau, verrait les noms d'hôtes inversés, comme ceci :

```
-Y mainoffice-gw.example.com -y minorbranch-gw.example.com
```

Remarquez que `spamd` gère également une authentification à clé partagée entre les partenaires de synchronisation. Si vous créez le fichier `/etc/mail/spamd.key` et que vous en distribuez des copies à tous les partenaires de synchronisation, son contenu sera utilisé pour calculer les sommes de contrôles nécessaires à l'authentification. Le fichier en lui-même peut correspondre à n'importe quel type de données, par exemple des données aléatoires tirées de `/dev/arandom`, comme le suggère la page de manuel de `spamd`.

Détecter l'utilisation de MX qui ne fonctionne pas

Une autre fonctionnalité sympathique et introduite dans OpenBSD 4.1 est la possibilité pour `spamd` de détecter l'utilisation de MX qui ne fonctionne pas. Contacter un échangeur de messagerie secondaire avant d'avoir essayé l'échangeur principal est une astuce de spammeurs assez connue et elle va à l'encontre du comportement attendu de la part d'un agent de transfert ordinaire.

En d'autres termes, si quelqu'un essaie de contacter les échangeurs de messagerie dans le mauvais ordre, il est presque certain que cette personne tente d'envoyer du spam. Reprenons le cas de notre domaine `example.com`, où le serveur mail principal se trouve à l'adresse IP `192.0.2.225` et le serveur de secours à l'adresse `192.0.2.224`. Ajouter `-M 192.0.2.224` aux options de démarrage de `spamd` fera en sorte que tout hôte essayant de contacter `192.0.2.224` via SMTP avant d'avoir contacté le serveur mail principal (`192.0.2.225`) sera ajouté à la liste `spamd-greytrap` locale et donc puni pour 24 heures.

Gérer les sites qui ne se comportent pas bien avec le système de listes grises

Malheureusement, il y a des situations où vous devez compenser les problèmes des configurations de messagerie électronique des autres. Nous savons déjà que le système de liste grise fonctionne essentiellement parce que les infrastructures de messagerie conformes aux standards doivent obligatoirement réessayer après un temps *raisonnable*. Cependant, et comme Murphy serait trop heureux de vous le rappeler, la vie n'est pas toujours aussi simple.

Quand un site qui ne vous a jamais contacté vous envoie un message, et tant que son adresse apparaît dans la liste grise, ce message est retardé pour une durée aléatoire, qui dépend principalement du délai au bout duquel l'expéditeur réitère sa tentative.

Dans certaines circonstances, un délai même minime n'est pas acceptable. Si, par exemple, vous avez des clients qui demandent une attention immédiate et urgente quand ils vous contactent, un délai initial pouvant atteindre quelques heures n'est pas une solution acceptable.

De plus, vous rencontrerez peut-être des serveurs mail mal configurés qui ne retentent jamais leur chance ou qui la retentent trop vite, ce qui peut bloquer les messages au bout de plusieurs tentatives, voire d'une seule.

Enfin, il y a des sites suffisamment grands pour disposer de plusieurs serveurs de messagerie sortants, peu compatibles avec le système de liste grise car il n'est pas garanti que l'adresse IP soit la même d'une tentative à l'autre. Ces sites prétendent sincèrement, et avec raison, se conformer au standard : les RFC ne déclarent pas que les nouvelles tentatives *doivent* émaner de la même adresse IP. C'est là un inconvénient évident du système de liste grise.

Si vous devez pallier ce type de problèmes dans votre infrastructure, c'est assez facile. Une bonne approche consiste à définir une table pour une liste blanche locale, qui sera peuplée par le contenu d'un fichier en cas de redémarrage :

```
table <localwhite> file "/etc/mail/whitelist.txt"
```

Pour s'assurer que le trafic SMTP émanant des adresses de cette table n'est pas envoyé à `spamd`, on ajoute une règle `no rdr` en haut du bloc de redirection :

```
no rdr proto tcp from <localwhite> to $emailserver port smtp
```

Une fois ces modifications intégrées au jeu de règles, on entre les adresses à protéger de la redirection dans le fichier `whitelist.txt` et on recharge le jeu de règles par `pfctl -f /etc/pf.conf`. Toutes les astuces possibles peuvent ensuite être utilisées sur la table `<localwhite>`, y compris la modification de son contenu après avoir changé le fichier `whitelist.txt`.

Notez qu'au moins quelques-uns des sites disposant de plusieurs serveurs SMTP sortants publient également les informations relatives aux hôtes autorisés à envoyer des e-mails vers leurs domaines, via des enregistrements SPF, qui font partie des informations DNS du domaine.

> Les enregistrements SPF sont stockés dans les zones DNS en tant qu'enregistrements TXT spéciaux ; voir http://www.openspf.org pour plus de détails. Remarquez que nous utilisons SPF seulement comme une source d'information possible. Une discussion complète sur les avantages et les inconvénients de l'architecture SPF et des buts qu'elle cherche à atteindre dépasse le cadre de ce livre.

Pour récupérer les enregistrements SPF de notre domaine `example.com`, vous pourriez utiliser l'option `-ttxt` de la commande `host`, comme ceci :

```
$ host -ttxt example.com
```

La commande donnerait une réponse comparable à :

```
example.com descriptive text "v=spf1 ip4:192.0.2.129/25 -all"
```

où le texte entre guillemets est l'enregistrement SPF du domaine `example.com`. Si les messages électroniques émanant du domaine `example.com` doivent arriver rapidement, et que ces personnes-là ne sont pas susceptibles d'envoyer ou de relayer du spam, copiez le rang d'adresses de l'enregistrement SPF dans votre fichier `whitelist.txt` et rechargez le contenu de la table `<localwhite>` à partir du fichier mis à jour.

En conclusion de notre expérience avec spamd

Pour résumer, des listes noires soigneusement sélectionnées et combinées avec `spamd` constituent des outils puissants, précis et efficaces de lutte contre le spam. La charge additionnelle sur la machine hébergeant `spamd` est minimale. Mais `spamd` ne se comportera jamais mieux que sa source de données la plus fragile, ce qui signifie que vous devrez superviser vos fichiers journaux et utiliser des listes blanches si nécessaire.

Il est tout à fait possible de faire fonctionner `spamd` en mode liste grise pur, sans liste noire. En fait, certains utilisateurs rapportent qu'une configuration `spamd` en mode liste grise seul n'est pas significativement moins efficace qu'une configuration à liste noire et que, dans certains cas, elle se révèle beaucoup plus performante que le filtrage de contenu.

On peut citer en exemple le message de Steve William sur la liste de diffusion OpenBSD-misc, daté du 20 octobre 2006 qui rapporte qu'une configuration à liste grise a permis de débarrasser son entreprise d'environ 95 % du spam qui lui était destiné (http://marc.info/?l=openbsd-misc&m=116136841831550&w=2).

Mon expérience personnelle me pousse à recommander la liste-piège de Bob Beck, générée par du greytrapping à grande échelle, en tant qu'unique liste noire importée. Cette liste sort du lot parce que le système de Bob retire automatiquement les adresses après 24 heures, ce qui se traduit par un très petit nombre de faux positifs.

Une fois que votre infrastructure vous convient, vous pourrez essayer d'introduire un greytrapping local. Vous attraperez sans doute quelques indésirables de plus et c'est, bien entendu, clair, net et amusant.

Des expériences limitées, menées pendant que j'écrivais ce chapitre[1], suggèrent même qu'il est très efficace de récolter les adresses invalides utilisées par les spammeurs, que ce soit depuis les journaux de votre serveur mail, ceux de `spamd` ou directement depuis votre liste grise, pour les ajouter à votre liste-piège. Publiez la liste sur une page web modérément visible, afin que les adresses qui y figurent soient enregistrées, encore et encore, par les robots récolteurs d'adresses. Vous améliorerez ainsi davantage votre matériel de greytrapping, car ces adresses seront probablement conservées par les spammeurs comme étant *valides*.

1. Chroniquées sur http://bsdly.blogspot.com, elles commencent par le texte se trouvant à l'adresse http://bsdly.blogspot.com/2007/07/hey-spammer-heres-list-for-you.html.

Files d'attente, calibrage et redondance

7

Ce chapitre aborde deux sujets qui, pris séparément ou ensemble, peuvent radicalement transformer votre vision des réseaux. Le thème récurrent de ce chapitre sera la gestion de la disponibilité des ressources. Dans une première partie, nous verrons comment utiliser le sous-système ALTQ de calibrage (*shaping*) de trafic pour allouer les ressources de bande passante efficacement et suivant une politique prédéfinie. Dans la seconde partie, nous verrons comment nous assurer que nos ressources restent disponibles grâce aux fonctionnalités de redondance offertes par les protocoles CARP et `pfsync`.

Diriger le trafic avec ALTQ

ALTQ, raccourci pour *ALTernate Queuing* (mise en file d'attente alternée), est un mécanisme très souple pour le calibrage du trafic réseau. ALTQ existait avant PF et il fut par la suite intégré au PF d'OpenBSD. Sous OpenBSD, ALTQ a été intégré au code de PF pour la sortie d'OpenBSD 3.3 ; il fut décidé que sa configuration se ferait dans `pf.conf`, essentiellement pour des raisons pratiques. Les ports de PF des autres systèmes BSD suivirent le mouvement et l'intégration d'ALTQ y est au moins optionnelle. Le processus d'intégration n'est pas encore terminé sur tous les systèmes et nous signalerons les différences quand cela sera nécessaire.

> **CULTURE** **Naissance d'Altq à USENIX 1999**
>
> Les recherches initiales sur ALTQ furent présentées dans un article de Kenjiro Cho destiné à la conférence USENIX 1999, intitulé « Managing Traffic with ALTQ ». Vous pouvez le lire en ligne.
> ▸ http://www.usenix.org/publications/library/proceedings/usenix99/cho.html

Les concepts de base d'ALTQ

La gestion de la bande passante ressemble fort à celle d'un chéquier ou de toute autre ressource rare ou disponible en quantité finie. L'approvisionnement est constant mais limité et vous devez allouer la ressource avec un maximum d'*efficacité*, selon les *priorités* définies dans votre *politique* ou *spécification*.

Le concept de file d'attente est au cœur de la gestion de bande passante avec ALTQ. Les *files d'attente* (*queues*) sont une sorte de tampon pour les paquets réseau. C'est là que sont placés les paquets avant d'être abandonnés ou envoyés, selon les critères qui s'appliquent à la file ; les paquets dépendent de la bande passante disponible de la file d'attente. Les files sont attachées à des interfaces spécifiques et la bande passante est gérée interface par interface, la bande passante disponible pour une interface donnée étant divisée entre les files d'attente définies par l'administrateur.

Les files sont définies soit par une quantité, soit par une proportion de la bande passante disponible. De plus, elles peuvent être soumises à une priorité hiérarchisée. Dans ce contexte, la *priorité* est un indicateur de préférence, qui permet de décider quelle file d'attente servir dans le plus court délai. Comme nous le verrons par la suite, certains types de files d'attente peuvent être configurés par une combinaison d'allocation et de priorité de bande passante. Pour affiner encore plus, certains types de files permettent d'allouer des portions de la bande passante de chaque file à des *sous-files*, c'est-à-dire des files résidant à l'intérieur d'autres files et partageant les ressources de la file parente. Une fois les files d'attente définies, l'intégration du calibrage dans le jeu de règles implique de réécrire les règles `pass` afin d'assigner le trafic à une file d'attente précise. Nous verrons cela plus en détail dans les pages à venir.

> Dans les configurations ALTQ, tout le trafic non explicitement assigné est regroupé avec le reste dans la file d'attente par défaut.

Ordonnanceurs de file d'attente ou disciplines de file

Dans la configuration réseau par défaut, sans file d'attente ALTQ, la pile TCP/IP et son sous-système de filtrage traitent les paquets dans leur ordre d'arrivée sur une interface. C'est ce que l'on appelle généralement la discipline *First In First Out* (*FIFO*, premier entré premier sorti).

Les files d'attente ALTQ peuvent être configurées pour obtenir des comportements assez différents. Chacun des trois algorithmes d'ordonnancement de file, ou *disciplines*, propose ses propres options :

`priq`

Les *files d'attente basées sur les priorités* (*priority-based queues*) sont purement définies en termes de priorité au sein de la bande passante totale. Pour les files `priq`, les priorités vont de 0 à 15 ; les valeurs les plus hautes bénéficient d'un traitement préférentiel. Les paquets qui correspondent aux critères des files de plus haute priorité sont servis avant ceux qui correspondent aux files de plus faible priorité.

`cbq`

Les *files d'attente basées sur les classes* (*class-based queues*) sont définies par une allocation de bande passante de taille constante (soit en pourcentage du total disponible, soit par unités de kilobits, mégabits ou gigabits par seconde). Une file `cbq` peut être divisée en sous-files qui se voient attribuer une priorité allant de 0 à 7, la plus forte priorité donnant ici aussi droit à un traitement préférentiel. Les paquets sont conservés dans la file jusqu'à ce que la bande passante soit disponible. Pour les files divisées en files priorisées et les files à allocation de bande passante, les paquets servis en premier sont ceux qui correspondent à la file de plus forte priorité.

`hfsc`

Cette discipline utilise l'algorithme *Hierarchical Fair Service Curve* (*HFSC*) pour garantir une allocation « équitable » de la bande passante parmi une hiérarchie de files d'attente. L'algorithme et la configuration correspondante sont assez compliqués, avec un grand nombre de paramètres modifiables. Pour cette raison, la plupart des utilisateurs d'ALTQ s'en tiennent aux types de files les plus simples, mais ceux qui affirment comprendre HSFC ne jurent que par lui.

Dans PF, la syntaxe générale des files ALTQ ressemble à ceci :

```
altq on interface type [options ... ] main_queue { sub_q1, sub_q2 ..}
  queue sub_q1 [ options ... ]
  queue sub_q2 [ options ... ] { subA, subB, ... }
[...]
pass [ ... ] queue sub_q1
pass [ ... ] queue sub_q2
```

Remarquez que les files `cbq` et `hsfc` peuvent admettre plusieurs niveaux de sous-files, alors que les files `priq` sont essentiellement plates (un seul niveau de file). Nous passerons en revue les spécificités syntaxiques de chacun de ces types un peu plus loin dans ce chapitre.

Configuration d'ALTQ

L'activation d'ALTQ (permettant d'utiliser la logique de file d'attente dans le jeu de règles PF) peut exiger quelques étapes supplémentaires selon le système d'exploitation utilisé.

ALTQ sous OpenBSD

Sous OpenBSD, toutes les disciplines de file d'attente sont compilées dans les noyaux `GENERIC` et `GENERIC.MP` ; la seule configuration nécessaire est donc celle de `pf.conf`.

ALTQ sous FreeBSD

Sous FreeBSD, vous devez vérifier que votre noyau contient bien ALTQ et ses options de disciplines de file d'attente ; rien de tout cela n'est activé par défaut dans le noyau `GENERIC` de FreeBSD. Les options à fournir sont les suivantes :

```
options         ALTQ
options         ALTQ_CBQ        # Class-Based Queuing (CBQ)
options         ALTQ_RED        # Random Early Detection (RED)
options         ALTQ_RIO        # RED In/Out
options         ALTQ_HFSC       # Hierarchical Packet Scheduler (HFSC)
options         ALTQ_PRIQ       # Priority Queuing (PRIQ)
options         ALTQ_NOPCC      # Required for SMP build
```

L'option `ALTQ` est nécessaire pour activer ALTQ dans le noyau et, sur les systèmes SMP, l'option `ALTQ_NOPCC` est *aussi* requise. Selon le type de file utilisé, vous devez activer au moins une option parmi `ALTQ_CBQ`, `ALTQ_PRIQ` et `ALTQ_HFSC`. Enfin, on peut activer les techniques visant à enrayer les congestions (*RED* pour *Random Early Detection* et *RED In/Out*), via les options respectives `ALTQ_RED` et `ALTQ_RIO`. Voir le Manuel de FreeBSD (le fameux *Handbook*) pour les informations relatives à la compilation et à l'installation d'un noyau personnalisé muni de ces options.

ALTQ sous NetBSD

À l'heure où j'écris ces lignes, ALTQ est en cours d'intégration dans l'implémentation de PF sur NetBSD 4.0. Comme pour FreeBSD, la configuration par défaut du noyau `GENERIC` de NetBSD n'inclut pas les options liées à ALTQ. Mais ces options figurent toutes dans le fichier de configuration de `GENERIC`, sous la forme de commentaires, afin d'en faciliter l'inclusion. Les principales options du noyau sont les suivantes :

```
options         ALTQ            # Manipulate network interfaces' output
queues
options         ALTQ_CBQ        # Class-Based Queuing
options         ALTQ_HFSC       # Hierarchical Fair Service Curve
options         ALTQ_PRIQ       # Priority Queuing
options         ALTQ_RED        # Random Early Detection
```

L'option `ALTQ` est nécessaire pour activer ALTQ dans le noyau. Selon le type de file utilisé, on doit activer au moins une option parmi `ALTQ_CBQ`, `ALTQ_PRIQ` et `ALTQ_HFSC`.

Pour les versions datant d'avant NetBSD 4.0, Peter Postma maintient un patch afin d'activer les options PF/ALTQ. Vous trouverez des informations actualisées sur ses pages concernant PF sous NetBSD, en particulier comment obtenir le patch ALTQ via `pkgsrc` ; consultez son site web à l'adresse http://nedbsd.nl/~ppostma/pf. La documentation de NetBSD relative à PF se trouve à l'adresse http://www.netbsd.org/Documentation/network/pf.html.

Vous devriez désormais disposer de toutes les informations nécessaires pour monter un système où ALTQ est activé.

De quelle quantité de bande passante disposez-vous réellement ?

Il peut être difficile de déterminer la bande passante réellement utilisable sur une interface donnée, ce qui est important pour gérer une file d'attente. Si aucune valeur de bande passante totale n'est précisée, c'est la bande passante totale disponible qui sera utilisée pour calculer les allocations. Mais certains types d'interfaces ne peuvent pas rapporter de façon fiable la bonne valeur de bande passante. Un exemple courant de ce type de décalage est le cas d'une passerelle d'interface externe Ethernet 100 Mbit, attachée à une ligne qui n'offre que 8 Mbit dans le sens descendant et 1 Mbit dans le sens ascendant. L'interface Ethernet rapportera alors de bonne foi une bande passante de 100 Mbit/s et non les valeurs de la ligne DSL. (Nul doute d'ailleurs que ces valeurs datent vraiment le livre et qu'elles sembleront obsolètes dans quelques années.)

Il est donc intéressant de régler la bande passante totale à une valeur fixée. Le problème est que cette valeur n'a aucun rapport avec les indications de votre FAI. Il y aura toujours un surplus variable suivant les technologies et implémentations. Sur un réseau usuel combinant TCP/IP et Ethernet filaire, le pourcentage de surplus est inférieur à 10 % ; dans le cas d'un réseau ATM ou ADSL, il n'est pas rare que le surplus atteigne 20 à 25 %. Si votre FAI ne se montre pas coopératif, vous devrez décider par vous-même et de manière réfléchie de la valeur de départ pour vos expérimentations. Dans tous les cas, gardez en tête que la bande passante totale disponible ne dépasse jamais celle du maillon le plus faible du chemin menant à votre réseau.

Il faut également noter que les files d'attente ne sont prises en charge que pour les connexions *sortant* du système qui assure la file d'attente. Pour planifier la gestion de la bande passante, il faut garder en tête la bande passante du maillon le plus faible du chemin menant au réseau et ce, même si les files d'attente sont configurées sur une autre interface.

Après ces préliminaires, vous devriez être prêt à voir quelques exemples de configurations réseau utilisant ALTQ.

Comprendre les files d'attente basées sur des priorités (priq)

Le concept de base des files d'attente basées sur des priorités (`priq`) est assez direct et c'est peut-être le plus simple à comprendre. Pour la bande passante totale allouée à la file principale, tout ce qui compte, c'est la priorité du trafic. On assigne aux files une valeur de priorité allant de 0 à 15 (plus la valeur est élevée, plus les requêtes de trafic de la file sont traitées rapidement).

Daniel Hartmeier en fournit un excellent exemple en conditions réelles. Il a découvert une méthode simple mais efficace pour améliorer le débit de son réseau personnel par le biais d'ALTQ. De manière assez courante, le réseau personnel de Daniel était rattaché à une connexion asymétrique, dont la bande passante utilisable était tellement faible qu'il fut poussé à en améliorer l'utilisation.

