OpenBSD

Langue : fr [équipes]
de en nl pl pt

[Index de La FAQ] [Suivant : Principes de Base]

PF : Le Filtre de Paquets d'OpenBSD

Table des Matières

Packet Filter (que nous appelerons désormais PF) est le système utilisé par OpenBSD pour filtrer le trafic TCP/IP et effectuer des opérations de Traduction d'Adresses IP ("NAT"). PF est aussi capable de normaliser, de conditionner le trafic TCP/IP, et de fournir des services de contrôle de bande passante et gestion des priorités des paquets. PF fait partie du noyau GENERIC d'OpenBSD depuis la version 3.0. Les précédentes versions d'OpenBSD utilisaient un ensemble logiciel pare- feu/NAT différent qui n'est plus supporté.

PF fût initialement développé par Daniel Hartmeier. Il est maintenant développé et maintenu par Daniel et le reste de l'équipe OpenBSD.

L'ensemble des documents listés dans la table des matières, disponible aussi au format PDF, est destiné à servir d'introduction générale au pare-feu PF tel qu'il est livré avec OpenBSD. Même s'il couvre les fonctionnalités majeures de PF, il est uniquement destiné à servir de complément aux pages du manuel; il ne les remplace en aucun cas.

Pour une couverture complète et approfondie de ce que peut faire PF, nous vous conseillons de lire d'abord la page du manuel pf(4).

De même que le reste de la FAQ, cet ensemble de documents est orienté vers les utilisateurs d' OpenBSD 3.8 (la version la plus récente disponible à ce jour). Etant donné que PF continue sans cesse de s'améliorer, il existe un certain nombre de changements et d'améliorations entre la version fournie avec 3.9-release et la version fournie avec OpenBSD-current ainsi que des différences entre la version 3.9 et les précédentes. Nous vous recommandons de lire les pages du manuel de la version d'OpenBSD que vous utilisez.

[Index de La FAQ] [Suivant : Principes de Base]


www@openbsd.org

$OpenBSD: index.html,v 1.32 2006/05/03 14:48:29 saad Exp $

[Index] [Suivant : Listes et Macros]

PF : Principes de Base

Table des Matières

Activation

Pour activer PF, ajoutez la ligne suivante :

```
pf=YES
```

au fichier /etc/rc.conf.local

Redémarrez votre système pour que les modifications soient prises en compte.

Vous pouvez aussi activer et désactiver PF en utilisant le programme pfctl(8) :

```
# pfctl -e
# pfctl -d
```

La première commande active PF (l'option "-e" correspond à "enable"). La seconde commande désactive PF (l'option "- d" correspond à "disable"). Ces commandes ne causent pas le chargement d'une base de règles. Cette dernière doit être chargée séparément avant ou après l'activation de PF.

Configuration

PF lit les règles de configuration à partir du fichier `/etc/pf.conf`, au démarrage de la machine. Ces règles sont chargées au démarrage par les scripts rc. `/etc/pf.conf` est le fichier par défaut chargé par ces scripts. C'est un fichier au format texte chargé et interprété par pfctl(8) puis inséré dans pf(4). Pour certaines applications, d'autres bases de règles peuvent être chargées à partir de fichiers différents durant la phase de démarrage. Comme n'importe quelle application Unix bien conçue, PF offre une grande flexibilité.

Le fichier `pf.conf` se compose de sept parties :

- **Macros :** Variables définies par l'utilisateur pouvant contenir des adresses IP, des noms d'interfaces, etc.
- **Tables :** Structures conçues pour contenir des listes d'adresses IP.

- **Options :** Diverses options pour contrôler le fonctionnement de PF.
- **Scrub :** Retraitement des paquets pour les normaliser et les défragmenter.
- **Gestion de La Bande Passante :** Gère la bande passante et la priorité des paquets.
- **Traduction ("Translation") :** Contrôle la Traduction d'Adresses IP et la redirection de paquets.
- **Règles de Filtrage :** Permet le filtrage sélectif ou le blocage des paquets lorsqu'ils traversent n'importe quelle interface.

A l'exception des macros et des tables, chaque section doit apparaître dans l'ordre précité dans le fichier de configuration. Cependant toutes les sections ne sont pas obligatoires.

Les lignes vides sont ignorées et les lignes commençant par # sont considérées comme des commentaires.

Contrôle

Après le démarrage, PF peut être contrôlé grâce au programme pfctl(8). Voici des exemples de commandes :

```
# pfctl -f /etc/pf.conf     Charge le fichier pf.conf
# pfctl -nf /etc/pf.conf    Analyse le fichier mais ne le charge pas
# pfctl -Nf /etc/pf.conf    Charge uniquement les règles NAT à partir du fichier
# pfctl -Rf /etc/pf.conf    Charge uniquement les règles de filtrage à partir du fichier

# pfctl -sn                 Affiche les règles NAT actuelles
# pfctl -sr                 Affiche les règles de filtrage actuelles
# pfctl -ss                 Affiche la table d'état actuelle
# pfctl -si                 Affiche les statistiques et les compteurs de filtrage
# pfctl -sa                 Affiche TOUT ce qu'on est capable d'afficher
```

Pour une liste complète de commandes, veuillez lire la page du manuel pfctl(8).

[Index] [Suivant : Listes et Macros]

---

www@openbsd.org

$OpenBSD: config.html,v 1.23 2006/08/08 07:14:20 saad Exp $

[Précédent : Principes de Base] [Index] [Suivant : Tables]

PF : Listes et Macros

Table des Matières

Listes

Une liste permet d'utiliser plusieurs critères similaires dans une même règle PF. Par exemple, plusieurs protocoles, plusieurs numéros de ports, plusieurs adresses IP, etc. Ainsi, au lieu d'écrire une règle de filtrage pour chaque adresse IP qui doit être bloquée, une seule règle peut être écrite en créant une liste avec les adresses IP à bloquer. Les listes sont définies en mettant des éléments entre accolades {}.

Quand pfctl(8) rencontre une liste durant le chargement de la base de règles, il crée de multiples règles : une règle pour chaque élément de la liste. Par exemple :

```
block out on fxp0 from { 192.168.0.1, 10.5.32.6 } to any
```

devient :

```
block out on fxp0 from 192.168.0.1 to any
block out on fxp0 from 10.5.32.6 to any
```

Plusieurs listes peuvent être utilisées dans une même règle. Elles ne sont pas limitées aux règles de filtrage :

```
rdr on fxp0 proto tcp from any to any port { 22 80 } -> \
   192.168.0.6
block out on fxp0 proto { tcp udp } from { 192.168.0.1, \
   10.5.32.6 } to any port { ssh telnet }
```

Les virgules entre les éléments d'une même liste sont optionnelles.

Attention aux expressions suivantes connues sous le nom de "listes négatives", elles constituent une erreur commune :

```
pass in on fxp0 from { 10.0.0.0/8, !10.1.2.3 }
```

A l'origine, le but recherché est d'appliquer la règle précitée à n'importe quelle adresse du réseau 10.0.0.0/8 à l'exception de 10.1.2.3. Mais en réalité, cette règle est interprétée de la façon suivante :

```
pass in on fxp0 from 10.0.0.0/8
pass in on fxp0 from !10.1.2.3
```

Ce qui correspond à accepter le traffic de n'importe quelle adresse. Pour éviter cela, utilisez une table.

Macros

Les macros sont des variables définies par l'utilisateur pouvant contenir des adresses IP, des numéros de ports, des noms d'interfaces, etc. Les macros peuvent réduire la complexité d'une base de règles PF et faciliter sa maintenance.

Les noms de macros doivent commencer par une lettre et peuvent contenir des lettres, des chiffres, et des barres de soulignement "_". Les noms des macros ne doivent pas être des noms réservés tels que `pass`, `out`, ou `queue`.

```
ext_if = "fxp0"

block in on $ext_if from any to any
```

L'exemple précédent montre comment créer une macro nommée `ext_if`. Quand une macro est appelée après avoir été créée, son nom est précédé d'un caractère $.

Les macros peuvent aussi représenter des listes tel que le montre l'exemple suivant :

```
friends = "{ 192.168.1.1, 10.0.2.5, 192.168.43.53 }"
```

Elles peuvent être définies de manière récursive. Vu qu'elles ne sont pas traduites dans des quotes "", la syntaxe suivante doit être employée :

```
host1 = "192.168.1.1"
host2 = "192.168.1.2"
all_hosts = "{" $host1 $host2 "}"
```

La macro `$all_hosts` correspond maintenant à la liste d'adresses IP {192.168.1.1 192.168.1.2}.

[Précédent : Principes de Base] [Index] [Suivant : Tables]


www@openbsd.org

$OpenBSD: macros.html,v 1.20 2006/08/08 07:14:20 saad Exp $

[Précédent : Listes et Macros] [Index] [Suivant : Règles de Filtrage]

PF : Tables

Table des Matières

Introduction

Une table est une structure utilisée pour regrouper un ensemble d'adresses IPv4 et/ou IPv6. Chercher une adresse IP dans une table est une opération très rapide qui consomme moins de mémoire et de temps processeur qu'une recherche dans une liste. C'est pourquoi une table est idéale pour contenir un grand nombre d'adresses IP vu que la consultation d'une table de 50 000 adresses ne prend qu'un tout petit peu plus de temps que celle d'une table de 50 adresses. Les tables peuvent être utilisées de l'une des manières suivantes :

- adresse source et/ou destination dans les règles de filtrage, scrub, NAT, et redirection.
- adresse de traduction dans les règles NAT.
- adresse de redirection dans les règles de redirection.
- adresse de destination dans les options des règles de filtrage `route-to`, `reply-to`, et `dup-to`.

Une table peut être créée dans `pf.conf` ou en utilisant pfctl(8).

Configuration

Dans `pf.conf`, les tables sont créées en utilisant la directive `table`. Les attributs suivants peuvent être spécifiés pour chaque table :

- `const` - le contenu de la table ne peut être modifié une fois la table créée. Lorsque cet attribut n'est pas spécifié, pfctl(8) peut être utilisé pour ajouter ou supprimer des adresses de la table à n'importe quel moment, même lorsque le système utilise un securelevel(7) supérieur ou égal à 2.
- `persist` - conduit le noyau à garder la table en mémoire même lorsqu'aucune règle ne s'y réfère. Sans cet attribut, le noyau supprimera automatiquement la table lorsque la dernière règle s'y référant aura été supprimée.

Exemple :

```
table <goodguys> { 192.0.2.0/24 }
table <rfc1918> const { 192.168.0.0/16, 172.16.0.0/12, \
   10.0.0.0/8 }
table <spammers> persist

block in on fxp0 from { <rfc1918>, <spammers> } to any
pass  in on fxp0 from <goodguys> to any
```

Les adresses peuvent aussi être spécifiées en utilisant l'opérateur de négation (ou "not"). Voici un exemple :

```
table <goodguys> { 192.0.2.0/24, !192.0.2.5 }
```

La table `goodguys` contient alors toutes les adresses dans le réseau 192.0.2.0/24 excepté 192.0.2.5.

Notez que les noms de tables sont toujours entourés des chevrons < et >.

Les tables peuvent aussi être peuplées à partir de fichiers texte contenant une liste d'adresses et de réseaux IP :

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

block in on fxp0 from <spammers> to any
```

Le fichier `/etc/spammers` doit contenir une liste d'adresses IP et/ou des blocs réseau en notation CIDR, une entrée par ligne. Toute ligne commençant par # est traitée comme un commentaire. Elle est donc ignorée.

Manipulation des Tables avec `pfctl`

Les tables peuvent être manipulées à la volée en utilisant pfctl(8). Par exemple, pour ajouter des entrées à la table <spammers> précédemment créée :

```
# pfctl -t spammers -T add 218.70.0.0/16
```

Ceci aura aussi pour effet de créer la table <spammers> si elle n'existe pas encore. Pour afficher le contenu d'une table :

```
# pfctl -t spammers -T show
```

L'argument `-v` peut aussi être utilisé avec `-T show` pour afficher des statistiques pour chaque entrée de la table. Pour supprimer des adresses de la table :

```
# pfctl -t spammers -T delete 218.70.0.0/16
```

Pour plus d'informations concernant la manipulation des tables avec `pfctl`, veuillez consulter la page de manuel pfctl(8).

Spécification des Adresses

Les hôtes peuvent être spécifiés par adresse IP ou par nom. Lorsque le nom est résolu en adresse IP, toutes les adresses IPv6 et IPv4 résultantes sont placées dans la table. Les adresses IP peuvent aussi être saisies dans la table en spécifiant le nom d'une

interface valide ou le mot-clé `self`. La table contiendra alors toutes les adresses IP affectées à cette interface ou à la machine (y compris les adresses de loopback), respectivement.

Il existe une restriction : lorsque des adresses sont spécifiées, les adresses `0.0.0.0/0` et `0/0` ne peuvent être utilisées avec des tables. L'alternative consiste à coder en dur ces adresses ou à utiliser une macro.

Recherche des Adresses Correspondantes

La recherche d'une adresse dans une table retournera l'entrée la plus proche de cette adresse. Ceci permet la création de tables telles que :

```
table <goodguys> { 172.16.0.0/16, !172.16.1.0/24, 172.16.1.100 }

block in on dc0 all
pass  in on dc0 from <goodguys> to any
```

Tout paquet arrivant par `dc0` verra son adresse source recherchée dans la table `<goodguys>`:

- 172.16.50.5 - l'entrée la plus proche est 172.16.0.0/16; le paquet a une entrée correspondante dans la table donc il passe
- 172.16.1.25 - l'entrée la plus proche est !172.16.1.0/24; le paquet a une entrée correspondante mais celle-ci est en négation (l'opérateur "!" est utilisé); le paquet n'a donc aucune entrée correspondante et sera bloqué
- 172.16.1.100 - l'entrée la plus proche est 172.16.1.100; le paquet a une entrée correspondante dans la table donc il passe
- 10.1.4.55 - n'a aucune adresse correspondante dans la table. Il sera bloqué

[Précédent : Listes et Macros] [Index] [Suivant : Filtrage de Paquets]

---

www@openbsd.org

$OpenBSD: tables.html,v 1.22 2006/10/29 10:58:53 jufi Exp $

[Section précédente : Tables] [Index] [Section suivante : Traduction des Adresses IP ("NAT")]

PF : Le Filtrage de Paquets

Table des Matières

Introduction

Le filtrage de paquets à l'aide de pf(4) consiste à autoriser ou bloquer le trafic réseau en fonction des propriétés des protocoles des couches 3 (IPv4 et IPv6) et 4 (TCP, UDP, ICMP, et ICMPv6). Les adresses et ports source et destination ainsi que les protocoles utilisés sont des critères fréquemment employés.

Les règles de filtrage énumèrent les critères auxquels doivent se conformer les paquets et spécifient les actions qui y sont associées : bloquer ou laisser passer. Ces règles sont évaluées de façon séquentielle de la première à la dernière (du haut vers le bas dans les fichiers de règles utilisés). Sauf utilisation du mot-clef `quick` dans l'une d'entre elles, chaque paquet est évalué à l'aide de *toutes* les règles avant qu'une décision finale ne soit prise. L'action (block ou pass) associée à la dernière règle dont les critères se rapportent au paquet traité est appliquée. La première règle est un `tout laisser passer` implicite de sorte que si aucune règle n'est applicable à un paquet, celui-ci est accepté (`pass`).

Syntaxe

L'écriture des règles obéit à la syntaxe *très simplifiée* suivante :

```
action [direction] [log] [quick] [on interface] [af] [proto protocol] \
   [from src_addr [port src_port]] [to dst_addr [port dst_port]] \
   [flags tcp_flags] [state]
```

action
: Ce mot-clef indique le type d'action associé à tout paquet correspondant aux critères contenus dans la règle. Les deux valeurs possibles sont `pass` et `block`. `pass` signifie que le paquet sera transféré au noyau pour être traité. Le blocage du paquet (`block`) sera fonction de la politique définie par les options `block-policy`. L'action associée par défaut peut être modifiée en spécifiant `block drop` ou `block return`.

direction
: Ce mot-clef spécifie le sens du trafic vu depuis l'interface réseau : celui-ci peut être entrant (`in`) ou sortant (`out`).

`log`
: La présence de ce mot-clef dans une règle déclenche la journalisation des paquets correspondants à la règle grâce à l'utilitaire pflogd(8). Si les options `keep state`, `modulate state`, ou `synproxy state` sont également renseignées, seul le paquet correspondant à l'ouverture de la session est conservé. Pour conserver tous les paquets d'une session, il faut utiliser le mot-clef `log (all)`.

`quick`

Si le mot-clef `quick` est utilisé dans une règle, celle-ci est considérée comme étant la dernière à prendre en compte et l'*`action`* spécifiée est prise.

`interface`

Ce mot-clef désigne l'interface sur laquelle le trafic est à analyser. Il est possible de grouper plusieurs interfaces. Dans ce cas, le groupe est désigné par le nom générique de l'interface non suivi d'un numéro. Par exemple : `ppp` ou `fxp`. La règle s'appliquera aux paquets traversant toute interface de type `ppp` ou `fxp` respectivement.

`af`

Il est possible de spécifier la famille d'adresses IP à laquelle appliquer une règle à l'aide de ce mot-clef. Les deux valeurs possibles sont `inet` pour les adresses IPv4 ou `inet6` pour IPv6. PF peut généralement déterminer à quelle famille appartient un paquet à partir de ses adresses source et destination.

`protocol`

Ce mot-clef identifie les protocoles de couche 4 utilisés. Il peut prendre les valeurs suivantes :

- `tcp`
- `udp`
- `icmp`
- `icmp6`
- tout protocole spécifié dans le fichier `/etc/protocols`
- le numéro d'un protocole compris entre 0 et 255
- un ensemble de protocoles passé dans une liste.

`src_addr`, *`dst_addr`*

Ces mots-clefs identifient respectivement les adresses source et destination du paquet telles que contenues dans son en-tête IP. Ces adresses peuvent être spécifiées :

- sous la forme d'adresse IPv4 ou IPv6.
- sous la forme d'un bloc d'adresses CIDR.
- sous la forme d'un domaine. Ce dernier fera l'objet d'une résolution DNS lors du chargement des règles par PF. La règle s'appliquera alors à tous les adresses du domaine.
- par l'intermédiaire du nom d'une interface réseau. Toute adresse IP affectée à cette interface sera valide pour la règle.
- par l'intermédiaire du nom d'une interface réseau suivi d'un *`/netmask`* (par exemple, `/24`). Toutes les adresses IP correspondant au bloc CIDR formé par l'adresse IP de l'interface et le masque de sous réseau seront valides pour cette règle.
- sous la forme du nom d'une interface réseau mis entre parenthèses `( )`. Cette mise en forme indique à PF qu'il doit mettre ses règles à jour en cas de changement de l'adresse IP affectée à l'interface en question. Cette technique est très utile dans le cas de liaisons PPP ou d'utilisation du protocole DHCP pour l'attribution d'adresses IP. Elle évite d'avoir à recharger les règles à chaque nouvelle affectation.
- grâce au nom d'une interface suivi d'un des paramètres suivants :
  - `:network` - identifie un bloc CIDR (par exemple : 192.168.0.0/24)
  - `:broadcast` - identifie une adresse de multi-diffusion (par exemple : 192.168.0.255)
  - `:peer` - identifie l'adresse d'un pair sur un lien point à point.

  De plus, le paramètre `:0` peut être ajouté aussi bien au nom d'une interface qu'à chacun des paramètres présentés ci-dessus. PF ne prend alors pas en compte les éventuels alias d'adresses IP affectés à l'interface. Ces paramètres peuvent aussi être utilisés dans le cas d'une interface entre parenthèses. Par exemple: `fxp0:network:0`
- à l'aide d'une table.
- sous toutes les formes décrites ci-dessus précédées du paramètre d'inversion `!` ("not").
- à l'aide d'une liste d'adresses.
- grâce au mot-clef `any` qui identifie n'importe quelle adresse.
- grâce au mot-clef `all` qui est un raccourci pour l'expression `from any to any` (n'importe quelle source vers n'importe quelle destination).

`src_port`, *`dst_port`*

Ces mots-clefs identifient respectivement les ports source et destination qui apparaissent dans les en-têtes des protocoles de couche 4. Ces ports peuvent être spécifiés :

- sous forme numérique par un nombre compris entre 1 et 65535.
- par le nom d'un service tel qu'identifié dans le fichier `/etc/services`.
- sous la forme d'une liste.
- sous la forme d'un intervalle de ports construit à l'aide des opérateurs suivants :
  - != (différent)
  - < (inférieur à)
  - > (supérieur à)
  - <= (inférieur ou égal à)
  - >= (supérieur ou égal à)
  - >< (intervalle)
  - <> (intervalle inverse)

    Les deux derniers opérateurs demandent deux arguments pour former un intervalle qui n'inclue pas ces arguments.
  - `:` (intervalle complet)

    L'opérateur d'intervalle complet est également un opérateur binaire incluant les arguments dans l'intervalle.

`tcp_flags`

Spécifie les drapeaux qui doivent être activés dans l'en-tête TCP lors de l'utilisation de `proto tcp`. Cette valeur est spécifiée ainsi : `flags` *`check/mask`*. Par exemple : `flags S/SA`. Dans ce cas, PF teste la valeur des drapeaux S et A (SYN et ACK) pour savoir s'ils sont positionnés. Si le drapeau SYN est positionné, et uniquement ce drapeau, alors la règle est applicable.

`state`

Ce mot-clef indique dans quel état doit être le paquet pour satisfaire à une règle.

- `keep state` - s'applique aux protocoles TCP, UDP, et ICMP.
- `modulate state` - ne s'applique qu'au protocole TCP. PF renforcera le caractère aléatoire des numéros de séquence initiaux (ISN) générés pour ces paquets.
- `synproxy state` - PF agit comme mandataire pour les connexions TCP entrantes. Cela renforce la protection contre les attaques par inondation de paquets SYN en provenance d'adresses usurpées. Cette option active implicitement les fonctionnalités `keep state` et `modulate state`.

Blocage par défaut

Il est recommandé d'adopter une approche de blocage par défaut lors de la configuration d'un pare-feu. Cela signifie que *tout* est interdit et que l'on autorise le trafic au cas par cas. Cette approche est assimilable à l'application d'un principe de précaution et simplifie l'écriture des règles.

Cette politique de blocage par défaut repose sur deux règles :

```
block in  all
block out all
```

Cela suffit à interdire tout trafic dans un sens comme dans l'autre et ce sur toutes les interfaces de la machine.

Laisser passer le trafic

Une fois notre politique restrictive mise en place, il faut spécifier quelles sont les connexions autorisées. C'est là qu'entrent en jeu les critères décrits précédemment : adresse et port source et destination, protocole, etc. Chaque fois qu'un paquet est autorisé à franchir les murs du pare-feu, les règles correspondantes devront être les plus restrictives possible : il s'agit de n'autoriser que le trafic voulu et lui seul.

Quelques exemples:

```
# Autoriser le trafic entrant sur l'interface dc0
# en provenance du réseau local 192.168.0.0/24,
# à destination de la machine dont l'adresse IP est 192.168.0.1.
# Dans le même temps, autoriser le trafic sortant par l'interface dc0.
pass in  on dc0 from 192.168.0.0/24 to 192.168.0.1
pass out on dc0 from 192.168.0.1 to 192.168.0.0/24