De plus, quand la ligne fonctionnait à plein régime ou presque, des choses étranges survenaient. Un symptôme en particulier semblait montrer un axe de progrès : le trafic entrant (téléchargements, e-mails entrants et autres) ralentissait de manière disproportionnée chaque fois qu'un trafic sortant démarrait. Ce phénomène dépassait ce qu'on pouvait expliquer en mesurant la quantité brute de données transférées. Il revint donc à une fonctionnalité basique de TCP/IP.

Après l'envoi d'un paquet TCP, l'expéditeur attend, pendant un délai fixé, un accusé de réception (sous la forme d'un paquet ACK) de la part du destinataire. Si le paquet ACK n'arrive pas dans le temps imparti, l'expéditeur suppose que le paquet initial n'a pas été reçu et il le renvoie donc.

Néanmoins, dans une configuration par défaut, les paquets sont servis séquentiellement par l'interface, au fur et à mesure de leur arrivée. Cela signifie forcément que les paquets ACK, qui ne contiennent pas vraiment de données utiles, font la queue pendant que les paquets de données sont transférés.

Voici une hypothèse sur laquelle nous pouvons travailler : si les petits paquets ACK, qui ne contiennent pratiquement aucune donnée, était capables de se glisser entre les paquets de données, qui sont plus gros, cela mènerait à une utilisation beaucoup plus efficace de la bande passante disponible. Le moyen le plus simple d'implémenter et de tester cette théorie consistait à mettre en place deux files d'attente avec deux priorités différentes et de les intégrer au jeu de règles.

Ces lignes montrent les parties correspondantes du jeu de règles :

```
ext_if="kue0"

altq on $ext_if priq bandwidth 100Kb queue { q_pri, q_def }
     queue q_pri priority 7
     queue q_def priority 1 priq(default)
```

```
pass out on $ext_if proto tcp from $ext_if to any flags S/SA \
        keep state queue (q_def, q_pri)

pass in on $ext_if proto tcp from any to $ext_if flags S/SA \
        keep state queue (q_def, q_pri)
```

Nous voyons ici que la file d'attente basées sur des priorités est configurée sur l'interface externe, avec deux files subordonnées. La première sous-file, `q_pri`, possède une forte priorité (de valeur `7`), tandis que la seconde, `q_def`, a une priorité beaucoup plus faible (de valeur `1`).

Ce jeu de règles, d'apparence simple, fonctionne en exploitant le traitement des files de priorités différentes par ALTQ. Après assignation d'une connexion à la file principale, ALTQ inspecte le champ Type de Service (ToS) de chaque paquet. Les paquets ACK portent un bit ToS *Delay* indiquant une valeur faible, c'est-à-dire une vitesse de transmission la plus rapide possible. Quand ALTQ voit un paquet à faible délai et qu'il est en présence de files de priorités différentes, il assigne alors le paquet à la file de plus haute priorité.

Cela signifie que les paquets ACK sont transmis avant les files de plus faible priorité, ce qui implique que les paquets de données sont également servis plus rapidement. Par conséquent, une configuration de de type fournit de meilleures performances qu'une configuration FIFO pure, avec le même matériel et la même bande passante disponible.

DOCUMENTATION

L'article de Daniel concernant cette version de sa configuration contient une analyse plus détaillée.

- http://www.benzedrine.cx/ackpri.html

Allocation de bande passante basée sur les classes, pour les petits réseaux (cbq)

Il est généralement judicieux de maximiser les performances réseau, mais chaque réseau peut avoir des besoins spécifiques. Par exemple, certains types de trafic (messagerie électronique ou autre service vital) peuvent nécessiter une quantité minimale permanente de bande passante, tandis que d'autres services (partage de fichiers pair à pair) peuvent être limités à une quantité de bande passante maximale. La discipline de file d'attente basée sur les classes (`cbq`) offre pour ce type de cas un jeu d'options légèrement plus conséquent.

Afin d'illustrer l'utilisation de `cbq`, nous allons voir un autre exemple qui repose sur les jeux de règles des chapitres précédents. Nous voulons permettre aux utilisateurs d'un petit réseau local de se connecter à un petit nombre de services prédéfinis, situés à l'extérieur de leur réseau, et nous voulons également autoriser l'accès depuis l'extérieur, à un serveur web situé sur le réseau local.

Ici, toutes les files d'attente se situent au niveau de l'interface externe, qui fait face à Internet. Cette approche est justifiée par la probabilité plus élevée que la bande passante soit limitée sur le lien externe que sur le réseau local. Cependant, et en principe, l'allocation des files d'attente et le calibrage de trafic peuvent se faire sur n'importe quelle interface réseau. Ici, la configuration comprend une file `cbq` pour une bande passante totale de 2 mégabits, avec six sous-files.

```
altq on $ext_if cbq bandwidth 2Mb queue { main, ftp, udp, web, ssh, icmp
}
     queue main bandwidth 18% cbq(default borrow red)
     queue ftp bandwidth 10% cbq(borrow red)
     queue udp bandwidth 30% cbq(borrow red)
     queue web bandwidth 20% cbq(borrow red)
    queue ssh bandwidth 20% cbq(borrow red) { ssh_interactive, ssh_bulk
}
          queue ssh_interactive priority 7 bandwidth 20%
          queue ssh_bulk priority 0 bandwidth 80%
     queue icmp bandwidth 2% cbq
```

Nous voyons que la sous-file `main`, qui occupe 18% de la bande passante, est désignée comme file d'attente par défaut. Cela signifie que tout trafic correspondant à une règle `pass` mais non explicitement assigné à une autre file d'attente termine dans la file `main`. Les mots-clés `borrow` et `red` signifient que la file peut « emprunter » (*borrow*) de la bande passante à la file parente, tandis que le système tente d'éviter les congestions en appliquant l'algorithme RED.

Les autres files suivent plus ou moins le même modèle, jusqu'à la file `ssh`, qui possède quant à elle deux sous-files, chacune dotée de sa propre priorité. Nous voyons ici une variante de l'exemple des priorités des paquets ACK : les transferts SSH de masse, typiquement les transferts SCP, possèdent un ToS indiquant *débit*, tandis que le trafic SSH interactif voit son drapeau ToS réglé sur *faible délai* et passe avant les transferts de masse. Le trafic interactif sera probablement moins consommateur de bande passante et il en obtient donc une part plus faible, mais il obtient un traitement préférentiel grâce à sa priorité élevée.

Le modèle améliore également la vitesse des transferts de fichiers SCP, car les paquets ACK correspondant à ces transferts seront assignés à une sous-file de forte priorité.

Nous avons enfin la file `icmp` qui réserve les 2 % de bande passante restants. Cela garantit une quantité minimale de bande passante pour le trafic ICMP que nous voulons laisser passer mais qui ne correspond pas aux critères d'assignation aux autres files d'attente.

Pour faire fonctionner tout cela, nous utilisons des règles `pass` traduisant l'assignation du trafic aux files d'attente et les critères associés :

```
set skip on { lo0, $int_if }
pass log quick on $ext_if proto tcp from any to any port ssh
flags S/SA \
    keep state queue (ssh_bulk, ssh_interactive)
pass in quick on $ext_if proto tcp from any to any port ftp flags S/SA \
    keep state queue ftp
pass in quick on $ext_if proto tcp from any to any port www flags S/SA \
    keep state queue http
pass out on $ext_if proto udp all keep state queue udp
pass out on $ext_if proto icmp all keep state queue icmp
pass out on $ext_if proto tcp from $localnet to any port $client_out
```

Les règles pour `ssh`, `www`, `udp` et `icmp` assignent toutes le trafic à leur file respective, alors que la dernière règle, jouant le rôle d'attrape-tout, laisse passer tout autre type de trafic émanant du réseau local et l'assigne à la file par défaut, `main`.

Files d'attente pour serveurs en DMZ

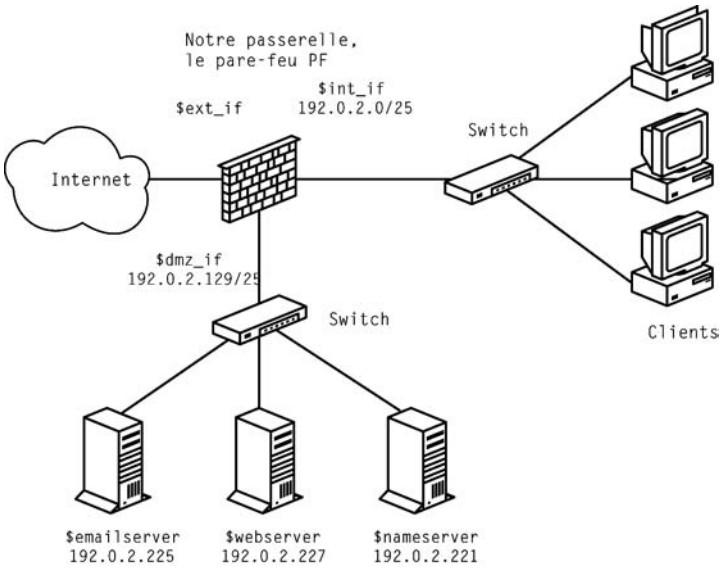
À la section « Un degré de séparation physique : présentation de la DMZ » , nous avons construit un réseau comportant une unique passerelle, mais dont tous les services visibles depuis l'extérieur tournaient sur des machines reléguées dans un réseau à part, nommé DMZ. De cette manière, tout le trafic destiné aux serveurs et provenant d'Internet ou du réseau local devait passer au travers de la passerelle.

La figure 7-1 représente un schéma du réseau, identique à la figure 5-2.

En prenant comme point de départ les règles du chapitre 5, nous ajouterons des files d'attente afin d'optimiser nos ressources réseau. La disposition physique et logique du réseau ne changera pas.

Le goulot d'étranglement le plus probable de ce réseau est la bande passante de la connexion entre l'interface externe de la passerelle et Internet. La bande passante du reste du réseau (c'est-à-dire de chacune des autres interfaces réseau concernées) n'est évidemment pas infinie, mais elle a peu de chances d'être un facteur limitant. Pour optimiser les performances des services, nous devons configurer les files de manière à ce que la bande passante disponible sur le site soit précisément dévolue aux types de trafic que nous souhaitons autoriser.

Figure 7–1
Un réseau à DMZ



Dans ce contexte, il est important de bien saisir la différence entre la bande passante d'une interface et celle réellement disponible pour la famille de connexions que nous voulons laisser passer. Dans notre exemple, il est probable que la bande passante de l'interface de la DMZ atteigne 100 Mbit ou 1 Gbit, alors que celle disponible pour les connexions émanant de l'extérieur du réseau local sera considérablement moindre. Cette considération se traduit dans la définition de nos files d'attente, où la principale limite est la bande passante réellement disponible pour le trafic externe.

```
total_ext = 2Mb
total_dmz = 100Mb
altq on $ext_if cbq bandwidth $total_ext queue { ext_main, ext_web,
ext_udp, ext_mail, ext_ssh }
queue ext_main bandwidth 25% cbq(default borrow red) { ext_hi, ext_lo }
     queue ext_hi priority 7 bandwidth 20%
     queue ext_lo priority 0 bandwidth 80%
queue ext_web bandwidth 25% cbq(borrow red)
queue ext_udp bandwidth 20% cbq(borrow red)
queue ext_mail bandwidth 30% cbq(borrow red)
altq on $dmz_if cbq bandwidth $total_dmz queue { ext_dmz, dmz_main,
dmz_web, dmz_udp, dmz_mail }
queue ext_dmz bandwidth $total_ext cbq(borrow red) queue { ext_dmz_web,
ext_dmz_udp, ext_dmz_mail }
     queue ext_dmz_web bandwidth 40% priority 5
     queue ext_dmz_udp bandwidth 10% priority 7
     queue ext_dmz_mail bandwidth 50% priority 3
```

```
queue dmz_main bandwidth 25Mb cbq(default borrow red) queue {
dmz_main_hi, dmz_main_lo }
     queue dmz_main_hi priority 7 bandwidth 20%
     queue dmz_main_lo priority 0 bandwidth 80%
queue dmz_web bandwidth 25Mb cbq(borrow red)
queue dmz_udp bandwidth 20Mb cbq(borrow red)
queue dmz_mail bandwidth 20Mb cbq(borrow red)
```

Remarquez que la limite `total_ext` détermine l'allocation de toutes les files gérant la bande passante extérieure. Pour utiliser la nouvelle infrastructure de file d'attente, nous devons aussi apporter quelques modifications aux règles de filtrage. Mieux vaut garder à l'esprit que tout le trafic non explicitement assigné à une file d'attente précise sera affecté à la file par défaut de l'interface. Il est donc important d'affiner les règles de filtrage et les définitions des files, afin qu'elles correspondent au mieux au trafic sur le réseau.

Après avoir ajouté les files d'attente, la partie principale des règles de filtrage possède l'allure suivante :

```
pass in on $ext_if proto { tcp, udp } from any to $nameservers port domain queue
ext_udp
pass in on $int_if proto { tcp, udp } from $localnet to $nameservers port domain
pass out on $dmz_if proto { tcp, udp } from any to $nameservers port domain queue
ext_dmz_udp
pass out on $dmz_if proto { tcp, udp } from $localnet to $nameservers port domain
queue dmz_udp
pass in on $ext_if proto tcp from any to $webserver port $webports queue ext_web
pass in on $int_if proto tcp from $localnet to $webserver port $webports
pass out on $dmz_if proto tcp from any to $webserver port $webports queue ext_dmz_web
pass out on $dmz_if proto tcp from $localnet to $webserver port $webports queue
dmz_web
pass in log on $ext_if proto tcp from any to $emailserver port smtp
pass in log on $ext_if proto tcp from $localnet to $emailserver port smtp
pass in log on $int_if proto tcp from $localnet to $emailserver port $email
pass out log on $dmz_if proto tcp from any to $emailserver port smtp queue ext_mail
pass in on $dmz_if from $emailserver to any port smtp queue dmz_mail
pass out log on $ext_if proto tcp from $emailserver to any port smtp queue
ext_dmz_mail
```

Vous remarquerez que seul le trafic passant par les interfaces DMZ ou externe se voit assigner à une file d'attente. Dans cette configuration, où aucun service accessible de l'extérieur n'est hébergé sur le réseau interne, la mise en file d'attente sur l'interface interne n'aurait pas beaucoup de sens : c'est en effet la partie du réseau qui subit peut-être le moins de restrictions sur la bande passante disponible.

Utiliser ALTQ pour gérer le trafic indésirable

Nous nous sommes jusqu'ici concentrés sur la mise en place de files d'attentes pour atteindre une efficacité optimale (compte tenu des conditions sur l'environnement du réseau) dans le traitement de classes de trafic précises. Pour conclure notre introduction aux files d'attente, nous allons étudier quelques exemples présentant une approche légèrement différente de l'identification et de la gestion du trafic non désiré. Ces exemples, par le biais de quelques astuces relatives aux files d'attente, peuvent vous aider à réduire un peu le niveau de bruit de votre réseau.

Dépassement de la charge d'une toute petite file d'attente

Revenons au début du chapitre 6, dans la section « Garder les méchants à distance » : nous y avons utilisé une combinaison d'options de suivi de l'état et de règles `overload` afin de remplir une table d'adresses. Nous réservions à ces adresses un traitement spécial (voir chapitre précédent) consistant à couper toutes les connexions. Il est toutefois également possible d'assigner le trafic `overload` à une file d'attente dédiée.

Considérez cette règle, vue plus haut au sujet de l'allocation de bande passante avec `cbq` :

```
pass log quick on $ext_if proto tcp from any to any port ssh flags S/SA
\
    keep state queue (ssh_bulk, ssh_interactive)
```

Si nous ajoutons des options de suivi d'état, comme ceci :

```
pass log quick on $ext_if proto tcp from any to any port ssh flags S/SA
\
    keep state (max-src-conn 15, max-src-conn-rate 5/3, \
    overload <bruteforce> flush global) queue (ssh_bulk,
ssh_interactive)
```

que nous réduisons légèrement l'allocation de l'une des files d'attente (mettons la file web) pour faire de la place aux contrevenants et que nous ajoutons ensuite :

```
queue smallpipe bandwidth 1% cbq
```

alors nous pouvons assigner le trafic indésirable à la file de petite bande passante grâce à cette règle :

```
pass inet proto tcp from <bruteforce> to any port $tcp_services queue
smallpipe
```

Il peut s'avérer utile d'ajouter à de telles règles une durée d'expiration des entrées de tables, comme décrit en début de chapitre 6 (section « Nettoyer vos tables avec pfctl »).

Assignation aux files d'attente basée sur l'empreinte du système d'exploitation

PF dispose d'un mécanisme de prise d'empreinte du système d'exploitation (*OS fingerprinting*) plutôt fiable, qui détecte le système d'exploitation utilisé à l'autre extrémité d'une connexion réseau en se basant sur les caractéristiques du paquet SYN ayant servi à l'initier. Notre dernier exemple concernant ALTQ développe le précédent jeu de règles simple, en se basant sur la probabilité élevée que les machines émettrices de spam utilisent un système d'exploitation particulier. S'il n'est pas possible, par exemple, de faire tourner `spamd` dans un environnement, une règle comme :

```
pass quick proto tcp from any os "Windows" to $ext_if port smtp queue
smallpipe
```

peut s'avérer une solution de rechange simple, lorsque l'on peut affirmer que personne n'enverra de message légitime depuis ce système d'exploitation précis. Ici, le trafic émanant d'hôtes fonctionnant sous le système d'exploitation spécifié se trouve limité à seulement 1% de la bande passante, sans emprunt possible.

Redondance et tolérance aux pannes : CARP et pfsync

De mémoire d'informaticien, la haute disponibilité et le service ininterrompu ont toujours été des termes très marketing et des objectifs cachés de tout administrateur réseau. CARP et `pfsync` furent ajoutés à OpenBSD 3.5 pour répondre à ces besoins et résoudre des problèmes qui leur sont liés.

CARP (*Common Address Redundancy Protocol*) a été développé en tant qu'alternative à *VRRP* (*Virtual Router Redundancy Protocol*, voir RFC 2281 et RFC 3768) ; ce dernier était grevé de brevets, mais il était bien parti pour devenir un standard sanctionné par l'IETF, malgré les problèmes potentiels non résolus.

Propriété intellectuelle

Les brevets en question sont détenus par Cisco, IBM et Nokia ; voir les RFC 2281 et 3768 pour de plus amples informations.

L'un des objectifs de CARP est de garantir la continuité de fonctionnement du réseau même si un pare-feu (ou un autre service) tombe, que ce soit à cause d'erreurs ou de maintenances planifiées (mises à jour, etc.). En complément de CARP, le protocole `pfsync` est conçu pour gérer la synchronisation des états PF entre des nœuds (ou des passerelles) de filtrage de paquets redondants. Ces deux protocoles sont prévus pour assurer la redondance des fonctionnalités réseau essentielles, avec récupération automatique en cas de panne.

CARP se base sur la configuration d'un groupe de machines, l'une étant désignée comme *maître* et les autres jouant le rôle de *sauvegarde* redondante, chacune de ces dernières étant équipée pour gérer une adresse IP commune. Le passage de relais d'un hôte CARP à un autre peut être authentifié via la configuration préalable d'un secret partagé qui, en pratique, ressemble fort à un mot de passe.

Dans le cas des pare-feux PF, `pfsync` peut être configuré de manière à gérer la synchronisation et, si cela est fait proprement, les connexions actives seraient alors transférées sans que l'on puisse remarquer d'interruption. En fait, `pfsync` est un type d'interface réseau virtuelle spécialement conçu pour synchroniser les informations d'état entre plusieurs pare-feux PF. Ses interfaces sont assignées à des interfaces physiques via `ifconfig`. Sur les réseaux où des exigences en matière de disponibilité très strictes nécessitent une récupération de panne automatisée, le nombre de connexions réseau simultanées (et donc le nombre d'états correspondants) peut devenir tellement grand qu'il devient évident de séparer physiquement le réseau `pfsync` du reste du réseau. `pfsync` ne gère en outre aucune authentification entre les partenaires de synchronisation et l'on ne peut donc garantir une synchronisation correcte que si l'on utilise des interfaces dédiées au trafic `pfsync`.

Spécifications du projet : deux passerelles redondantes

Avant toute chose, il est judicieux de rédiger une spécification claire et nette. Pour illustrer une configuration utilisant CARP et `pfsync`, imaginons un réseau disposant d'une unique passerelle vers le monde.