# Autoriser le trafic TCP entrant sur l'interface fxp0
# à destination d'un serveur HTTP.
# Le nom de l'interface - fxp0 - est utilisé comme adresse de destination
# pour les paquets autorisés. pass in on fxp0 proto tcp from any to fxp0 port www
```

L'option `quick`

Comme nous l'avons vu, chaque paquet est testé au regard de toutes les règles de filtrage, de la première à la dernière. Par défaut, un paquet est marqué à chaque test. Le résultat final peut ainsi changer d'une règle à l'autre jusqu'à ce que toutes aient été parcourues. Rappelons que c'est **la dernière règle à laquelle correspond un paquet qui l'emporte** sur les autres. Il y a cependant une exception : si l'option `quick` est présente dans une règle et qu'un paquet correspond aux critères de cette règle, il n'y a plus de tests : la règle en question est alors considérée par PF comme étant la dernière. Les exemples suivants illustrent ce cas de figure :

Mauvais :

```
block in on fxp0 proto tcp from any to any port ssh
pass  in all
```

La ligne `block` n'aura aucun effet. Les paquets seront bien évalués suivant ses critères, mais la ligne suivante annulera tout effet de cette première règle.

Mieux :

```
block in quick on fxp0 proto tcp from any to any port ssh
pass  in all
```

A première vue c'est la même chose. Si la ligne `block` correspond, l'option `quick` provoque l'arrêt des tests pour chaque paquet qui correspond aux critères de la première règle. Les paquets qui n'y satisfont pas seront quant à eux testés au regard des critères de la règle suivante.

Conserver l'état

Une des fonctionnalités les plus importantes de PF est sa capacité à conserver l'état des connexions. PF est capable d'évaluer un paquet non plus unitairement mais dans le contexte de la connexion à laquelle il appartient. PF utilise pour cela une table d'état grâce à laquelle PF peut rapidement déterminer si un paquet fait partie d'une connexion déjà établie et autorisée. Si tel est le cas, le paquet est transféré sans test complémentaire.

Conserver l'état des connexions a pour avantage de simplifier les règles de filtrage et d'améliorer les performances. PF évalue les paquets *quel que soit leur sens* : il n'est alors plus nécessaire d'écrire les règles pour les paquets des flux retour. PF consacre ainsi beaucoup moins de temps à inspecter les paquets.

Quand l'option `keep state` est utilisée dans une règle, le premier paquet qui déclenche l'activation de celle-ci provoque la création d'un enregistrement dans la table d'état des connexions en cours. Par la suite, non seulement les paquets allant de l'expéditeur au destinataire sont rattachés à cette table et donc autorisés à passer, mais également les paquets qui appartiennent aux réponses du destinataire. Par exemple :

```
pass out on fxp0 proto tcp from any to any keep state
```

Cette règle autorise les connexions TCP sortantes sur l'interface `fxp0` mais aussi les paquets retour. L'option keep state permet donc d'augmenter les performances du pare-feu car la recherche dans la table d'état est beaucoup plus rapide que l'opération consistant à évaluer un paquet au regard de toutes les règles de filtrage.

L'option `modulate state` fonctionne de la même façon que l'option `keep state` à ceci près qu'elle ne s'applique qu'aux paquets TCP sortants. L'option `modulate state` renforce le caractère aléatoire de leurs numéros de séquence initiaux (ISN). Cette option permet de renforcer la sécurité de certains systèmes d'exploitation ne sachant pas générer des ISN suffisamment aléatoires. A partir de la version 3.5 d'OpenBSD, l'option `modulate state` peut être utilisée pour les autres protocoles que TCP.

Pour conserver l'état des connexions TCP, UDP et ICMP tout en renforçant la sécurité des ISN :

```
pass out on fxp0 proto { tcp, udp, icmp } from any \
    to any modulate state
```

Un des avantages de conserver l'état des connexions tient à ce que les messages ICMP relatifs à celles-ci seront traités comme faisant partie de la connexion. Si des messages ICMP sont émis par une machine pour signaler une congestion par exemple, et que l'option `keep state` est activée, les messages seront pris en compte et acceptés par le pare-feu. Sans cela, ils auraient été bloqués ou ignorés.

La portée d'une entrée dans la table d'état dépend des options `state-policy` d'une manière globale ou bien des options `if-bound`, `group-bound` et `floating-state`. Les valeurs appliquées règle par règle on la même signification que lorsqu'elles sont utilisées avec l'option `state-policy`. Par exemple :

```
pass out on fxp0 proto { tcp, udp, icmp } from any \
    to any modulate state (if-bound)
```

Selon cette règle, les paquets ne trouveront une correspondance dans la table d'état que s'ils transitent par l'interface `fxp0`.

Il faut noter que les règles `nat`, `binat` et `rdr` créent implicitement des entrées dans la table d'état.

Conserver l'état des connexions UDP

On entend souvent dire qu'il est impossible d'utiliser la table d'état avec UDP car c'est un protocole sans état. S'il est vrai qu'une connexion UDP n'utilise pas stricto sensu le concept d'état tel que le fait une connexion TCP (à savoir une ouverture et une terminaison explicites de la connexion), cela n'a aucune conséquence sur la capacité qu'a PF de créer et gérer des états pour les connexions UDP. Pour ce type de protocole sans début ou fin de connexion explicites, PF conserve simplement une trace de la durée d'une session. S'il s'écoule un certain laps de temps sans échange de paquets entre les deux parties, l'entrée associée à la session dans la table d'état est supprimée. Ce laps de temps peut être configuré dans la section options du fichier `pf.conf`.

Options de Suivi Stateful

Quand une règle de filtrage crée une entrée dans la table d'états suite à l'utilisation des mots clé `keep state`, `modulate state` ou `synproxy state`, certaines options peuvent être spécifiées afin de contrôler le comportement de ces créations d'états. Les options suivantes sont disponibles :

`max` *`number`*
: Limite le nombre maximum d'entrées d'états que la règle peut créer à *number*. Si le maximum est atteint, les paquets qui devraient normalement créer un état sont rejetés jusqu'à ce que le nombre d'états existants diminue.

`source-track`
: Cette option active le suivi du nombre d'états créés par adresse IP source. Cette option a deux formats :
  - `source-track rule` - Le nombre maximum d'états créés par cette règle est limité par les options `max-src-nodes` et `max-src-states` de la règle. Seules les entrées d'états crées par cette règle particulière comptent pour la limite des règles.

- `source-track global` - Le nombre d'états créés par toutes les règles qui utilisent cette option est limité. Chaque règle peut spécifier différentes options `max-src-nodes` et `max-src-states`, cependant, les entrées d'états crées par toute règle participante comptent en vue d'une limite individuelle éventuelle.

Le nombre total d'adresses IP source suivies globalement peut être contrôlé via l'option `src-nodes` runtime.

`max-src-nodes` *`nombre`*
: Lorsque l'option `source-track` est utilisée, `max-src-nodes` limitera le nombre d'adresses IP source pouvant créer simultanément une entrée. Cette option peut seulement être utilisée avec une règle `source-track`.

`max-src-states` *`nombre`*
: Lorsque l'option `source-track` est utilisée, `max-src-states` limitera le nombre d'entrées d'états simultanées pouvant être crées par adresse IP source. La portée de cette limite (les états créés par cette règle uniquement ou les états créés par toutes les règles utilisant `source-track`) est dépendante de l'option `source-track` spécifiée.

Une règle d'exemple :

```
pass in on $ext_if proto tcp to $web_server \
    port www flags S/SA keep state \
    (max 200, source-track rule, max-src-nodes 100, max-src-states 3)
```

La règle ci-dessus définit le comportement suivant :

- Limiter le nombre maximum absolu d'états pouvant être créés par cette règle à 200
- Activer le suivi de la source; limiter la création d'états pour cette règle uniquement
- Limiter le nombre maximum de noeuds pouvant simultanément créer des états à 100
- Limiter le nombre maximum d'états simultanés par adresse IP source à 3

Un jeu séparé de restrictions peut être placé sur les connexions TCP stateful qui ont une poignée de main "3-way handshake" complète.

`max-src-conn` *`nombre`*
: Limiter le nombre maximum de connexions TCP simultanées ayant réalisé la poignée de main qu'un hôte peut initier.

`max-src-conn-rate` *`nombre / intervalle`*
: Limiter le taux de nouvelles connexions à un certaine fréquence.

Ces deux options font appel à l'option `source-track rule` et sont incompatibles avec `source-track global`.

Ces limites étant placées sur les connexions TCP ayant réalisé la poignée de main TCP, des connexions plus agressives pourront toujours avoir lieu depuis les adresses IP concernées.

`overload <`*`table`*`>`
: Mettre l'adresse d'un hôte concerné dans la table désignée.

`flush [global]`
: Tue toutes les autres connexions qui correspondent à cette règle et qui ont été créées par cette adresse IP source. Quand `global` est spécifié, cela tue tous les états correspondant à cette adresse IP source, sans discernement de la règle qui a créé cet état.

Un exemple :

```
table <abusive_hosts> persist
block in quick from <abusive_hosts>

pass in on $ext_if proto tcp to $web_server \
    port www flags S/SA keep state \
    (max-src-conn 100, max-src-conn-rate 15/5, overload <abusive_hosts> flush)
```

Ceci permet de :

- Limiter le nombre maximum de connexions par source à 100
- Limiter le taux du nombre de connexions à 15 dans une durée de 5 secondes
- Mettre l'adresse IP de tout hôte qui dépasse ces limites dans la table `<abusive_hosts>`
- Pour des adresses IP concernées, supprimer tous les états créés par cette règle.

Les drapeaux TCP

On utilise souvent les drapeaux TCP dans des règles pour traiter les ouvertures de sessions. Ces drapeaux et leur signification sont présentés dans la liste suivante :

- **F** : FIN - Fin de session
- **S** : SYN - Synchronize ; correspond à une ouverture de session
- **R** : RST - Reset ; met fin à une session
- **P** : PUSH - Push ; le paquet est envoyé immédiatement
- **A** : ACK - Acknowledgement ; atteste de l'acquittement d'une des parties
- **U** : URG - Urgent
- **E** : ECE - Explicit Congestion Notification Echo ; avis de congestion
- **W** : CWR - Congestion Window Reduced ; avis de réduction de la fenêtre TCP

Le mot-clef `flags` doit apparaître dans une règle si l'on souhaite que PF prenne en compte la valeur des drapeaux TCP d'un paquet. La syntaxe est la suivante :

```
flags check/mask
```

La partie *`mask`* de la règle indique la liste des drapeaux que PF doit inspecter. La partie *`check`* quant à elle spécifie les drapeaux qui doivent être positionnés pour que la règle s'applique au paquet traité.

```
pass in on fxp0 proto tcp from any to any port ssh flags S/SA
```

Cette règle s'applique aux paquets TCP dont le drapeau SYN est positionné. PF limite son inspection aux drapeaux SYN et ACK. La règle s'applique donc à un paquet dont les drapeaux SYN et ECE sont positionnés mais pas à un paquet dont les drapeaux SYN et ACK ou dont seul le drapeau ACK sont positionnés.

Notez que les précédentes versions d'OpenBSD acceptaient cette syntaxe :

```
. . . flags S
```

Ce qui n'est plus vrai : le masque doit maintenant *toujours* être renseigné.

Les drapeaux sont souvent utilisés avec l'option `keep state` pour mieux contrôler la création des entrées dans la table d'état :

```
pass out on fxp0 proto tcp all flags S/SA keep state
```

Une entrée est créée pour tous les paquets TCP sortants dont le drapeau SYN est positionné.

Il faut manipuler les drapeaux avec prudence et se méfier des mauvais conseils. Certaines personnes suggèrent de ne créer des entrées que pour les paquets dont le drapeau SYN est positionné. Ce qui peut aboutir à cette règle :

```
. . . flags S/FSRPAUEW  mauvaise pioche !!
```

En théorie, une session TCP commence par un paquet dont le drapeau SYN est positionné. Toujours en théorie, il ne faut créer une entrée dans la table d'état que pour ce genre de paquets. Mais certains systèmes utilisent le drapeau ECN en début de session. Ces paquets sont rejetés par la règle précédente. Une meilleure solution est :

```
. . . flags S/SAFR
```

Si le trafic est normalisé, il peut être pratique et sûr de ne pas tester la valeur des drapeaux FIN et RST. Dans ce cas, PF rejette tout paquet entrant dont les drapeaux TCP sont positionnés de manière illicite (par exemple SYN et RST) et les combinaisons potentiellement ambiguës (telles que SYN et FIN) seront normalisées. Il est fortement recommandé de toujours `normaliser` (scrub) le trafic entrant :

```
scrub in on fxp0
.
.
.
pass in on fxp0 proto tcp from any to any port ssh flags S/SA \
   keep state
```

Mandataire TCP SYN

Normalement, quand un client ouvre une connexion TCP vers un serveur, PF relaie les paquets d'ouverture (handshake) au fur et à mesure qu'ils arrivent. PF peut agir en tant que mandataire (proxy). Dans ce cas, PF va traiter la demande en lieu et place du serveur et ne transfèrera qu'ensuite les paquets à ce dernier. Aucun paquet n'est transmis au serveur avant que le client n'ait terminé l'échange initial (handshake). L'avantage de cette méthode est de protéger le serveur des attaques

par inondation de paquets SYN lancées à l'aide de paquets falsifiés.

Le mandataire TCP SYN est activé à l'aide de l'option `synproxy state` :

```
pass in on $ext_if proto tcp from any to $web_server port www \
   flags S/SA synproxy state
```

Toutes les connexions à destination du serveur HTTP seront mandatées par PF.

L'option `synproxy state` apporte les mêmes avantages que les options `keep state` et `modulate state`.

Par contre, l'option synproxy ne fonctionne pas quand PF est installé en passerelle transparente (bridge(4)).

Bloquer les paquets usurpés

On parle d'usurpation quand un utilisateur mal intentionné maquille son adresse IP dans le but d'anonymiser ou de cacher son identité afin de lancer des attaques sans que leur origine soit détectable. Il peut également essayer et parfois réussir à avoir accès à des services réservés à certaines adresses.

PF permet de se prémunir de ce type d'attaques grâce à l'option `antispoof` :

```
antispoof [log] [quick] for interface [af]
```

`log`
: Journalise les paquets via pflogd(8).

`quick`
: Si un paquet correspond à la règle, celle-ci est appliquée immédiatement.

`interface`
: Désigne l'interface sur laquelle s'applique la protection. Il est possible de passer une liste d'interfaces en paramètre.

`af`
: Spécifie le type d'adresse : `inet` pour IPv4 ou `inet6` pour IPv6.

Exemple:

```
antispoof for fxp0 inet
```

Quand les règles sont chargées, toutes les occurrences du mot `antispoof` sont décodées dans deux filtres. Si l'interface `fxp0` dont l'adresse IP est 10.0.0.1 pour un masque de sous- réseau de 255.255.255.0 (soit /24) est protégée, l'option `antispoof` sera décodée ainsi :

```
block in on ! fxp0 inet from 10.0.0.0/24 to any
block in inet from 10.0.0.1 to any
```

Cette règle déclenche deux actions :

- elle bloque tout le trafic en provenance du réseau 10.0.0.0/24 s'il ne passe *pas* par l'interface `fxp0`. Puisque le réseau 10.0.0.0/24 est branché sur l'interface `fxp0`, aucun paquet en provenance de celui-ci ne devrait arriver ailleurs.
- elle bloque tout le trafic entrant dont l'adresse source est 10.0.0.1, à savoir celle de l'interface `fxp0`. Cette machine ne devrait en effet jamais émettre de paquets à travers une interface externe et par conséquent ne doit pas recevoir de paquets ayant son adresse IP comme adresse source.

REMARQUE : le filtrage activé par l'option `antispoof` d'une règle s'applique également aux paquets envoyés sur l'adresse de bouclage interne (loopback). Le filtrage est communément désactivé sur ces interfaces, et cela devient primordial lors de l'utilisation de règles antispoof :

```
set skip on lo0

antispoof for fxp0 inet
```

L'utilisation de l'option `antispoof` est réservée aux interfaces qui possèdent une adresse IP. Utiliser `antispoof` sur une interface sans adresse IP aboutit au filtrage suivant :

```
block drop in on ! fxp0 inet all
block drop in inet all
```

Avec ce genre de règles, le risque est réel de bloquer *tout* le trafic entrant sur *toutes* les interfaces.

Reconnaissance passive d'OS par leurs empreintes

La reconnaissance passive d'OS par leurs empreintes ("OS Fingerprinting" ou OSFP) est une méthode qui permet de reconnaître à distance quel système d'exploitation tourne sur une machine. Cette reconnaissance se base sur les caractéristiques des paquets TCP SYN renvoyés par une machine. Ces informations peuvent être utilisées comme critères dans des règles de filtrage.

PF utilise le fichier d'empreintes `/etc/pf.os` pour reconnaître les systèmes d'exploitation auxquels il a affaire. Lorsque PF s'exécute, la liste des empreintes reconnues peut être consultée grâce à la commande suivante :

```
# pfctl -s osfp
```

Dans une règle, une empreinte peut être désignée sous la forme d'une classe, d'une version ou d'un sous-type d'OS. La liste de ces éléments est affichée à l'aide de la commande `pfctl`. Pour désigner une empreinte dans une règle, il faut utiliser le mot-clef `os` :

```
pass  in on $ext_if from any os OpenBSD keep state
block in on $ext_if from any os "Windows 2000"
block in on $ext_if from any os "Linux 2.4 ts"
block in on $ext_if from any os unknown
```

`unknown` est une classe spéciale désignant les systèmes d'exploitation dont l'empreinte n'est pas connue.

Notez bien que :

- La reconnaissance peut échouer face à des paquets spécifiquement construits pour tromper la détection d'empreintes.
- L'application de correctifs peut modifier le comportement de la pile TCP/IP d'un système d'exploitation et faire également échouer ou tromper la reconnaissance de l'OS.
- L'option OSFP n'est applicable qu'aux paquets TCP SYN. Elle est inefficace avec d'autres protocoles et pour les sessions déjà établies.

Les options IP

PF bloque par défaut tous les paquets qui utilisent les options IP. Cela rend moins aisé le travail des outils de reconnaissance d'empreintes tels que nmap. Si une application utilise ces options (par exemple IGMP ou les diffusions multicast) il est possible d'utiliser l'option `allow-opts` :

```
pass in quick on fxp0 all allow-opts
```

Exemple de règles de filtrage

Vous trouverez ci-dessous un exemple de règles de filtrage pour un pare-feu PF destiné à protéger un petit réseau connecté à Internet. Seules les règles de filtrage sont mentionnées ; queueing, `nat`, `rdr`, etc. ont été volontairement laissées de côté.

```
ext_if  = "fxp0"
int_if  = "dc0"
lan_net = "192.168.0.0/24"

# Déclaration du tableau référençant toutes les adresses IP affectées au
# pare-feu.
table <firewall> const { self }

# Ne pas filtrer sur l'interface de bouclage
set skip on lo0

# Normalisation de tous les paquets entrants.
scrub in all

# Mise en place d'une politique d'interdiction par défaut.
block all

# Activation de la protection contre l'usurpation sur l'interface
# externe.
antispoof quick for $int_if inet
```

```
# Les connexions ssh ne sont autorisées qu'en provenance du réseau local
# et de la machine 192.168.0.15. "block return" provoque l'émission d'un
# paquet TCP RST pour mettre fin aux connexions illicites. "quick"
# assure que cette règle n'est pas contredite par les règles "pass".
block return in quick on $int_if proto tcp from ! 192.168.0.15 \
   to $int_if port ssh flags S/SA

# Autoriser le trafic sortant et entrant sur le réseau local.
pass in  on $int_if from $lan_net to any
pass out on $int_if from any to $lan_net

# Autoriser les connexions sortantes tcp, udp et icmp sur l'interface
# externe.
# Activer le suivi des états pour les protocoles udp et icmp.
# Activer l'option modulate state sur les paquets tcp.

pass out on $ext_if proto tcp all modulate state flags S/SA
pass out on $ext_if proto { udp, icmp } all keep state

# Autoriser les connexions ssh sur l'interface externe du moment
# qu'elles ne sont pas destinées au pare-feu lui-même. Journaliser le
# paquet qui initie la session afin de pouvoir déterminer qui s'est
# connecté. Activer un service mandataire SYN.
pass in log on $ext_if proto tcp from any to ! <firewall> \
   port ssh flags S/SA synproxy state
```

[Section précédente : Tables] [Index] [Section suivante : Traduction des Adresses IP ("NAT")]


www@openbsd.org

$OpenBSD: filter.html,v 1.24 2006/10/29 10:58:52 jufi Exp $

[Précédent : Filtrage de Paquets] [Index] [Suivant : Redirection du Trafic]

PF : Traduction des Adresses IP ("NAT")

Table des Matières

Introduction

La traduction d'adresses IP ("NAT") est un mécanisme destiné à faire correspondre un réseau entier (ou des réseaux) à une seule adresse IP. Un tel mécanisme est nécessaire lorsque le nombre des adresses IP qui vous sont attribuées par votre Fournisseur d'Accès à Internet est plus petit que le nombre de machines qui doivent pouvoir bénéficier d'une connexion Internet. La NAT est décrite dans la RFC 1631, "The IP Network Address Translator (NAT)".

La NAT vous permet de bénéficier des blocs d'adressage privés décrits dans la RFC 1918, "Address Allocation for Private Internets". Typiquement, votre réseau interne sera paramétré pour utiliser un ou plusieurs des blocs réseau suivants :

```
10.0.0.0/8        (10.0.0.0 - 10.255.255.255)
172.16.0.0/12     (172.16.0.0 - 172.31.255.255)
192.168.0.0/16    (192.168.0.0 - 192.168.255.255)
```

Un système OpenBSD mettant en oeuvre la NAT aura au moins deux cartes réseau dont une est connectée à Internet; l'autre étant connectée à votre réseau interne. La NAT traduit les requêtes en provenance de votre réseau interne de telle façon à les faire apparaître comme si elles étaient générées par votre système de NAT OpenBSD.

Comment Fonctionne la NAT

Lorsqu'un client du réseau interne entre en communication avec une machine sur Internet, il envoie des paquets IP à destination de cette machine. Ces paquets contiennent toutes les informations sur leur émetteur et leur destinataire nécessaires à leur bon acheminement. La NAT prend en compte les informations suivantes :

- L'adresse IP source (192.168.1.35 par exemple)
- Port TCP ou UDP source (2132 par exemple)

Lorsque les paquets passent à travers la passerelle NAT, ils sont modifiés de telle façon à ce qu'ils semblent provenir de la passerelle NAT. Cette passerelle enregistrera les modifications effectuées dans sa table d'état afin de a) inverser les modifications pour les paquets de retour et b) s'assurer que les paquets de retour sont autorisés à travers le pare-feu et ne sont pas bloqués. Par exemple, les modifications suivantes peuvent être effectuées :

- Adresse IP source : remplacée par l'adresse externe de la passerelle (24.5.0.5 par exemple)
- Port source : remplacé par un port non utilisé sur la passerelle et choisi de manière aléatoire (53136 par exemple)

Aucune des deux extrêmités de la communication ne se rend compte de ces modifications. Pour la machine interne, le système qui effectue la NAT est simplement une passerelle Internet. Pour l'hôte sur Internet, les paquets semblent provenir directement du système de NAT; il ne sait même pas que la machine interne existe.

Lorsque l'hôte sur Internet répond aux paquets de la machine interne, ils seront envoyés à l'adresse IP externe de la passerelle NAT (24.5.0.5) sur le port de traduction (53136). Dès réception de ces paquets, la passerelle NAT cherchera dans sa table d'état si ces paquets de retour correspondent à une connexion déjà établie. Une correspondance unique sera trouvée, basée sur la combinaison IP/port qui permet à PF de voir que les paquets appartiennent à une connexion initiée par la machine interne 192.168.1.35. PF effectuera les modifications inverses à celles effectuées sur les paquets sortants puis enverra ces paquets à la machine interne.

La traduction de paquets ICMP s'effectue de manière similaire mais sans la modification du port source.

NAT et Filtrage de Paquets

REMARQUE : Les paquets traduits doivent, comme n'importe quel autre paquet, être évalués par le mécanisme pare-feu suivant les règles de filtrage définies. La *seule* exception à cette règle est lorsque le mot-clé `pass` est utilisé dans la règle de `nat`. Ce mot-clé permettra aux paquets traduits d'être autorisés immédiatement par le pare-feu.

Il est à noter que la traduction a lieu *avant* le filtrage. Ce qui veut dire que le pare-feu verra le paquet *traduit* avec l'adresse IP et le port traduits tel que c'est expliqué dans Comment Fonctionne la NAT.

Routage IP

Vu que la NAT est utilisée la plupart du temps sur des routeurs et des passerelles réseau, il sera probablement nécessaire d'activer le routage IP pour permettre aux paquets de traverser les interfaces réseau du système OpenBSD. Le routage IP peut être activé à l'aide du mécanisme sysctl(3) :