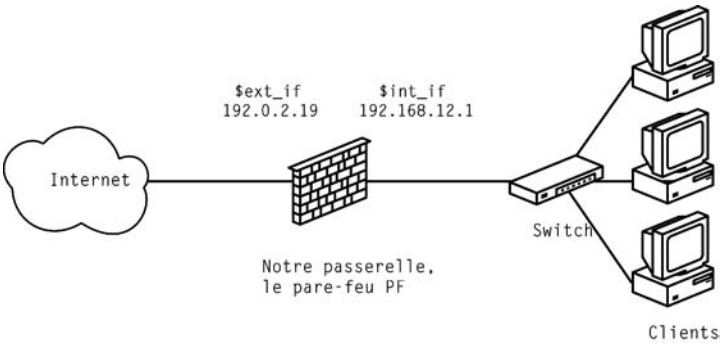
Les détails exacts du jeu de règles PF ne sont pas importants pour le moment ; ce qui l'est en revanche, c'est le but de l'exercice.

À la fin de la refonte de la configuration, le réseau doit répondre aux exigences suivantes :

- continuer à fonctionner comme avant ;
- offrir une meilleure disponibilité, sans que l'on puisse ressentir d'indisponibilité ;
- récupérer efficacement en cas de panne et ce, sans interrompre les connexions actives.

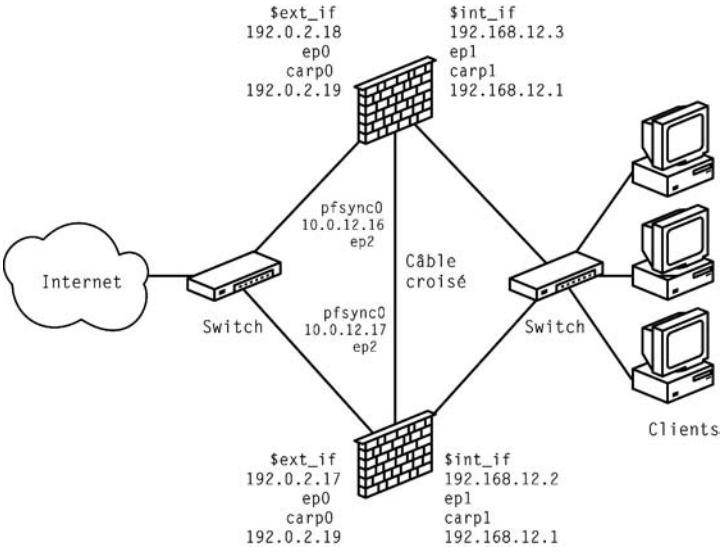
Nous allons commencer avec le réseau du chapitre 3, qui est relativement simple, analogue à celui de la figure 7-2.

Figure 7–2
Réseau à passerelle unique



Nous remplaçons la passerelle unique par deux machines redondantes, reliées par un réseau privé pour les mises à jour des informations d'état par `pfsync`. Le résultat ressemble à la figure 7-3.

Figure 7–3
Réseau à passerelles redondantes



Passons ensuite aux détails de cette configuration. Comme dans les chapitres précédents, la référence est un système OpenBSD, mais nous signalerons les différences avec les autres systèmes BSD aux endroits nécessaires.

Il est important de comprendre que les adresses CARP sont des adresses virtuelles. À moins d'avoir accès à la console de toutes les machines du groupe CARP, il faut quasiment toujours assigner une adresse IP aux interfaces physiques pour communiquer avec l'hôte en étant certain de la machine avec laquelle a lieu l'interaction.

Par convention, l'adresse IP assignée à l'interface physique appartient au même sous-réseau que l'adresse IP virtuelle partagée. En fait, le noyau tentera par défaut d'assigner l'adresse CARP à une adresse physique déjà configurée, appartenant au même sous-réseau. On peut outrepasser cette sélection d'interface en renseignant la valeur désirée dans l'option `carpdev` de la commande `ifconfig` lancée à l'activation de l'interface CARP.

> Quand on reconfigure un réseau et que l'adresse de la passerelle par défaut devient une adresse virtuelle (elle n'est donc plus fixée à une interface et un hôte précis), il est extrêmement difficile d'éviter une perte de connectivité temporaire.

Mise en place de CARP : options du noyau, sysctl et commandes ifconfig

Mettre en place une configuration redondante implique essentiellement de câbler, régler des options `sysctl` et lancer des commandes `ifconfig`. Nous ne décrivons ici le câblage qu'en termes généraux (le schéma principal découle naturellement de l'illustration, adapté le cas échéant). Toutefois, dans certains systèmes, il faudra vérifier que le noyau intègre bien les options nécessaires. Voici quelques instructions spécifiques aux différents systèmes d'exploitation.

CARP sous OpenBSD

Les périphériques `carp` et `pfsync` figurent dans la configuration par défaut des noyaux `GENERIC` et `GENERIC.MP`. À moins d'utiliser un noyau personnalisé où ces options ont été retirées, aucune configuration n'est nécessaire pour le noyau.

CARP sous FreeBSD

Vérifiez que votre noyau intègre les pseudo-périphériques `carp` et `pfsync`. Par défaut, le noyau `GENERIC` n'intègre pas ces options. Voir le *FreeBSD Handbook* pour la marche à suivre afin de compiler et d'installer un noyau personnalisé contenant ces options.

CARP sous NetBSD

Vérifiez que votre noyau intègre le pseudo-périphérique `carp`. La configuration par défaut du noyau `GENERIC` de NetBSD n'intègre pas `carp`.

Dans tous les systèmes capables de faire tourner CARP, les fonctions de base sont gouvernées par quelques variables `sysctl`. La principale d'entre elles, `net.inet.carp.allow`, est activée par défaut. Sur un système OpenBSD normal, on verra apparaître :

```
$ sysctl net.inet.carp.allow
net.inet.carp.allow=1
```

ce qui signifie que le système est convenablement équipé pour faire fonctionner CARP.

Avertissement

Si le noyau ne comprend pas de périphérique CARP, alors cette commande produira une erreur du type `unknown oid 'net.inet.carp.allow'` (sous FreeBSD) ou `sysctl: third level name 'carp' in 'net.inet.carp.allow'` (sous NetBSD).

Pour vérifier que le système est correctement configuré et visualiser toutes les variables liées à `carp`, on peut utiliser la commande `sysctl` suivante :

```
$ sysctl net.inet.carp
net.inet.carp.allow=1
net.inet.carp.preempt=0
net.inet.carp.log=0
net.inet.carp.arpbalance=0
```

FreeBSD

Sous FreeBSD, vous rencontrerez aussi la variable `net.inet.carp.suppress_preempt`, qui est une variable en lecture seule indiquant si la préemption est possible ou non.

Les variables importantes sont les deux premières et les deux autres ne devraient pas nécessiter de modification. Pour information, régler `net.inet.carp.log` à `1` renverra des informations de débogage concernant le trafic CARP journalisé, mais cette fonctionnalité est désactivée par défaut. De la même manière, la variable `net.inet.carp.arpbalance` peut servir à activer l'*équilibrage ARP* dans CARP, qui gère un équilibrage de charge limité entre les hôtes d'un réseau local. Mais pour que la récupération de panne entre nos passerelles puisse se faire correctement, nous devons activer la variable `net.inet.carp.preempt` :

```
$ sudo sysctl net.inet.carp.preempt=1
```

Cela signifie que, sur les hôtes disposant de plusieurs interfaces réseau, ce qui est le cas de nos passerelles, toutes les interfaces CARP passeront leur `advskew` (dont nous allons parler dans un instant) à la valeur extrêmement forte de `240 ;` de cette façon, les autres hôtes du groupe CARP sont contraints de démarrer la récupération de panne quand l'une des interfaces tombe. À la mise en place du système, il faut répéter ce réglage sur tous les hôtes.

Ensuite, il faut activer les interfaces réseau. Si nous regardons le diagramme du réseau, nous voyons que le réseau local utilise les adresses du réseau `192.168.12.0`, tandis que l'interface externe (qui fait face à Internet) fait partie du réseau `192.0.2.0`. En considérant ces rangs d'adresses et le comportement par défaut des interfaces CARP, les commandes de configuration des interfaces virtuelles découlent naturellement.

Sur la machine devant jouer le rôle de maître initial du groupe, on utilise ces commandes :

```
$ sudo ifconfig carp0 192.0.2.19 vhid 1
$ sudo ifconfig carp1 192.168.1.1 vhid 2
```

Remarquez que nous n'avons pas à indiquer explicitement l'interface physique. Les interfaces virtuelles `carp0` et `carp1` se lieront d'elles-mêmes aux interfaces physiques portant une adresse appartenant au même sous-réseau qu'elles. On peut ensuite vérifier que les interfaces CARP sont bien configurée comme il se doit grâce à la commande `ifconfig` :

```
$ ifconfig carp0
carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:00:5e:00:01:01
        carp: MASTER carpdev ep0 vhid 1 advbase 1 advskew 0
        groups: carp
        inet 192.0.2.19 netmask 0xffffff00 broadcast 192.0.2.255
        inet6 fe80::200:5eff:fe00:101%carp0 prefixlen 64 scopeid 0x5
```

La sortie de la commande `ifconfig` pour les interfaces CARP devrait être similaire. Remarquez la ligne `carp:`, qui indique le statut `MASTER`. Sur la machine de secours, la configuration est presque identique, sauf qu'il faut ajouter le paramètre `advskew` :

```
$ sudo ifconfig carp0 192.0.2.19 vhid 1 advskew 100
$ sudo ifconfig carp1 192.168.1.1 vhid 2 advskew 100
```

La paramètre `advskew` demande quelques explications. Pour faire court, il indique le niveau de *moindre préférence* de la machine pour la prise de contrôle, en cas de chute du maître actuel. En fait, `advskew` (et son compagnon `advbase`) sont utilisés pour cal-

culer l'intervalle entre deux annonces d'état de l'hôte maître, une fois que celui-ci a pris le contrôle. Les valeurs par défaut de `advbase` et `advskew` sont respectivement `1` et `0`. Dans notre exemple, le maître s'annonce toutes les secondes (1 + 0/256) et la machine de secours attend 1 + 100/256 secondes. Avec `net.inet.carp.preempt=1`, lorsque le maître arrête ses annonces ou signale son indisponibilité, les renforts prennent le relais et le nouveau maître commence à s'annoncer suivant son propre rythme. Une valeur faible pour `advskew` implique un intervalle d'annonce plus court et donc une plus forte probabilité de devenir le nouveau maître. Si plusieurs hôtes ont le même `advskew`, le maître actuel garde son statut.

Depuis OpenBSD 4.1, un nouveau facteur a fait son apparition dans l'équation déterminant l'hôte qui prend le rôle de maître CARP. Le *compteur de dégradation* (*demotion counter*) est une valeur annoncée par chaque hôte CARP au sujet de son groupe d'interfaces CARP. Il s'agit d'un indicateur du niveau de fiabilité des interfaces CARP. Zéro signifie que l'hôte est complètement prêt, tandis que des valeurs positives mesurent le niveau de dégradation. On peut régler le compteur de dégradation en utilisant `ifconfig -g`, mais c'est habituellement le système qui gère lui-même sa valeur (les valeurs les plus fortes apparaissant généralement au cours du processus de démarrage). Si toutes les autres valeurs sont égales, c'est l'hôte portant le plus faible compteur de dégradation qui remporte l'élection du nouveau maître CARP.

Sur la machine de secours, on doit vérifier que les interfaces CARP sont toutes configurées correctement, par le biais de l'inévitable `ifconfig` :

```
$ ifconfig carp0
carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:00:5e:00:01:01
        carp: BACKUP carpdev ep0 vhid 1 advbase 1 advskew 100
        groups: carp
        inet 192.0.2.19 netmask 0xffffff00 broadcast 192.0.2.255
        inet6 fe80::200:5eff:fe00:101%carp0 prefixlen 64 scopeid 0x5
```

Ici, la sortie n'est que légèrement différente ; remarquez que la ligne `carp:` indique le statut `BACKUP` (machine de secours) et les valeurs respectives des paramètres `advbase` et `advskew`.

Pour un usage en production, il est recommandé d'ajouter une mesure de sécurité contre une activité CARP non autorisée. On assigne pour cela une phrase secrète et partagée entre les membres du groupe CARP, par exemple :

```
$ sudo ifconfig carp0 pass mekmitasdigoat 192.0.2.19 vhid 1
$ sudo ifconfig carp1 pass mekmitasdigoat 192.168.1.1 vhid 2
```

Remplacez, bien entendu, le mot de passe de l'exemple[1]. Tout trafic CARP l'exigera à partir de ce moment : prenez donc bien soin de configurer toutes les interfaces CARP d'un groupe avec la même phrase – ou alors faites sans !

Une fois les choix de configuration arrêtés, on insère les réglages dans les fichiers adéquats de `/etc/` : sous OpenBSD, les paramètres de `ifconfig` vont dans `hostname.carp0` et `hostname.carp1` ; sous FreeBSD et NetBSD, il faut les placer dans les variables `ifconfig_carp0=` et `ifconfig_carp1=` du fichier `rc.conf`.

Synchronisation des tables d'état : pfsync

La dernière pièce du puzzle, avant d'aborder le jeu de règles PF, consiste à configurer la synchronisation des tables d'état entre les hôtes du groupe de pare-feux redondants. Il faudrait vraiment que le ciel nous tombe sur la tête pour que le trafic soit interrompu pendant la reprise d'une panne. Comme nous l'avons déjà signalé, nous avons seulement besoin d'un ensemble d'interfaces `pfsync` correctement configurées.

> À ce jour, malheureusement, NetBSD ne prend pas encore en charge `pfsync` et ce, à cause d'un problème de numérotation de protocole. Cette situation devrait changer dans une version future.

Si l'on a pris la peine de planifier un peu les choses, la configuration des interfaces `pfsync` se fait en quelques commandes `ifconfig` simples. On peut configurer `pfsync` sur n'importe quelle interface réseau active mais, en général, il vaut mieux réserver une interface réseau à la synchronisation `pfsync`.

C'est d'ailleurs ce que nous avons fait pour notre configuration d'exemple (voir figure 7-3). Un câble croisé relie les deux interfaces Ethernet. Si le groupe de reprise de panne comporte davantage de machines, il faudra faire appel à un switch, un hub ou un vlan.

Pour cet exemple, nous avons assigné les adresses IP `10.0.12.16` et `10.0.12.17` aux interfaces prévues pour la synchronisation. Comme la configuration TCP/IP basique a déjà été faite, la configuration complète de `pfsync` pour les deux interfaces partenaires de synchronisation se limite à :

```
$ sudo ifconfig pfsync0 syncdev ep2
```

1. Cette phrase a une signification pour les initiés – une recherche sur le Web vous permettra de le découvrir... Si vous n'avez pas envie de chercher, rendez-vous à l'adresse http://marc.info/?l=openbsd-misc&m=98027812528843&w=2.

Cela illustre à la fois l'avantage d'avoir des configurations matérielles identiques et celui de faire passer le trafic `pfsync` sur un réseau physiquement à part.

En lui-même, le protocole `pfsync` n'offre que peu de fonctionnalités de sécurité. Il ne dispose d'aucun mécanisme d'authentification et communique par défaut via du trafic IP multicast. S'il n'est vraiment pas possible de construire un sous-réseau physiquement séparé du reste, il reste possible de renforcer la sécurité de `pfsync`. Deux options sont envisageables : obliger `pfsync` à se synchroniser uniquement avec un pair précis :

```
$ sudo ifconfig pfsync0 syncpeer 10.0.12.16 syncdev ep2
```

ou protéger le trafic de synchronisation en utilisant IPsec :

```
$ sudo ifconfig pfsync0 syncpeer 10.0.12.16 syncdev enc0
```

ce qui signifie que le périphérique `syncdev` devient l'interface d'encapsulation `enc0` au lieu de l'interface physique.

> Autant que possible, il faut viser une synchronisation dans un sous-réseau physiquement séparé.

Nous atteignons la fin de la configuration réseau de base pour une reprise de panne basée sur CARP. Dans la section suivante, nous allons voir ce qu'il faut garder en tête pour rédiger un jeu de règles PF dans le cadre d'une configuration redondante et tolérante aux pannes.

Assemblage du jeu de règles

Après tous ces efforts pour configurer le réseau, que reste-t-il à faire pour migrer les règles du fichier `pf.conf` actuel vers la nouvelle configuration ?

Eh bien, pas grand chose en vérité. L'essentiel des modifications apportées est invisible au reste du monde et un jeu de règles destiné à une passerelle simple fonctionnera généralement bien dans une configuration redondante, s'il est bien conçu. Nous avons cependant introduit deux nouveaux protocoles, CARP et `pfsync`, ce qui implique des modifications mineures pour que la reprise de panne se passe correctement.

Laisser passer le trafic CARP sur les interfaces adéquates

La solution la plus lisible consiste à introduire une définition de macro pour les périphériques CARP et à l'accompagner d'une règle `pass` du type :

```
pass on $carpdevs proto carp keep state
```

Laisser passer le trafic pfsync sur les interfaces adéquates

La solution la plus lisible consiste à introduire une définition de macro pour les périphériques `pfsync` et à l'accompagner d'une règle `pass` du type :

```
pass on $syncdev proto pfsync
```

ou, si l'on souhaite soustraire le périphérique `pfsync` du filtrage :

```
set skip on $syncdev
```

Il faut aussi réfléchir aux rôles joués par l'interface virtuelle CARP par rapport à l'interface physique, notamment pour son adresse. Du point de vue de PF, tout le trafic entrant viendra des interfaces physiques, mais le trafic peut porter l'adresse IP de l'interface CARP comme source ou comme destination.

Dans une configuration tolérante aux pannes, il n'est pas toujours nécessaire de synchroniser toutes les règles (connexions aux services hébergés par la passerelle en elle-même). On peut citer en exemple une règle typique, qui vise à autoriser SSH en entrée pour l'administrateur :

```
pass in on $int_if from $ssh_allowed to self
```

Pour ces règles, l'option d'état `no-sync` peut servir à empêcher la synchronisation des changements d'états pour les connexions sans intérêt après la reprise de panne :

```
pass in on $int_if from $ssh_allowed to self keep state (no-sync)
```

Cette configuration vous permettra de planifier les mises à jour des systèmes d'exploitation des membres du groupe CARP, ainsi que les opérations de maintenance exigeant une interruption de service, sans pour autant que les utilisateurs puissent mesurer ou remarquer la moindre interruption.

Journalisation, supervision et statistiques

8

Les lecteurs de ce livre, professionnels ou non, cherchent à contrôler leur réseau. Pour atteindre cet objectif, il est nécessaire de savoir tout ce qui se passe sur le réseau. Heureusement pour nous, PF est capable, comme la plupart des composants des systèmes Unix, de générer des journaux d'activité réseau.

PF propose un certain nombre d'options pour définir le niveau de détail de la journalisation, le traitement des fichiers journaux et l'extraction de certains types de données. Les outils existant dans le système de base permettent déjà d'accomplir beaucoup mais il en existe d'autres, disponibles via le gestionnaire de paquetages du système BSD. On peut y faire appel pour collecter, étudier et présenter les données de journalisation de diverses manières. Dans ce chapitre, nous verrons plus en détail la journalisation de PF et certains des outils que nous venons d'évoquer.

La base des journaux de PF

C'est à vous de décider ce que PF enregistre dans son journal et avec quel niveau de détail : ces informations sont définies depuis le jeu de règles. Le principe est simple : il faut ajouter le mot-clé `log` à chaque règle que l'on souhaite journaliser.

Quand on charge un jeu de règles où figure le mot-clé `log`, tout paquet initiant une connexion correspondant aux règles concernées (qu'il soit bloqué ou qu'il passe) est copié vers un périphérique `pflog`.

PF stockera également des données supplémentaires, telles que l'horodatage, l'interface, le fait que le paquet ait été autorisé ou non à passer et le numéro de règle du jeu chargé. Les données de journalisation de PF sont ensuite collectées par le démon de journalisation `pflogd` lancé par défaut, lorsque PF est activé, au démarrage du sys-

tème. Par défaut, les données de journalisation sont stockées dans `/var/log/pflog` ; le journal est cependant écrit dans le format binaire utilisé par `tcpdump`.

> **Format binaire du journal**
>
> Les outils supplémentaires pour extraire et afficher les informations de votre fichier journal seront décrits plus tard. Ne vous inquiétez pas, il s'agit d'un format binaire bien connu et largement pris en charge.

Pour commencer, voyons un exemple simple de journalisation. Ajoutons aux règles à journaliser le mot-clé `log` :

```
block log all
pass log quick proto { tcp, udp } from any to any port ssh
```

Rechargeons le jeu de règles : on remarque que l'horodatage du fichier `/var/log/pflog` change à mesure que le fichier grossit. Pour visualiser les données qui y sont stockées, on peut faire appel à `tcpdump` et son option `-r` (lecture du fichier).

Si la journalisation tourne depuis quelque temps, la commande :

```
$ sudo tcpdump -n -ttt -r /var/log/pflog
```

peut afficher une grande quantité de lignes qui défileront très rapidement à l'écran. La séquence suivante reproduit seulement les toutes premières lignes de la sortie d'un `tcpdump` long de plusieurs écrans, la plupart des éléments s'étalant sur plusieurs lignes :

```
$ sudo tcpdump -n -ttt -r /var/log/pflog
tcpdump: WARNING: snaplen raised from 96 to 116
Sep 13 13:00:30.556038 rule 10/(match) pass in on epic0: 194.54.107.19.34834 >
194.54.103.66.113: S 3097635127:3097635127(0) win 16384
 ➥ <mss 1460,nop,nop,sackOK,nop,wscale
0,[|tcp]> (DF)
Sep 13 13:00:30.556063 rule 10/(match) pass out on fxp0: 194.54.107.19.34834 >
194.54.103.66.113: S 3097635127:3097635127(0) win 16384
 ➥ <mss 1460,nop,nop,sackOK,nop,wscale
0,[|tcp]> (DF)
Sep 13 13:01:07.796096 rule 10/(match) pass in on epic0: 194.54.107.19.29572 >
194.54.103.66.113: S 2345499144:2345499144(0) win 16384
 ➥ <mss 1460,nop,nop,sackOK,nop,wscale
0,[|tcp]> (DF)
Sep 13 13:01:07.796120 rule 10/(match) pass out on fxp0: 194.54.107.19.29572 >
194.54.103.66.113: S 2345499144:2345499144(0) win 16384
 ➥ <mss 1460,nop,nop,sackOK,nop,wscale
0,[|tcp]> (DF)
```

```
Sep 13 13:01:15.096643 rule 10/(match) pass in on epic0: 194.54.107.19.29774 >
194.54.103.65.53: 49442 [1au][|domain]
Sep 13 13:01:15.607619 rule 12/(match) pass in on epic0: 194.54.107.19.29774 >
194.54.107.18.53: 34932 [1au][|domain]
```

Vous vous rendrez compte que `tcpdump` est un programme très souple, en particulier grâce au large choix d'options pour la mise en forme de la sortie. La présentation dans cet exemple découle des options passées à `tcpdump`. Le programme affiche presque toujours la date et l'heure d'apparition du paquet (ici, l'option `-ttt` spécifie le format long). Ensuite, `tcpdump` indique le numéro de la règle et l'interface où le paquet s'est présenté, puis les adresses et ports source et de destination (l'option `-n` indique à `tcpdump` d'afficher les adresses IP au lieu des noms d'hôtes). Enfin apparaissent diverses propriétés du paquet. Chacun peut trouver la combinaison d'options qui lui convient en lisant la page de manuel de `tcpdump`.

Déchiffrer les numéros de règles

Il est intéressant de noter que les numéros de règles dans les fichiers de journalisation se réfèrent au jeu de règles *chargé en mémoire*. Le jeu de règles passe par plusieurs étapes automatisées au cours du processus de démarrage (optimisations, développement des macros, etc.). De ce fait, le numéro de règle stocké dans le journal ne correspond probablement pas au numéro de la ligne où figure la règle dans le fichier `/etc/pf.conf`. Si la règle correspondant au paquet ne paraît pas évidente, on peut la retrouver en étudiant la sortie de la commande `pfctl -vvs rules` (ou, équivalente mais plus courte, `pfctl -vvsr`).

Dans l'exemple que nous venons de donner, la dixième règle du jeu chargé semble être une règle attrape-tout, qui correspond à la fois aux requêtes `IDENT` et aux recherches de noms de domaines. C'est le genre de sortie importante dans un cadre de débogage. En fait, on doit garder ces données à portée de main pour rester au courant de tout ce qui se passe sur le réseau. Avec un petit effort et une lecture attentive des pages de manuel de `tcpdump`, on peut extraire toutes les données de journalisation dont on a besoin, quelles qu'elles soient.

Pour afficher en temps réel le trafic journalisé, on peut utiliser `tcpdump` pour lire ces informations directement depuis le périphérique de journalisation. Pour ce faire, on utilise l'option `-i` afin de spécifier sur laquelle `tcpdump` doit lire ; le reste des options à employer est déjà connu :

```
$ sudo tcpdump -nettti pflog0
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0, link-type PFLOG
```

```
Sep 13 15:26:52.122002 rule 17/(match) pass in on epic0: 91.143.126.48.46618 >
194.54.103.65.22: [|tcp] (DF)
Sep 13 15:28:02.771442 rule 12/(match) pass in on epic0: 194.54.107.19.8025 >
194.54.107.18.8025: udp 50
Sep 13 15:28:02.773958 rule 10/(match) pass in on epic0: 194.54.107.19.8025 >
194.54.103.65.8025: udp 50
Sep 13 15:29:27.882888 rule 10/(match) pass in on epic0: 194.54.107.19.29774 >
194.54.103.65.53:[|domain]
Sep 13 15:29:28.394320 rule 12/(match) pass in on epic0: 194.54.107.19.29774 >
194.54.107.18.53:[|domain]
```

Le format de sortie doit vous sembler familier. Cette séquence démarre par une connexion SSH. Les deux connexions suivantes sont des synchronisations `spamd`, suivies de recherches de noms de domaines. Si on laisse la commande se poursuivre, les lignes affichées vont défiler sur tout l'écran, mais on peut bien entendu rediriger les données vers un fichier ou un programme, pour traitement ultérieur.

> Dans certaines situations, on s'intéresse principalement au trafic émanant ou à destination d'hôtes spécifiques, ou encore correspondant à des critères plus fins que ceux des règles journalisées. C'est précisément dans ces cas-là que les fonctionnalités de filtrage propres à `tcpdump` sont utiles : elles peuvent servir à extraire les données nécessaires. Voir `man tcpdump` pour de plus amples détails.

Journaliser tous les paquets : log (all)

Pour la plupart des tâches de débogage et pour les petites tâches de supervision, journaliser uniquement le premier paquet d'une connexion suffit, car il fournit assez d'informations. Cependant, on peut souhaiter journaliser tous les paquets correspondant à une règle donnée : on procède alors en ajoutant l'option de journalisation `(all)` aux règles concernées. Dans notre jeu de règles minimal, cela donne :

```
block log (all) all
pass log (all) quick proto tcp from any to any port ssh keep state
```

Cela rend les journaux beaucoup plus verbeux. Pour illustrer le volume supplémentaire généré par `log (all)`, voici un autre fragment de jeu de règles, qui laisse passer les recherches de noms de domaines et les synchronisations d'horloge par réseau :

```
udp_services = "{ domain, ntp }"
pass log (all) inet proto udp from any to any port $udp_services
```

Le `log (all)` de la règle ci-dessus est illustré par la séquence suivante, où un serveur de noms russe envoie une requête de nom de domaine à l'un de mes serveurs :

```
$ sudo tcpdump -n -ttt -i pflog0 port domain
tcpdump: WARNING: pflog0: no IPv4 address assigned
tcpdump: listening on pflog0, link-type PFLOG
Sep 30 14:27:41.260190 212.5.66.14.53 > 194.54.107.19.53:[|domain]
Sep 30 14:27:41.260253 212.5.66.14.53 > 194.54.107.19.53:[|domain]
Sep 30 14:27:41.260267 212.5.66.14.53 > 194.54.107.19.53:[|domain]
Sep 30 14:27:41.260638 194.54.107.19.53 > 212.5.66.14.53:[|domain]
Sep 30 14:27:41.260798 194.54.107.19.53 > 212.5.66.14.53:[|domain]
Sep 30 14:27:41.260923 194.54.107.19.53 > 212.5.66.14.53:[|domain]
```

Cela produit six entrée au lieu d'une seule. Même si `tcpdump` filtre ici tout ce qui ne correspond pas au port domaine, il va sans dire qu'ajouter `log (all)` à une règle augmente considérablement la quantité de données journalisées. Si la capacité de stockage de la passerelle est limitée et que l'on doit journaliser tout le trafic, il faut prévoir un espace de stockage en conséquence.

Journaliser sur plusieurs interfaces pflog

Dans les versions de PF antérieures à OpenBSD 4.1, il n'y avait qu'une seule interface `pflog`. OpenBSD 4.1 introduisit la possibilité de cloner l'interface `pflog`. Cela signifie qu'on peut utiliser des commandes `ifconfig` pour créer plusieurs interfaces `pflog` en plus du `pflog0` par défaut. cela facilite notamment la journalisation, en permettant à chaque partie du jeu de règle de diriger son journal vers sa propre interface `pflog`. Les données résultantes peuvent ainsi être traitées séparément.

Les modifications à apporter à la configuration sont subtiles, mais efficaces. Pour journaliser sur plusieurs interfaces, on doit s'assurer que toutes les interfaces de journalisation utilisées par le jeu de règles sont créées avant le chargement des règles. Si le jeu de règles journalise sur une interface qui n'existe pas, les données de journalisation sont tout simplement perdues.

On ajoute les interfaces nécessaires grâce à la commande suivante :

```
$ sudo ifconfig create pflog1
```

que l'on répète autant de fois qu'il y a besoin d'interfaces. On précise ensuite pour chaque règle le périphérique de journalisation à côté du mot-clé `log`, comme ceci :

```
pass log (to pflog1) proto tcp from any to $emailserver port $email
pass log (to pflog1) proto tcp from $emailserver to any port smtp
```

Pour rendre cette configuration permanente sous OpenBSD, il faut créer autant de fichiers `hostname.pflogX` (où `X` est un nombre supérieur ou égal à `1`) que nécessaire, contenant simplement :

```
up
```

Sous FreeBSD, la configuration des interfaces `pflog` clonées intervient dans le fichier `/etc/rc.conf`, sous la forme :

```
ifconfig_pflog1="up"
```

Avertissement

NetBSD ne prend en charge le clônage d'interfaces `pflog` que depuis la branche 5.0, non encore sortie au moment de la traduction de cet ouvrage ; si vous utilisez un NetBSD d'une branche antérieure, cela ne fonctionnera pas. La marche à suivre est la même que pour FreeBSD.

Note du traducteur

L'auteur indique qu'à la date de sortie du livre ce n'était pas pris en charge. La page de manuel de NetBSD 5.0 http://netbsd.gw.com/cgi-bin/man-cgi?pflog+4+NetBSD-5.0, version actuellement en développement, signale que ça l'est maintenant, mais celle de la version stable actuelle, NetBSD 4.0.1 http://netbsd.gw.com/cgi-bin/man-cgi?pflog+4+NetBSD-4.0.1, n'en fait pas mention.

Comme nous l'avons vu au chapitre 6, rediriger les informations de journalisation des diverses parties du jeu de règles PF vers des interfaces dédiées à chacune de ces parties permet de les transmettre à des applications différentes. Cela simplifie leur traitement par des applications telles que `spamlogd`, car celles-ci reçoivent uniquement les informations pertinentes, pendant que d'autres applications traitent le reste des données journalisées.

Journaliser avec syslog, localement ou à distance

L'une des solutions pour limiter l'encombrement des journaux par les données de PF consiste à configurer la passerelle pour transférer les données vers les fichiers journaux sur une autre machine. Si l'on dispose déjà d'une infrastructure de journalisation centralisée, cela paraît plutôt logique, même si les mécanismes de journalisation de PF n'ont pas été réellement conçus pour fonctionner dans le mode traditionnel de journalisation du type `syslog`.

Si cet avertissement ne vous décourage pas de journaliser l'activité de PF via `syslog`, voici une courte recette expliquant comment procéder.

Petit avertissement concernant syslog

Tous les vieux routiers de BSD vous le diront : l'infrastructure traditionnelle de journalisation système, `syslog`, est un peu naïve dans sa gestion des données reçues par UDP en provenance des autres hôtes. En particulier, il existe un risque d'attaque de type déni de service, qui conduirait à saturer les disques. Il existe aussi un risque de perte d'informations de journalisation à cause d'une forte charge sur l'une ou l'autre des machines en jeu (le client et le serveur). Pour ces raisons, un système de journalisation distante ne doit être envisagé *que* si tous les hôtes impliqués communiquent via un réseau bien sécurisé et bien supervisé. Sur la plupart des systèmes BSD, `syslogd` est configuré par défaut pour refuser les données de journalisation d'autres hôtes. Si vous envisagez d'utiliser la journalisation `syslog` à distance, lisez la page de manuel de `syslogd` ; vous y trouverez des informations sur la marche à suivre pour activer l'écoute sur le réseau.

Dans une configuration ordinaire de PF, `pflogd` gère les données de journalisation et les copie dans le fichier journal. Dans une configuration où ces données doivent être stockées sur un système distant, il faut désactiver l'accumulation de données de `pflog`, en faisant pointer le fichier de journalisation de `pflog` vers `/dev/null`, via les options de démarrage du démon. Sous OpenBSD, c'est dans `rc.conf.local` :

```
pflogd_flags="-f /dev/null"
```

Sous FreeBSD et NetBSD, c'est la ligne `pflog_logfile=` du fichier `rc.conf` qu'il faut modifier de la même manière :

```
pflog_logfile="/dev/null"
```

Ensuite, il faut interrompre puis redémarrer le processus `pflogd` muni de ses nouveaux paramètres.

L'étape suivante consiste à vérifier que les données de journalisation, qui ne sont désormais plus collectées par `pflogd`, sont correctement transmises au système de traitement des journaux. Cela implique deux opérations : tout d'abord, configurer l'outil de journalisation pour qu'il transmette ses données aux système distant ; en second lieu, utiliser `tcpdump` et `logger` pour convertir les données et les injecter dans le système `syslog`.

Configurer `syslogd` pour qu'il traite les données nécessite simplement de choisir l'*infrastructure de journalisation* (*log facility*), le *niveau de journalisation* (*log level*) et enfin l'*action*, puis de reporter ces choix dans le fichier `/etc/syslog.conf`. Supposons que l'outil de journalisation de l'hôte `loghost.example.com` a déjà été configuré

pour recevoir les données, que l'infrastructure de journalisation choisie est `local2` et le niveau de journalisation, `info` ; dans ce cas, la ligne à écrire est :

```
local2.info                                @loghost.example.com
```

Après cette modification, on doit redémarrer `syslogd` afin qu'il lise les nouveaux paramètres. On doit ensuite demander à `tcpdump` de convertir les données de journalisation du périphérique `pflog` et de transmettre le résultat à `logger`, qui les enverra à son tour à l'outil de journalisation système. Nous réutilisons ici la commande `tcpdump` des exemples de base, avec quelques ajouts :

```
$ sudo nohup tcpdump -lnettti pflog0 | logger -t pf -p local2.info &
```

La commande `nohup` garantit que le processus continuera à tourner même s'il n'est contrôlé par aucun terminal ou qu'il se retrouve en tâche de fond (ce qui est le cas ici, car nous avons ajouté un caractère `&` à la fin de la commande). L'option `-l` précise que les lignes sorties par `tcpdump` seront mises en mémoire tampon, ce qui est utile ici pour les rediriger vers un autre programme. De l'autre côté du pipeline, `logger` ajoute l'étiquette `pf` pour identifier les données PF au sein du flux et règle la priorité de journalisation à `local2.info` via l'option `-p`. Le résultat est enregistré dans le fichier spécifié sur l'hôte de journalisation, fichier dont les entrées ressembleront à ceci :

```
pf: Sep 21 14:05:11.492590 rule 93/(match) pass in on ath0: 10.168.103.11.15842 >
82.117.50.17.80: [|tcp] (DF)
pf: Sep 21 14:05:11.492648 rule 93/(match) pass out on xl0: 194.54.107.19.15842 >
82.117.50.17.80: [|tcp] (DF)
pf: Sep 21 14:05:11.506289 rule 93/(match) pass in on ath0: 10.168.103.11.27984 >
82.117.50.17.80: [|tcp] (DF)
pf: Sep 21 14:05:11.506330 rule 93/(match) pass out on xl0: 194.54.107.19.27984 >
82.117.50.17.80: [|tcp] (DF)
pf: Sep 21 14:05:11.573561 rule 136/(match) pass in on ath0: 10.168.103.11.6430 >
10.168.103.1.53:[|domain]
pf: Sep 21 14:05:11.574276 rule 136/(match) pass out on xl0: 194.54.107.19.26281 >
209.62.178.21.53:[|domain]
```

Le fragment de journal affiché ici montre surtout une activité de surf sur le Web et les recherches de noms de domaines associées.

Simplifier les statistiques de chaque règle grâce aux labels

Les informations séquentielles obtenues par récupération des données de journalisation reflètent les mouvements de paquets au cours du temps. Dans d'autres contextes, la séquence ou l'historique des connexions sont moins importants que cer-

taines données statistiques telles que le nombre de paquets ou d'octets ayant correspondu à une règle depuis la dernière remise à zéro des compteurs.

Nous avons déjà vu en fin de chapitre 2 (section « Les statistiques de pfctl »), comment utiliser `pfctl -s all` pour visualiser les compteurs globaux et d'autres données. Si l'on souhaite voir plus de détails sur les données, on peut garder la trace des compteurs globaux, règle par règle, grâce à une autre option de la commande pfctl : `pfctl -vs rules`, qui affiche les statistiques relatives à chaque règle.

```
$ pfctl -vs rules
pass inet proto tcp from any to 192.0.2.225 port = smtp flags S/SA keep state
label "mail-in"
  [ Evaluations: 1664158   Packets: 1601986   Bytes: 763762591   States: 0 ]
  [ Inserted: uid 0 pid 24490 ]
pass inet proto tcp from 192.0.2.225 to any port = smtp flags S/SA keep state
label "mail-out"
  [ Evaluations: 2814933   Packets: 2711211   Bytes: 492510664   States: 0 ]
  [ Inserted: uid 0 pid 24490 ]
```

Ce format est simple à lire car il vise à donner une vue d'ensemble de l'activité du réseau.

Mais il n'est pas très adapté au traitement par un script ou un programme. Si l'on envisage d'extraire ces statistiques pour les passer à un script, afin de décider quelles règles valent la peine d'être suivies, alors les *labels* de règles sont l'outil adapté.

Les labels ne se limitent pas à identifier des règles pour traiter certaines sortes de trafic. Ils facilitent également l'extraction des statistiques du trafic. En attachant un label à certaines règle, on stocke des données supplémentaires sur les parties intéressantes du jeu ou celles qui demandent une attention spéciale. C'est, par exemple, une solution adaptée pour mesurer l'utilisation de la bande passante à des fins de facturation.

Nous attribuons ici les labels `mail-in` et `mail-out` aux règles `pass` respectives pour le trafic mail entrant et sortant :

```
pass log proto { tcp, udp } from any to $emailserver port smtp label "mail-in"
pass log proto { tcp, udp } from $emailserver to any port smtp label "mail-out"
```

Attendons un peu après avoir rechargé le jeu de règles, puis vérifions les données via la commande `pfctl -vsl` :

```
$ sudo pfctl -vsl
mail-in 1664158 1601986 763762591 887895 682427415 714091 81335176
mail-out 2814933 2711211 492510664 1407278 239776267 1303933 252734397
```

La sortie de cette commande montre le label, suivi du nombre de fois où la règle a été évaluée et du nombre total de paquets qu'elle a laissé passer. La troisième valeur correspond au nombre total d'octets qui sont passés, puis viennent le nombre de paquets passés en entrée, le nombre d'octets passés en entrée, le nombre de paquets passés en sortie et le nombre d'octets passés en sortie. Bien que cela manque de détails pour un être humain, ce format de liste est particulièrement adapté au traitement par un script, via un pipeline.

Les compteurs se mettent à tourner dès que le jeu de règles est chargé et jusqu'à être remis à zéro. Il est souvent intéressant d'installer une tâche `cron` qui lit les valeurs des labels à intervalles fixes et lesenregistre de façon permanente. Si on lance la collecte des données à intervalles fixes, il faut envisager de le faire plutôt via `pfctl -vszl`. L'option `-z` remet les compteurs à zéro après que `pfctl` les ait lus. L'outil de collecte des données rapatrie alors des *données périodiques*, c'est-à-dire les données accumulées depuis le dernier passage du script.

> Notez bien que les règles contenant des macros et des listes se développeront en plusieurs règles distinctes. Si l'on attache un label à des règles contenant des macros et des listes, le résultat chargé en mémoire sera composé de plusieurs règles portant toutes le même label. Cela peut brouiller un peu la sortie de `pfctl -vsl`, mais ce n'est pas vraiment un problème tant que l'application ou le script qui reçoit les données est capable de les interpréter correctement, en faisant la somme des labels identiques.