```
# sysctl net.inet.ip.forwarding=1
# sysctl net.inet6.ip6.forwarding=1 (si Ipv6 est utilisé)
```

Pour rendre cette modification permanente, les lignes suivantes doivent être ajoutées au fichier `/etc/sysctl.conf` :

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

Ces lignes sont déjà présentes dans l'installation par défaut mais sont commentées (préfixées avec un caractère #). Supprimez le # et sauvegardez le fichier. Le routage IP sera désormais activé automatiquement à chaque redémarrage.

Configurer la NAT

Le format général des règles de NAT dans `pf.conf` est le suivant :

```
nat [pass [log]] on interface [af] from src_addr [port src_port] to \
   dst_addr [port dst_port] -> ext_addr [pool_type] [static-port]
```

`nat`
: Le mot-clé qui permet de créer une règle de NAT.

`pass`
: Permet aux paquets traduits de contourner complètement les règles de filtrage et d'être autorisés systématiquement par le pare-feu.

`log`
: Lorsque `pass` est spécifié, les paquets peuvent être tracés à l'aide de pflogd(8). Normalement, uniquement le premier paquet qui correspond à la règle sera tracé. Pour tracer tous les paquets correspondants, utilisez `log (all)`.

`interface`
: Le nom de l'interface réseau sur laquelle les paquets seront traduits.

`af`
: La famille d'adresses : `inet` pour IPv4 ou `inet6` pour IPv6. PF est normalement capable de déterminer ce paramètre à partir des adresses source/destination.

`src_addr`
: L'adresse source (interne) des paquets qui seront traduits. L'adresse source peut être spécifiée de plusieurs manières :
  - Une seule adresse IPv4 ou IPv6.
  - Un bloc réseau en notation CIDR.
  - Un nom de domaine au format FQDN qui sera résolu via DNS lorsque les règles seront chargées. Ce nom de domaine sera remplacé par toutes les adresses IP résultant de la résolution.
  - Un nom d'interface réseau. Ce nom sera remplacé par toutes les adresses IP attribuées à cette interface au moment du chargement des règles.
  - Un nom d'interface réseau suivi d'un *`/netmask`* (`/24` par exemple). Chaque adresse IP attribuée à l'interface sera combinée avec le netmask pour former un bloc réseau en notation CIDR. Le résultat remplacera la spécification initiale dans la règle.
  - Le nom d'une interface réseau suivi par un modificateur parmi les modificateurs suivants :
    - `:network` - remplace le bloc réseau en notation CIDR (192.168.0.0/24 par exemple)
    - `:broadcast` - remplace l'adresse de broadcast réseau (192.168.0.255 par exemple)
    - `:peer` - remplace l'adresse IP du pair dans une communication point-à-point

      De plus, le modificateur `:0` peut être ajouté au nom de l'interface ou à l'un des modificateurs précités pour indiquer à PF que les alias IP ne doivent pas être inclus dans la substitution. Ces modificateurs peuvent aussi être utilisés lorsque le nom de l'interface est entre parenthèses. Exemple : `fxp0:network:0`
  - Une table.
  - N'importe quelle spécification précitée préfixée du modificateur `!` ("not") pour indiquer la négation.
  - Un ensemble d'adresses en utilisant une liste
  - Le mot-clé `any` signifiant toutes les adresses

`src_port`
: Le port source dans l'en-tête du paquet (couche 4). Les ports peuvent être spécifiés de la manière suivante :
  - Un nombre entre 1 et 65535
  - Un nom de service valide figurant dans le fichier `/etc/services`
  - Un ensemble de ports sous forme de liste
  - Un intervalle :
    - `!=` (différent de)
    - `<` (inférieur à)
    - `>` (supérieur à)
    - `<=` (inférieur ou égal à)
    - `>=` (supérieur ou égal à)
    - `><` (intervalle)
    - `<>` (intervalle inverse)

      Les deux derniers opérateurs sont des opérateurs binaires (ils prennent deux arguments) et

n'incluent pas les arguments dans l'intervalle.

- `:` (intervalle avec inclusion)
  L'opérateur d'intervalle avec inclusion est aussi un opérateur binaire mais contrairement aux deux opérateurs précédents, il inclut les arguments dans l'intervalle.

L'option `port` n'est pas habituellement utilisée dans les règles `nat` vu que le but est souvent de traduire tout le trafic peu importe le(s) port(s) utilisé(s).

`dst_addr`
L'adresse destination des paquets à traduire. L'adresse de destination est spécifiée de la même manière que l'adresse source.

`dst_port`
Le port destination dans l'en-tête du paquet (couche 4). Ce port est spécifié de la même manière que le port source.

`ext_addr`
L'adresse externe (de traduction) sur la passerelle NAT. C'est l'adresse qui figurera dans les paquets une fois ceux-ci traduits. L'adresse externe peut être spécifiée comme suit :

- Une seule adresse IPv4 ou IPv6.
- Un bloc réseau en notation CIDR.
- Un nom de domaine au format FQDN qui sera résolu via DNS lorsque les règles seront chargées. Ce nom de domaine sera remplacé par toutes les adresses IP résultant de la résolution.
- Le nom de l'interface réseau externe. Ce nom sera remplacé par toutes les adresses IP attribuées à cette interface au moment du chargement des règles.
- Le nom de l'interface réseau externe entouré de parenthèses `( )`. Ceci permet à PF de mettre à jour la règle si l'adresse ou les adresses IP de cette interface change(nt). C'est très utile lorsque l'interface obtient son adresse IP via DHCP ou un système dial-up. Dans ce cas, les règles n'ont pas à être rechargées chaque fois que l'adresse change.
- Le nom d'une interface réseau suivi par un modificateur parmi les modificateurs suivants :
  - `:network` - remplace le bloc réseau en notation CIDR (192.168.0.0/24 par exemple)
  - `:peer` - remplace l'adresse IP du pair dans une communication point-à-point
    De plus, le modificateur `:0` peut être ajouté au nom de l'interface ou à l'un des modificateurs précités pour indiquer à PF que les alias IP ne doivent pas être inclus dans la substitution. Ces modificateurs peuvent aussi être utilisés lorsque le nom de l'interface est entre parenthèses. Exemple : `fxp0:network:0`
- Un ensemble d'adresses en utilisant une liste

`pool_type`
Spécifie le type de pool d'adresses à utiliser pour la traduction.

`static-port`
Si cette option est utilisée, PF ne traduira pas le port source des paquets TCP et UDP.

Ceci nous donne une ligne de la forme basique suivante :

```
nat on tl0 from 192.168.1.0/24 to any -> 24.5.0.5
```

Cette règle permet d'effectuer la NAT sur l'interface `tl0` sur tout paquet provenant de 192.168.1.0/24 et remplace l'adresse IP source par 24.5.0.5.

Bien que cette règle soit correcte, ce n'est pas la forme recommandée. La maintenance peut s'avérer difficile vu que toute modification des adresses réseau externe ou interne nécessitera une modification de la règle. La forme suivante est plus facile à maintenir (`tl0` est l'interface externe, `dc0` est l'interface interne) :

```
nat on tl0 from dc0:network to any -> tl0
```

L'avantage de cette forme doit être clair maintenant : vous pouvez changer les adresses IP des deux interfaces sans changer la règle.

Lorsque vous spécifiez un nom d'interface comme adresse de traduction tel que c'est fait dans la règle ci-dessus, l'adresse IP est déterminée lors du *chargement* de pf.conf. Elle n'est pas déterminée à la volée. Si vous utilisez DHCP pour configurer votre interface externe, ceci peut poser problème. Par exemple, si l'adresse qui vous est attribuée change, le système de NAT continuera

à traduire les paquets sortants en utilisant l'ancienne adresse IP. Les connexions sortantes ne fonctionneront donc plus. Pour éviter ce problème, vous pouvez dire à PF de mettre à jour automatiquement l'adresse de traduction en mettant des parenthèses autour du nom de l'interface :

```
nat on tl0 from dc0:network to any -> (tl0)
```

Cette méthode fonctionne aussi bien pour la traduction d'adresses IPv6 que pour la traduction d'adresses IPv4.

Mise en Correspondance Bidirectionnelle (mise en correspondance 1:1)

Une mise en correspondance bidirectionnelle (traduction 1 à 1) peut être établie en utilisant une règle `binat`. Une règle `binat` établit une correspondance 1 à 1 entre une adresse IP interne et une adresse externe. Ceci peut être pratique par exemple pour permettre à un serveur web interne d'avoir sa propre adresse de traduction externe. Les connexions en provenance d'Internet à destination de cette adresse externe seront acheminées vers l'adresse interne du serveur web. Quant aux requêtes émanant du serveur web (telles que les requêtes DNS), elles seront traduites de telle façon à remplacer l'adresse interne par l'adresse externe de traduction attribuée au serveur web. Contrairement aux règles `nat`, les règles `binat` ne modifient jamais les ports TCP.

Exemple :

```
web_serv_int = "192.168.1.100"
web_serv_ext = "24.5.0.6"

binat on tl0 from $web_serv_int to any -> $web_serv_ext
```

Exceptions aux Règles de Traduction

Des exceptions aux règles de traduction peuvent être faites en utilisant le mot-clé `no`. Par exemple, si la règle de NAT précédente était modifiée de la manière suivante :

```
no nat on tl0 from 192.168.1.208 to any
nat on tl0 from 192.168.1.0/24 to any -> 24.2.74.79
```

Alors tous les paquets du réseau 192.168.1.0/24 seront traduits avec l'adresse externe 24.2.74.79 excepté pour la machine 192.168.1.208.

Il est à noter que la première règle à laquelle un paquet correspond est appliquée et aucune évaluation supplémentaire n'est effectuée. S'il y a une règle `no`, alors le paquet n'est pas traduit. Le mot-clé `no` peut être utilisé avec les règles `binat` et `rdr`.

Vérification de l'état de la NAT

Pour voir les traductions actives, il faut utiliser pfctl(8) avec l'option `-s state`. Cette option affichera une liste de toutes les sessions NAT en cours :

```
# pfctl -s state
fxp0 TCP 192.168.1.35:2132 -> 24.5.0.5:53136 -> 65.42.33.245:22 TIME_WAIT:TIME_WAIT
fxp0 UDP 192.168.1.35:2491 -> 24.5.0.5:60527 -> 24.2.68.33:53   MULTIPLE:SINGLE
```

Explications (première ligne uniquement) :

fxp0
: Indique l'interface à laquelle est rattachée la session. Le mot-clé `self` apparaîtra si la session est de type `floating`.

TCP
: Le protocole utilisé par la connexion.

192.168.1.35:2132
: L'adresse IP (192.168.1.35) de la machine sur le réseau interne. Le port source (2132) est affiché après l'adresse. C'est aussi l'adresse qui sera remplacée au niveau de l'en-tête IP.

24.5.0.5:53136
: L'adresse IP (24.5.0.5) et le port (53136) sur la passerelle utilisés pour la traduction.

65.42.33.245:22
: L'adresse IP (65.42.33.245) et le port (22) auxquels la machine interne est connectée.

TIME_WAIT:TIME_WAIT
: Indique l'état de la connexion TCP, vu par PF.

[Précédent : Filtrage de Paquets] [Index] [Suivant : Redirection du Trafic]


www@openbsd.org

$OpenBSD: nat.html,v 1.18 2006/10/29 10:58:52 jufi Exp $

[Précédent : Traduction d'Adresses Réseau] [Index] [Suivant : Raccourcis pour la Création de Jeux de Règles]

PF : Redirection de Trafic ("Forwarding" de Ports)

Table des Matières

Introduction

Quand vous utilisez la NAT, les machines sur votre réseau peuvent accéder à la totalité des ressources Internet. Qu'en est-il lorsqu'une de vos machines derrière la passerelle NAT a besoin d'être accédée depuis Internet ? la redirection est utilisée pour résoudre ce genre de problèmes. Elle permet au trafic entrant d'être acheminé vers une machine derrière la passerelle NAT.

Prenons un exemple :

```
rdr on tl0 proto tcp from any to any port 80 -> 192.168.1.20
```

La ligne ci-dessus redirige le trafic à destination du port TCP 80 (serveur web) vers une machine sur le réseau d'adresse IP 192.168.1.20. Ainsi, même si cette machine est derrière votre passerelle, le monde externe peut y avoir accès.

La partie `from any to any` de la ligne `rdr` précitée peut être assez utile. Si vous voulez restreindre les adresses ou les sous-réseaux autorisés à avoir accès au serveur web sur le port 80, vous pouvez le faire de la façon suivante :

```
rdr on tl0 proto tcp from 27.146.49.0/24 to any port 80 -> \
   192.168.1.20
```

Ceci aura pour effet de rediriger uniquement le trafic en provenance du sous-réseau spécifié. Notez que cela implique la possibilité de rediriger différentes machines en entrée vers des machines derrière la passerelle. Une telle fonctionnalité est utile. Par exemple, vous pouvez autoriser des utilisateurs sur des sites distants à accèder à leur propre machine dans la mesure où vous connaissez leur adresse IP :

```
rdr on tl0 proto tcp from 27.146.49.14 to any port 80 -> \
   192.168.1.20
rdr on tl0 proto tcp from 16.114.4.89 to any port 80 -> \
   192.168.1.22
rdr on tl0 proto tcp from 24.2.74.178 to any port 80 -> \
   192.168.1.23
```

Une plage de ports peut aussi être redirigée par une même règle :

```
rdr on tl0 proto tcp from any to any port 5000:5500 -> \
   192.168.1.20
rdr on tl0 proto tcp from any to any port 5000:5500 -> \
   192.168.1.20 port 6000
rdr on tl0 proto tcp from any to any port 5000:5500 -> \
   192.168.1.20 port 7000:*
```

Les exemples précédents permettent de rediriger tout port entre 5000 et 5500 inclus vers 192.168.1.20. Dans la première règle, on a une correspondance de ports 1 à 1. Ainsi le port 5000 est redirigé vers le port 5000 de la machine destination, le port 5001 est redirigé vers le port 5001 et ainsi de suite. Dans la deuxième règle, la plage de ports toute entière est redirigée vers le port 6000. Enfin, dans la troisième règle, le port 5000 est redirigé vers le port 7000, 5001 vers 7001 et ainsi de suite.

Redirection et Filtrage de Paquets

REMARQUE : Les paquets redirigés doivent encore être autorisés par le moteur de filtrage. Ils seront bloqués ou autorisés selon les règles de filtrage qui ont été définies.

La *seule* exception à cette règle est lorsque le mot-clé `pass` est utilisé dans la règle `rdr`. Dans ce cas, les paquets redirigés passeront à travers le moteur de filtrage et l'état de connexion sera préservé : Les règles de filtrage ne seront pas considérées pour ces paquets. Ceci est un racourci utile pour éviter de rajouter des règles de filtrage `pass` pour chaque règle de redirection. Il faut voir cela comme une règle `rdr` normale (sans mot-clé `pass`) associée à une règle de filtrage `pass` avec le mot-clé `keep state`. Cependant, si vous voulez mettre en oeuvre des options de filtrage spécifiques telles que `synproxy`, `modulate state`, etc. vous devrez utilisez une règle de filtrage `pass` dédiée étant donné que ces options ne sont pas implémentées au niveau des règles de redirection.

Notez aussi que la traduction est effectuée *avant* le filtrage. Le moteur de filtrage verra le paquet *traduit*, ç.à.d. après que son adresse IP destination et/ou port de destination soient modifiés pour correspondre à l'adresse/au port de redirection spécifié(e) dans la règle `rdr`. Considérons le cas suivant :

- 192.0.2.1 - hôte sur Internet
- 24.65.1.13 - adresse IP externe du routeur OpenBSD
- 192.168.1.5 - adresse IP interne du serveur web

Règle de redirection :

```
rdr on tl0 proto tcp from 192.0.2.1 to 24.65.1.13 port 80 \
   -> 192.168.1.5 port 8000
```

Avant que la règle de redirection `rdr` ne soit utilisée, le paquet est traité :

- Adresse source : 192.0.2.1
- Port source : 4028 (choisi de manière arbitraire par le système d'exploitation)
- Adresse de destination : 24.65.1.13
- Port de destination : 80

Puis la règle de redirection `rdr` est évaluée :

- Adresse source : 192.0.2.1
- Port source : 4028
- Adresse de destination : 192.168.1.5
- Port de destination : 8000

Le moteur de filtrage verra le paquet IP tel qu'il apparaît après la traduction effectuée par le mécanisme de redirection.

Implications au niveau de la Sécurité

La redirection a des implications au niveau de la sécurité. La création d'une ouverture pour autoriser le trafic à destination du réseau interne protégé peut causer la compromission d'une machine sur ce dernier. Par exemple, si le trafic est redirigé vers un serveur web interne et que par malheur, une vulnérabilité est découverte dans le service web ou dans un script CGI executé par le serveur, alors la machine peut être compromise par un utilisateur distant. Une fois cette compromission réalisée, l'utilisateur malveillant peut rebondir à partir de cette machine vers le réseau interne ce qui, bien entendu, ne peut être bloqué par le pare-feu étant donné qu'il ne voit pas ce traffic.

Ces risques peuvent être minimisés en confinant de manière stricte le système accédé depuis l'extérieur à un réseau séparé. Ce réseau est souvent appelé zone démilitarisée (DMZ) ou réseau de service privé (Private Service Network, PSN). De cette façon, si le serveur web est compromis, les effets peuvent être limités au réseau DMZ/PSN en filtrant soigneusement le trafic autorisé entre DMZ/PSN et vos autres réseaux.

Redirection et Réflexion

Souvent, les règles de redirection sont utilisées pour faire suivre des connexions Internet entrantes à un serveur local disposant d'une adresse privée sur le réseau interne comme le montre l'exemple suivant :

```
server = 192.168.1.40

rdr on $ext_if proto tcp from any to $ext_if port 80 -> $server \
   port 80
```

Mais lorsque la règle de redirection est testée par un client du LAN, elle ne fonctionne pas. C'est normal car les règles de redirection ne s'appliquent qu'aux paquets qui passent à travers l'interface spécifiée dans chacune d'elles (dans l'exemple précité, c'est l'interface externe `$ext_if`). Cependant, une connexion émanant d'un client interne et à destination de l'adresse externe du pare-feu n'implique pas que les paquets vont passer à travers l'interface externe du pare-feu. La pile TCP/IP sur le pare- feu compare l'adresse de destination des paquets entrants avec ses propres adresses et aliases et détecte les connexions qui lui sont destinées lorsque les paquets constituant celles-ci passent son interface interne. De tels paquets ne passent pas physiquement à travers l'interface externe, et de toute façon la pile ne simule pas ce passage. Ainsi, PF ne voit jamais ces paquets sur l'interface externe. Du coup, la règle de redirection sur l'interface externe n'est jamais appliquée.

Que doit-on faire alors ? ajouter une seconde règle de redirection sur l'interface interne ? non, ça ne fonctionnera pas mieux. Lorsque le client local se connecte à l'interface externe du pare-feu, le paquet initial de l'échange TCP atteint le pare-feu via son interface interne. La nouvelle règle de redirection s'applique et l'adresse de destination est remplacée par celle du serveur interne. Le paquet et alors redirigé par l'interface interne du pare-feu vers le serveur interne. Mais l'adresse source n'a pas été modifiée. Elle contient l'adresse du client local. Le serveur envoie alors directement les réponses au client. Le pare-feu ne voit jamais le retour et n'a aucune chance de traduire correctement les paquets de retour. Le client reçoit une réponse à partir d'une source inattendue. Il met alors fin à cette connexion.

Cependant, il est souvent souhaitable que les clients sur le LAN se connectent au même serveur interne que les clients externes et

de manière aussi transparente. Il existe plusieurs solutions à ce problème :

DNS en "Split-Horizon"

Il est possible de configurer les serveurs DNS pour fournir une réponse différente selon la provenance de la requête : clients internes ou externes. Ainsi les clients internes recevront l'adresse interne du serveur en réponse à leur demande de résolution de nom. Ils pourront alors se connecter directement au serveur local sans impliquer le pare- feu. Ce dernier sera alors moins sollicité.

Déplacer le Serveur Vers un Réseau Local Séparé

Une autre solution consiste à ajouter une carte réseau au pare-feu et à déplacer le serveur local vers un réseau dédié de type DMZ. Les connexions des clients locaux seront alors redirigées de la même manière que les connexions en provenance d'Internet. L'utilisation de réseaux séparés a plusieurs avantages. Entre autres, une amélioration du niveau de sécurité en isolant le serveur des machines internes. Si le serveur (qui, nous le rappelons, est joignable à partir d'Internet) est compromis, il ne peut se connecter à des machines internes directement vu que toutes ces connexions doivent passer à travers le pare-feu.

Mandater les Connexions TCP ("TCP Proxying")

Un mandataire TCP générique peut être mis en place sur le pare-feu. Ce mandataire devra écouter sur le port de redirection ou accepter les connexions sur l'interface interne et redirigées sur le port sur lequel il écoute. Lorsqu'un client se connecte sur le pare-feu, le mandataire accepte la connexion, établit une seconde connexion au serveur interne, et véhicule les données entre les deux connexions.

Des mandataires simples peuvent être crées avec inetd(8) et nc(1). L'entrée suivante dans le fichier `/etc/inetd.conf` crée une socket d'écoute rattachée à l'adresse de loopback (127.0.0.1) et le port 5000. Les connexions sont redirigées sur le port 80 du serveur 192.168.1.10.

```
127.0.0.1:5000 stream tcp nowait nobody /usr/bin/nc nc -w \
   20 192.168.1.10 80
```

La règle de redirection suivante redirige le port 80 sur l'interface interne du mandataire :

```
rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> \
   127.0.0.1 port 5000
```

Combinaison RDR et NAT

A l'aide d'une règle NAT supplémentaire sur l'interface interne, la traduction manquante d'adresse source décrite plus haut peut être réalisée.