Quelques outils supplémentaires pour les journaux et les statistiques de PF

Il est également important de pouvoir observer l'état courant du système. Dans cette section, nous verrons quelques outils de supervision utiles dans ce cadre.

Il ne s'agit pas d'une liste exhaustive des outils permettant d'interagir avec une configuration PF. Tous les logiciels présentés ici sont disponibles via le système de paquetages d'OpenBSD et de FreeBSD (et de NetBSD, à une exception près).

Garder un œil sur ce qui se passe avec pftop

Si vous voulez surveiller ce qui entre sur votre réseau et en sort, l'outil `pftop` de Can Erkin Acar se révèle très utile. Comme son nom l'indique, `pftop` montre un instantané du trafic, dans un format très inspiré de celui de l'outil de visualisation de processus `top` (un outil traditionnel des systèmes UNIX).

Voici un exemple de sortie produite sur l'une de mes passerelles, réduite à son minimum :

```
pfTop: Up State 1-21/67, View: default, Order: none, Cache: 10000 19:52:28
PR  DIR SRC                  DEST                STATE  AGE  EXP   PKTS BYTES
tcp Out 194.54.103.89:3847   216.193.211.2:25    9:9     28    67  29   3608
tcp In  207.182.140.5:44870  127.0.0.1:8025      4:4     15  86400  30   1594
tcp In  207.182.140.5:36469  127.0.0.1:8025      10:10  418    75 810  44675
tcp In  194.54.107.19:51593  194.54.103.65:22    4:4    146  86395 158  37326
tcp In  194.54.107.19:64926  194.54.103.65:22    4:4    193  86243 131  21186
tcp In  194.54.103.76:3010   64.136.25.171:80    9:9    154     59  11   1570
tcp In  194.54.103.76:3013   64.136.25.171:80    4:4      4  86397   6   1370
tcp In  194.54.103.66:3847   216.193.211.2:25    9:9     28     67  29   3608
tcp Out 194.54.103.76:3009   64.136.25.171:80    9:9    214      0   9   1490
tcp Out 194.54.103.76:3010   64.136.25.171:80    4:4     64  86337   7   1410
udp Out 194.54.107.18:41423  194.54.96.9:53      2:1     36      0   2    235
udp In  194.54.107.19:58732  194.54.103.66:53    1:2     36      0   2    219
udp In  194.54.107.19:54402  194.54.103.66:53    1:2     36      0   2    255
udp In  194.54.107.19:54681  194.54.103.66:53    1:2     36      0   2    271
```

On peut trier les connexions suivant divers critères, notamment la règle PF, le volume, l'âge, les adresses source et de destination.

Ce programme n'est pas inclus dans le système de base, probablement parce que l'on peut faire appel à diverses options de `pfctl` pour obtenir des informations équivalentes (mais pas en temps réel). `pftop` est cependant disponible sous la forme d'un paquetage, sous le nom `sysutils/pftop`, dans les ports d'OpenBSD et de FreeBSD et dans le pkgsrc de NetBSD.

Afficher des graphiques de synthèse du trafic avec pfstat

Une fois que le système est actif, fonctionnel, et génère des données intéressantes, il est judicieux de représenter le trafic sous forme de courbes. Heureusement, il est assez simple d'y parvenir

`pfstat` constitue une solution populaire répondant à ce besoin. Il s'agit d'un petit utilitaire, développé par Daniel Hartmeier, visant à extraire et présenter les données statistiques générées automatiquement par PF. `pfstat` est disponible, sous forme de paquetage, dans le système de ports d'OpenBSD sous le nom `net/pfstat` et, dans le système de ports de FreeBSD et le pkgsrc de NetBSD, sous le nom `sysutils/pfstat`.

Le programme collecte les données (spécifiées dans le fichier de configuration) et les présente sous forme de fichiers graphiques JPG et PNG. Il peut y avoir deux types de sources de données : d'une part, le PF qui tourne sur le système local, via le péri-

phérique `/dev/pf` et, d'autre part, les données collectées provenant d'un ordinateur distant, sur lequel on aura pris soin de lancer le démon `pfstatd`.

La mise en place de `pfstat` demande surtout de choisir quels éléments des données PF doivent figurer dans les graphiques et sous quelle forme. Il faut ensuite écrire le fichier de configuration, puis lancer des tâches `cron` de collecte des données à exploiter.

Le programme est livré avec un exemple de fichier de configuration bien documenté, ainsi qu'une page de manuel courte mais utile. L'exemple de configuration fourni est une bonne base pour la rédaction d'un fichier personnalisé.Dans ce qui suit, je vous présente un petit exemple.

L'extrait de `pfstat.conf` ci-dessous est assez proche de celui disponible dans l'exemple de configuration.

```
collect 8 = global states inserts diff
collect 9 = global states removals diff
collect 10 = global states searches diff

image "/var/www/users/peter/bsdly.net/pfstat-states.jpg" {
       from 1 days to now
       width 980 height 300
       left
              graph 8 "inserts" "states/s" color 0 192 0 filled,
              graph 9 "removals" "states/s" color 0 0 255
       right
              graph 10 "searches" "states/s" color 255 0 0
}
```

Les valeurs `color` de l'exemple donnent un graphe à lignes rouges, bleues et vertes. Pour la version monochrome, nous avons modifié les couleurs pour avoir des niveaux de gris : `0 192 0` est devenu `105 105 105`, `0 0 255` est devenu `192 192 192` et `255 0 0` est devenu `0 0 0`.

La collecte, une fois par minute, des insertions, suppressions et recherches d'états, sur l'ensemble de la journée passée, produit une graphe ressemblant à celui de la figure 8-1. Les données représentées proviennent de l'une de mes passerelles les moins chargées.

Pour une vue plus détaillées des mêmes données, j'ai décidé de représenter les données correspondant à l'heure passée, dans une résolution légèrement plus grande. J'ai changé la période en `from 1 hours to now` et les dimensions en `width 600 height 300`, ce qui a produit le graphe de la figure 8-2.

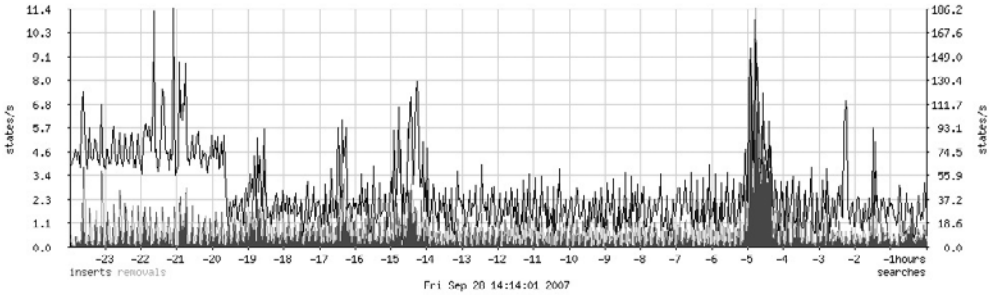


Figure 8–1 Statistiques de la table d'états, sur une durée de 24 heures

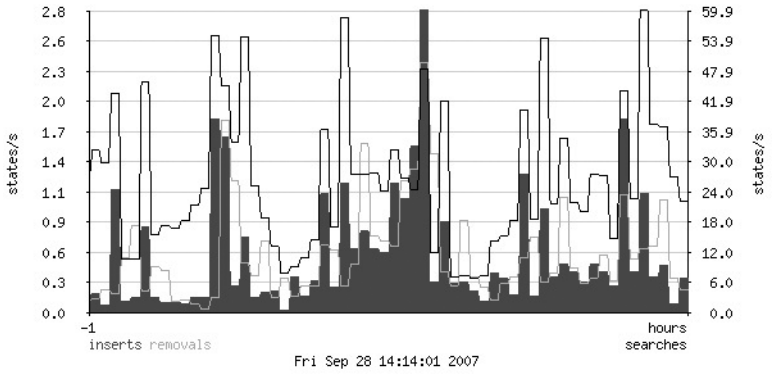


Figure 8–2 Statistiques de la table d'états, sur une durée d'une heure

La page principale de `pfstat`, qui se trouve à l'adresse http://www.benzedrine.cx/pfstat.html, propose un certain nombre d'autres exemples et des démonstrations (graphes mis à jour en temps réel, illustrant les données des passerelles du domaine benzedrine.cx). En lisant les exemples et en puisant dans la connaissance que vous avez de votre propre trafic, vous devriez être en mesure de créer des configurations `pfstat` adaptées aux besoins de votre site.

Collecter les données NetFlow avec pfflowd

NetFlow est une méthode de collecte et d'analyse de données réseau prise en charge par une grande famille d'outils d'enregistrement et d'analyse des données relatives aux connexions TCP/IP. Elle provient, à l'origine, de chez Cisco, mais divers équipements réseau l'ont adoptée au fil du temps pour la gestion et l'analyse.

Si vous utilisez déjà des outils NetFlow, alors il est intéressant – et peut-être même crucial – de savoir que les données de PF peuvent être mises à disposition des outils NetFlow via le paquetage `pfflowd`.

Le modèle de données NetFlow définit un *flux réseau* comme étant une séquence unidirectionnelle de paquets partageant les mêmes adresses de source et de destination, ainsi que le même protocole. Cela s'accorde très bien avec les informations d'état de PF et `pfflowd` est conçu pour enregistrer les changements d'état à partir du périphérique `pfsync` du système local. Une fois activé, `pfflowd` agit en tant que senseur NetFlow, convertissant les données `pfsync` au format NetFlow pour qu'elles soient transmises à un collecteur NetFlow sur le réseau.

L'outil `pfflowd` a été développé et est maintenu par Damien Miller ; il est disponible depuis son site web (http://www.mindrot.org/projects/pfflowd) et au travers des systèmes de paquetages d'OpenBSD et de FreeBSD sous le nom `net/pfflowd`. NetBSD ne prenant en charge `pfsync` que depuis très peu de temps, `pfflowd` n'a pas encore été porté et inclus dans le système pkgsrc.

Les outils SNMP et les MIB SNMP liés à PF

Le protocole de gestion simple du réseau (SNMP, pour *Simple Network Management Protocol*) a été conçu pour permettre aux administrateurs réseau de collecter et superviser les données clés sur le fonctionnement de leurs systèmes et les éventuels changements de configuration de plusieurs nœuds d'un système centralisé.

Le protocole fut inauguré par la RFC 1067 en août 1988 et il en est à présent à sa troisième version, définie dans les RFC 3411 à 3418. Le protocole SNMP est fourni avec une interface bien définie et une méthode permettant d'étendre la *base de gestion des informations* (*MIB* pour *Management Information Base*), qui détermine les périphériques et objets gérés.

La gestion par SNMP est devenue un composant tellement demandé dans les gros équipements réseau que, si vous administrez un grand réseau, vous avez probablement jeté un œil à cette section avant de décider d'acheter ce livre.

Les systèmes de gestion et de supervision réseau, propriétaires ou open source, prennent généralement en charge SNMP d'une manière ou d'une autre et c'est même une fonctionnalité de base de certains produits. Dans les systèmes de la famille BSD, la prise en charge de SNMP est généralement fournie sous la forme du paquetage `net-snmp`, intégrant les programmes nécessaires pour récupérer les données SNMP et collecter les données utilisées par les systèmes de gestion. Le paquetage est disponible dans OpenBSD et NetBSD sous le nom `net/net-snmp` et dans FreeBSD sous le nom `net-mgmt/net-snmp`.

Heureusement, il existe une extension de ce paquetage grâce à laquelle les données PF sont disponibles pour la supervision par SNMP. Joel Knight maintient les MIB concernant la récupération des données de PF, les MIB de CARP et ceux des senseurs du noyau d'OpenBSD. Ils sont téléchargeables sous forme de patches pour `net-snmp` depuis http://www.packetmischief.ca/openbsd/snmp.

Une fois le paquetage et son extension installés, les systèmes de supervision SNMP sont en mesure de surveiller les données de PF, avec tous les détails souhaités. Il faut de plus remarquer que le `bsnmpd` de FreeBSD comprend un module PF. Consultez voir la page de manuel de `bsnmpd` pour de plus amples informations.

Souvenez-vous : des données de journalisation pertinentes forment la base d'un débogage efficace

Dans ce chapitre, nous avons parcouru les bases de la collecte, de l'affichage et de l'interprétation des données d'un système où PF est activé. Il est utile, à plus d'un titre, de connaître les méthodes pour rechercher et exploiter les informations relatives au comportement du système.

Suivre l'état d'un système en cours de fonctionnement est déjà utile en soi, mais pouvoir lire et interpréter les données de journalisation est encore plus important lorsque l'on cherche à déterminer si le comportement du système est conforme aux spécifications. Les données de journalisation sont également utiles tracer l'effet de toute modification apportée à la configuration, notamment lorsque l'on affine les réglages du système pour obtenir des performances optimales.

Vérifier et affiner la configuration pour atteindre des performances optimales, à partir (entre autres) des données de journalisation, voilà pour l'essentiel ce qui nous attend au chapitre suivant.

Affiner les réglages pour les adapter parfaitement

9

Vous venez de passer un certain temps à concevoir votre réseau et à mettre en œuvre ces spécifications dans votre configuration de PF. Affiner vos réglages pour qu'ils vous conviennent parfaitement et corriger les quelques bugs restants peuvent s'avérer des tâches ardues dans certains cas. Dans ce chapitre, nous discuterons de quelques options et méthodes destinées à obtenir le résultat qui vous conviendra pleinement. Nous allons commencer par voir les options globales et quelques réglages qui peuvent grandement influencer le comportement de votre configuration.

Ce que vous pouvez modifier et ce à quoi vous ne devriez pas toucher

Les configurations réseau disposent, par essence, d'un très grand nombre de paramètres. Si vous parcourez la page de manuel de `pf.conf` ou d'autres sources de documentation de référence, vous vous sentirez peut-être submergé par le nombre d'options et de réglages qu'il est possible d'ajuster pour obtenir une configuration optimisée au maximum.

Il est important de garder à l'esprit qu'avec PF, *les valeurs par défaut sont correctes* dans la plupart des cas. Certains paramètres et variables demandent parfois à être ajustés ; d'autres devraient faire l'objet d'un avertissement bien visible, indiquant de ne les modifier que dans des circonstances vraiment inhabituelles. Ce chapitre évoque certains de ces cas de figure.

Commençons par voir les paramètres globaux que tout administrateur doit connaître, même s'il n'aura pas forcément à les modifier. Si vous lisez la page de manuel de `pf.conf`, vous découvrirez l'existence de quelques autres options, mais elles ne sont pas forcément pertinentes pour tester un réseau et affiner ses performances.

Dans le fichier `pf.conf`, les options globales, que l'on écrit sous la forme `set`*`optionparamètre`*, se placent après les définitions de macros, mais avant les règles de traduction ou de filtrage. Les sections suivantes en donnent quelques exemples et les expliquent.

L'option block-policy

Cette option détermine l'éventuel retour transmis par PF aux hôtes tentant d'initier des connexions qui se voient bloquées. Cette option peut prendre deux valeurs : `drop`, qui abandonne les connexions sans prendre la peine d'y répondre, et `return`, qui retourne un code d'état du type `Connection refused`.

La stratégie à suivre pour établir une politique de blocage a fait l'objet d'un grand nombre de discussions au fil des ans. Le réglage par défaut de l'option `block-policy` est `drop`, ce qui signifie que le paquet est silencieusement abandonné sans le moindre retour. Cela augmente cependant la probabilité de voir l'expéditeur tenter à nouveau d'envoyer les paquets pour lesquels ils n'a pas reçu d'accusé de réception, au lieu d'abandonner. PF continuera donc à bloquer les paquets jusqu'à ce que le délai limite soit écoulé. À moins d'avoir une bonne raison de ne pas modifier cette option, il est recommandé de passer `block-policy` à `return` :

```
set block-policy return
```

Cela signifie que la pile réseau de l'expéditeur reçoit un signal clair indiquant que la connexion a été refusée. Notez également que ce réglage définit le comportement par défaut de la politique de blocage et ce, au niveau *global*. Si nécessaire, on peut toujours faire varier le type de blocage au niveau des règles.

On peut, par exemple, modifier le jeu de règles de protection contre les attaques par force brute (présenté dans la section « Garder les méchants à distance ») en appliquant une stratégie `block-policy return` tout en utilisant `block drop quick from <bruteforce>` : de cette manière, on fait perdre leur temps aux personnes malveillantes qui s'entêtent après avoir été ajoutées à la table `<bruteforce>`. On peut aussi, au niveau de l'interface externe, abandonner le trafic provenant d'adresses non routables.

L'option skip

L'option `skip` permet d'exclure des interfaces données de tout traitement PF, ce qui revient à une règle `pass` totale pour une interface, comme `pass on $int_if`. On emploie souvent l'option `skip` pour désactiver le filtrage sur l'interface de bouclage (*loopback*) où, dans la plupart des cas, le filtrage n'est ni pratique, ni significatif d'un point de vue sécurité :

```
set skip on lo0
```

En fait, il n'est quasiment jamais utile de filtrer l'interface de bouclage et cela peut donner des résultats étranges avec bon nombre de programmes et de services courants. `skip` est désactivé par défaut, ce qui signifie que toutes les interfaces configurées sont susceptibles d'être filtrées par PF. En plus de simplifier légèrement le jeu de règles, activer l'option `skip` sur les interfaces où l'on ne compte pas filtrer implique un léger gain de performance.

L'option state-policy

L'option `state-policy` spécifie la manière dont PF fait correspondre les paquets à la table d'états. Les valeurs possibles sont `floating` et `if-bound`. Les différences entre les deux tiennent au traitement des paquets ultérieurs, après la création d'une entrée dans la table d'états.

Avec la politique `floating` par défaut, le trafic peut correspondre à un état sur toutes les interfaces et pas seulement l'interface où l'état a été créé. Avec la politique `if-bound`, le trafic ne peut correspondre qu'au niveau de l'interface où l'état a été créé. Le trafic des autres interfaces ou groupes d'interfaces ne correspondra pas à l'état existant. Comme l'option `block-policy`, cette option spécifie la politique *globale* de correspondance d'état. On peut néanmoins contourner la politique d'état règle par règle. Par exemple, dans un jeu de règles où s'applique la politique par défaut, `floating`, on peut avoir une règle comme celle-ci :

```
pass out on egress inet proto tcp to any port $allowed modulate state
(if-bound)
```

Avec cette règle, tout trafic de retour devra passer sur la même interface que celle où l'état a été créé, pour correspondre à l'entrée de la table d'états.

> **Avertissement**
>
> Les cas où `state-policy if-bound` est utile sont suffisamment rares pour que l'on recommande de conserver le réglage par défaut.

L'option timeout

L'option `timeout` définit les délais pour diverses interactions avec les éléments de la table d'états. La majorité des paramètres est constituée de valeurs spécifiques à un protocole, mesurées en secondes et préfixées des chaînes `tcp.`, `udp.`, `icmp.` et `others.`. Les valeurs `adaptive.start` et `adaptive.end` décrivent quant à elles le nombre d'entrées de la table d'états.

> **Avertissement**
>
> Ces options peuvent être utilisées dans le cadre d'une optimisation des performances, mais modifier les réglages d'un protocole crée un risque important d'abandon prématuré ou de blocage pur et simple de connexions valides mais inactives.

Les options de délais les plus susceptibles d'être modifiées sont les suivantes :

Tableau 9–1 Options de timeout les plus couramment modifiées

`adaptive.start` et `adaptive.end`	Ces valeurs fixent les limites d'abaissement des valeurs, une fois que le nombre d'entrées a atteint la valeur `adaptive.start`. Quand le nombre d'états atteint `adaptive.end`, tous les délais sont remis à zéro, ce qui fait immédiatement expirer tous les états. Les valeurs par défaut sont, respectivement, `6000` et `12 000`. Ces réglages sont intimement liés aux paramètres de réserve mémoire, que l'on fixe via l'option `limit` expliquée ci-après.
`interval`	Cette valeur décrit le délai, en secondes, entre les purges des états et des fragments expirés. La valeur par défaut est `10` secondes.
`frag`	La valeur `frag` décrit la durée de conservation d'un fragment sous forme désassemblée avant sa destruction. La valeur par défaut est `30` secondes.
`src.track`	Si la valeur `src.track` est définie, elle décrit la durée de conservation des données de retraçage des sources après expiration du dernier état. La valeur par défaut est `0` secondes.