```
rdr on $int_if proto tcp from $int_net to $ext_if port 80 -> \
   $server
no nat on $int_if proto tcp from $int_if to $int_net
nat on $int_if proto tcp from $int_net to $server port 80 -> \
   $int_if
```

Ceci aura pour effet d'effectuer une autre traduction d'adresse du paquet initial envoyé par le client lorsque celui-ci est redirigé à travers l'interface interne. Cette seconde opération de traduction remplacera l'adresse source du client par l'adresse interne du pare-feu. Le serveur interne répondra alors à l'adresse interne du pare-feu, qui effectuera les opérations de NAT et de RDR inverses avant de faire suivre le paquet au client. Cette méthode est relativement complexe. Elle crée deux états séparés pour chaque connexion redirigée. Il faut prendre des précautions pour éviter que la règle de NAT ne s'applique au reste du trafic tel que les

connexions en provenance de hôtes externes (via d'autres redirections) ou en provenance du pare-feu lui-même. La règle `rdr` précitée soumettra à la pile TCP/IP des paquets en provenance du réseau interne et à destination de ce dernier.

Nous recommandons cependant de privilégier plutôt les autres solutions précédemment mentionnées.

[Précédent : Traduction d'Adresses Réseau] [Index] [Suivant : Raccourcis pour la Création de Jeux de Règles]


www@openbsd.org

$OpenBSD: rdr.html,v 1.18 2006/05/02 17:09:33 jufi Exp $

[Précédent : Redirection de Trafic ("Forwarding" de Ports)] [Index] [Suivant : Options de Fonctionnement]

PF : Raccourcis pour la Création de Jeux de Règles

Table des Matières

Introduction

PF offre plusieurs moyens pour simplifier un jeu de règles. Quelques bons exemples sont les macros et les listes. De plus, le langage d'écriture des jeux de règles, ou grammaire, offre aussi quelques raccourcis pour rendre un jeu de règles plus simple. En règle générale, plus un jeu de règles est facile plus il est facile de le comprendre et le maintenir.

Utiliser les Macros

Les macros sont utiles car elles fournissent une alternative au codage en dur des adresses, des numéros de ports, des noms d'interfaces etc. dans un jeu de règles. Est-ce que l'adresse IP d'un serveur a été modifiée ? Pas de problème, il suffit de mettre à jour la macro; nul besoin de retoucher les règles de filtrage que vous avez mis du temps et de l'énergie à construire selon vos besoins.

Une convention usuelle dans l'écriture des jeux de règles PF est de définir une macro pour chaque interface réseau. Si une carte réseau doit être remplacée par une autre carte qui utilise un pilote différent (en changeant une 3Com par une Intel par exemple), la macro est mise à jour et les règles de filtrage fonctionneront comme avant. Un autre bénéfice est le déploiement du même jeu de règles sur plusieurs machines. Certaines de ces machines peuvent avoir des cartes réseau différentes. L'utilisation de macros pour définir les cartes réseau permet d'installer les jeux de règles avec un minimum d'édition. L'utilisation de macros pour définir des informations dans un jeu de règles sujet à changements, telles que les numéros de ports, les adresses IP, et les noms d'interfaces, est une pratique recommandée.

```
# définition de macros pour chaque interface réseau
IntIF = "dc0"
ExtIF = "fxp0"
DmzIF = "fxp1"
```

Une autre convention usuelle est d'utiliser les macros pour définir des adresses IP et les blocs réseau. Ceci aura pour effet de réduire de manière significative les opérations de maintenance d'un jeu de règles lorsque les adresses IP y figurant changent.

```
# définition de nos réseaux
IntNet = "192.168.0.0/24"
ExtAdd = "24.65.13.4"
DmzNet = "10.0.0.0/24"
```

Si le réseau interne doit être étendu ou modifié, il suffit de mettre à jour la macro :

```
IntNet = "{ 192.168.0.0/24, 192.168.1.0/24 }"
```

Une fois le jeu de règles rechargé, tout fonctionnera comme avant.

Utiliser les Listes

Prenons comme exemple quelques bonnes règles à inclure dans vos jeux de règles pour gérer les adresses RFC 1918 qui ne doivent pas être routées sur Internet et qui d'habitude, sont utilisées dans un but malicieux :

```
block in  quick on tl0 inet from 127.0.0.0/8 to any
block in  quick on tl0 inet from 192.168.0.0/16 to any
block in  quick on tl0 inet from 172.16.0.0/12 to any
block in  quick on tl0 inet from 10.0.0.0/8 to any
block out quick on tl0 inet from any to 127.0.0.0/8
block out quick on tl0 inet from any to 192.168.0.0/16
block out quick on tl0 inet from any to 172.16.0.0/12
block out quick on tl0 inet from any to 10.0.0.0/8
```

Simplifions maintenant les règles ci-dessus :

```
block in  quick on tl0 inet from { 127.0.0.0/8, 192.168.0.0/16, \
   172.16.0.0/12, 10.0.0.0/8 } to any
block out quick on tl0 inet from any to { 127.0.0.0/8, \
   192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }
```

Le jeu de règles a été réduit de six lignes : on est passé de huit lignes à deux. On peut encore simplifier en utilisant des macros en conjonction d'une liste :

```
NoRouteIPs = "{ 127.0.0.0/8, 192.168.0.0/16, 172.16.0.0/12, \
   10.0.0.0/8 }"
ExtIF = "tl0"
block in  quick on $ExtIF from $NoRouteIPs to any
block out quick on $ExtIF from any to $NoRouteIPs
```

Notez que les macros et les listes simplifient le fichier `pf.conf` mais que les lignes sont en réalité interprétées par pfctl(8) en plusieurs lignes. Ainsi l'exemple précédent est interprété comme suit :

```
block in  quick on tl0 inet from 127.0.0.0/8 to any
block in  quick on tl0 inet from 192.168.0.0/16 to any
block in  quick on tl0 inet from 172.16.0.0/12 to any
block in  quick on tl0 inet from 10.0.0.0/8 to any
block out quick on tl0 inet from any to 10.0.0.0/8
```

```
block out quick on tl0 inet from any to 172.16.0.0/12
block out quick on tl0 inet from any to 192.168.0.0/16
block out quick on tl0 inet from any to 127.0.0.0/8
```

Comme vous pouvez le voir, la simplification des règles figurant dans le fichier `pf.conf` est une facilité offerte à l'auteur et à la personne chargée de la maintenance de ce fichier. Ce n'est pas une simplification réelle des règles traitées par pf(4).

Les macros peuvent être utilisées n'importe où dans le fichier de règles PF et pas seulement pour définir des adresses et des ports :

```
pre = "pass in quick on ep0 inet proto tcp from "
post = "to any port { 80, 6667 } keep state"

# classe de David
$pre 21.14.24.80 $post

# logement de Nick
$pre 24.2.74.79 $post
$pre 24.2.74.178 $post
```

Les macros précédentes sont interprétées comme suit :

```
pass in quick on ep0 inet proto tcp from 21.14.24.80 to any \
   port = 80 keep state
pass in quick on ep0 inet proto tcp from 21.14.24.80 to any \
   port = 6667 keep state
pass in quick on ep0 inet proto tcp from 24.2.74.79 to any \
   port = 80 keep state
pass in quick on ep0 inet proto tcp from 24.2.74.79 to any \
   port = 6667 keep state
pass in quick on ep0 inet proto tcp from 24.2.74.178 to any \
   port = 80 keep state
pass in quick on ep0 inet proto tcp from 24.2.74.178 to any \
   port = 6667 keep state
```

Grammaire de PF

La grammaire de PF est assez flexible ce qui permet une grande flexibilité dans les jeux de règles. PF est capable de supposer certains mots-clés ce qui veut dire qu'ils n'ont pas besoin d'être explicitement inclus dans une règle, et l'ordonnancement des mots-clés est relâché de telle façon à ce qu'il ne soit pas nécessaire de mémoriser une syntaxe stricte.

Élimination de mots-clé

Pour définir une politique de blocage par défaut, deux règles sont utilisés :

```
block in  all
block out all
```

Ceci peut être réduit à :

```
block all
```

Quand aucune direction n'est spécifiées, PF suppose que la règle s'applique aux paquets passant dans les deux sens.

De manière similaire, les clauses "`from any to any`" et "`all`" peuvent ne pas figurer dans une règle. Par exemple :

```
block in on rl0 all
pass  in quick log on rl0 proto tcp from any to any port 22 keep state
```

peut être simplifiée de la manière suivante :

```
block in on rl0
pass  in quick log on rl0 proto tcp to port 22 keep state
```

La première règle bloque tous les paquets en entrée de n'importe où vers n'importe où sur l'interface rl0, et la deuxième règle laisse passer le trafic TCP sur rl0 à destination du port 22.

Simplification des Règles avec `return`

Un jeu de règles utilisé pour bloquer des paquets et répondre avec un TCP RST ou un ICMP Unreachable peut être écrit comme suit :

```
block in all
block return-rst in proto tcp all
block return-icmp in proto udp all
block out all
block return-rst out proto tcp all
block return-icmp out proto udp all
```

Il peut être largement simplifié comme suit :

```
block return
```

Quand PF voit le mot-clé `return`, il "sait" ce qu'il faut envoyer comme réponse (ou pas en envoyer du tout), selon le protocole du paquet bloqué.

Ordonnancement de Mots-clés

L'ordre selon lequel les mots-clés sont spécifiés est flexible dans la plupart des cas. Par exemple, une règle écrite comme suit :

```
pass in log quick on rl0 proto tcp to port 22 \
   flags S/SA keep state queue ssh label ssh
```

Peut aussi être écrite de cette façon :

```
pass in quick log on rl0 proto tcp to port 22 \
   queue ssh keep state label ssh flags S/SA
```

Des variations similaires fonctionneront aussi.

[Précédent : Redirection de Trafic ("Forwarding" de Ports)] [Index] [Suivant : Options de Fonctionnement]


www@openbsd.org

$OpenBSD: shortcuts.html,v 1.15 2006/08/08 07:14:20 saad Exp $

[Précédent : Raccourcis pour La Création des Jeux de Règles] [Index] [Suivant : Scrub (Normalisation de Paquets)]

PF : Options de Fonctionnement

Des options sont utilisées pour contrôler le fonctionnement de PF. Celles-ci sont spécifiées dans le fichier `pf.conf` en utilisant la directive `set`.

REMARQUE : A partir d'OpenBSD 3.7, le comportement des options de lancement a changé. Auparavant, lorsqu'une option était utilisée, elle n'était jamais remise à sa valeur par défaut, sauf si le jeu de règles était rechargé. A présent, à chaque fois qu'un jeu de règles est chargé, les options de lancement sont remises à leurs valeurs par défaut avant que le jeu de règles ne soit analysé. Ainsi, si une option est activée puis désactivée depuis le jeu de règles et que le jeu de règles est rechargé, l'option sera remise à sa valeur par défaut.

`set block-policy` *`option`*

Paramètre le comportement pour par défaut des règles de filtrage qui ont pour effet de bloquer les paquets (mot-clé "`block`").

- `drop` - Le paquet est détruit sans aucune notification.
- `return` - un paquet TCP RST est renvoyé pour les paquets TCP et un paquet ICMP Unreachable est renvoyé pour tous les autres.

Il est à noter que les règles de filtrage individuelles peuvent avoir leur propre réponse. `drop` est l'option par défaut.

`set debug` *`option`*

Permet de paramétrer le niveau de débogage de pf.

- `none` - aucun message de débogage n'est affiché.
- `urgent` - messages de débogage générés pour les erreurs sérieuses.
- `misc` - messages de débogage générés pour des erreurs diverses (par exemple pour voir le status du sous-système scrub et pour les échecs de création d'état).
- `loud` - messages de débogage générés dans des conditions courantes (par exemple pour voir le status de l'analyseur passif de signatures OS).

`urgent` est l'option par défaut.

`set debug` *`option`*

Permet de paramétrer le fichier à partir duquel seront chargées les signatures de systèmes d'exploitation. Utilisé avec l'analyse passive de l'empreinte des OS. La valeur par défaut est `/etc/pf.os`.

`set limit` *`option value`*

Permet de paramétrer différentes limites.

- `frags` - nombre maximal d'entrées dans la zone mémoire utilisée pour le réassemblage de paquets (règles scrub). La valeur par défaut est 5000.
- `src-nodes` - nombre maximal des entrées dans la zone mémoire utilisée pour assurer un suivi des adresses IP sources (généré par les options `sticky-address` et `source-track`). La valeur par défaut est 10000.
- `states` - nombre maximal d'entrées dans la zone mémoire utilisée pour les entrées de la table d'état (règles de filtrage qui contiennent le mot-clé `keep state`). La valeur par défaut est 10000.

`set loginterface` *`interface`*

Paramètre l'interface pour laquelle PF devra récupérer des statistiques telles que le nombre d'octets entrants/sortant les paquets acceptés/bloqués. Les statistiques ne peuvent être récupérées que pour *une* interface à la fois. Il est à noter que les indicateurs `match`, `bad-offset`, etc., et les indicateurs de la table d'état sont enregistrés que `loginterface` soit positionnée ou pas. Pour désactiver cette option, positionnez la à `none`. `none` est l'option par défaut.

`set optimization` *`option`*

Optimise PF pour l'un de ces environnements réseau :

- `normal` - convient à pratiquement tous les réseaux.
- `high-latency` - réseaux à haute latence tels que les réseaux à connexion satellite.
- `aggressive` - les connexions expirent plus rapidement de la table d'état. Les pré-requis mémoire sont ainsi fortement réduits sur un pare-feu particulièrement chargé au risque de terminer des connexions inactives trop rapidement.
- `conservative` - paramétrage extrêmement conservateur. Contrairement à aggressive, ce paramétrage évite de terminer les connexions inactives ce qui se traduit par une plus grande utilisation mémoire et une utilisation du processeur un peu plus soutenue.

`normal` est l'option par défaut.

`set skip on` *`interface`*

Saute *tous* les traitements PF sur l'*`interface`*. Ceci peut être utile sur une interface de loopback pour laquelle le filtrage, la normalisation, la mise en queue etc. ne sont pas nécessaires. Elle peut être utilisée plusieurs fois. Elle n'est pas activée par défaut.

`set state-policy` *`option`*

Permet de paramétrer le comportement de PF vis-à-vis de la préservation de l'état des connexions. Ce comportement peut être contourné au niveau de chaque règle. Pour plus de détails, consultez la section de la FAQ intitulée Préservation de l'État.

- `if-bound` - les états sont liés à l'interface sur lesquels ils ont été créés. Si le trafic correspond à une entrée de la table d'états mais qu'il ne traverse pas l'interface sur laquelle l'état a été enregistré, cette correspondance est rejetée. Le paquet doit alors correspondre à une règle de filtrage ou il sera tout simplement détruit/rejeté.
- `group-bound` - même comportement que `if-bound` mis à part que les paquets sont autorisés à travers des interfaces du même groupe, par exemple toutes les interfaces `ppp` etc.
- `floating` - les états peuvent correspondre à des paquets sur n'importe quelle interface. Peu importe l'interface qu'il traverse, un paquet sera accepté tant qu'il correspond à une entrée dans la table d'états et qu'il transite dans le même sens qu'il le fit initialement lors de la création de l'état, peu importe l'interface qu'il traverse, il passera.

`floating` est l'option par défaut.

`set timeout` *`option value`*

Permet de paramétrer différents timeouts (en secondes).

- `interval` - nombre de secondes entre les purges d'états expirés et des fragments de paquets. La valeur par défaut est `10`.
- `frag` - nombre de secondes avant qu'un fragment non assemblé n'expire. La valeur par défaut est `30`.
- `src.track` - nombre de secondes pendant lesquelles garder l'entrée source tracking en mémoire après que le dernier état ait expiré. La valeur par défaut est `0` (zero).

Exemple :

```
set timeout interval 10
set timeout frag 30
set limit { frags 5000, states 2500 }
set optimization high-latency
set block-policy return
set loginterface dc0
set fingerprints /etc/pf.os.test
set skip on lo0
set state-policy if-bound
```

[Précédent : Raccourcis pour La création des Jeux de Règles] [Index] [Suivant : Scrub (Normalisation de Paquets)]


www@openbsd.org

$OpenBSD: options.html,v 1.13 2006/03/10 16:35:07 saad Exp $

[Précédent : Options de Fonctionnement] [Index] [Suivant : Ancres]

PF : Scrub (Normalisation de Paquets)

Table des Matières

Introduction

Le "scrubbing" est la normalisation de paquets utilisée pour supprimer toute ambiguïté dans l'interprétation d'un paquet qui sera effectuée par la destination finale de ce dernier. La directive scrub réassemble aussi des paquets fragmentés, afin de protéger certains systèmes d'exploitation de quelques types d'attaques. Cette directive rejette aussi les paquets TCP contenants des combinaisons invalides de drapeaux. Voici un exemple simple d'utilisation de la directive scrub :

```
scrub in all
```

Ceci aura pour effet d'appliquer le "scrub" sur tous les paquets en entrée de chaque interface.

Il existe des cas où il ne faut pas appliquer le "scrub". Par exemple, une interface qui véhicule du trafic NFS à travers PF. Certaines plates-formes non basées sur OpenBSD envoient (et s'attendent à recevoir) des paquets étranges -- des paquets fragmentés avec le bit "do not fragment" (ne pas fragmenter) positionné, qui sont rejetés (comportement normal) par `scrub`. On peut résoudre ce problème en utilisant l'option `no-df`. Un autre exemple est l'utilisation de jeux multi-joueurs qui ont des problèmes de connexion à travers PF lorsque `scrub` est activé. Mis à part ces cas quelque peu inhabituels, la normalisation de paquets "scrub" est une pratique *hautement recommandée*.

La syntaxe de la directive `scrub` est très similaire à la syntaxe de filtrage ce qui rend aisé la normalisation de certains paquets et pas les autres. Le mot-clé `no` peut être utilisé devant la directive `scrub` pour ne pas normaliser certains paquets. La première règle correspondante est appliquée, comme pour les règles `nat`.

Vous pouvez trouver plus d'informations concernant le principe et les concepts de la normalisation de paquets dans l'article *Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics*.

Options

Les options de `scrub` sont les suivantes :

`no-df`
: *Supprime le bit don't fragment de l'en-tête du paquet IP. Quelques systèmes d'exploitation sont connus pour générer des paquets fragmentés avec le bit don't fragment positionné. Ceci est particulièrement vrai dans le cas de NFS.* `scrub` *va bloquer tous les paquets qui sont dans ce cas sauf si l'option* `no-df` *est spécifiée. Vu que certains systèmes d'exploitation génèrent des paquets don't fragment avec un champ d'identification à zéro au niveau de l'en-tête IP, il est recommandé d'utiliser* `no-df` *conjointement avec l'option* `random-id`*.*

`random-id`
: *Remplace le champ d'identification IP des paquets avec des valeurs aléatoires pour contourner les valeurs prévisibles utilisées par certains systèmes d'exploitation. Cette option s'applique aux paquets qui ne sont pas fragmentés après le réassemblage optionnel des paquets.*

`min-ttl num`
: *S'assure que le Time To Live (TTL) est au moins égal à la valeur donnée en argument dans les en-têtes des paquets IP.*

`max-mss num`
: *S'assure que le Maximum Segment Size (MSS) est au plus égal à la valeur donnée en argument dans les en-têtes des paquets TCP.*

`fragment reassemble`
: *Met dans une mémoire tampon les fragments de paquets et les réassemble en paquet complet avant de les transmettre au moteur de filtrage. Les règles de filtrage peuvent ainsi se charger de filtrer le paquet complet sans se soucier des fragments. L'inconvénient est la mémoire additionnelle nécessaire pour le tampon contenant les fragments de paquets. C'est le comportement par défaut lorsqu'aucune option* `fragment` *n'est spécifiée. C'est aussi la seule option* `fragment` *qui fonctionne avec la NAT.*

`fragment crop`
: *Supprime les fragments dupliqués et tout chevauchement entre fragment. Contrairement à* `fragment reassemble`*, les fragments ne sont pas gardés en mémoire tampon mais sont transmis dès leur arrivée.*

`fragment drop-ovl`
: *Assez similaire à* `fragment crop`*. Tous les paquets fragments de paquets dupliqués et se chevauchant sont supprimés ainsi que tous les fragments suivants qui correspondent à ces fragments.*

`reassemble tcp`
: *Normalise de manière "stateful" les connexions TCP. Lorsque* `scrub reassemble tcp` *est utilisée, une direction (in/out) peut ne pas être spécifiée. Les normalisations suivantes sont effectuées :*
  - *Aucune extrêmité de la connexion n'est autorisée à réduire le TTL IP. Cette normalisation est appliquée pour assurer une protection contre un attaquant qui envoie un paquet de telle façon à ce que ce dernier atteigne le pare-feu, affecte les informations d'état mémorisées pour cette connexion, et expire avant d'atteindre sa destination finale. Le TTL de tous les paquets est positionné à la valeur la plus haute observée pour la connexion.*
  - *Module les "timestamps" RFC1323 dans l'en-tête des paquets TCP avec un nombre aléatoire. Ceci peut empêcher un observateur de déduire le "uptime" d'un hôte ou de deviner combien de hôtes sont derrière une passerelle NAT.*

Exemples :

```
scrub in on fxp0 all fragment reassemble min-ttl 15 max-mss 1400
scrub in on fxp0 all no-df
scrub    on fxp0 all reassemble tcp
```

[Précédent : Options de Fonctionnement] [Index] [Suivant : Ancres]


www@openbsd.org

$OpenBSD: scrub.html,v 1.14 2006/10/29 10:58:53 jufi Exp $

[Précédent : Scrub (Normalisation de Paquets)] [Index] [Suivant : Gestion de La Bande Passante]

PF : Ancres

Table des Matières

Introduction

En plus de la base de règles principale, PF peut aussi évaluer des bases de règles secondaires. Vu que les bases de règles secondaires ("sub rulesets" en anglais) peuvent être manipulées à la volée en utilisant pfctl(8), elles fournissent un bon moyen de modifier dynamiquement l'ensemble des règles actives. Alors qu'une table est utilisée pour contenir une liste dynamique d'adresses, une base de règle secondaire est utilisée pour contenir un ensemble dynamique de règles de filtrage, `nat`, `rdr`, et `binat`.

Les bases de règles secondaires sont attachées à la base de règles principale à l'aide d'ancres. Il existe quatre types de règles `anchor` (ancre en anglais) :

- `anchor` *`nom`* - évalue toutes les règles de filtrage de l'ancre *`nom`*
- `binat-anchor` *`nom`* - évalue toutes les règles de traduction `binat` de l'ancre *`nom`*
- `nat-anchor` *`nom`* - évalue toutes les règles de traduction `nat` de l'ancre *`nom`*
- `rdr-anchor` *`nom`* - évalue toutes les règles `rdr` de l'ancre *`nom`*

Les ancres peuvent être encastrées, ce qui permet aux sous-règles d'êtres enchainées ensembles. Les règles d'ancres seront évaluées par rapport à l'ancre depuis laquelle elles sont chargées. Par exemple, les règles d'ancres de la base de régles principale peuvent créer des points d'ancrage avec la base de règles principale comme parent, et les règles d'ancres chargées à partir de fichiers à l'aide de la directive `load anchor` créeront des points d'ancrage définissant cette ancre comme parent.

Ancres

Une ancre est une collection de règles de filtrage et/ou de traduction, des tables, et d'autres ancres auxquelles un nom a été affecté. Lorsque PF lit une règle `anchor` dans le jeu de règles principal, il va évaluer les règles contenues dans ce point d'ancre de la même manière qu'il évalue des règles dans la base de règles principale. Le traitement de la base de règles principale se poursuivra ensuite à moins que le paquet en cours de traitement ne corresponde à une règle de filtrage qui utilise le mot-clé `quick` ou à une règle de traduction dans l'ancre, auquel cas la correspondance est déclarée comme finale et l'évaluation sera arrêtée aussi bien dans les bases de règles rattachées à l'ancre que dans la base de règles principale.

Par exemple :

```
ext_if = "fxp0"

block on $ext_if all
pass  out on $ext_if all keep state
anchor goodguys
```

Cette base de règles implémente une politique dite restrictive (tout ce qui n'est pas connu est interdit par défaut) sur l'interface `fxp0` aussi bien pour le trafic entrant que le trafic sortant. Le trafic sortant est ensuite autorisé et un état est gardé. Ensuite une règle d'ancrage est créée avec le nom `goodguys`. Les ancres peuvent être peuplées avec des règles de deux façons différentes :

- en utilisant une règle de chargement `load`
- en utilisant pfctl(8)

La règle de chargement `load` est utilisée pour ordonner à `pfctl` de peupler l'ancre spécifiée à partir d'un fichier texte. La règle `load` doit être placée après la règle `anchor`. Par exemple :

```
anchor goodguys
load anchor goodguys from "/etc/anchor-goodguys-ssh"
```

Pour ajouter des règles à une ancre à l'aide de `pfctl`, le type de commandes suivant devra être employé :

```
# echo "pass in proto tcp from 192.0.2.3 to any port 22" \
   | pfctl -a goodguys -f -
```

Les règles peuvent aussi être sauvegardées et chargées à partir d'un fichier texte :

```
# cat >> /etc/anchor-goodguys-www
pass in proto tcp from 192.0.2.3 to any port 80
pass in proto tcp from 192.0.2.4 to any port { 80 443 }

# pfctl -a goodguys -f /etc/anchor-goodguys-www
```

Des règles de filtrage et de traduction peuvent être chargées dans une base de règles nommée en utilisant les mêmes syntaxe et options utilisées par les règles définies dans la base de règles principale. Une exception est à signaler cépendant dans le cas des macros. Les macros utilisées dans une base de règles nommée doivent aussi être définies dans la base de règles nommée qui les utilisent; les macros définies dans la base de règles principale ne sont *pas* visibles par les bases de règles nommées.

Etant donné que les ancres peuvent être encastrées, il est possible de demander l'évaluation de toutes les ancres fille :

```
anchor "spam/*"
```

Cette syntaxe cause l'évaluation de chaque règle attachée à l'ancre `spam`. Les ancres fille seront évaluées dans l'ordre alphabétique mais ne sont pas parcourues récursivement. Les ancres sont toujours relatives à l'ancre dans laquelle elles sont définies.

Chaque ancre, ainsi que la base de règles principale, existe séparemment des autres jeux de règles. Les opérations effectuées sur une base de règles, telles que la suppression des règles, n'a aucun effet sur les autres bases. De plus, la suppression d'un point d'ancrage dans la base de règles principale ne détruit pas l'ancre ni ancres attachées à celles- ci. Une ancre n'est détruite que lorsque toutes les ancres qu'elle contient sont supprimées à l'aide de pfctl(8).

Options d'Ancrage

Optionnellement, les règles d'ancrage `anchor` peuvent spécifier une interface, un protocole, un port source, un port destination, une balise ... en utilisant la même syntaxe que celle employée dans les règles de filtrage. Lorsque de telles informations sont fournies, les règles d'ancrage `anchor` sont uniquement évaluées lorsque le paquet en cours de traitement correspond à la définition de la règle `anchor`. Par exemple :

```
ext_if = "fxp0"

block on $ext_if all
pass  out on $ext_if all keep state
anchor ssh in on $ext_if proto tcp from any to any port 22
```

Les règles de l'ancre `ssh` sont uniquement évaluées pour les paquets TCP destinés au port 22 en entrée de `fxp0`. Ces règles sont alors ajoutées à l'ancre (`anchor`) de la façon suivante :

```
# echo "pass in from 192.0.2.10 to any" | pfctl -a ssh -f -
```

Ainsi, même si la règle de filtrage ne spécifie ni l'interface, ni le protocole ni d'autres paramètres tels que le port, le hôte 192.0.2.10 sera uniquement autorisé à faire des connexions SSH vu la définition de la règle d'ancrage (`anchor`).

Manipulation des Ancres

La manipulation des ancres se fait à l'aide de `pfctl`. Cette commande peut être utilisée pour ajouter ou supprimer des règles d'une ancre sans nécessiter le rechargement de la base de règles principale.

Pour lister toutes les règles de l'ancre `ssh` :

```
# pfctl -a ssh -s rules
```

Pour supprimer toutes les règles de filtrage de la même ancre :

```
# pfctl -a ssh -F rules
```

Pour une liste complète des commandes, veuillez consulter pfctl(8).

[Précédent : Scrub (Normalisation de Paquets)] [Index] [Suivant : Gestion de La Bande Passante]


www@openbsd.org

$OpenBSD: anchors.html,v 1.14 2006/05/02 17:09:33 jufi Exp $

[Précédent : Ancres] [Sommaire] [Suivant : Ensembles d'adresses ("Pools") et Partage de Charge]

PF : Gestion de La Bande Passante

Table des Matières

Mise en queue

Mettre en queue quelque chose signifie stocker, de manière ordonnée, cette chose pendant qu'elle attend un traitement. Dans un réseau informatique, quand les paquets de données sont envoyés à l'extérieur de l'hôte, ils entrent dans une queue où ils attendent un traitement par le système d'exploitation. Celui-ci décide ensuite quelle queue et quel(s) paquet(s) de cette queue doivent être traités. L'ordre dans lequel le système d'exploitation selectionne les paquets à traiter peut affecter les performances du réseau. Par exemple, immaginez un utilisateur lançant deux applications réseau : SSH et FTP. Idéalement, les paquets SSH devraient être traités avant les paquets FTP à cause de la sensibilité au temps de SSH; quand une clef est entrée dans le client SSH, une réponse immédiate est attendue, mais un transfert FTP retardé de quelques secondes n'attirera pratiquement pas l'attention. Mais que se passe t'il si le routeur manipulant ces connexions traite un grand nombre de paquets provenant de la connexion FTP avant de traiter la connexion SSH ? Les paquets de la connexion SSH resteront dans la queue (ou seront rejetés par le routeur si la queue n'est pas assez grande pour acceuillir tous les paquets) et la session SSH apparaitra certainement comme ralentie ou haletante. En modifiant la stratégie de mise en queue utilisée, la bande bassante du réseau peut être partagée de façon juste entre les différents applications, utilisateurs et ordinateurs.

Notez que la mise en queue est utile uniquement pour les paquets sortant ("*outbound*"). Une fois qu'un paquet arrive sur une interface dans la direction entrante il est déjà trop tard pour le mettre en queue -- la bande passante a déjà été consommée et l'interface le reçoit simplement. La seule solution est d'activer la mise en queue sur un routeur adjacent ou, si l'hôte recevant le paquet agit comme un routeur, d'activer la mise en queue sur l'interface interne sur laquelle les paquets quittent le routeur.

Planificateurs

L'algorithme décide des queues à traiter et dans quel ordre. Par défaut, OpenBSD utilise un algorithme "First In First Out" (FIFO). Une queue FIFO travaille comme une caisse de supermarché -- le premier article de la queue est le premier traité. Au fur et à mesure que des nouveaux paquets arrivent, ils sont ajoutés à la queue. Si la queue est pleine - et c'est là que l'analogie avec la caisse de supermarché s'arrête - les nouveaux paquets sont rejetés. Ceci est connu sous le nom de "tail- drop".

OpenBSD supporte deux algorithmes additionels :

- Mise en queue par classes
- Mise en queue par priorités

Mise en queue par classes

La Mise en queue par classes (CBQ) est un algorithme de mise en queue qui divise la bande passante des multiples connexions réseau en queues ou en classes. Chaque queue se voit ainsi assigner un trafic sur les bases d'une adresse source ou de destination, d'un numéro de port, d'un protocol, etc. Une queue peut

optionnellement être configurée pour emprunter de la bande passante aux queues parent si celles-ci sont sous- utilisées. Les queues ont aussi une priorité selon qu'elles contiennent du trafic interactif, comme SSH, et peuvent avoir leurs paquets traités avant les queues contenant le trafic volumineux, comme FTP.

Les queues CBQ sont ordonnées de manière hiérarchique. Au sommet de la hiérarchie se trouve la queue "root" qui définit la totalité de la bande passante disponible. Les queues enfant sont crées sous la queue "root", chacune d'elle peut se voir assigner une partie de la bande passante de la queue "root". Par exemple, les queues peuvent être définies comme suit :

```
Root Queue (2Mbps)
        Queue A (1Mbps)
        Queue B (500Kbps)
        Queue C (500Kbps)
```

Dans ce cas, la totalité de la bande passante est fixée à 2 megabits par seconde (Mbps). Cette bande passante est ensuite subdivisée en trois queues enfant.

La hiérarchie peut encore être approfondie en définissant des queues dans les queues. Pour subdiviser équitablement la bande passante entre les différents utilisateurs et aussi classer leur trafic pour éviter que certains protocoles ne privent de bande passante d'autres protocoles, une structure de mise en queue comme celle-ci peut être définie :

```
Root Queue (2Mbps)
        UserA (1Mbps)
                ssh (50Kbps)
                bulk (950Kbps)
        UserB (1Mbps)
                audio (250Kbps)
                bulk (750Kbps)
                        http (100Kbps)
                        other (650Kbps)
```

Notez qu' à chaque niveau, la somme des bandes passantes assignées à chaque queue ne doit pas être supérieure à celle assignée à la queue parent.

Une queue peut être configurée pour emprunter de la bande passante aux queues parent si ces dernières en ont en excès suite à la non- utilisation par d'autres queues enfant. Considérez le réglage de mise en queue suivant :

```
Root Queue (2Mbps)
        UserA (1Mbps)
                ssh (100Kbps)
                ftp (900Kbps, borrow)
        UserB (1Mbps)
```

Si le trafic dans la queue `ftp` excède 900Kbps et le trafic dans la queue `UserA` est inférieur à 1Mbps (parce que la queue `ssh` utilise moins que les 100Kbps qu'elle dispose), la queue `ftp` empruntera l'excès de bande passante de `UserA`. Dans cette optique la queue `ftp` est en mesure d'utiliser davantage de bande passante que ce qui lui est assigné quand elle fait face à une surcharge. Quand la queue `ssh` verra sa charge augmenter, la bande passante empruntée sera restituée.

CBQ assigne une priorité à chaque queue. Les queues avec une priorité elevée sont préferées aux queues avec une priorité inférieure pendant une congestion tant que les deux queues partagent le même parent (en d'autres termes, tant que les deux queues appartiennent à une même branche de la hiérarchie). Les queues avec une même priorité sont traitées dans un mode "round-robin". Par exemple :

```
Root Queue (2Mbps)
        UserA (1Mbps, priority 1)
                ssh (100Kbps, priority 5)
                ftp (900Kbps, priority 3)
        UserB (1Mbps, priority 1)
```

CBQ traitera les queues `UserA` et `UserB` dans un mode "round-robin" -- aucune queue ne sera préferée à une autre. Pendant le temps ou la queue `UserA` est traitée, CBQ traitera aussi les queues enfant. Dans ce cas, la queue `ssh` a une priorité plus élevée et aura un traitement préferentiel vis à vis de la queue `ftp` si le réseau est congestionné. Notez que les queues `ssh` et `ftp` n'ont pas leurs priorités comparées à celles de `UserA` et `UserB` à cause du fait qu'elles ne sont pas dans une même branche de la hiérarchie.

Pour un regard plus détaillé sur la théorie du fonctionnement de CBQ, veuillez consulter les Références sur CBQ.

Mise en queue par priorités

La Mise en queue par priorités (PRIQ) assigne des queues à une interface réseau et chaque queue posséde une priorité distincte. Une queue avec une priorité plus

élevée est *toujours* traitée avant une queue avec une priorité moindre.

La structure des queues de PRIQ est plate -- vous ne pouvez pas définir de queue au sein d'une autre queue. La queue "root" est définie et possède la totalité de la bande passante disponible, et les sous-queues sont définies ensuite en aval. Considérez l'exemple suivant :

```
Root Queue (2Mbps)
        Queue A (priority 1)
        Queue B (priority 2)
        Queue C (priority 3)
```

La queue "root" est définie comme ayant 2Mbps de bande passante disponible et trois sous-queues sont définies. La queue avec la plus haute priorité (le nombre priorité le plus grand) est servie en premier. Une fois que tous les paquets dans cette queue ont été traités, ou si la queue est vide, PRIQ s'occupe de la queue avec la priorité suivante. Au sein d'une queue donnée, les paquets sont traités selon la manière "First In First Out" (FIFO).

Il est important de noter que si vous utilisez PRIQ, vous devez créer vos queues très prudemment. A cause du fait que PRIQ traite *toujours* une queue avec une grande priorité avant une queue avec une priorité basse, il est possible qu'une queue avec une priorité élevée cause la suppression ou la disparition des paquets d'une queue avec une priorité basse si la queue avec la priorité élevée reçoit un flux constant de paquets.

Détection Aléatoire Anticipée

La Détection Aléatoire Anticipée (RED) est un algorithme d'évitement des congestions. Sa fonction est d'éviter les congestions en s'assurant que la queue ne devienne pas pleine. Elle réalise cela en calculant en permanence la largeur moyenne (taille) de la queue et en la comparant à deux valeurs de seuil, un seuil minimum et un seuil maximum. Si la taille moyenne de la queue est au dessous du seuil minimal, aucun paquet n'est rejeté. Si la moyenne est au dessus du seuil maximum alors *tous* les nouveaux paquets arrivant sont rejetés. Si la moyenne est comprise entre les seuils minimum et maximum les paquets sont rejetés suivant une probabilité calculée grâce à la taille de la queue. En d'autres mots, quand l'occupation de la queue approche le seuil maximum, de plus en plus de paquets sont rejetés. Quand il rejete des paquets, RED choisi aléatoirement la connexion pour laquelle il va rejeter les paquets. Les connexions utilisant largement la bande passante sont une plus grande probabilité de voir leurs paquets rejetés.

RED est utile car il évite la situation connue sous le nom de synchronisation globale et il peut s'adapter à des bonds du trafic. La synchronisation globale se réferre à une perte totale des données sortantes due à la suppression simultanée de paquets de plusieurs connexions. Par exemple, si la congestion apparait sur un routeur supportant le trafic de 10 connexions FTP et tous les paquets de ces connexions sont rejetés (comme c'est le cas avec une mise en queue FIFO), la totalité du trafic sortant sera rejeté; brusquement. Ce n'est pas une situation idéale car elle entraîne une réduction du trafic de toutes les connexions FTP ce qui signifie que le réseau n'est plus utilisé avec son potentiel maximum. RED évite cela en choisissant aléatoirement une de ces connexions plutôt que de choisir toutes les connexions. Les connexions utilisant une large partie de la bande passante ont une probabilité supérieure de voir leurs paquets rejetés. Dans cette optique, les connexions avec une haute bande passante seront étranglées, la congestion sera évitée, et les pertes violentes de trafic de sortie n'auront pas lieu. De plus, RED est capable de supporter des explosions du trafic car il commence la suppression des paquets *avant* que la queue ne devienne pleine. Si une explosion du trafic se produit, il y aura assez d'espace pour mémoriser les nouveaux paquets.

RED ne devrait être utilisé que quand le protocole est capable de répondre aux indicateurs de congestion du réseau. Dans beaucoup de cas cela signifie que RED devrait être utilisé pour mettre en queue du trafic TCP et non UDP ou ICMP trafic.

Pour une regard plus détaillé sur la théorie derrière RED, veuillez consulter References on RED.

Notification Explicite de Congestion

La Notification Explicite de Congestion (Early Congestion Notification ou ECN) travaille en conjonction avec RED pour notifier deux hôtes communicant via le réseau d'une congestion le long du chemin de communication. Ceci est fait en autorisant RED à fixer un drapeau dans l'entête du paquet plutôt que de rejeter le paquet. Si l'on part du principe que l'hôte expediteur supporte ECN, il pourra lire ce drapeau et diminuer son trafic réseau en fonction.

Pour plus d'informations sur ECN, veuillez consulter RFC 3168.

Configuration de la Mise en queue

Depuis OpenBSD 3.0 la Mise en queue alternée (ALTQ), implémentation de la mise en queue, fait partie du système de base. En commençant avec OpenBSD 3.3 ALTQ a été intégrée dans PF. L'implémentation de ALTQ dans OpenBSD supporte les algorithmes de Mise en Queue par classes (CBQ) et de Mise en Queue par priorités (PRIQ). Elle supporte aussi la Détection Aléatoire Anticipée (RED) et la Notification de Congestion Explicite (ECN).

Parce que ALTQ a été fusionnée avec PF, ce dernier doit être activé pour que la mise en queue fonctionne. Les instructions sur la procédure d'activation de PF peuvent être trouvées dans les Principes de base.

La Mise en queue est configurée dans `pf.conf`. Il y a deux types de directives utilisées pour configurer la Mise en Queue :

- `altq on` - active la mise en queue sur une interface, définit l'algorithme à utiliser, et créée la queue "root"
- `queue` - définit les propriétés d'une queue enfant

La syntaxe de la directive `altq on` est :

```
altq on interface scheduler bandwidth bw qlimit qlim \
   tbrsize size queue { queue_list }
```

- *`interface`* - l'interface réseau sur laquelle activer la mise en queue.
- *`scheduler`* - l'algorithme à utiliser. Les valeurs possibles sont `cbq` et `priq`. Actuellement, seul un algorithme à la fois peut être activé sur une interface.
- *`bw`* - la quantité totale de bande passante disponible pour l'algorithme. Elle peut être spécifiée avec une valeur absolue en utilisant les suffixes `b`, `Kb`, `Mb`, et `Gb` signifiant bits, kilobits, megabits, et gigabits par seconde, en valeur ou en pourcentage de la bande passante de l' *`interface`*.
- *`qlim`* - le nombre maximum de paquets à stocker dans une queue. Ce paramètre est optionnel. Il est par défaut de 50.
- *`size`* - la taille du régulateur en octets. Si elle n'est pas spécifiée, la taille utilisée est basée sur la bande passante de l' *`interface`*.
- *`queue_list`* - une liste de queues enfant à créer en dessous de la queue "root".

Par exemple :

```
altq on fxp0 cbq bandwidth 2Mb queue { std, ssh, ftp }
```

Ceci active CBQ sur l'interface `fxp0`. La totalité de la bande passante disponible est fixée à 2Mbps. Trois queues enfant sont définies : `std`, `ssh`, et `ftp`.

La syntaxe pour la directive `queue` est :

```
queue name [on interface] bandwidth bw [priority pri] [qlimit qlim] \
   scheduler ( sched_options ) { queue_list }
```

- *`name`* - le nom de la queue. Il doit correspondre à un des noms de queues définies dans la directive `altq on` *`queue_list`*. Pour `cbq` il peut aussi correspondre au nom d'une queue dans la directive `queue` précédente *`queue_list`*. Les noms de queue ne doivent pas être plus longs que 15 caractères.
- *`interface`* - l'interface réseau pour laquelle la queue est valide. Cette valeur est optionnelle et si elle n'est pas précisée, la queue sera valide pour toutes les interfaces.
- *`bw`* - la quantité totale de bande passante disponible pour la queue. Elle doit être spécifiée avec une valeur absolue en utilisant les suffixes `b`, `Kb`, `Mb`, et `Gb` signifiant bits, kilobits, megabits, et gigabits par seconde, en valeur ou en pourcentage de la bande passante de la queue parent. Ce paramètre n'est applicable que lorsque l'on utilise l'algorithme `cbq`. Si non spécifié, 100% de la bande passante de la queue mère est utilisé.
- *`pri`* - la priorité de la queue. Pour `cbq` la gamme de priorités va de 0 à 7 et pour `priq` la gamme va de 0 à 15. La priorité 0 est la priorité la plus basse. Quand elle n'est pas spécifiée, elle est par défaut de 1.
- *`qlim`* - le nombre maximum de paquets acceptables dans une queue. Quand il n'est pas spécifié, la valeur utilisée par défaut est de 50.
- *`scheduler`* - l'algorithme utilisé, soit `cbq` soit `priq`. Il doit être le même que celui de la queue "root".
- *`sched_options`* - d'autres options peuvent être passées à l'algorithme pour contrôler son comportement :
  - `default` - définit une queue par défaut dans laquelle tous les paquets ne répondant pas à une quelconque règle vont. Cette queue par défaut doit être unique.
  - `red` - active la Détection Aléatoire Anticipée (RED) pour cette queue.
  - `rio` - active RED dans les sens ENTREE/SORTIE. Dans ce mode, RED maintiendra des longueurs moyennes de queues et des valeurs de seuil, une pour chaque niveau de Qualité de service.
  - `ecn` - active la Notification Explicite de Congestion (ECN) pour cette queue. `Ecn` implique `red`.
  - `borrow` - la queue peut emprunter de la bande passante aux queues parent. Ceci n'est valable que lorsque vous utilisez l'algorithme `cbq`.
- *`queue_list`* - une liste de queues enfant à créer sous cette queue. Une *`queue_list`* ne peut être définie que lorsque l'on utilise l'algorithme `cbq`.

Suite de l'exemple ci-dessus :

```
queue std bandwidth 50% cbq(default)
queue ssh bandwidth 25% { ssh_login, ssh_bulk }
  queue ssh_login bandwidth 25% priority 4 cbq(ecn)
  queue ssh_bulk bandwidth 75% cbq(ecn)
queue ftp bandwidth 500Kb priority 3 cbq(borrow red)
```

Ici, les paramètres des queues enfant définies précédemment sont définis. La queue `std` se voit attribuer une bande passante correspondant à 50% de celle de la queue "root" (1 Mbps) et est définie comme queue par défaut. La queue `ssh` se voit assigner 25% de la bande passante de la queue racine (500kb) et contient également deux autres queues, `ssh_login` et `ssh_bulk`. La queue, `ssh_login` a une priorité plus élevée que `ssh_bulk` et les deux ont ECN activé. La queue `ftp` obtient une bande passante de 500Kbps et obtient une priorité de 3. Elle peut aussi emprunter de la bande passante quand une quantité supplémentaire est disponible et RED est activé.

REMARQUE : Chaque définition de queue enfant a sa bande passante spécifiée. Sans spécifier la bande passante, PF donnera 100% de la bande passante de la

queue parente. Dans cette situation, ceci causera une erreur lors du chargement des règles sauf s'il y a une queue avec 100% de la bande passante, aucune autre queue ne peut être déclarée à ce niveau, car il n'y a plus de bande passante à allouer.

Assignation du Trafic à une Queue

Pour assigner du trafic à une queue, le mot-clé `queue` est utilisé en conjonction avec les règles de filtrage de PF. Par exemple, considérons un jeu de règles de filtrage contenant une ligne comme :

```
pass out on fxp0 from any to any port 22
```

Les paquets répondant à cette règle peuvent être assignés à une queue spécifique en utilisant le mot-clé `queue` :

```
pass out on fxp0 from any to any port 22 queue ssh
```

Lorsque l'on utilise le mot-clé `queue` avec les directives `block`, les paquets TCP RST ou ICMP Unreachable résultants sont assignés dans la queue spécifiée.

Notez que la désignation des queues est possible sur une interface différente de celle définie dans la directive `altq on` :

```
altq on fxp0 cbq bandwidth 2Mb queue { std, ftp }
queue std bandwidth 500Kb cbq(default)
queue ftp bandwidth 1.5Mb

pass in on dc0 from any to any port 21 queue ftp
```

La Mise en queue est activée sur `fxp0` mais la désignation concerne `dc0`. Si les paquets répondant à la règle `pass` sortent de l'interface `fxp0`, ils seront mis en queue dans `ftp`. Ce type de mise en queue peut être très pratique sur les routeurs.

Normalement, un seul nom de queue est donné avec le mot-clé `queue`, mais si un deuxième nom est spécifié cette queue sera utilisée pour les paquets avec un Type de Service (ToS) "low-delay" et pour les paquets TCP ACK sans donnée. Un bon exemple de ceci est l'utilisation de SSH. Les connexions de sessions SSH mettent le ToS en "low-delay" contrairement aux sessions SCP et SFTP. PF peut utiliser cette information pour mettre les paquets correspondant à une connexion dans une queue différente que celles pour les connexions sans ouverture de session. Ceci peut être utile pour donner la priorité aux connexions avec ouverture de session sur les paquets de transfert de fichiers.