Vous pouvez inspecter les réglages effectifs de tous les paramètres de l'option `timeout` grâce à la commande `pfctl -s timeouts`. La commande ci-dessous montre les valeurs par défaut d'un système en cours de fonctionnement :

```
$ sudo pfctl -s timeouts
tcp.first                  120s
```

```
tcp.opening               30s
tcp.established        86400s
tcp.closing              900s
tcp.finwait               45s
tcp.closed                90s
tcp.tsdiff                30s
udp.first                 60s
udp.single                30s
udp.multiple              60s
icmp.first                20s
icmp.error                10s
other.first               60s
other.single              30s
other.multiple            60s
frag                     30s
interval                 10s
adaptive.start         6000 states
adaptive.end          12000 states
src.track                 0s
```

L'option limit

L'option `limit` fixe la taille des réserves de mémoire que PF utilise pour ses tables d'états et d'adresses. Il s'agit de limites dures, qu'il peut être nécessaire d'affiner. Si le réseau est trop chargé pour les valeurs par défaut ou si la configuration requiert de grandes tables d'adresses (ou, tout simplement, un grand nombre de tables), alors cette section est importante.

Une remarque préliminaire à garder en tête : la quantité de mémoire disponible *via* ces réserves est prise sur l'*espace mémoire du noyau* ; elle dépend donc de la quantité de mémoire disponible pour le noyau. Le noyau alloue une certaine quantité de mémoire, fixée au démarrage du système, pour son propre usage ; cependant, la mémoire noyau n'étant jamais mise en swap, la quantité qu'il s'alloue ne peut ni atteindre ni dépasser la quantité de mémoire physique dont dispose le système. Si cela arrivait, il ne resterait plus de mémoire pour les programmes de l'espace utilisateur. L'exacte quantité de mémoire en réserve dépend de la plate-forme matérielle et de diverses variables difficiles à prédire car spécifiques au système.

Pour l'architecture i386, le maximum se trouve entre 768 Mo et 1 Go ; divers facteurs interviennent, notamment le nombre et la nature des périphériques mémoire attachés au système. La quantité réellement disponible pour allocation aux réserves découle de ce total, mais elle dépend encore d'un certain nombre de paramètres spécifiques au système.

On peut visualiser les réglages effectifs de ces limites en utilisant `pfctl -sm`. Cette commande produit généralement un résultat comparable à :

```
$ sudo pfctl -sm
states        hard limit    10000
src-nodes     hard limit    10000
frags         hard limit     5000
tables        hard limit     1000
table-entries hard limit   200000
```

Pour modifier ces valeurs, on édite le fichier `pf.conf` pour y renseigner de nouvelles valeurs limites. Par exemple, pour élever la limite dure du nombre d'états à `25 000` et du nombre d'entrées des tables à `300 000`, on doit insérer les lignes suivantes :

```
set limit states 25000
set limit table-entries 300000
```

On peut également fixer plusieurs paramètres à la fois en les plaçant entre accolades, comme ceci :

```
set limit { states 25000, src-nodes 25000, table-entries 300000 }
```

Au final, il est fortement déconseillé de modifier ces limites. Toutefois, si cela s'avère nécessaire, il faut surveiller attentivement les fichiers journaux pour identifier le moindre impact négatif ou problème de mémoire disponible. Pour cela, il peut être intéressant d'élever le niveau de débogage.

L'option debug

L'option `debug` détermine les informations d'erreur que PF génère au niveau du journal `kern.debug`. La valeur par défaut est `urgent`, qui signifie que seules les erreurs sérieuses sont journalisées. Les autres réglages possibles sont `none` (aucun message), `misc` (qui est un peu plus détaillé que `urgent`) et `loud` (qui produit des messages pour la plupart des opérations). Après avoir laissé tourné ma passerelle personnelle au niveau `loud` pendant un certain temps, voici à quoi ressemblait mon fichier `/var/log/messages` :

```
$ tail -f /var/log/messages
Oct 4 11:41:11 skapet /bsd: pf_map_addr: selected address 194.54.107.19
Oct 4 11:41:15 skapet /bsd: pf: loose state match:
 ➥ TCP 194.54.107.19:25 194.54.107.19:25
```

```
158.36.191.135:62458 [lo=3178647045 high=3178664421
 ➥ win=33304 modulator=0 wscale=1]
[lo=3111401744 high=3111468309 win=17376 modulator=0 wscale=0]
 ➥ 9:9 R seq=3178647045
(3178647044) ack=3111401744 len=0 ackskew=0 pkts=9:12
Oct 4 11:41:15 skapet /bsd: pf: loose state match:
 ➥ TCP 194.54.107.19:25 194.54.107.19:25
158.36.191.135:62458 [lo=3178647045 high=3178664421
 ➥ win=33304 modulator=0 wscale=1]
[lo=3111401744 high=3111468309 win=17376 modulator=0 wscale=0]
 ➥ 10:10 R seq=3178647045
(3178647044) ack=3111401744 len=0 ackskew=0 pkts=10:12
Oct 4 11:42:24 skapet /bsd: pf_map_addr: selected address 194.54.107.19
```

De toute évidence, le niveau `loud` donne un niveau de détail où PF rapporte répétitivement les adresses IP pour l'interface dont il s'occupe à un instant précis. Entre les messages `selected address`, pour un même paquet, PF signale à deux reprises que son numéro de séquence est à l'extrême limite autorisée. Au premier regard, ce niveau de détail peut sembler trop précis, mais il représente parfois le meilleur moyen de diagnostiquer un problème et de vérifier le résultat d'une solution.

Notez bien que l'on peut régler cette option depuis la ligne de commande par `pfctl -x` suivi du niveau de débogage désiré. La commande `pfctl -x loud` donne le maximum d'informations, alors que `pfctl -x none` ne renvoie rien. Gardez à l'esprit que le réglage `loud` peut générer une grande quantité de données – et, dans des cas extrêmes, freiner les performances jusqu'à provoquer un auto-déni de service.

L'option ruleset-optimization

L'option `ruleset-optimization` règle le mode d'optimisation du jeu de règles. Sa valeur par défaut est `basic`, ce qui signifie qu'aucune optimisation automatique n'est effectuée. Si l'on inclut la ligne :

```
set ruleset-optimization basic
```

au fichier `pf.conf` et que l'on recharge la configuration, le jeu de règles fait l'objet de quelques traitements supplémentaires avant d'être chargé.

Si l'on active l'optimisation de niveau `basic`, alors la routine chargée de l'optimisation procède aux actions suivantes :

1. suppression des doublons dans les règles ;
2. suppression des règles qui sont des cas particuliers d'autres règles plus générales ; Mettons, par exemple, que vous ayez une macro `tcp_services = { ssh, www, https }`, combinée à la règle `pass proto tcp from any to self port`

`$tcp_services`. Ailleurs dans le fichier de configuration figure une autre règle, `pass proto tcp from any to self port ssh`. La seconde règle est clairement un cas particulier de la première et on peut donc les fusionner en une seule et même règle. On peut citer un autre cas courant entrant dans cette catégorie : une règle `pass` comme `pass proto tcp from any to int_if:network port $tcp_services`, combinée à d'autres règles `pass` identiques où les adresses cibles font toutes partie du rang `int_if:network`.

3. fusion de règles en tables si c'est possible et approprié ; habituellement, les optimisations de type règle vers table se font sur les règles qui autorisent, redirigent ou bloquent en se basant sur des critères identiques, sauf pour l'adresse source ou cible.
4. modification de l'ordre des règles afin d'améliorer les performances.

Si l'optimisation du jeu de règles est définie à la valeur `profile`, alors la routine responsable de l'optimisation analyse le jeu chargé par rapport au trafic réseau réel, afin de déterminer l'ordre optimal des règles `quick`.

On peut aussi régler la valeur de l'option d'optimisation depuis la ligne de commande :

```
$ sudo pfctl -o basic
```

Cet exemple active le mode d'optimisation `basic`.

> Puisque l'optimisation peut supprimer ou réordonner les règles, la signification de certaines statistiques (notamment le nombre d'évaluations des règles) peut varier de manière difficilement prévisible. L'effet est cependant négligeable dans la plupart des cas.

L'option optimization

L'option `optimization` définit les profils pour gérer les délais d'expiration des états. Les valeurs possibles sont `normal`, `high-latency`, `satellite`, `aggressive` et `conservative`. Il est recommandé de s'en tenir à la valeur par défaut, `normal`, à moins d'avoir des besoins très spécifiques.

Les valeurs `high-latency` et `satellite` sont synonymes et elles signifient que les états expirent plus lentement afin de compenser un délai de latence éventuellement important.

Le réglage `aggressive` fait expirer les états plus rapidement afin d'économiser la mémoire (attention toutefois au fait que cela peut conduire à abandonner des connexions valides mais inactives, si le système se trouve déjà proche de ses limites de charge et de trafic).

Enfin, le réglage `conservative` fait l'effort de conserver plus longtemps les états (et les connexions inactives), au prix d'une consommation de mémoire plus importante.

Nettoyer le trafic : scrub et antispoof

Les deux fonctionnalités que nous allons présenter dans cette section, `scrub` et `antispoof`, ont en commun de fournir une protection automatisée contre certaines incongruités potentiellement dangereuses dans le trafic de votre réseau. On parle souvent d'elles en les qualifiant d'outil d'« hygiène », simplement parce qu'elles assainissent considérablement le réseau.

scrub

Le mot-clé `scrub` active la normalisation de trafic. Avec `scrub`, les paquets fragmentés sont réassemblés et les fragments invalides, comme les fragments se chevauchant, sont ignorés. Ainsi, le paquet résultant est complet et sans ambiguïté. L'activation de `scrub` protège contre certains types d'attaques basés sur une gestion incorrecte des fragments de paquets.

> **Plus d'infos**
>
> Certaines techniques d'attaques célèbres, notamment plusieurs dénis de services (DoS) historiques, ont exploité des bugs dans la gestion des fragments ; ces bugs ont pu conduire à saturer la mémoire (ou d'autres ressources limitées) des victimes. L'un de ces exploits, ciblant la gamme de pare-feux PIX de chez Cisco, est décrit dans un article en ligne.
>
> - http://www.cisco.com/en/US/products/products_security_advisory09186a008011e78d.shtml

Il y a un certain nombre d'options supplémentaires, mais la forme la plus simple, que nous montrons ci-dessous, convient à la plupart des configurations :

```
scrub in all
```

L'activation de scrub empêche certains services de fonctionner, à moins d'activer certaines de leurs options. Nous pouvons citer l'exemple courant de NFS. En effet, certaines combinaisons de serveurs et de clients ne peuvent notoirement pas fonctionner si `scrub` est activé, à moins d'utiliser le paramètre `no-df`. Certaines combinaisons de services, de systèmes d'exploitation et de configurations réseau peuvent exiger des options plus exotiques encore. Pour déboguer un problème lié à `scrub`, on doit commencer par étudier la page de manuel de `pf.conf` et consulter les gourous des listes de diffusion adéquates.

ATTENTION Refonte de la directive Scrub entre OpenBSD 4.5 et 4.6

Entre OpenBSD 4.5 et OpenBSD 4.6, la directive scrub a fait l'objet d'une refonte, et elle est désormais intégrée au principal jeu de règles. L'activation de la normalisation de paquets se fait donc maintenant en ajoutant le paramètre scrub (options) aux règles de filtrage et/ou via une règle `match`. Cela ajoute un degré de souplesse : on peut toujours gérer la normalisation de paquets de manière globale (règle match), mais on peut désormais aussi la gérer au cas par cas en ajoutant scrub à une règle `pass`. Par exemple, pour remplacer la règle `scrub in all no-df max-mss 1440` on peut utiliser ceci :

```
match in all scrub (no-df max-mss 1440)
```

Cette règle n'est plus une règle de normalisation de paquet mais bel et bien une règle de filtrage, il conviendrait donc de la déplacer afin de la mettre en tête des règles de filtrage (c'est-à-dire avant les règles `pass quick`) ; ce n'est cependant plus obligatoire car les développeurs d'OpenBSD ont décidé que l'ordre des règles ne devrait plus être respecté de manière stricte (l'option `set require-order` est désormais par défaut à `no`).

Henning Brauer explique dans le message accompagnant la modification apportée à PF que `match` est une nouvelle action, exactement comme `pass` et `block`. Elle s'utilise exactement comme ces dernières, mais elles ne modifient pas l'état `pass`/`block` d'un paquet, c'est-à-dire que `pass match` laisse passer le paquet, et que `block match` le bloque.

À chaque fois qu'une règle match correspond (et pas seulement quand c'est la dernière règle correspondante), les actions suivantes entrent en jeu :

- l'assignation à une file d'attente. Cela peut être écrasé plus loin, la dernière règle établissant une file étant la gagnante. Ce n'est pas équivalent à faire gagner la dernière règle correspondante, car si la dernière règle correspondante n'assigne pas à une file d'attente et que l'avant-dernière règle correspondante assignait à une file d'attente, alors cette assignation est prise en compte.
- l'assignation `r-table`, qui fonctionne comme l'assignation à une file d'attente ;

```
-set-tos, min-ttl, max-mss, no-df, random-id, reassemble tcp
```

fonctionnent toutes comme décrit ci-dessus ;

- la journalisation. Toute règle correspondant entraîne la journalisation du paquet. Cela signifie qu'un paquet unique peut être journalisé plusieurs fois (par exemple, dans le cas de plusieurs interfaces équipées de plusieurs receveurs, comme pflogd et spamlogd).

- http://marc.info/?m=123901961726016
- http://marc.info/?m=123902038227368

antispoof

Certaines actions courantes relatives à la gestion des paquets pourraient être écrites sous forme de règles PF, mais ces règles seraient longues, compliquées et sujettes à erreurs. `antispoof` a donc été implémenté pour un cas particulier de filtrage et de blocage. Ce mécanisme protège contre l'activité d'adresses IP usurpées ou forgées. Il procède notamment par blocage des paquets qui apparaissent sur des interfaces et se déplacent dans un sens logiquement impossible.

Avec `antispoof`, nous pouvons préciser que tout le trafic usurpé provenant du reste du monde doit être éliminé, de même que tout paquet usurpé prétendant venir de notre propre réseau (ce qui est peu probable). La figure 9-1 illustre ce concept.

Pour établir le type de protection dépeinte dans le diagramme, spécifions `antispoof` pour les deux interfaces du réseau illustré, grâce à ces deux lignes :

```
antispoof for $ext_if
antispoof for $int_if
```

Ces lignes seront développées en règles complexes. La première, par exemple, bloque le trafic entrant dont la source semble appartenir au réseau directement connecté à l'interface définie, mais qui se présente sur une autre interface. `antispoof` n'est cependant pas conçu pour détecter à distance les adresses usurpées sur des réseaux qui ne sont pas directement connectés à la passerelle PF.

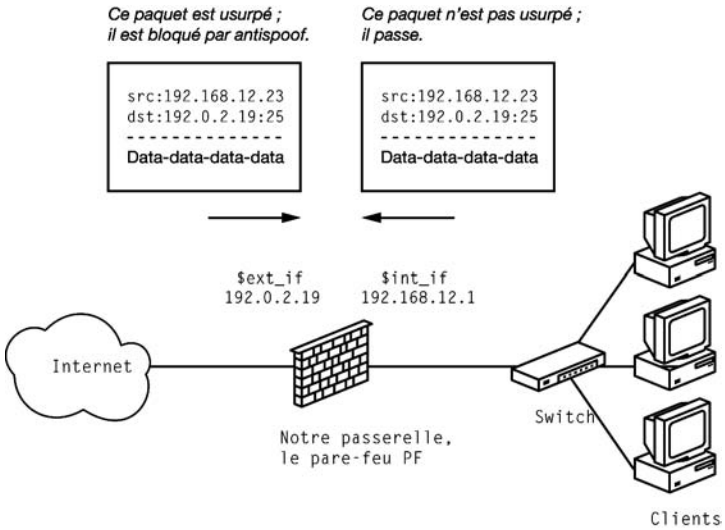


Figure 9–1
Antispoof élimine les paquets n'émanant pas du bon réseau.

Tester son réseau

Nous allons à présent nous concentrer sur les tests ; il est donc temps d 'exhiber les spécifications qui décrivent précisément le fonctionnement *théorique* de notre configuration. Nous avons mis de côté ces informations depuis quelques chapitres afin de présenter les fonctionnalités intéressantes, mais il est à présent essentiel de l'avoir à portée de main pour s'y référer. En fait, dès que l'on sort des considérations triviales, il est toujours judicieux d'avoir ce document sous la main.

Les spécifications sur lesquelles nous avons travaillé tout au long de ce livre reposent essentiellement sur l'idée d'un réseau centré autour d'une *passerelle*, connectée à Internet via `$ext_if`. Le *réseau local* est attaché à la passerelle via `$int_if` ; il contient des sta-

tions de travail et éventuellement un ou plusieurs serveur(s) pour un usage local. Enfin, nous disposons d'une DMZ, connectée à `$dmz_if`, peuplée de serveurs ouverts au réseau local et à Internet. La figure 9-2 montre la disposition logique de notre réseau.

Les spécifications du jeu de règles associé correspondent aux éléments suivants :

- les machines situées à l'extérieur de notre réseau doivent avoir accès aux services hébergés par les serveurs de la DMZ, mais aucun accès au réseau local ;
- les machines de notre réseau local, attaché à `$int_if`, doivent avoir accès aux services hébergés par les serveurs de la DMZ, ainsi qu'à une liste définie de services extérieurs ;
- les machines de la DMZ doivent avoir accès à certains services réseau du monde extérieur.

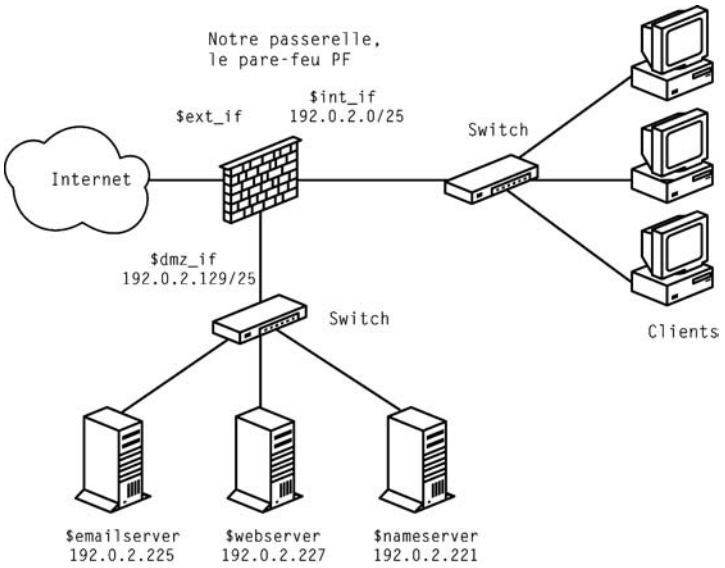


Figure 9–2
Un réseau avec des serveurs en DMZ.

Nous devons donc nous assurer que notre jeu de règles met réellement en œuvre ces spécifications et tester la configuration. Le Tableau 9.2 présente une batterie de tests adaptée.

Tableau 9–2 Exemple de séquence de tests d'un jeu de règles

Action de test	Résultat attendu
Tenter une connexion du réseau local vers chacun des ports autorisés sur les serveurs de la DMZ.	La connexion doit passer.
Tenter une connexion du réseau local vers chacun des ports autorisés sur des serveurs extérieurs au réseau.	La connexion doit passer.

Tableau 9–2 Exemple de séquence de tests d'un jeu de règles (suite)

Action de test	Résultat attendu
Tenter une connexion depuis la DMZ vers n'importe quel port du réseau local.	La connexion doit être bloquée.
Tenter une connexion depuis la DMZ vers chacun des ports autorisés sur des serveurs extérieurs au réseau.	La connexion doit passer.
Tenter une connexion depuis l'extérieur vers `$webserver` (dans la DMZ) sur chacun des ports de `$webports`.	La connexion doit passer.
Tenter une connexion depuis l'extérieur vers `$webserver` (dans la DMZ) sur le port 25 (SMTP).	La connexion doit être bloquée.
Tenter une connexion depuis l'extérieur vers `$emailserver` (dans la DMZ) sur le port 80 (HTTP).	La connexion doit être bloquée.
Tenter une connexion depuis l'extérieur vers `$emailserver` (dans la DMZ) sur le port 25 (SMTP).	La connexion doit passer.
Tenter une connexion depuis l'extérieur vers une ou plusieurs des machines du réseau local.	La connexion doit être bloquée.

Votre configuration peut nécessiter d'autres tests ou des variantes des tests proposés ici. Votre batterie de tests doit aussi vérifier la journalisation des paquets et des connexions. L'important est de décider, avant de mener les tests et pour chacun des cas, quels sont les résultats attendus.

En général, il faut conduire les tests à partir des applications supposées des utilisateurs (par exemple leurs navigateurs web, leurs clients de messagerie électronique), sur les systèmes d'exploitation qu'ils sont susceptibles d'utiliser. Les connexions doivent simplement réussir ou échouer, conformément aux spécifications. Si certains tests donnent des résultats inattendus, von passe alors au *débogage* du jeu de règles.

Déboguer son jeu de règles

Que se passe-t-il lorsqu'une configuration ne se comporte pas comme on l'attend ? Il est possible qu'il y ait une erreur dans la logique du jeu de règles et, si c'est le cas, on doit la trouver et la corriger. La traque de ces erreurs logiques peut prendre beaucoup de temps et nécessiter d'évaluer manuellement le jeu de règles, à la fois dans sa version stockée dans les fichier `pf.conf` et dans la version chargée en mémoire après développement des macros et optimisation.

Avant de plonger dans le jeu de règles, on peut facilement déterminer si c'est bien la configuration de PF qui pose problème. On désactive PF par la commande `pfctl -d` pour voir si le problème disparaît ; ce test peut nous éviter de nombreux problèmes.

Sur les listes de diffusion, groupes de nouvelles et forums, nous voyons souvent des utilisateurs tenant PF pour responsable de problèmes qui, au final, s'avèrent être des problèmes réseau élémentaires. En fait, les coupables sont plus souvent qu'à leur tour une interface réseau réglée avec de mauvais paramètres de duplex, un masque de sous-réseau erroné ou même un matériel réseau défectueux.

Si le problème persiste quand PF est désactivé, c'est donc qu'il ne peut venir de la configuration de PF. Dans ce cas, on doit se tourner vers le débogage des autres parties du réseau. Mais, avant d'ajuster la configuration PF, mieux vaut s'assurer que PF est bien activé et que son jeu de règles est bien chargé. On utilise pour cela la commande suivante :

```
$ sudo pfctl -si | grep Status
Status: Enabled for 20 days 06:28:24            Debug: Loud
```

Ici, `Status: Enabled` nous apprend que PF est activé. Essayons donc de visualiser les règles chargées au moyen d'une autre commande `pfctl` :

```
$ sudo pfctl -sr
scrub in all fragment reassemble
block return log all
block return log quick from <bruteforce> to any
anchor "ftp-proxy/*" all
```

Ici, `pfctl -sr` est équivalent à `pfctl -s rules`. La sortie obtenue sera probablement plus longue que celle-ci, mais c'est un bon exemple de ce à quoi on peut s'attendre quand un jeu de règles est correctement chargé. En phase de débogage, mieux vaut ajouter le drapeau `-vv` à la commande `pfctl`, de manière à voir les numéros des règles, ainsi que certaines informations supplémentaires utiles au débogage :

```
$ sudo pfctl -vvsr
@0 scrub in all fragment reassemble
  [ Evaluations: 67274995 Packets: 34231784 Bytes: 9800756925 States: 0     ]
  [ Inserted: uid 0 pid 1013 ]
@0 block return log all
  [ Evaluations: 618114 Packets: 15833 Bytes: 1444217 States: 0     ]
  [ Inserted: uid 0 pid 1013 ]
@1 block return log quick from <bruteforce:2> to any
  [ Evaluations: 618114 Packets: 13208 Bytes: 792140 States: 0     ]
  [ Inserted: uid 0 pid 1013 ]
@2 anchor "ftp-proxy/*" all
  [ Evaluations: 604906 Packets: 3498832 Bytes: 2803255822 States: 0     ]
  [ Inserted: uid 0 pid 1013 ]
```

On doit ensuite parcourir le jeu de règles étape par étape pour trouver les règles qui correspondent aux paquets suspects. Quelle est la dernière d'entre elles ? Si l'on trouve plusieurs règles, l'une d'entre elles est-elle une règle `quick` ? On doit suivre la trace de l'évaluation des règles, jusqu'à arriver au bout ou rencontrer une règle `quick` qui mette fin au processus. Si le jeu de règles conduit à un endroit qui n'est pas celui attendu, on a trouvé votre erreur de logique.

> **Rappel**
>
> Nous l'avions déjà indiqué dans un chapitre précédent : lorsqu'un paquet correspond à une règle `quick`, l'évaluation s'arrête et cette règle est immédiatement appliquée au paquet.

Les erreurs de logique d'un jeu de règles se répartissent en trois catégories :

- la règle ne correspond pas parce qu'elle n'est jamais évaluée ; c'est une règle `quick` placée en amont qui a obtenu en premier la correspondance et l'évaluation s'est arrêtée ;
- la règle est évaluée, mais elle ne correspond pas au paquet parce que l'un de ses critères est erroné ;
- la règle est évaluée, elle correspond, mais le paquet correspond également à une autre règle placée en aval ; c'est la dernière règle correspondante qui détermine le sort de la connexion.

Au chapitre 8, nous avons présenté `tcpdump`. Nous avons dit de lui qu'il est un excellent outil pour la lecture et l'interprétation des journaux de PF. Ce programme convient aussi très bien à la visualisation du trafic passant par une interface donnée. Tout ce que vous avez appris à propos des journaux de PF et des fonctionnalités de filtrage de `tcpdump` vous sera très utile quand vous voudrez savoir quel paquet atteint quelle interface.

Ici, nous utilisons `tcpdump` pour observer le trafic TCP de l'interface `xl0` (sauf le trafic SSH et le trafic SMTP) et pour afficher le résultat en mode très très détaillé (`-vvv`) :

```
$ sudo tcpdump -nvvvpi xl0 tcp and not port ssh and not port smtp
tcpdump: listening on xl0, link-type EN10MB
21:41:42.395178 194.54.107.19.22418 > 137.217.190.41.80: S [tcp sum ok]
3304153886:3304153886(0) win 16384 <mss 1460,nop,nop,sackOK,nop,wscale
0,nop,nop,timestamp
1308370594 0> (DF) (ttl 63, id 30934, len 64)
21:41:42.424368 137.217.190.41.80 > 194.54.107.19.22418: S [tcp sum ok]
1753576798:1753576798(0) ack 3304153887 win 5792 <mss
1460,sackOK,timestamp 168899231
1308370594,nop,wscale 9> (DF) (ttl 53, id 0, len 60)
```

Nous observons ici une connexion réussie vers un site web.

Mais il y a des problèmes plus intéressants à élucider, par exemple les connexions qui échouent ou, à l'inverse, qui sont autorisées, contrairement aux spécifications.

Dans ces cas-là, il ne reste qu'à retracer le parcours du paquet au travers de la configuration. Ici aussi, mieux vaut commencer par vérifier que PF est activé et que le résultat diffère si on le désactive. À partir du résultat de ce test, on suit les mêmes étapes d'analyse que nous venons de décrire. Quand on tient une théorie valable sur la circulation du paquet par rapport au jeu de règles et à l'interface réseau, on fait appel à `tcpdump` pour visualiser le trafic de toutes les interfaces, l'une après l'autre. Les fonctionnalités de filtrage de cet outil permettent d'afficher uniquement les paquets qui devraient correspondre au cas précis sur lequel on travaille, comme `port smtp` et `dst 192.0.2.19`.

On doit localiser l'endroit exact où les suppositions commencent à diverger avec la réalité du trafic réseau. On active la journalisation sur les règles impliquées, puis on laisse tourner `tcpdump` sur l'interface `pflog` concernée afin de voir quelles règles correspondent effectivement aux paquets.

Voilà les grandes lignes de la procédure de testfixées . Si vous êtes certain que le problème vient de votre configuration PF, tout se résume alors à trouver les règles qui correspondent et celle qui détermine finalement si le paquet passe ou est bloqué.

Connaître son réseau, garder le contrôle

Le thème récurrent de ce livre était de voir avec quelle facilité PF – et les outils qui lui sont liés – vous permettent de prendre le contrôle de votre réseau et de diriger son comportement suivant vos souhaits. En d'autres termes, l'objet de ce livre est de construire le réseau dont vous avez besoin.

Faire tourner un réseau peut être une tâche amusante et j'espère que vous avez apprécié cette présentation d'un outil que je considère comme le meilleur existant. Pour présenter PF, j'ai sciemment choisi de mettre en avant sa façon de penser et ses méthodes, via des configurations intéressantes et fonctionnelles, au lieu de faire de ce livre une *référence complète*. De toute façon, la référence complète existe déjà : ce sont les pages de manuel, mises à jour tous les six mois dans les nouvelles versions d'OpenBSD. Vous trouverez, dans les annexes suivant ce chapitre, une liste de références web et papier, utiles et toutes accompagnées de commentaires. Vous trouverez ensuite une note sur le matériel, une sur les divers types de prises en charge et une autre sur la marche à suivre pour interagir avec la communauté des utilisateurs et celle des développeurs.

Maintenant que vous connaissez un peu ce dont PF est capable, vous pouvez commencer à construire des réseaux suivant vos propres besoins. C'est entièrement à vous de jouer. Vous vous rendrez compte que vous êtes désormais capable de fouiller par vous-même dans les pages de manuel et de repérer seul l'information dont vous avez besoin. C'est là que tout cela devient amusant !

Ressources A

Ces ressources devraient vous aider à retirer le meilleur de votre configuration. Malgré mes efforts, il s'est avéré impossible de couvrir l'intégralité des possibilités offertes par PF ; j'espère que ces ressources combleront quelques lacunes ou donneront une perspective légèrement différente. Certaines d'entre elles sont même des lectures amusantes. Avec un peu de chance, elles resteront utiles et à jour.

Ressources Internet générales sur les réseaux et les systèmes BSD

Voici les ressources web généralistes citées tout au long du livre. Notez bien qu'il vaut mieux consulter les sites web des divers projets BSD pour trouver les informations les plus à jour.

L'*OpenBSD Journal* est particulièrement intéressant pour les utilisateurs d'OpenBSD. Il diffuse des nouvelles et articles concernant OpenBSD et les projets qui lui sont liés.

- http://undeadly.org

Note du traducteur

undeadly.org est en fait le successeur de deadly.org. Après que les mainteneurs du second aient jeté l'éponge, Daniel Hartmeier, l'auteur de PF, a repris le flambeau, avec un nouveau nom de domaine et un nouveau site web. Le traducteur de ce livre fait depuis peu partie de l'équipe de rédaction.

Le site web d'OpenBSD est la principale référence sur OpenBSD. Si vous utilisez ce système, consultez-le de temps à autre.

- http://www.openbsd.org

Vous trouverez un certain nombre de présentations et d'articles rédigés par les développeurs d'OpenBSD. C'est une bonne source d'informations sur l'avancée du développement d'OpenBSD.

- http://www.openbsd.org/papers

La *Documentation* et les *Questions Fréquemment Posées* d'OpenBSD sont davantage un guide d'utilisation qu'un traditionnel document de questions-réponses. C'est là que vous trouverez toutes les informations générales sur OpenBSD et les instructions étape par étape pour l'installation d'un système.

- http://www.openbsd.org/faq/fr/index.html

PF : Le Filtre de Paquets d'OpenBSD est la documentation officielle de PF, maintenue par l'équipe d'OpenBSD. Le guide d'utilisation de PF est mis à jour pour chaque version et c'est une ressource de référence, extrêmement utile pour les utilisateurs de PF.

- http://www.openbsd.org/faq/pf/fr/index.html

La présentation animée par Bob Beck au NYCBUG 2006, intitulée « *PF. It's not just for firewalls anymore* », couvre les fonctionnalités de redondance et de fiabilité de PF, avec des exemples réels tirés du réseau de l'Université d'Alberta (Canada).

- http://www.ualberta.ca/~beck/nycbug06/pf

Les pages que Daniel Hartmeier a consacrées à PF sur son site web personnel contiennent des liens vers vers d'autres ressources web.

- http://www.benzedrine.cx/pf.html

L'article présenté par Daniel Hartmeier à la conférence USENIX 2002, intitulé « *Design and Performance of the OpenBSD Stateful Packet Filter (PF)* », décrit la conception et l'implémentation initiale de PF.

- http://www.benzedrine.cx/pf-paper.html

La série de trois articles publiés en septembre 2006 par Daniel Hartmeier sur *undeadly.org*. Ces articles étaient prévus à l'origine pour être des chapitres d'un livre qui fut malheureusement annulé. Ces trois articles sont respectivement intitulés « *PF: Firewall Ruleset Optimization* », « *PF: Testing Your Firewall* » et « *PF: Firewall Management* ». Ils couvrent leurs sujets respectifs en détail, tout en restant très lisibles.

- http://undeadly.org/cgi?action=article&sid=20060927091645
- http://undeadly.org/cgi?action=article&sid=20060928081238
- http://undeadly.org/cgi?action=article&sid= 20060929080943

La RFC 1631, « The IP Network Address Translator (NAT) », datée du mois de mai 1994, est la première partie des spécifications de la traduction d'adresses, qui s'est avérée plus dure à cuire que ses auteurs l'avaient apparemment voulu. C'est toujours une ressource importante pour comprendre le fonctionnement du système NAT, même si elle a été largement supplantée par la RFC 3022, datée du mois de janvier 2001.

- http://www.ietf.org/rfc/rfc1631.txt?number=1631

La RFC 1918, « Address Allocation for Private Internets », datée du mois de février 1996[1], est la deuxième pièce du puzzle constitué par le système NAT et l'allocation de l'espace d'adressage privé et non routable. Cette RFC décrit les motivations conduisant à allouer un espace d'adressage privé et non routable, et définit ces espaces. La RFC 1918 a été reconnue comme pratique à suivre (*Best Current Practice*).

- http://www.ietf.org/rfc/rfc1918.txt?number=1918

1. NdT : On peut trouver une traduction de cette RFC à l'adresse http://abcdrfc.free.fr/rfc-vf/rfc1918.html.

Exemples de configuration

Un certain nombre de personnes ont eu la gentillesse de partager leur expérience et de mettre à la disposition des internautes des exemples de configuration. Voici quelques-uns de mes préférés.

L'article « The Six Dumbest Ideas in Computer Security » de Marcus Ranum (septembre 2005) est depuis longtemps l'un de mes favoris. Cet article explore certains des défauts de compréhension les plus courants concernant la sécurité et leurs implications malencontreuses.

> http://www.ranum.com/security/computer_security/editorials/dumb/index.html

Le HOWTO « Transparent Packet Filtering with OpenBSD » de Nate Underwood, daté de 2002, illustre une configuration de pont filtrant.

> http://ezine.daemonnews.org/200207/transpfobsd.html

L'article « Monitoring Net Traffic with OpenBSD's Packet Filter » de Randal L. Schwartz présente un exemple réaliste de supervision de trafic et d'utilisation des labels pour facturation. Certains détails ont changé depuis, notamment en ce qui concerne les labels, mais l'article est toujours assez lisible et il présente bien plusieurs concepts importants.

> http://www.samag.com/documents/s=9053/sam0403j/0403j.htm

L'article « Brandvägg med OpenBSD » du groupe suédois Unix.se et ses exemples de configuration, notamment celles sur ALTQ, me furent plutôt utiles au début. Ce site rappelle en outre que les efforts bénévoles, par exemple les groupes d'utilisateurs locaux (*LUG*), peuvent être d'excellentes sources d'information.

> http://unix.se/Brandv%E4gg_med_OpenBSDpages Web

L'article du blog de Randal L. Schwartz daté du jeudi 29 janvier 2004 décrit comment il a résolu un problème ennuyeux, grâce à un usage astucieux d'ALTQ et de la prise d'empreinte de système d'exploitation.

> http://use.perl.org/~merlyn/journal/17094

L'article « Managing Traffic with ALTQ » de Kenjiro Cho est l'article initial sur ALTQ, qui décrit la conception et le début de son implémentation dans FreeBSD.

- http://www.usenix.org/publications/library/proceedings/usenix99/cho.html

L'article « Failover Firewalls with OpenBSD and CARP » de Jason Dixon, paru dans le numéro de mai 2005 de *SysAdmin Magazine* offre une vue d'ensemble de CARP et de `pfsync`, comprenant quelques exemples pratiques.

- http://www.samag.com/documents/s=9658/sam10505e.html

La présentation que Theo de Raadt a animée lors de l'OpenCON 2006, « Open Documentation for Hardware: Why hardware documentation matters so much and why it is so hard to get » fut une importante inspiration pour la note de l'annexe B concernant le matériel et les systèmes d'exploitation libres en général et OpenBSD en particulier.

- http://openbsd.org/papers/opencon06-docs/index.html

PF dans les autres systèmes BSD

PF a été porté d'OpenBSD vers les autres systèmes BSD. Bien que le but soit, bien évidemment, de minimiser par cet effort de portage le décalage entre les versions présentes dans ces systèmes et son développement dans OpenBSD, mieux vaut suivre la trace de l'intégration de PF dans ces systèmes.

La page web dédiée au projet d'intégration de PF dans FreeBSD (http://pf4freebsd.love2party.net) décrit les premières étapes et les objectifs du projet. De fait, cette page n'est plus vraiment à jour mais, avec un peu de chance, elle reviendra à la vie lorsque Max Laier se rendra compte qu'il est référencé dans un livre.

La page « PF Loadable Kernel Module for NetBSD 2 » (http://nedbsd.nl/~ppostma/pf) diffuse des patches et de la documentation pour le PF de NetBSD, comprenant certaines des fonctionnalités relativement récentes qui ne sont pas encore intégrées dans l'arbre des sources de NetBSD.

Livres sur BSD et sur les réseaux

En plus des ressources web, dont le nombre ne peut que croître sans limite, plusieurs livres peuvent s'avérer utiles, en accompagnement ou en complément à ce livre.

- Emmanuel Dreyfus, *BSD – Les dessous d'Unix*, deuxième édition (Eyrolles 2004)
- Jacek Artymiak, *Building Firewalls with OpenBSD and PF*, troisième édition (devGuide.net, 2003). C'est *le* livre traditionnellement recommandé sur PF ; il couvre en grand détail le PF d'OpenBSD 4.5/4.6.
- Michael W. Lucas, *Absolute OpenBSD* (No Starch Press, 2003). Rédigé à l'époque d'OpenBSD 3.4, cet ouvrage présente la plupart des fonctionnalités du système grâce à de nombreux exemples pratiques.
- Brandon Palmer et Jose Nazario, *Secure Architectures with OpenBSD* (Addison-Wesley, 2004). Ce livre fournit une vue d'ensemble des fonctionnalités d'OpenBSD, avec pour but de construire un système fiable et sécurisé. Il prend OpenBSD 3.4 comme version de référence.
- Douglas R. Mauro and Kevin J. Schmidt, *Essential SNMP*, deuxième édition. (O'Reilly Media, 2005). Comme son titre l'indique, c'est-là une référence essentielle sur SNMP.
- Jeremy C. Reed (editor), *The OpenBSD PF Packet Filter Book* (Reed Media Services, 2006). Ce livre se base sur le *Guide d'Utilisation de PF*, étendu pour couvrir FreeBSD, NetBSD et DragonFly BSD, avec quelques ajouts concernant certains outils tiers qui interagissent avec PF.

Ressources sur les réseaux sans fil

Le support de cours de Kjell Jørgen Hole est une excellente ressource pour comprendre les réseaux sans fil. C'est le support du cours que suivent les étudiants du Professeur Hole à l'Université de Bergen ; il est librement disponible et vaut vraiment le coup d'œil.

> http://www.kjhole.com/Standards/WiFi/WiFiDownloads.html

Pour se tenir au courant des avancées dans le monde des réseaux sans fil, le site WNN *Wi-Fi Net News* est un bon point de départ. On peut y trouver les articles sur la sécurité.

> http://wifinetnews.com/archives/cat_security.html

Une dernière ressource fortement recommandée en matière de sécurité des réseaux sans fil : « The Unofficial 802.11 Security Web Page ».

- http://www.drizzle.com/~aboba/IEEE

Ressources sur spamd et les listes grises

Si vous devez gérer des services de messagerie électronique (ou si vous pensez que vous y serez amené), vous avez probablement apprécié la description de `spamd`, du *tarpitting* et du système de listes grises. Si vous cherchez des informations en plus de celles des RFC, consultez les documents suivants :

Le site web greylisting.org contient une collection bien fournie d'articles sur le système de listes grises et les sujets liés, ainsi que SMTP de manière générale.

- http://www.greylisting.org

L'article « The Next Step in the Spam Control War: Greylisting » d'Evan Harris est l'article d'origine sur le système de liste grise.