```
pass out on fxp0 from any to any port 22 queue(ssh_bulk, ssh_login)
```

Ceci assigne les paquets correspondant aux connexions d'ouverture de session SSH à la queue `ssh_login` et les paquets correspondant aux connexions SCP et SFTP dans la queue `ssh_bulk`. Les connexions d'ouverture de session auront leurs paquets traités avant ceux des connexions SCP et SFTP car la queue `ssh_login` a une priorité plus élevée.

L'assignation des paquets TCP ACK à une queue ayant une priorité élevée est utile pour les connexions asymétriques, c'est à dire les connexions qui ont des bandes passantes différentes en envoi et en réception, comme les lignes ADSL. Avec une ligne ADSL, si le canal d'envoi est utilisé à son maximum et si un téléchargement est lancé, le téléchargement en souffrira car les paquets TCP ACK devant être envoyés seront congestionnés quand ils essaieront de traverser le canal d'envoi. Des tests ont montré que pour réaliser les meilleurs résultats, la bande passante de la queue d'envoi doit être inférieure à la capacité maximale. Par exemple, si une ligne ADSL a un envoi maximum de 640Kbps, le fait de régler la `bande passante` de la queue "root" sur une valeur comme 600Kb entraînera de meilleures performances. C'est après des essais et des erreurs que vous obtiendrez les meilleurs réglages de `bande passante`.

Quand on utilise le mot-clé `queue` avec des règles qui `gardent l'état` comme :

```
pass in on fxp0 proto tcp from any to any port 22 flags S/SA \
   keep state queue ssh
```

PF enregistrera la queue dans la table d'états ce qui entraînera que les paquets traversant `fxp0` au retour et répondant à la connexion "stateful" termineront dans la queue `ssh`. Notez que même si le mot-clé `queue` est utilisé dans une règle filtrant le trafic entrant, le but est de spécifier une queue pour le trafic sortant correspondant; la règle ci-dessus ne mettra pas en queue les paquets entrants.

Exemple #1 : Réseau domestique

```
[ Alice ]    [ Charlie ]
    |             |                              ADSL
 ---+-----+-------+------ dc0 [ OpenBSD ] fxp0 -------- ( Internet )
```

```
    |
[ Bob ]
```

Dans cet exemple, OpenBSD est utilisé sur une passerelle Internet pour un petit réseau à domicile avec trois stations de travail. La passerelle réalise le filtrage de paquets et la NAT. La connexion Internet via une ligne ADSL possède une bande passante descendante de 2MBps et montante de 640Kbps.

La politique de mise en queue pour ce réseau :

- Réserver une bande passante de 80Kbps en téléchargement pour Bob afin qu'il puisse jouer aux jeux en ligne sans être ralenti par les téléchargements d'Alice et de Charlie. Autoriser Bob à utiliser plus de 80Kbps quand ceci est possible.
- Le trafic généré par le SSH interactif et la messagerie instantanée aura une plus haute priorité que le trafic régulier.
- Les requêtes et les réponses DNS auront la seconde plus haute priorité.
- Les paquets TCP ACK sortants auront une priorité plus haute que tout autre trafic sortant.

Le jeu de règles ci-dessous permet cette politique réseau. Notez que seul les directives de `pf.conf` qui concernent directement la politique ci-dessus sont présentes; `nat`, `rdr`, options, etc., ne sont pas montrées.

```
# active la mise en queue sur l'interface externe pour contrôler
# le trafic allant sur Internet. utilise l'algorithme priq pour
# contrôler uniquement les priorités.
# fixe la bande passante à 610Kbps pour avoir les meilleures
# performances à la sortie de la queue TCP ACK.

altq on fxp0 priq bandwidth 610Kb queue { std_out, ssh_im_out, dns_out, \
        tcp_ack_out }

# définit les paramètres pour les queues enfant.
# std_out      - la queue standard. toute règle de filtrage ci-dessous
#                qui ne spécifie pas explicitement une queue aura
#                son trafic ajouté à cette queue.
# ssh_im_out   - trafic généré par le SSH interactif et
#                la messagerie instantanée
#                variés.
# dns_out      - requêtes DNS.
# tcp_ack_out  - paquets TCP ACK sans donnée.

queue std_out     priq(default)
queue ssh_im_out  priority 4 priq(red)
queue dns_out     priority 5
queue tcp_ack_out priority 6

# active la mise en queue sur l'interface interne en vue de contrôler
# le trafic provenant d'Internet. utilise l'algorithme cbq pour
# contrôler la bande passante. la bande passante maximale est de
# 2Mbps.

altq on dc0 cbq bandwidth 2Mb queue { std_in, ssh_im_in, dns_in, bob_in }

# définit les paramètres pour les queues enfant.
# std_in      - la queue standard. toute règle de filtrage ci-dessous
#               qui ne spécifie pas explicitement une queue aura
#               son trafic ajouté à cette queue.
# ssh_im_in   - trafic généré par le SSH interactif et
#               par la messagerie instantanée.
# dns_in      - réponses DNS.
# bob_in      - bande passante réservée à la station
#               de travail de Bob. l'autorise à emprunter.

queue std_in    bandwidth 1.6Mb cbq(default)
queue ssh_im_in bandwidth 200Kb priority 4
queue dns_in    bandwidth 120Kb priority 5
queue bob_in    bandwidth 80Kb cbq(borrow)


# ... dans la section filtrage de pf.conf ...

alice         = "192.168.0.2"
bob           = "192.168.0.3"
charlie       = "192.168.0.4"
local_net     = "192.168.0.0/24"
ssh_ports     = "{ 22 2022 }"
```

```
im_ports      = "{ 1863 5190 5222 }"

# règles de filtrage pour l'entrée en fxp0
block in on fxp0 all

# règles de filtrage pour la sortie en fxp0
block out on fxp0 all
pass  out on fxp0 inet proto tcp from (fxp0) to any flags S/SA \
        keep state queue(std_out, tcp_ack_out)
pass  out on fxp0 inet proto { udp icmp } from (fxp0) to any keep state
pass  out on fxp0 inet proto { tcp udp } from (fxp0) to any port domain \
        keep state queue dns_out
pass  out on fxp0 inet proto tcp from (fxp0) to any port $ssh_ports \
        flags S/SA keep state queue(std_out, ssh_im_out)
pass  out on fxp0 inet proto tcp from (fxp0) to any port $im_ports \
        flags S/SA keep state queue(ssh_im_out, tcp_ack_out)

# règles de filtrage pour l'entrée en dc0
block in on dc0 all
pass  in on dc0 from $local_net

# règles de filtrage pour la sortie en dc0
block out on dc0 all
pass  out on dc0 from any to $local_net
pass  out on dc0 proto { tcp udp } from any port domain to $local_net \
        queue dns_in
pass  out on dc0 proto tcp from any port $ssh_ports to $local_net \
        queue(std_in, ssh_im_in)
pass  out on dc0 proto tcp from any port $im_ports to $local_net \
        queue ssh_im_in
pass  out on dc0 from any to $bob queue bob_in
```

Exemple #2 : Réseau d'entreprise

Dans cet exemple, l' hôte OpenBSD agit en firewall pour le réseau d'une entreprise. L'entreprise utilise un serveur WWW dans la partie DMZ de son réseau où les clients envoient leurs sites via FTP. Le département IT a son propre sous-réseau connecté au réseau principal, et le directeur a un PC sur son bureau utilisé pour les emails et le surf sur Internet. La connexion à Internet se fait via une ligne T1 avec une bande passante de 1.5Mbps dans les deux directions. Tous les autres segments du réseau utilisent Fast Ethernet (100Mbps).

L'administrateur réseau a decidé de la politique suivante :

- Limiter tout le trafic entre le serveur WWW et Internet à 500Kbps in
  - Réserver 250Kbps pour le trafic HTTP.
  - Réserver 250Kbps pour le reste (trafic non-HTTP etc.)
  - Permettre autrement de pouvoir emprunter dans les 500Kbps.
  - Donner au trafic HTTP entre le serveur WWW et Internet une priorité plus élevée que les autres trafics entre le serveur WWW et Internet (comme les envois FTP).
- Le trafic entre le serveur WWW et le érseau interne peut utiliser davantage que les 100Mbps offerts par le réseau.
- Réserver 500Kbps pour le réseau du département IT pour qu'ils puissent télécharger les dernières mises à jour de logiciels de façon rapide. Ils devront pouvoir utiliser plus de 500Kbps quand de la bande passante additionnelle sera disponible.
- Donner une priorité plus élevée au trafic entre le PC du directeur et Internet que tout autre trafic allant ou venant d'Internet.

Le jeu de règles ci-dessous permet cette politique réseau. Notez que seul les directives de `pf.conf` qui concernent directement la politique ci-dessus sont présentes; `nat`, `rdr`, options, etc., ne sont pas montrées.

```
# active la mise en queue sur l'interface externe pour les paquets sortant sur
# Internet. utilise l'algorithme cbq, afin que la bande passante de chaque
# queue puisse être contrôlée. la bande passante sortante
# maximale est de 1.5Mbps.

altq on fxp0 cbq bandwidth 1.5Mb queue { std_ext, www_ext, boss_ext }

# définit les paramètres pour les queues enfant.
# std_ext        - la queue standard. également queue par défaut
#                  pour le trafic sortant sur fxp0.
# www_ext        - queue contenant les queues du serveur WWW. limitée à
#                  500Kbps.
#   www_ext_http - trafic http du serveur WWW; plus haute priorité.
#   www_ext_misc - tout le trafic non-http du serveur WWW.
# boss_ext       - trafic venant de l'ordinateur du patron.

queue std_ext        bandwidth 500Kb cbq(default borrow)
queue www_ext        bandwidth 500Kb { www_ext_http, www_ext_misc }
  queue www_ext_http bandwidth 50% priority 3 cbq(red borrow)
  queue www_ext_misc bandwidth 50% priority 1 cbq(borrow)
queue boss_ext       bandwidth 500Kb priority 3 cbq(borrow)

# active la mise en queue sur l'interface interne pour contrôler le trafic
# venant d'Internet ou de la DMZ. utilise l'algorithme cbq pour contrôler
# la bande passante de chaque queue. la bande passante sur cette interface
# est fixée au maximum. le trafic venant de la DMZ sera en mesure
# d'utiliser la totalité de cette bande passante alors que le trafic
# venant de l'Internet sera limité à 1.0Mbps (car 0.5Mbps
# (500Kbps) sont alloués à fxp1).

altq on dc0 cbq bandwidth 100% queue { net_int, www_int }

# définit les paramètres pour les queues enfant.
# net_int    - queue contenant le trafic provenant de Internet. la bande
#              passante est de 1.0Mbps.
#   std_int  - la queue standard. également queue par défaut pour
#              le trafic sortant en dc0.
#   it_int   - trafic vers le réseau du département informatique;
#                                     réserver 500Kbps.
#   boss_int - trafic vers le PC du patron; assigner la plus haute
#                              priorité.
# www_int    - trafic du serveur WWW de la DMZ; vitesse maximale.

queue net_int    bandwidth 1.0Mb { std_int, it_int, boss_int }
  queue std_int  bandwidth 250Kb cbq(default borrow)
  queue it_int   bandwidth 500Kb cbq(borrow)
  queue boss_int bandwidth 250Kb priority 3 cbq(borrow)
queue www_int    bandwidth 99Mb cbq(red borrow)

# active la mise en queue sur l'interface de la DMZ pour contrôler
# le trafic destiné au serveur WWW. cbq sera utilisé
# sur cette interface tant que le contrôle détaillé de la bande
# passante sera nécessaire. la bande passante sur cette interface
# est fixée au maximum. le trafic provenant du réseau
# interne sera en mesure d'utiliser la totalité de la bande passante
# alors que le trafic provenant d'Internet sera limité à 500Kbps.

altq on fxp1 cbq bandwidth 100% queue { internal_dmz, net_dmz }

# définit les paramètres pour les queues enfant.
# internal_dmz   - trafic provenant du réseau interne.
# net_dmz        - queue contenant le trafic provenant de Internet.
#   net_dmz_http - trafic http; plus haute priorité.
#   net_dmz_misc - tout autre trafic. c'est aussi la queue par défaut.

queue internal_dmz   bandwidth 99Mb cbq(borrow)
queue net_dmz        bandwidth 500Kb { net_dmz_http, net_dmz_misc }
  queue net_dmz_http bandwidth 50% priority 3 cbq(red borrow)
  queue net_dmz_misc bandwidth 50% priority 1 cbq(default borrow)


# ... dans la section filtrage du pf.conf ...

main_net  = "192.168.0.0/24"
it_net    = "192.168.1.0/24"
```

```
int_nets  = "{ 192.168.0.0/24, 192.168.1.0/24 }"
dmz_net   = "10.0.0.0/24"

boss      = "192.168.0.200"
wwwserv   = "10.0.0.100"

# refut par défaut
block on { fxp0, fxp1, dc0 } all

# règles de filtrage pour l'entrée de fxp0
pass in on fxp0 proto tcp from any to $wwwserv port { 21, \
        > 49151 } flags S/SA keep state queue www_ext_misc
pass in on fxp0 proto tcp from any to $wwwserv port 80 \
        flags S/SA keep state queue www_ext_http

# règles de filtrage pour la sortie de fxp0
pass out on fxp0 from $int_nets to any keep state
pass out on fxp0 from $boss to any keep state queue boss_ext

# règles de filtrage pour l'entrée de dc0
pass in on dc0 from $int_nets to any keep state
pass in on dc0 from $it_net to any queue it_int
pass in on dc0 from $boss to any queue boss_int
pass in on dc0 proto tcp from $int_nets to $wwwserv port { 21, 80, \
        > 49151 } flags S/SA keep state queue www_int

# règles de filtrage pour la sortie de dc0
pass out on dc0 from dc0 to $int_nets

# règles de filtrage pour l'entrée de fxp1
pass in on fxp1 proto { tcp, udp } from $wwwserv to any port 53 \
        keep state

# règles de filtrages pour la sortie de fxp1
pass out on fxp1 proto tcp from any to $wwwserv port { 21, \
        > 49151 } flags S/SA keep state queue net_dmz_misc
pass out on fxp1 proto tcp from any to $wwwserv port 80 \
        flags S/SA keep state queue net_dmz_http
pass out on fxp1 proto tcp from $int_nets to $wwwserv port { 80, \
        21, > 49151 } flags S/SA keep state queue internal_dmz
```

[Précédent : Ancres] [Index] [Suivant : Ensembles d'adresses ("Pools") et Partage de Charge]


www@openbsd.org

$OpenBSD: queueing.html,v 1.12 2006/05/02 17:09:33 jufi Exp $

Pools d'Adresses et Partage de Charge

Table des Matières

Introduction

Un pool d'adresses, constitué de deux adresses ou plus, permet à un groupe d'utilisateurs une utilisation partagée de ces adresses. Un tel ensemble peut être utilisé comme adresse de redirection dans les règles `rdr`, comme adresse de traduction dans les règles `nat`, et comme adresse destination dans les options `route-to`, `reply-to`, `dup-to` et filter.

Il existe quatre méthodes pour utiliser un pool d'adresses :

- `bitmask` - supprime la partie réseau d'une adresse IP (adresse utilisée comme adresse source dans des règles `nat` ou adresse destination dans des règles `rdr`) et la remplace avec la partie réseau de l'adresse correspondante du pool d'adresses. Exemple : si l'adresse du pool est 192.0.2.1/24 et l'adresse à modifier est la 10.0.0.50, l'adresse correspondante au niveau du pool est la 192.0.2.50. Si le pool d'adresses est le bloc 192.0.2.1/25 et l'adresse à modifier est la 10.0.0.130, alors l'adresse correspondante au niveau du pool est la 192.0.2.2.
- `random` - sélectionne aléatoirement une adresse du pool.
- `source-hash` - utilise un hash de l'adresse source pour choisir l'adresse du pool à utiliser. Cette méthode permet de guarantir qu'une adresse source donnée sera toujours associée avec la même adresse du pool. La clé avec laquelle est alimentée l'algorithme de hachage peut être spécifiée en option après le mot-clé `source-hash` au format hex ou chaîne de caractères. Par défaut, pfctl(8) va générer une clé aléatoire à chaque fois que la base de règles est chargée.
- `round-robin` - attribue les adresses du pool séquentiellement. C'est la méthode par défaut et c'est aussi la seule méthode autorisée lorsque le pool d'adresses est spécifié en utilisant une table.

Dans tous les cas, à l'exception notable de la méthode `round-robin`, le pool d'adresses doit être spécifié sous forme d'un bloc réseau en notation CIDR (Classless Inter-Domain Routing). La méthode `round-robin` accepte des adresses individuelles à partir d'une liste ou d'une table.

L'option `sticky-address` peut être utilisée conjointement avec les types de pool `random` et `round- robin` afin de s'assure qu'à une adresse source donnée corresponde toujours la même adresse de redirection.

Pools d'Adresses NAT

Un pool d'adresses peut être utilisé comme adresse de traduction dans les règles `nat`. Les adresses source dans chaque paquet de connexion sont traduites avec une adresse du pool selon la méthode choisie. Ceci peut être utile dans des situations où PF doit faire des opérations de NAT pour un réseau très grand. Etant donné que le nombre de connexions traduites par adresse de traduction est limité, l'ajout d'adresses de traduction permettra à la passerelle NAT d'être scalable afin d'assurer le service pour un grand nombre d'utilisateurs.

Dans l'exemple suivant, un pool de deux adresses est utilisé pour traduire les paquets sortants. Pour chaque connexion sortante, PF fera une attribution séquentielle avec rotation selon la méthode round-robin.

```
nat on $ext_if inet from any to any -> { 192.0.2.5, 192.0.2.10 }
```

Cependant, cette méthode a un inconvénient. En effet, des connexions successives à partir de la même adresse interne ne seront pas toujours traduites avec la même adresse de traduction. Ceci peut causer des interférences lors, par exemple, de la navigation sur des sites web qui utilisent les adresses IP comme éléments d'identification des utilisateurs. Une approche alternative consiste à utiliser la méthode `source- hash`. Ainsi, chaque adresse interne est toujours traduite avec la même adresse de traduction. Dans ce cas, le pool d'adresses doit être obligatoirement un bloc réseau en notation CIDR.

```
nat on $ext_if inet from any to any -> 192.0.2.4/31 source-hash
```

Cette règle de `nat` utilise le pool d'adresses 192..2.4/31 (192.0.2.4 - 192.0.2.5) comme adresse de traduction de paquets sortants. Chaque adresse interne sera toujours traduite avec la même adresse de traduction grâce au mot-clé `source-hash`.

Partage de Charge Des Connexions Entrantes

Des pools d'adresses peuvent aussi être utilisés pour partager la charge sur les connexions entrantes. Par exemple, des connexions web entrantes peuvent être partagées entre serveurs web d'une même ferme :

```
web_servers = "{ 10.0.0.10, 10.0.0.11, 10.0.0.13 }"

rdr on $ext_if proto tcp from any to any port 80 -> $web_servers \
    round-robin sticky-address
```

Les connexions successives seront redirigées vers les serveurs web selon la méthode round-robin : les connexions à partir de la même adresse source seront toujours envoyés au même serveur web. Cette connexion "collante" ("sticky") existera aussi longtemps qu'il y aura des états faisant référence à cette connexion. Une fois les états expirés, la connexion collante expirera aussi. Les futures connexions à partir de ce hôte seront redirigés vers le prochain serveur dans le round robin.

Partage de Charge Des Connexions Sortantes

Les pools d'adresses peuvent être utilisés en combinaison de l'option de filtrage `route-to` pour faire du partage de charge entre deux ou plusieurs connexions Internet lorsqu'un protocole de routage multi-lien (tel que BGP4) n'est pas disponible. En utilisant `route-to` avec un pool d'adresses et la méthode `round-robin`, les connexions sortantes peuvent être distribuées entre plusieurs liens.

Pour que le partage de charge fonctionne, il faut aussi connaître l'adresse IP du routeur adjacent sur chaque connexion Internet. Ces informations sont données à l'option `route-to` afin de contrôler la destination des paquets sortants.

L'exemple suivant partage le trafic sortant entre deux connexions Internet :

```
lan_net = "192.168.0.0/24"
int_if  = "dc0"
ext_if1 = "fxp0"
ext_if2 = "fxp1"
ext_gw1 = "68.146.224.1"
ext_gw2 = "142.59.76.1"

pass in on $int_if route-to \
   { ($ext_if1 $ext_gw1), ($ext_if2 $ext_gw2) } round-robin \
   from $lan_net to any keep state
```

L'option `route-to` est utilisée pour le trafic *entrant* sur l'interface *internal* afin de spécifier les interfaces réseau sur lesquelles le trafic sortant doit être partagé ainsi que les passerelles respectives de chaque interface réseau. Il est à noter que l'option `route-to` doit être présente dans *chaque* règle de filtrage pour laquelle le trafic doit être équilibré. Les paquets de retour seront acheminés vers la même interface externe à partir de laquelle les paquets d'origine ont ressortis (c'est ce qui est effectué par les FAIs). Ces paquets de retour seront ensuite acheminés vers l'interface du réseau interne normalement.

Pour s'assurer que les paquets avec une adresse source appartenant à `$ext_if1` sont toujours acheminés vers `$ext_gw1` (et, de même, pour `$ext_if2` et `$ext_gw2`), les deux lignes suivantes doivent être incluses dans le jeu de règles :

```
pass out on $ext_if1 route-to ($ext_if2 $ext_gw2) from $ext_if2 \
   to any
pass out on $ext_if2 route-to ($ext_if1 $ext_gw1) from $ext_if1 \
   to any
```

Enfin, la NAT peut aussi être utilisée sur chaque interface de sortie :

```
nat on $ext_if1 from $lan_net to any -> ($ext_if1)
nat on $ext_if2 from $lan_net to any -> ($ext_if2)
```

Voici un exemple complet d'équilibrage de charge du trafic sortant :

```
lan_net = "192.168.0.0/24"
int_if  = "dc0"
ext_if1 = "fxp0"
ext_if2 = "fxp1"
ext_gw1 = "68.146.224.1"
ext_gw2 = "142.59.76.1"

#  nat outgoing connections on each internet interface
nat on $ext_if1 from $lan_net to any -> ($ext_if1)
nat on $ext_if2 from $lan_net to any -> ($ext_if2)

#  default deny
block in  from any to any
block out from any to any

#  pass all outgoing packets on internal interface
pass out on $int_if from any to $lan_net
#  pass in quick any packets destined for the gateway itself
pass in quick on $int_if from $lan_net to $int_if
#  load balance outgoing tcp traffic from internal network.
pass in on $int_if route-to \
    { ($ext_if1 $ext_gw1), ($ext_if2 $ext_gw2) } round-robin \
    proto tcp from $lan_net to any flags S/SA modulate state
#  load balance outgoing udp and icmp traffic from internal network
pass in on $int_if route-to \
    { ($ext_if1 $ext_gw1), ($ext_if2 $ext_gw2) } round-robin \
    proto { udp, icmp } from $lan_net to any keep state