- http://greylisting.org/articles/whitepaper.shtml

La présentation « OpenBSD spamd – greylisting and beyond », animée par Bob Beck pour le NYCBUG, explique le fonctionnement de `spamd` et décrit son rôle dans l'infrastructure de l'Université d'Alberta. Notez bien que, parmi les éléments présentés comme *travail futur*, de nombreux points ont en fait déjà été implémentés.

- http://www.ualberta.ca/~beck/nycbug06/spamd

« The Silent Network: Denying the spam and malware chatter using free tools » est le titre de l'article que j'ai présenté lors de la conférence BSDCan 2007. Il s'agit en fait d'une liste de bonne pratiques pour utiliser des listes grises, `spamd`, OpenBSD et divers autres outils libres, afin de gagner la bataille du spam et des logiciels malveillants.

- http://home.nuug.no/~peter/malware-talk/silent-network.pdf

Ressources web concernant ce livre

Pour trouver des actualités et des mises à jour à propos de ce livre, des documents à télécharger et les errata, commencez par consulter la page du livre sur mon site personnel, où je publierai des mises à jour et des ressources liées au livre ; en voici un lien direct : http://www.bsdly.net/bookofpf.

Je maintiens le manuscrit de mon didacticiel sur PF *Firewalling with OpenBSD's PF packet filter*, qui n'est autre que l'ancêtre de ce livre. La façon dont je procède est simple : je fais les mises à jour aux moments opportuns, habituellement quand je me rends compte que certaines fonctionnalités de PF ont été ajoutées ou modifiée, ce qui arrive en fait quand je me prépare pour une présentation ou une conférence. Ce didacticiel est disponible sous licence BSD et dans différents formats, sur mon site personnel à l'adresse http://home.nuug.no/~peter/pf. J'y mettrai en ligne des versions mises à jour au fil de mes apparitions lors de divers événements.

COMMUNAUTÉ Si vous avez aimé ce livre, achetez des CD OpenBSD et faites un don !

Si vous avez aimé ce livre ou l'avez simplement trouvé utile, n'hésitez pas à vous rendre sur la page des commandes du site web OpenBSD.org (à l'adresse http://www.openbsd.org/orders.html) pour acheter un CD. À défaut d'acheter, vous pouvez soutenir le travail de développement du projet OpenBSD par un don, en passant par la page adéquate que vous trouverez à l'adresse ;

- http://www.openbsd.org/donations.html

Si vous représentez une organisation commerciale pour qui il est plus pratique de donner à une autre organisation formelle, vous pouvez contacter la Fondation OpenBSD. Il s'agit d'une organisation à but non lucratif sise au Canada, créée en 2007 dans l'unique but de collecter les dons institutionnels. Pour plus d'informations, vous pouvez consulter son site web à l'adresse ;

- http://www.openbsdfoundation.org.

Si vous avez trouvé ce livre lors d'une conférence, c'est que vous vous trouvez probablement à proximité d'un stand OpenBSD, où vous pourrez acheter des CD, des Tshirts et d'autres « goodies » encore.

Souvenez-vous : les logiciels libres et gratuits exigent du travail et de l'argent, pour leur développement et leur maintenance.

Remarque sur la prise en charge du matériel

B

Et qu'en est-il de la prise en charge matérielle ? J'entends souvent cette question et, habituellement, j'ai la réponse suivante : d'après mon expérience, OpenBSD et les autres systèmes libres tendent à fonctionner.

Mais, pour diverses raisons, il subsiste une idée reçue suivant laquelle passer à un système libre implique de lutter pour trouver du matériel compatible.

Il y a peut-être un fond de vérité à cela– je me souviens très bien d'avoir bataillé avec FreeBSD 2.0.5, qui parvenait à démarrer son processus d'installation sans jamais parvenir jusqu'au bout, car mon lecteur de CD-ROM n'était pas pris en charge.

Mais attendez : c'était en *juin 1995*.

Au cas où vous ne vous souviendriez pas de cette époque, c'était le temps où les lecteurs CD des PC étaient souvent fournis avec une interface IDE qui ne gérait pas complètement la norme IDE, et qui étaient attachés à la carte son (*paquetage multimédia*) avec laquelle ils étaient vendus. C'est le matériel dont je disposais à l'époque. Les systèmes BSD viennent du monde des *vrais ordinateurs*, c'est-à-dire des *serveurs* (pour parler en termes modernes), et les normes comme SCSI étaient mieux sprises en charge ; demander de l'aide dans un groupe de nouvelles BSD conduisait, à cette époque, à des commentaires dénigrant le « très mauvais matériel » – ce qui, en langage hacker de l'époque, signifiait « équipement trop mal conçu ou trop primitif pour être digne de l'intérêt d'un hacker ».

C'était ma foi plutôt vrai ; à cette époque, les PC bon marché n'intégraient généralement pas de circuit réseau non plus. La configuration d'un réseau impliquait typiquement de déplacer des cavaliers sur les circuits électroniques de la carte réseau, voire de la carte-mère elle-même, ou d'utiliser un logiciel bizarre et propriétaire. Et encore : vous aviez là la chance de disposer d'une interface Ethernet. Les connexions téléphoniques et RNIS étaient les types de connexion Internet les plus fréquents.

De nos jours, on peut raisonnablement s'attendre à ce que tous les composants importants d'un système fonctionnent sous OpenBSD. La configuration optimale peut demander un peu de prudence, mais mieux vaut prendre le temps de bien concevoir et planifier son infrastructure que d'agir impulsivement, non ?

Étude de cas : l'histoire d'un petit réseau sans fil

La prise en charge des réseaux sans fil dans OpenBSD et, plus généralement, dans les systèmes de la famille BSD, s'améliore de jour en jour. Cela ne veut pas dire pour autant que tout sera facile pour vous.

Voici l'histoire de la mise en place de mon réseau sans fil personnel. Elle commence par l'achat de deux cartes CNet CWP-854, qui auraient dû être prises en charge sous OpenBSD 3.7 par le nouveau pilote `ral`. Celle que j'ai branchée dans la machine Dell flambant neuve, équipée d'un système d'exploitation non libre, a fonctionné directement. Ma passerelle, qui tournait sans problème depuis la version 3.3, a cependant posé quelques problèmes. La carte était reconnue et configurée, mais quand la machine Dell tentait d'obtenir une adresse IP, la passerelle tombait pour cause de panique du noyau. Les détails complets sont disponibles publiquement sous la référence OpenBSD PR 4217. J'ai promis de tester la carte à nouveau, avec un nouveau snapshot, dès que pourrais l'insérer ailleurs[1].

J'ai ensuite décidé que je voulais essayer une carte `ath` et j'ai acheté une D-Link DWL-G520, que j'ai réussi à perdre pendant mon déménagement. J'ai ensuite investi dans une DWL-G520+, pensant que le signe « plus » était de meilleur augure. Malheureusement, il signifiait qu'elle embarquait un circuit complètement différent, le TI ACX111, qui coûte moins cher mais dont les concepteurs ne fournissent aucune documentation aux développeurs de logiciels libres. La magasin a heureusement accepté sans problème de la reprendre et de me la rembourser.

J'étais à ce moment plutôt frustré et j'ai entrepris de traverser toute la ville pour me rendre à un magasin qui avait plusieurs DWL-AG520 en stock. Elles étaient un peu plus chères, mais elles ont immédiatement fonctionné. Quelques semaines plus tard, c'était le tour de la G520, qui bien entendu a également fonctionné. Mon ordinateur portable (qui tournait à l'époque sous FreeBSD) intégrait une carte sans fil mini-PCI Realtek 8180, que je n'ai jamais réussi à faire fonctionner. J'ai fini par acheter une carte DWL-AG650 au format CardBus, qui fonctionne parfaitement avec le pilote `ath`.

1. Après plus de deux ans passés à repousser cela au lendemain, la carte est probablement toujours dans l'une des machines que nous avons emballées pour le déménagement. J'espère tomber à nouveau dessus avant que les cartes PCI ne deviennent obsolètes et inutiles !

Plus de deux ans après, le pilote `acx` (introduit dans OpenBSD 4.0) a apporté aux systèmes BSD la prise en charge des cartes basées sur les circuits ACX 1nn (ce pilote a été développé par ingénierie inverse). Beaucoup de temps et d'efforts ont été nécessaires et le développement s'est fait contre la volonté du constructeur, mais c'est là un thème que nous développerons en toute fin d'annexe lorsque nous traiterons des problèmes que rencontrent les développeurs dans la prise en charge matérielle. Ce qu'il faut retenir de tout cela, c'est qu'il est important d'établir un plan.

Choisir le bon matériel

Pour choisir le bon matériel, il faut surtout vérifier la correspondance entre ce qu'accepte votre système et les besoins de votre réseau. Il est toujours judicieux de vérifier les listes de compatibilité matérielle du site web de votre système d'exploitation. Vous pouvez aussi lire les pages de manuel ou utiliser des commandes `apropos` *`mot-clé`* (ou *`mot-clé`* est le type de périphérique que vous recherchez). Pensez également à fouiller dans les archives des listes de diffusions concernées, si vous souhaitez plus d'informations générales.

Vous devez cependant savoir que certains matériels sont fournis avec d'étranges restrictions ; je pense en particulier aux matériels qui dépendent d'un firmware chargé sur la carte. Il arrive souvent que le constructeur refuse que le firmware soit redistribué et les systèmes d'exploitation tels qu'OpenBSD ne peuvent donc pas l'empaqueter dans leurs versions.

On rencontre souvent ce type de problèmes avec les ordinateurs portables. Si vous avez cherché à en acheter un récemment, vous avez probablement jeté un œil à des ordinateurs embarquant du matériel réseau Intel PRO/Wireless 3945ABG 802.11a/b/g. Ce matériel est plutôt populaire et pris en charge par un grand nombre de systèmes d'exploitation ; c'est le cas d'OpenBSD, par le biais du pilote `wpi(4)`. Mais ce matériel ne fonctionne pas du tout si vous ne disposez pas, sur votre système, des bons fichiers de firmware ; Intel exige par ailleurs que vous les téléchargiez depuis son site web et que vous acceptiez pour cela les termes d'une licence.

Cela signifie que, malgré son excellente prise en charge des installations réseau, OpenBSD ne peut pas être installé via un réseau sans fil sur un ordinateur portable embarquant ce matériel Intel PRO/Wireless. Après discussion, Intel a refusé d'autoriser l'intégration des fichiers nécessaires dans le média d'installation.

Dans le cas du pilote et du firmware `wpi`, la lecture des pages de manuel révèle que le mainteneur du pilote a rassemblé les fichiers du firmware et en a fait un paquetage installable. Vous pouvez télécharger ce paquetage depuis son site personnel ; je ne

suis pas sûr que cela soit strictement légal vis-à-vis des termes de la licence, mais l'installation du paquetage résout le problème.

Ce circuit Intel PRO/Wireless n'est pas le seul périphérique souffrant de telles restrictions, mais c'est ce qui m'est arrivé avec mon ThinkPad R60, un excellent système par ailleurs. Notez bien que, dans OpenBSD, les pages de manuel des matériels pris en charge mais souffrant de ce type de restrictions mentionnent en général le problème ; dans certains cas, elles donnent même les adresses e-mail des personnes qui ont le pouvoir de faire changer la politique du constructeur.

Mon conseil est le suivant : si vous achetez en ligne, gardez les pages de manuel ouvertes dans un onglet ou une fenêtre ; si vous vous rendez dans un vrai magasin, dites bien aux vendeurs que vous utilisez un système BSD. Si vous avez un doute sur le matériel qu'ils essaient de vous vendre, voyez si vous pouvez emprunter une machine pour surfer sur le Web, afin de lire les pages de manuel et autres sources de documentation. Discuter avec les vendeurs peut aider à obtenir un remboursement au cas où le matériel ne fonctionnerait pas ; leur signaler que le matériel fonctionne fera un peu de publicité au système que vous utilisez.

Les problèmes que rencontrent les développeurs à propos de la prise en charge matérielle

Les systèmes tels qu'OpenBSD et les autres membres de la famille BSD, ne sont pas sortis tels quels de la tête d'un quelconque dieu (quoique certains prétendront que le processus ne fut pas si différent que cela). Ils sont le résultat d'années d'efforts, fournis par un certain nombre de personnes intelligentes et dévouées que l'on appelle développeurs.

Les développeurs sont des personnes hautement qualifiées et extrêmement dévouées, travaillant d'arrache-pied – la plupart sur leur temps libre – pour produire des résultats incroyables. Mais ils ne vivent pas dans une bulle et n'ont pas forcément accès à tout ce dont ils ont besoin. Ils ne disposent pas forcément du matériel qu'ils désirent voir pris en charge ou de la documentation nécessaires à son support ; parfois, ils n'ont ni l'un ni l'autre. La documentation peut être fournie sous condition de non divulgation, ce qui limite l'exploitation que les développeurs peuvent faire des informations[1].

1. Il s'agit-là d'un sujet récurrent ; voir, par exemple, la présentation animée par Theo de Raadt lors de l'OpenCON 2006, « Open Documentation for Hardware: Why hardware documentation matters so much and why it is so hard to get », que l'on trouve à l'adresse http://www.openbsd.org/papers/opencon06-docs/index.html.

Même sans documentation, les développeurs peuvent écrire des pilotes pour gérer du matériel, par le biais d'un processus nommé *ingénierie inverse* (*reverse engineering*). Ce processus est néanmoins très complexe, car il implique de deviner à l'aveuglette le fonctionnement du matériel, de développer un pilote et de le tester. Cela peut être amusant si vous savez comment faire, mais c'est une méthode qui présente deux inconvénients majeurs : d'une part, elle demande un temps conséquent et, d'autre part, elle peut poser des problèmes juridiques dans plusieurs endroits du monde, problèmes que seuls les avocats et les lobbyistes sont à-mêmes de comprendre.

Alors, que faire pour aider les développeurs à obtenir le matériel et la documentation dont ils ont besoin ?

Comment aider à la prise en charge du matériel ?

Si vous êtes capable de fournir du code de qualité, les projets BSD accueilleront probablement volontiers vos contributions. Si vous n'êtes pas développeur, vous pouvez quand même aider de différentes manières :

Achetez votre matériel auprès de constructeurs collaborant avec le monde open source.

Si c'est vous qui décidez ou qui recommandez les achats d'équipement de votre organisation, n'hésitez pas à dire à vos fournisseurs potentiels que leur niveau de bienveillance à l'égard de l'open source est un facteur que vous prenez en compte dans vos décisions.

Dites aux constructeurs de matériel ce que vous pensez de leur prise en charge (ou de leur manque de prise en charge) de votre système d'exploitation favori.

Certains constructeurs ont beaucoup aidé, en fournissant à la fois du matériel et de la documentation s'y rapportant. D'autres ont été moins coopératifs et certains ont même été franchement hostiles à l'égard des développeurs qui demandaientà accéder au matériel ou à de la documentation. Tous ces constructeurs doivent recevoir les retours qu'ils méritent. Écrivez-leur, dites-leur ce qui vous semble bien chez eux, ce qu'ils devraient améliorer et comment. Si, par exemple, un constructeur a refusé de fournir de la documentation ou alors l'a assortie d'une clause de confidentialité (*non-disclosure agreement*, NDA), peut-être qu'une lettre raisonnée et bien tournée, provenant d'un client potentiel, l'inciterait à coopérer.

Participez aux tests du système et suivez l'évolution des pilotes du matériel qui vous intéresse.

Qu'un pilote soit finalisé ou en développement, les développeurs aiment généralement recevoir des rapports leur disant comment se comporte leur code chez d'autres personnes. Les rapports indiquant que tout va bien sont toujours appréciés, mais les rapports de bugs le sont encore plus. C'est particulièrement vrai s'ils contiennent une description détaillée des problèmes rencontrés, car ils sont essentiels pour créer et maintenir un système de haute qualité.

Donnez du matériel ou de l'argent.

Les développeurs ont toujours besoin de matériel et d'argent pour travailler au quotidien au développement d'OpenBSD. Si vous avez les moyens de donner de l'argent ou du matériel, rendez-vous sur la page des dons sur le site web d'OpenBSD, à l'adresse http://www.openbsd.org/donations.html ; vous trouverez la liste des équipements dont le projet a besoin à l'adresse http://www.openbsd.org/want.html. Faire un don à OpenBSD aidera sûrement un peu PF mais, si vous préférez donner à FreeBSD, NetBSD ou DragonFly BSD, vous trouverez les informations nécessaires sur les sites web respectifs de chacun de ces projets.

Si vous représentez une organisation commerciale pour qui il est plus pratique de donner à une autre organisation formelle, vous pouvez contacter la Fondation OpenBSD. Il s'agit d'une organisation à but non lucratif sise au Canada, créée en 2007 dans l'unique but de collecter les dons institutionnels. Pour plus d'information, vous pouvez consulter son site web à l'adresse http://www.openbsdfoundation.org.

Quelle que soit la nature des relations que vous entretenez avec les systèmes BSD et votre matériel, j'espère que cela vous aidera à prendre les bonnes décisions en matière d'achats et à interagir avec les ordinateurs et les personnes.

Espérons que c'est aussi le cas du reste de ce livre.

Index

Cahiers de l'Admin

Packet Filter

Ce cahier de l'Admin aidera tous les administrateurs système et réseau qui n'ont pas abandonné leur quête d'un Graal moderne : la qualité réseau. Cet ouvrage démystifie Packet Filter, pare-feu filtrant des Unix libres BSD.

Écrit par un pédagogue hors pair, ce cahier vous mènera à travers les subtilités des flux de tout réseau. Il vous apprendra à créer des jeux de règles pour toutes sortes de trafics, tant sur un simple réseau local domestique que derrière une NAT ou à travers une DMZ, à lutter contre le spam à l'aide de listes grises, à équilibrer la charge et mettre en œuvre une qualité de service en gérant des files d'attente, et enfin à superviser votre réseau.

Packet Filter et Linux • Filtrage et blocage • Traduction d'adresses réseau (NAT) • Prise en charge d'IPv6 • **Première configuration** • Sous OpenBSD, FreeBSD et NetBSD • Listes et macros • Statistiques de pfctl • **Une passerelle simple avec NAT** • in, out et on • FTP au travers d'une NAT • Adresses routables : ftpsesame, pftpx et ftp-proxy • Dépannage du réseau • Ping et traceroute • Découverte de la MTU du chemin • **Réseau sans fil** • La norme IEE 802.11 • Filtrage d'adresses MAC • WEP et WPA • Choisir le bon matériel • Le jeu de règles PF pour le point d'accès • **IPsec et les solutions VPN** • SSH • Échange de clés par UDP (IKE/ISAKMP) • Filtrage sur l'interface d'encapsulation • Authpf, gardien du réseau sans fil • Une passerelle authentifiante • **Réseaux plus complexes** • Filtrage de services • Serveur web et serveur de messagerie • Adresses routables ou NAT • Séparation physique avec la DMZ • Répartir la charge • Hoststated • Puissance des étiquettes (tags) • Pare-feu ponté • Adresses non routables venues d'ailleurs • **Défense pro-active : SSH et listes noi-res, grises et blanches** • Nettoyer les tables avec pfctl • expiretable • Lutte contre les spams avec spamd • Configurer et suivre : spamd.conf, spamlogd, spamdb • Construire ses listes • Gérer les entrées piégées • Synchroniser les listes • Détecter les mauvais MX • **Files d'attente, calibrage et redondance** • Diriger le trafic avec ALTQ • Par priorité : priq. Allouer la bande passante par classe : cbq • Trafic indésirable : empreinte du système d'exploitation • Tolérance aux pannes : CARP et pfsync • **Journalisation, supervision et statistiques** • Pflog, syslog • Les labels. Pftop, pfstat, pfflowd, SNMP et MIB SNMP • **Affiner les réglages** • Options block-policy, skip, state-policy, timeout, limit, debug, ruleset-optimization, optimisation • Nettoyer le trafic : scrub et antispoof • Tester et déboguer son réseau • **Ressources, RFC** • Prise en charge du matériel.

Consultant et administrateur système, **Peter N. M. Hansteen** vit à Bergen (Norvège) et promeut les systèmes Unix libres depuis de nombreuses années. Il anime très souvent des conférences sur FreeBSD et OpenBSD. Son premier ouvrage, *Le livre de PF*, est une version largement améliorée du célèbre didacticiel sur PF qu'il maintient en ligne depuis 2005 sur son site web.

Maxime Derche collabore activement à la communauté OpenBSD francophone. Il utilise OpenBSD depuis sa version 3.8.

Code éditeur : G12516
ISBN : 978-2-212-12516-0

www.editions-eyrolles.com
Groupe Eyrolles | Diffusion Geodif | Distribution Sodis