#  general "pass out" rules for external interfaces
pass out on $ext_if1 proto tcp from any to any flags S/SA modulate state
pass out on $ext_if1 proto { udp, icmp } from any to any keep state
pass out on $ext_if2 proto tcp from any to any flags S/SA modulate state
pass out on $ext_if2 proto { udp, icmp } from any to any keep state

#  route packets from any IPs on $ext_if1 to $ext_gw1 and the same for
#  $ext_if2 and $ext_gw2
pass out on $ext_if1 route-to ($ext_if2 $ext_gw2) from $ext_if2 to any
pass out on $ext_if2 route-to ($ext_if1 $ext_gw1) from $ext_if1 to any
```

[Précédent : Mise en Queue des Paquets et Gestion des Priorités] [Index] [Suivant : Balisage de Paquets]


www@openbsd.org

$OpenBSD: pools.html,v 1.13 2006/05/02 17:09:33 jufi Exp $

[Précédent : Ensembles d'Adresses ("Pools") et Partage de Charge] [Index] [Suivant : Journal des Evénements]

PF : Balisage des Paquets (Filtrage par Politique)

Table des Matières

Introduction

Le balisage de paquets est une méthode pour marquer les paquets avec un identifiant interne qui peut être utilisé comme critère dans les règles de filtrage et de traduction d'adresses. Grâce au balisage, il est possible de créer des paquets dits "de confiance" entre des interfaces et de déterminer si des paquets ont été traités par les règles de traduction d'adresses. Il est aussi possible de faire du filtrage suivant une politique au lieu de faire du filtrage par règle.

Affectation de Balises aux Paquets

Pour ajouter une balise à un paquet, utilisez le mot-clé `tag` :

```
pass in on $int_if all tag INTERNAL_NET keep state
```

La balise `INTERNAL_NET` sera ajoutée à tout paquet qui correspondra à la règle précitée.

Une balise peut aussi être affectée grâce à une macro. Par exemple :

```
name = "INTERNAL_NET"
pass in on $int_if all tag $name keep state
```

On peut utiliser un ensemble de macros prédéfinies.

- `$if` - L'interface
- `$srcaddr` - L'adresse IP source
- `$dstaddr` - L'adresse IP destination
- `$srcport` - Le port source
- `$dstport` - Le port destination
- `$proto` - Le protocole
- `$nr` - Le nombre de la règle

Ces macros sont interprétées lors du chargement des règles, PAS en "runtime".

L'affectation de balises observe les règles suivantes :

- Les balises sont adhérentes ("sticky"). Une fois une balise est appliquée à un paquet par une règle correspondante, elle n'est jamais supprimée. Cependant, elle peut être remplacée par une balise différente.
- A cause de cette forte adhérence d'une balise, un paquet peut avoir une balise même si la dernière règle correspondante n'utilise pas le mot-clé `tag`.
- Un paquet ne peut avoir qu'une seule balise à un moment donné.

- Les balises sont des identifiants *internal*. Elles ne sont pas envoyées sur le réseau.

Prenons le jeu de règles suivant comme exemple :

```
(1) pass in on $int_if tag INT_NET keep state
(2) pass in quick on $int_if proto tcp to port 80 tag \
        INT_NET_HTTP keep state
(3) pass in quick on $int_if from 192.168.1.5 keep state
```

- Les paquets arrivant sur l'interface `$int_if` se verront attribuer une balise `INT_NET` par la règle #1.
- Les paquets TCP arrivant sur l'interface `$int_if` et destinés au port 80 se verront d'abord attribuer une balise `INT_NET` par la règle #1. Cette balise sera ensuite remplacée par la balise `INT_NET_HTTP` à cause de la règle #2.
- Les paquets arrivant sur l'interface `$int_if` et en provenance de 192.168.1.5 seront autorisés par la règle #3 car c'est la dernière règle qui correspond au filtrage de ces paquets. Cependant, ces paquets se verront attribuer la balise `INT_NET_HTTP` s'ils sont destinés au port TCP 80; sinon ils se verront attribuer la balise `INT_NET`.

De même que pour les règles de filtrage, les balises peuvent être appliquées par des règles `nat`, `rdr`, et `binat` en utilisant le mot-clé `tag`.

Vérification des Balises Appliquées

Pour vérifier les balises précédemment appliquées, utilisez le mot-clé `tagged` comme dans l'exemple suivant :

```
pass out on $ext_if tagged INT_NET keep state
```

Les paquets sortant à partir de `$ext_if`doivent être balisés avec la balise `INT_NET` pour que la règle ci-dessus corresponde à ces paquets. La correspondance inverse peut aussi être réalisée avec l'opérateur `!` :

```
pass out on $ext_if ! tagged WIFI_NET keep state
```

Les règles de réécriture (`nat`/`rdr`/`binat`) peuvent aussi utiliser le mot clé `tagged` pour correspondre aux paquets.

Filtrage par Politique

Le filtrage par politique utilise une approche différente pour l'écriture d'un jeu de règles. Une politique est définie par rapport aux types de trafic : règles pour les types de trafic à passer, règles pour les types de trafic à bloquer. Les paquets sont ensuite classifiés au sein de la politique selon les critères traditionnels : adresse IP source/destination, protocole, etc. Examinez la politique de filtrage qui suit :

- Le trafic provenant du LAN interne et à destination d'Internet est permis (LAN_INET) et doit être réécrit (LAN_INET_NAT)
- Le trafic provenant du LAN interne et à destination de la DMZ est autorisé (LAN_DMZ)
- Le trafic provenant d'Internet et à destination de serveurs dans la DMZ est autorisé (INET_DMZ)
- Le trafic provenant d'Internet et redirigé vers spamd(8) est autorisé (SPAMD)
- Tout autre trafic est bloqué

Notez que la politique couvre *tout* le trafic qui transite par le pare-feu. Le mot entre parenthèses indique le nom de la balise qui sera utilisée pour chaque élément de la politique.

Des règles de filtrage et de traduction doivent à présent être écrites pour classifier les paquets au sein de la politique.

```
rdr on $ext_if proto tcp from <spamd> to port smtp \
   tag SPAMD -> 127.0.0.1 port 8025
nat on $ext_if tag LAN_INET_NAT tagged LAN_INET -> ($ext_if)

block all
pass in on $int_if from $int_net tag LAN_INET keep state
pass in on $int_if from $int_net to $dmz_net tag LAN_DMZ keep state
pass in on $ext_if proto tcp to $www_server port 80 tag INET_DMZ keep state
```

Maintenant les règles qui constituent la politique sont définies.

```
pass in  quick on $ext_if tagged SPAMD keep state
pass out quick on $ext_if tagged LAN_INET_NAT keep state
```

```
pass out quick on $dmz_if tagged LAN_DMZ keep state
pass out quick on $dmz_if tagged INET_DMZ keep state
```

Maintenant que le jeu de règles a été paramétré, les modifications futures sont à apporter uniquement dans les règles de classification. Par exemple, si un serveur POP3/SMTP est ajouté à la DMZ, il sera nécessaire d'ajouter des règles de classification pour le trafic POP3 et SMTP comme le montre l'exemple suivant :

```
mail_server = "192.168.0.10"
...
pass in on $ext_if proto tcp to $mail_server port { smtp, pop3 } \
   tag INET_DMZ keep state
```

Le trafic mail sera autorisé car il fait partie de la classification INET_DMZ.

Voici le jeu de règles complet :

```
# macros
int_if  = "dc0"
dmz_if  = "dc1"
ext_if  = "ep0"
int_net = "10.0.0.0/24"
dmz_net = "192.168.0.0/24"
www_server = "192.168.0.5"
mail_server = "192.168.0.10"

table <spamd> persist file "/etc/spammers"

# classification -- classifier les paquets selon la politique
# définie
rdr on $ext_if proto tcp from <spamd> to port smtp \
    tag SPAMD -> 127.0.0.1 port 8025
nat on $ext_if tag LAN_INET_NAT tagged LAN_INET -> ($ext_if)

block all
pass in on $int_if from $int_net tag LAN_INET keep state
pass in on $int_if from $int_net to $dmz_net tag LAN_DMZ keep state
pass in on $ext_if proto tcp to $www_server port 80 tag INET_DMZ keep state
pass in on $ext_if proto tcp to $mail_server port { smtp, pop3 } \
    tag INET_DMZ keep state

# filtrage -- autoriser/bloquer suivant la politique.
pass in  quick on $ext_if tagged SPAMD keep state
pass out quick on $ext_if tagged LAN_INET_NAT keep state
pass out quick on $dmz_if tagged LAN_DMZ keep state
pass out quick on $dmz_if tagged INET_DMZ keep state
```

Balisage des Trames Ethernet

Le balisage peut être effectué au niveau Ethernet si la machine de balisage/filtrage est aussi un pont (bridge(4)). En créant des règles de filtrage pour bridge(4) qui utilisent le mot-clé `tag`, PF peut filtrer les paquets d'après leur adresse MAC source ou destination. Les règles pour bridge(4) sont créés avec la commande brconfig(8). Exemple :

```
# brconfig bridge0 rule pass in on fxp0 src 0:de:ad:be:ef:0 \
   tag USER1
```

Puis dans `pf.conf` :

```
pass in on fxp0 tagged USER1
```

[Précédent : Ensembles d'Adresses ("Pools") et Partage de Charge] [Index] [Suivant : Journal des Evénements]


www@openbsd.org

$OpenBSD: tagging.html,v 1.20 2006/10/29 10:58:53 jufi Exp $

[Précédent : Balisage de Paquets] [Index] [Suivant : Performances]

PF : Journal des Evénements

Table des Matières

Introduction

L'enregistrement des Evénements dans PF est effectué par pflogd(8) qui écoute sur l'interface `pflog0` et écrit les paquets dans un fichier d'événements (normalement /var/log/pflog) au format binaire tcpdump(8). Les flux correspondant aux règles de filtrage qui contiennent les mots-clés `log` ou `log (all)` sont enregistrés de cette manière.

Traçage De Paquets

Afin de tracer les paquets passant à travers PF, le mot-clef `log` doit être utilisé dans les règles de NAT/rdr et de filtrage. Il est à noter que PF ne peut tracer que les paquets qu'il bloque ou qu'il laisse passer; vous ne pouvez pas créer de règle qui ne fait que du traçage de paquets.

Le mot-clef `log` indique à PF de tracer les paquets correspondant à la règle dans laquelle ce mot-clef figure. Dans le cas où la règle crée un état, seul le premier paquet "vu" (celui qui entraîne la création de l'état) sera tracé.

Les options qui peuvent être fournies au mot-clef `log` sont :

`all`
: Tous les paquets seront tracés et pas uniquement le paquet initial. Utile pour les règles à état.

`user`
: Trace aussi les identifiants UNIX utilisateur et groupe qui possèdent la socket à partir de laquelle le paquet est émis ou vers laquelle il est destiné (socket local bien entendu), en plus des informations standard.

Les options sont fournies entre parenthèses après le mot-clef `log`; plusieurs options peuvent être séparées par une virgule ou un espace.

```
pass in log (all) on $ext_if inet proto tcp to $ext_if port 22 keep state
```

Trace tous les paquets entrants à destination du port 22.

Lire un Fichier d'Evénements

pflogd écrit le fichier d'événements dans un format binaire compatible tcpdump(8). Il ne peut donc n'être analysé que par tcpdump(8) ou tout autre outil acceptant des fichiers dans ce format.

Pour lire le fichier d'événements :

```
# tcpdump -n -e -ttt -r /var/log/pflog
```

La lecture du fichier *pflog* à l'aide de tcpdump(8) ne donne pas un affichage en temps réel. Cependant, il est possible d'obtenir un tel affichage en utilisant l'interface `pflog0` :

```
# tcpdump -n -e -ttt -i pflog0
```

REMARQUE : Lorsque vous examinez les événements, une attention particulière doit être accordée au décodage verbeux des protocoles effectué par tcpdump (décodage activé par l'option `-v`). Les décodeurs protocolaires de tcpdump ne possèdent pas un historique sécurité parfait. En théorie du moins, une attaque reste possible en utilisant les charges utiles partielles enregistrées par le périphérique d'enregistrement d'événements. Il est recommandé de déplacer les fichiers d'événements du pare-feu vers une autre machine avant de les traiter à l'aide de tcpdump(8).

Des précautions supplémentaires doivent êtres prises pour sécuriser l'accès aux événements. Par défaut, pflogd enregistrera 96 octets du paquet dans le fichier d'événements. L'accès aux fichiers d'événements pourrait fournir un accès partiel à des charges utiles de paquets sensibles (tels que les noms d'utilisateurs et les mots de passe telnet(1) ou ftp(1)).

Filtrer les Evénements en Sortie

Etant donné que pflogd enregistre les événements au format binaire tcpdump, la totalité des fonctionnalités tcpdump peut être utilisée pour analyser les événements. Par exemple, pour voir uniquement les paquets qui correspondent à un certain port :

```
# tcpdump -n -e -ttt -r /var/log/pflog port 80
```

Ceci peut être encore affiné en limitant l'affichage des paquets à une certaine combinaison de hôte et de port :

```
# tcpdump -n -e -ttt -r /var/log/pflog port 80 and host 192.168.1.3
```

La même idée peut être appliquée quand on lit les événements directement à partir de l'interface `pflog0` :

```
# tcpdump -n -e -ttt -i pflog0 host 192.168.4.2
```

Il est à noter que ces manipulations n'ont aucun impact sur les paquets enregistrés dans le fichier d'événements pflogd; les commandes ci-dessus ne feront qu'afficher les paquets au fur et à mesure qu'ils sont enregistrés.

En plus de l'utilisation des règles de filtrage standard tcpdump(8), le langage de filtrage du tcpdump a été étendu pour lire la sortie de pflogd :

- `ip` - la famille d'adresses est IPv4.
- `ip6` - la famille d'adresses est IPv6.
- `on` *`int`* - le paquet est passé à travers l'interface *int*.
- `ifname` *`int`* - idem `on` *`int`*.
- `ruleset` *`name`* - le jeu de règles/ancre auquel correspond le paquet.
- `rulenum` *`num`* - la règle de filtrage à laquelle correspondait le paquet était la règle numéro *num*.
- `action` *`act`* - l'action prise pour le paquet. Les actions possibles sont `pass` et `block`.
- `reason` *`res`* - la raison pour laquelle l'`action` a été prise. Les raisons possibles sont `match`, `bad-offset`, `fragment`, `short`, `normalize`, `memory` et `bad-timestamp`, `congestion`, `ip-option`, `proto-cksum`, `state-mismatch`, `state-insert`, `state-limit`, `src-limit`, and `synproxy`.
- `inbound` - le paquet était entrant.
- `outbound` - le paquet était sortant.

Exemple :

```
# tcpdump -n -e -ttt -i pflog0 inbound and action block and on wi0
```

Cette commande affichera l'événement en temps réel créé par des paquets entrants bloqués sur l'interface wi0.

Enregistrement des Evénements à Travers Syslog

Dans plusieurs situations, il est souhaitable de disposer des événements enregistrés par le pare-feu sous format ASCII ou de les envoyer à un serveur d'événements central distant. Tout ceci peut être effectué par un petit script shell, quelques modifications mineures aux fichiers de configuration OpenBSD, et syslogd(8), le service qui permet d'enregistrer les événements. Syslogd enregistre les événements au format ASCII et il est aussi capable de les envoyer à un serveur d'événements distant.

Créez le script suivant :

`/etc/pflogrotate`

```
#!/bin/sh
PFLOG=/var/log/pflog
FILE=/var/log/pflog5min.$(date "+%Y%m%d%H%M")
kill -ALRM $(cat /var/run/pflogd.pid)
if [ -r $PFLOG ] && [ $(stat -f %z $PFLOG) -gt 24 ]; then
   mv $PFLOG $FILE
   kill -HUP $(cat /var/run/pflogd.pid)
   tcpdump -n -e -ttt -r $FILE | logger -t pf -p local0.info
   rm $FILE
fi
```

Editez la liste des tâches cron du super-utilisateur root :

```
# crontab -u root -e
```

Ajoutez-y les deux lignes suivantes :

```
# effectuer une rotation du fichier d'événements
# toutes les 5 minutes 0-59/5 * * * * /bin/sh /etc/pflogrotate
```

Ajoutez la ligne suivante au fichier `/etc/syslog.conf` :

```
local0.info     /var/log/pflog.txt
```

Si vous voulez les envoyer aussi à un serveur d'événements distant, ajoutez la ligne suivante

```
local0.info     @syslogger
```

Assurez vous que le hôte *syslogger* a bien été défini dans le fichier hosts(5).

Créez le fichier `/var/log/pflog.txt` afin de permettre à syslog d'enregistrer les événements dans ce fichier et donnez lui les mêmes permissions qu'au fichier pflog.

```
# touch /var/log/pflog.txt
# chmod 600 /var/log/pflog.txt
```

Relancez syslogd pour que les modifications soient prises en compte :

```
# kill -HUP $(cat /var/run/syslog.pid)
```

Tous les événements enregistrés sont maintenant envoyés vers `/var/log/pflog.txt`. Si la seconde ligne a été ajoutée, les événements sont aussi envoyés au serveur d'événements distant *syslogger*.

Le script `/etc/pflogrotate` traite désormais `/var/log/pflog` et le supprime à la fin du traitement. La rotation du fichier `pflog` par newsyslog(8) n'est plus nécessaire désormais et devrait être désactivée. Cependant `/var/log/pflog.txt` remplace `/var/log/pflog` et sa rotation devrait être activée. Modifiez `/etc/newsyslog.conf` comme suit :

```
#/var/log/pflog        600    3    250    *    ZB /var/run/pflogd.pid
/var/log/pflog.txt     600    7    *      24
```

PF, à l'aide du mécanisme précité, générera des événements qui seront enregistrés au format ASCII dans le fichier `/var/log/pflog.txt`. Si `/etc/syslog.conf` est configuré en conséquence, ces événements pourront aussi être envoyés à un serveur de logs distant. L'enregistrement des événements n'est pas immédiate. Elle prend jusqu'à 5-6 minutes (l'intervalle d'exécution de la tâche cron) avant que les événements enregistrés apparaissent dans le fichier.

[Précédent : Balisage de Paquets] [Index] [Suivant : Performances]


www@openbsd.org

$OpenBSD: logging.html,v 1.25 2006/10/29 10:58:52 jufi Exp $

[Précédent : Journal des Evénements] [Index] [Suivant : Gestion du Protocole FTP]

PF : Performances

"Combien de bande passante PF peut-il gérer ?"
"Quelle puissance de machine est suffisante pour gérer mon accès Internet ?"

Il n'y a pas de réponse facile à ces questions. Pour certaines applications, un 486/66 avec une paire de bonnes cartes réseau ISA pourrait filtrer et faire de la NAT à quelques 5Mbps, mais d'autres applications requièrent une machine beaucoup plus puissante avec des cartes réseau PCI plus efficaces. La vraie question n'est pas le nombre de bits par seconde mais plutôt le nombre de paquets par seconde et la complexité des jeux de règles.

Les performances de PF sont déterminées par plusieurs variables :

- Nombre de paquets par seconde. Pratiquement le même traitement doit être effectué sur un paquet avec une charge utile de 1400 octets que sur un paquet avec une charge utile d'un octet. Le nombre de paquets par seconde détermine le nombre de fois que la table d'états et, au cas où il n'y a pas de correspondance, les règles de filtrage doivent être évaluées chaque seconde, déterminant ainsi à quel point on sollicite le système.
- Performance du bus système. Le bus ISA a une bande passante maximale de 8MB/sec, et lorsque le processeur y accède, il doit diminuer sa vitesse à la vitesse effective d'un 80286 cadencé à 8MHz, peu importe la vitesse réelle du processeur. Le bus PCI a une bande passante effective beaucoup plus grande et a moins d'impact sur le processeur.
- Efficacité de la carte réseau. Certaines cartes réseau sont plus efficaces que d'autres. Les cartes basées sur le Realtek 8139 (rl(4)) ont tendance a avoir de faibles performances alors que les cartes basées sur l'Intel 21143 (dc(4)) ont tendance à avoir d'excellentes performances. Pour des performances maximales, utilisez des cartes Ethernet gigabit même si vous ne vous connectez pas à des réseaux gigabit. Ces cartes ont une gestion de la mémoire tampon beaucoup plus avancée que les autres cartes.
- Complexité et conception de votre jeu de règles. Plus le jeu de règles est complexe, plus lent le pare-feu sera. Plus vous utiliserez des règles `keep state` et `quick` pour filtrer les paquets, meilleures les performances seront. Plus le nombre de lignes devant être évaluées pour chaque paquet sera grand, plus les performances seront faibles.
- Mentionnons tout de même deux éléments : le CPU et la RAM. Vu que PF est un processus noyau, il n'utilisera pas d'espace de pagination (swap). Donc si vous avez suffisamment de RAM, il fonctionne, sinon il se met en mode panique à cause de l'épuisement du pool(9). Des quantités énormes de RAM ne sont guère nécessaires. 32MB devraient permettre de contenir 30 000 états ce qui un nombre énorme pour des applications SoHo. La plupart des utilisateurs trouveront qu'une machine "recyclée" est plus que suffisante pour un système PF. Un système cadendé à 300MHz pourra gérer un très grand nombre de paquets rapidement, dans la mesure où il a des bonnes cartes réseau et un bon jeu de règles.

Les utilisateurs demandent souvent des benchmarks PF. Le seul benchmark qui compte vraiment est la performance de *votre* système dans *votre* environnement. Un benchmark qui ne réplique pas votre environnement ne vous aidera pas à planifier correctement votre système pare-feu. La meilleure action possible est de tester PF vous-même dans les mêmes conditions réseau ou du moins dans les conditions les plus proches possibles que votre pare-feu actuel sur le même matériel.

PF est utilisé dans certaines applications très conséquentes avec un trafic important, et les développeurs sont des "power users" de PF. Il y a donc beaucoup de chance pour que PF fonctionne dans votre environnement avec des performances correctes.

[Précédent : Journal des Evénements] [Index] [Suivant : Gestion du Protocole FTP]


www@openbsd.org

$OpenBSD: perf.html,v 1.17 2006/05/02 17:09:33 jufi Exp $

[Précédent : Performances] [Index] [Suivant : Authpf : Shell Utilisateur pour les Passerelles d'Authentification]

PF : Gestion du Protocole FTP

Table des Matières

Modes FTP

FTP est un protocole qui date d'une époque où Internet était un petit ensemble de machines avec des relations de confiance mutuelle et où tout le monde connaissait tout le monde. A cette époque le besoin de filtrer ou d'assurer une sécurité stricte était inexistant. FTP n'était pas conçu pour filtrer, pour faire transiter du trafic à travers les pare- feux, ou pour fonctionner avec la NAT.

FTP peut fonctionner dans deux modes : passif et actif. De manière générale, le choix du mode actif ou passif consiste à déterminer l'extrêmité de communication FTP qui a un problème avec le filtrage pare- feu. Dans le meilleur des mondes, il faudrait supporter les deux modes pour rendre vos utilisateurs heureux.

Dans le mode FTP actif, quand un utilisateur se connecte à un serveur FTP distant et demande une information ou un fichier, le serveur FTP effectue une connexion vers le client pour transférer les données demandées. Cette connexion est appelée la *connexion de données*. Pour commencer, le client FTP choisit un port aléatoire qui lui servira pour recevoir les données. Le client envoie le numéro de port qu'il a choisi au serveur FTP puis se met à l'écoute des connexions entrantes à destination de ce port. Le serveur FTP initie ensuite une connexion vers l'adresse du client sur le port choisi et transfère les données. Ce mode de fonctionnement est problématique pour les utilisateurs essayant d'accéder à des serveurs FTP lorsqu'ils sont derrière une passerelle NAT. Vu la manière dont fonctionne la NAT, le serveur FTP initie la connexion de données en se connectant à l'adresse externe de la passerelle NAT sur le port choisi. La passerelle NAT n'a pas de correspondance pour le paquet reçu dans sa table d'état. Le paquet sera ignoré et ne sera pas délivré au client.

Dans le mode FTP passif (le mode par défaut utilisé par le client ftp(1) d'OpenBSD), le client demande au serveur de choisir un port aléatoire sur lequel ce dernier se mettra à l'écoute en attente de la connexion de données. Le serveur communique au client le port qu'il a choisi pour le transfert des données. Malheureusement, ce n'est pas toujours possible ou souhaitable à cause de la possibilité d'existence d'un pare-feu devant le serveur FTP qui bloquerait les connexions de données en entrée. Le client ftp(1) d'OpenBSD utilise le mode passif par défaut; pour forcer le mode FTP actif, utilisez le drapeau -A du client ftp(1). Il est aussi possible de désactiver le mode passif (et donc d'activer par cette action le mode actif) en utilisant la commande "`passive off`" à l'invite de commandes "`ftp>`".

Client FTP derrière un pare-feu

Comme précédemment indiqué, FTP ne fonctionne pas très bien à travers des mécanismes de NAT et des pare-feux.

Packet Filter fournit une solution à ce cas en redirigeant le trafic FTP à travers un serveur mandataire FTP (proxy). Ce processus agit de telle façon à "guider" votre trafic FTP à travers la passerelle NAT/pare-feu en activant des règles pour PF puis en les désactivant lorsque le transfert est terminé à l'aide du système des ancres PF. Le proxy FTP utilisé par OpenBSD 3.9 et PF est ftp-proxy(8). (remarque : les versions précédentes d'OpenBSD utilisaient un proxy différent du même nom qui est documenté dans la page de manuel OpenBSD 3.8 ftp-proxy(8)v3.8).

Pour l'activer, ajustez la ligne suivante selon vos besoins et ajoutez la à la section NAT de `pf.conf` :

```
nat-anchor "ftp-proxy/*"
rdr-anchor "ftp-proxy/*"
rdr on $int_if proto tcp from any to any port 21 -> 127.0.0.1 \
   port 8021
```

Les deux premières lignes sont les ancres utilisées par ftp-proxy afin d'ajouter les règles à chaud afin de gérer votre trafic FTP. La dernière ligne redirige FTP en provenance de vos clients vers le program ftp-proxy(8) qui écoute sur votre machine au port 8021.

Vous avez également besoin d'une dans vos règles :

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

Il est évident que le serveur mandataire doit être démarré et exécuté sur la machine OpenBSD. Ceci peut être effectué en insérant la ligne suivante dans `/etc/rc.conf.local` :

```
ftpproxy_flags=""
```

Le program ftp-proxy peut être démarré en tant que root afin d'avoir à éviter de redémarrer.

ftp-proxy écoute sur le port 8021, le même port auquel la ligne `rdr` précédente renvoie le trafic FTP.

Pour activer les connexions actives, vous devez utiliser l'option "-r" de ftp-proxy(8) (avec l'ancien proxy, vous utilisiez l'option "-u root").

Veuillez noter que ftp-proxy(8) est utilisé pour aider les **clients FTP** derrière un pare-feu PF ; il n'est pas utilisé pour prendre en charge un **serveur FTP** derrière un tel pare-feu.

Serveur FTP protégé par un pare-feu PF local

Dans ce cas, PF est activé sur le serveur FTP lui-même plutôt que sur un pare-feu dédié. Pour les besoins d'une connexion FTP passive, FTP utilisera un haut port TCP choisi de manière aléatoire pour les données entrantes. Par défaut, le serveur FTP natif d'OpenBSD ftpd(8) utilise la plage 49152 à 65535. Ces ports doivent être autorisés en entrée ainsi que le port 21 (le port de contrôle FTP) :

```
pass in on $ext_if proto tcp from any to any port 21 keep state
pass in on $ext_if proto tcp from any to any port > 49151 \
   keep state
```

Si vous le souhaitez, vous pouvez restreindre cette plage considérablement. Dans le cas du programme ftpd(8) d'OpenBSD, ceci peut être effectué en utilisant les variables sysctl(8) `net.inet.ip.porthifirst` et `net.inet.ip.porthilast`.

Serveur FTP protégé par un pare-feu PF externe avec NAT

Dans ce cas, le pare-feu doit rediriger tout le trafic vers le serveur FTP. De plus, il ne doit bloquer aucun des ports requis. Pour fournir un exemple concret, on supposera encore une fois que le serveur FTP est le serveur standard sous OpenBSD : ftpd(8), utilisant la plage de ports par défaut.

Voici un exemple de règles qui pourraient être utilisées dans ce cas de figure :

```
ftp_server = "10.0.3.21"

rdr on $ext_if proto tcp from any to any port 21 -> $ftp_server \
   port 21
rdr on $ext_if proto tcp from any to any port 49152:65535 -> \
   $ftp_server port 49152:65535

# in on $ext_if
pass in quick on $ext_if proto tcp from any to $ftp_server \
   port 21 keep state
pass in quick on $ext_if proto tcp from any to $ftp_server \
   port > 49151 keep state

# out on $int_if
pass out quick on $int_if proto tcp from any to $ftp_server \
   port 21 keep state
pass out quick on $int_if proto tcp from any to $ftp_server \
   port > 49151 keep state
```

Plus d'Informations sur FTP

Vous pouvez obtenir plus d'informations concernant le filtrage et le fonctionnement de FTP de manière générale dans le livre blanc suivant :

- FTP Reviewed

[Précédent : Performances] [Index] [Suivant : Authpf : Shell Utilisateur pour les Passerelles d'Authentification]


www@openbsd.org

$OpenBSD: ftp.html,v 1.17 2006/05/12 06:32:29 jufi Exp $

[Précédent : Problèmes avec FTP] [Index] [Suivant : Haute-disponibilité des pare-feux avec CARP et pfsync]

PF : Authpf : Shell Utilisateur pour les Passerelles d'Authentification

---

Table des Matières

Introduction

Authpf(8) est un shell utilisateur pour les passerelles d'authentification. Une passerelle d'authentification est comme n'importe quelle passerelle réseau normale (un routeur) excepté que les utilisateurs doivent d'abord s'authentifier sur la passerelle avant que celle-ci ne permette au trafic généré par les utilisateurs de la traverser. Quand le shell d'un utilisateur est `/usr/sbin/authpf` (au lieu que son shell soit ksh(1), csh(1), etc) et que l'utilisateur se connecte en utilisant SSH, authpf fera les modifications nécessaires au jeu de règles pf(4) actif de telle manière à permettre au trafic généré par l'utilisateur de passer à travers les filtres et/ou d'être traduit en utilisant la Traduction d'Adresses IP ("NAT") ou d'être redirigé. Une fois que l'utilisateur se déconnecte ou que sa session est terminée, authpf supprimera toutes les règles chargées pour l'utilisateur et enlevera toutes les connexions avec maintien d'état que l'utilisateur aura ouvertes. Ainsi, le trafic de l'utilisateur ne passera à travers la passerelle que si l'utilisateur garde sa session SSH ouverte.

Authpf charge les règles de filtres et de NAT dans un point d'ancrage unique. L'ancre est nommée par la combinaison de l'identifiant UNIX de l'utilisateur et de l'id du processus authpf dans le format "`identifiant(PID)`". Chaque ancre utilisateur est stockée dans l'ancre `authpf` qui est elle même ancrée dans le jeu de règles principal. Le "chemin d'ancrage complet" devient dès lors :

```
jeu_de_regles_principal/authpf/identifiant(PID)
```

Les règles qu'authpf charge peuvent être configurées par utilisateur ou globalement.

Voici quelques exemples d'utilisation d'authpf :

- Exiger que les utilisateurs s'authentifient avant de permettre un accès à Internet.
- Permettre à certains utilisateurs -- tels que les administrateurs -- l'accès à certaines parties restreintes du réseau.
- Permettre uniquement aux utilisateurs connus l'accès au reste du réseau ou à Internet depuis un segment de réseau sans fil.
- Permettre à des travailleurs d'accéder à des ressources du système d'information de l'entreprise depuis chez eux, lorsqu'ils sont sur la route... Les utilisateurs en dehors des bureaux peuvent avoir accès au réseau d'entreprise et aussi être redirigés vers des ressources spécifiques (leur propre station de travail par exemple) selon le nom d'utilisateur qu'ils utilisent pour s'authentifier.
- Dans une configuration telle que celle d'une bibliothèque ou un autre lieu équipé de terminaux avec un accès Internet public, PF peut être configuré pour permettre un accès limité à Internet aux utilisateurs de passage. Authpf peut alors être utilisé pour permettre aux utilisateurs connus un accès complet.

Authpf enregistre le nom d'utilisateur et l'adresse IP de chaque utilisateur qui réussit à s'authentifier ainsi que le début et la fin de leur session. L'enregistrement se

fait via syslogd(8). En utilisant cette information, un administrateur peut déterminer qui s'est connecté et responsabiliser les utilisateurs par rapport à leur trafic réseau.

Configuration

La section suivante fournit les étapes de base nécessaires pour configurer authpf. Pour une description complète de la configuration d'authpf, veuillez consulter la page du manuel authpf.

Activer Authpf

Authpf ne se lancera pas si le fichier de configuration `/etc/authpf/authpf.conf` n'est pas présent. Même si le fichier est vide (de taille nulle), il doit être présent ou Authpf se fermera automatiquement après qu'un utilisateur s'authentifie correctement.

Les directives de configuration suivantes peuvent êtres placées dans `authpf.conf` :

- `anchor=`*`nom`* - Utiliser l'ancre nom spécifiée au lieu de "authpf".
- `table=`*`nom`* - Utiliser la table table nom spécifiée au lieu de "authpf_users".

Attacher Authpf au Jeu de Règles Principal

Authpf peut être rattaché au jeu de règles principal en utilisant des règles d'ancrage. Ces règles utilisent le mot-clé `anchor` :

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

PF "sortira" du jeu de règles principal pour évaluer les règles authpf à l'endroit où les règles `anchor` sont placées dans le jeu de règles. Il n'est pas nécessaire que les quatre règles `anchor` soient présentes. Par exemple, si authpf ne doit charger aucune règle de traduction `nat`, la règle `nat-anchor` peut être omise.

Configurer les Règles Chargées

Authpf charge les règles à partir d'un des deux fichiers suivants :

- `/etc/authpf/users/$USER/authpf.rules`
- `/etc/authpf/authpf.rules`

Le premier fichier contient des règles qui seront uniquement chargées lorsque l'utilisateur `$USER` (cette variable étant à remplacer par le nom d'utilisateur) se connecte. La configuration spécifique par utilisateur est utilisée lorsqu'un utilisateur donné -- un administrateur par exemple -- nécessite un jeu de règles différent du jeu de règles authpf par défaut. Le deuxième fichier contient les règles par défaut qui sont chargées pour tout utilisateur qui ne possède pas son propre fichier `authpf.rules`. Si le fichier spécifique à l'utilisateur existe, il est chargé au lieu du fichier par défaut. Au moins un des fichiers doit exister. Autrement authpf ne fonctionnera pas.

Les règles de filtrage et de traduction ont la même syntaxe que pour n'importe quel autre jeu de règles PF avec une seule exception : Authpf permet l'utilisation de deux macros prédéfinies :

- `$user_ip` - l'adresse IP de l'utilisateur connecté
- `$user_id` - le nom d'utilisateur de l'utilisateur connecté

Il est recommandé d'utiliser la macro `$user_ip` pour ne permettre que le trafic provenant de la machine de l'utilisateur authentifié.

En plus de la macro `$user_ip`, authpf utilisera la table `authpf_users` (si elle existe) pour stocker les adresses IP de tous les utilisateurs authentifiés. Soyez sûrs de définir la table avant de vouloir l'utiliser :

```
table <authpf_users> persist
pass in on $ext_if proto tcp from <authpf_users> \
    to port smtp flags S/SA keep state
```

Cette table ne devrait être utilisée que pour les règles devant s'appliquer à tous les utilisateurs authentifiés.

Les Listes de Contrôle d'Accès

On peut empêcher les utilisateurs d'utiliser authpf en créant un fichier sous le répertoire `/etc/authpf/banned/`. Le fichier doit avoir comme nom le nom de l'utilisateur banni. Le contenu du fichier sera affiché à l'utilisateur avant qu'authpf ne le déconnecte. Cette fonctionnalité fournit une méthode pratique pour notifier un utilisateur ou lui dire pourquoi son accès n'est pas autorisé et la personne qu'il doit contacter pour la restauration de son accès.

Autrement, il est aussi possible d'autoriser uniquement l'accès à des utilisateurs spécifiques en mettant leur nom d'utilisateur dans le fichier `/etc/authpf/authpf.allow`. Si le fichier `/etc/authpf/authpf.allow` n'existe pas ou contient "`*`", authpf permettra l'accès à n'importe quel utilisateur qui a réussi à se connecter via SSH et qui n'est pas explicitement banni.

Si authpf n'est pas capable de déterminer s'il doit autoriser ou interdire l'accès à un utilisateur, il affichera un message bref et déconnectera l'utilisateur. Une entrée dans le fichier `/etc/authpf/banned/` est toujours prise en compte avant une entrée dans le fichier `/etc/authpf/authpf.allow`.

Présenter un Message de Login

Chaque fois qu'un utilisateur s'authentifie correctement à authpf, un message indiquant que l'utilisateur est correctement authentifié est affiché.

```
Hello charlie. You are authenticated from host "64.59.56.140"
```

Ce message peut être suivi par un message personnalisé dans `/etc/authpf/authpf.message`. Le contenu de ce fichier sera affiché après le message d'accueil par défaut.

Attribuer Authpf comme Shell Utilisateur

Afin qu'authpf fonctionne, il doit être attribué à l'utilisateur en tant que shell de connexion ("login shell"). Lorsque l'utilisateur s'authentifie selon les mécanismes offerts par sshd(8), authpf sera executé comme shell de l'utilisateur. Authpf vérifiera ensuite si l'utilisateur est autorisé à utiliser authpf, chargera les règles à partir du fichier approprié, etc.

Il existe plusieurs méthodes pour attribuer authpf à un utilisateur comme shell de connexion :

1. Manuellement pour chaque utilisateur en utilisant chsh(1), vipw(8), useradd(8), usermod(8), etc.
2. En affectant les utilisateurs à une classe de connexion et en changeant l'option `shell` dans `/etc/login.conf`.

Créer une classe d'authentification authpf

Lors de l'utilisation d'authpf sur un système qui dispose de comptes normaux et de comptes utilisateurs authpf, il peut être intéressant de créer une classe d'authentification authpf séparée pour les utilisateurs authpf. Cela autorise certains changements sur ces comptes à être faits de manière globale ainsi que de placer des politiques différentes sur les comptes normaux et les comptes authpf. Quelques exemples de ces polices peuvent être :

- **shell** - Spécifier un shell utilisateur de connexion. Cela peut être utilisé pour forcer un shell utilisateur à `authpf` quelle que soit l'entrée dans la base passwd(5).
- **welcome** - Spécifie quel fichier de motd(5) utiliser lorsqu'un utilisateur se connecte. Cela est très utile pour afficher des messages ne concernant que les utilisateurs authpf.

Les classes d'authentification sont créées dans le fichier login.conf(5). Un exemple de classe d'authentification pour les utilisateurs authpf :

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

Les utilisateurs sont assignés à une classe d'authentification par l'édition du champ `class` de la base d'utilisateurs passwd(5). Une méthode pour le faire est l'utilisation de la commande chsh(1).

Voir Les Personnes Connectées

Une fois l'utilisateur connecté et les règles chargées par authpf, ce dernier changera son nom de processus pour indiquer le nom d'utilisateur et l'adresse IP de

l'utilisateur connecté :

```
# ps -ax | grep authpf
23664 p0  Is+     0:00.11 -authpf: charlie@192.168.1.3 (authpf)
```

Dans l'exemple ci-dessus, `charlie` est connecté depuis la machine 192.168.1.3. En envoyant un signal SIGTERM au processus authpf, on peut déconnecter l'utilisateur. Authpf supprimera aussi toute règle chargée pour l'utilisateur et supprimera toutes les connexions à état que l'utilisateur aura ouvert.

```
# kill -TERM 23664
```

Exemple

Dans cet exemple, authpf est utilisé sur une passerelle OpenBSD pour authentifier des utilisateurs sur un réseau sans fil faisant partie d'un réseau de campus plus grand. Une fois les utilisateurs authentifiés, et en supposant qu'ils ne font pas partie de la liste des utilisateurs bannis, ils seront autorisés à établir des sessions SSH vers l'extérieur et à naviguer sur le web (y compris sur les sites web sécurisés). De plus, ils pourront accéder à un des serveurs DNS du campus.

Le fichier `/etc/authpf/authpf.rules` contient les règles suivantes :

```
wifi_if = "wi0"

pass in quick on $wifi_if proto tcp from $user_ip to port { ssh, http, \
   https } flags S/SA keep state
```

L'administrateur `charlie` devra être capable d'accéder aux serveurs SMTP et POP3 du campus en plus des droits d'accès précités. Les règles suivantes font partie du fichier `/etc/authpf/users/charlie/authpf.rules` :

```
wifi_if = "wi0"
smtp_server = "10.0.1.50"
pop3_server = "10.0.1.51"

pass in quick on $wifi_if proto tcp from $user_ip to $smtp_server \
   port smtp flags S/SA keep state
pass in quick on $wifi_if proto tcp from $user_ip to $pop3_server \
   port pop3 flags S/SA keep state
pass in quick on $wifi_if proto tcp from $user_ip to port { ssh, http, \
   https } flags S/SA keep state
```

Voici le contenu du jeu de règles principal figurant dans le fichier `/etc/pf.conf` :

```
# macros
wifi_if = "wi0"
ext_if  = "fxp0"
dns_servers = "{ 10.0.1.56, 10.0.2.56 }"

table <authpf_users> persist

scrub in all

# filtrage
block drop all

pass out quick on $ext_if inet proto tcp from \
   { $wifi_if:network, $ext_if } flags S/SA modulate state
pass out quick on $ext_if inet proto { udp, icmp } from \
   { $wifi_if:network, $ext_if } keep state

pass in quick on $wifi_if inet proto tcp from $wifi_if:network to $wifi_if \
   port ssh flags S/SA keep state

pass in quick on $wifi_if inet proto { tcp, udp } from <authpf_users> \
```

```
   to $dns_servers port domain keep state

anchor "authpf/*" in on $wifi_if
```

Le jeu de règles est très simple :

- Bloquer tout par défaut.
- Autoriser les trafics TCP, UDP et ICMP sortants sur l'interface externe en provenance du réseau sans fil et depuis la passerelle elle même.
- Autoriser le trafic SSH entrant en provenance du réseau sans fil et destiné à la passerelle. Cette règle est nécessaire pour permettre aux utilisateurs de s'authentifier.
- Autoriser les requêtes DNS entrantes provenant de tous les utilisateurs authpf authentifiés et destinées aux serveurs DNS du campus.
- Créer le point d'ancrage "authpf" pour le trafic entrant sur l'interface sans fil.

Le jeu de règles par défaut est utilisé pour bloquer tout trafic non strictement nécessaire pour le fonctionnement de l'architecture. Le trafic sortant de l'interface externe est autorisé. Cependant le trafic en entrée de l'interface sans fil est interdit. Une fois l'utilisateur authentifié, son trafic est autorisé à traverser l'interface sans fil et à accéder au reste du réseau. Le mot-clé `quick` est utilisé un peu partout afin que PF n'évalue pas tous les jeux de règles nommés quand une nouvelle connexion traverse la passerelle.

[Précédent : Problèmes avec FTP] [Index] [Suivant : Haute-disponibilité des pare-feux avec CARP et pfsync]


www@openbsd.org

$OpenBSD: authpf.html,v 1.19 2006/05/02 17:09:33 jufi Exp $

[Précédent : Authpf : Shell Utilisateur pour les Passerelles d'Authentifications] [Index] [Suivant : Pare-feu pour réseau domestique ou petite société]

PF: Haute-disponibilité des pare-feux avec CARP et pfsync

Table des Matières

Introduction à CARP

CARP veut dire "Common Address Redundancy Protocol". L'objectif premier de ce protocole est de permettre à un groupe d'hôtes sur un même segment réseau de partager une adresse IP. CARP est une alternative sécurisée et libre aux protocoles Virtual Router Redundancy Protocol (VRRP)et Hot Standby Router Protocol (HSRP).

On appelle un groupe d'hôtes utilisant CARP un "groupe de redondance". Le groupe de redondance se voit attribuer une adresse IP partagée entre les membres du groupe. Au sein de ce groupe, un hôte est désigné comme "maître". Les autres membres sont appellés "esclaves". L'hôte maître est celui qui "prend" l'adresse IP partagée. Il répond à tout trafic ou requête ARP à l'attention de cette adresse. Chaque hôte peut appartenir à plusieurs groupes de redondance.

Une utilisation commune de CARP est la création d'un groupe de pare-feux redondants. L'adresse IP virtuelle attribuée au groupe de redondance est désignée comme l'adresse du routeur par défaut sur les machines clientes. Dans le cas où le pare-feu maître rencontre une panne ou est déconnecté du réseau, l'adresse IP virtuelle sera prise par un des pare-feux esclaves et le service continuera à être rendu sans interruption.

CARP supporte IPv4 et IPv6.

Fonctionnement de CARP

L'hôte maître du groupe envoit des annonces régulières sur le réseau local afin que les hôtes esclaves puissent savoir qu'il est toujours disponible. Si les hôtes esclaves ne recoivent plus d'annonce de la part du maître durant une période de temps configurable, alors l'un d'eux devient le nouveau maître. Ce dernier est celui dont les valeurs configurées pour `advbase` et `advskew` sont les plus faibles.

Il est possible de faire co-exister plusieurs groupes CARP sur un même segment réseau. Les annonces de CARP contiennent un identifiant appelé "Virtual Host ID" qui permet à des membres d'un même groupe d'identifier le groupe de redondance auquel une annonce donnée appartient.

Afin d'empêcher un utilisateur malicieux sur le segment réseau d'usurper les annonces CARP, chaque groupe peut être doté d'un mot de passe. Ainsi chaque paquet CARP envoyé au groupe est protégé par SHA1 MAC.

Etant donné que CARP est un protocole à part entière, il doit être explicitement autorisé au niveau des règles de filtrage :

```
pass out on $carp_dev proto carp keep state
```

`$carp_dev` est l'interface physique que CARP utilise pour la communication.

Configurer CARP

Chaque groupe de redondance est représenté par une interface réseau virtuelle de type carp(4). Ainsi, CARP est configuré à l'aide de la commande ifconfig(8).

```
ifconfig carpN create

ifconfig carpN vhid vhid [pass password] [carpdev carpdev] \
   [advbase advbase] [advskew advskew] [state state] ipaddress \
   netmask mask
```

carpN
: Le nom de l'interface virtuelle carp(4). *N* est un entier qui représente le numéro de l'interface (par exemple carp10).

vhid
: L'identifiant Virtual Host ID. C'est un numéro unique utilisé pour identifier le groupe de redondance sur le réseau. Les valeurs acceptables sont de 1 à 255.

password
: Le mot de passe d'authentification à utiliser lors de la communication avec d'autres hôtes CARP dans le même groupe de redondance. Ce mot de passe doit être partagé entre tous les membres du groupe.

carpdev
: Ce paramètre optionnel spécifie l'interface réseau physique qui appartient à ce groupe de redondance. Par défaut, CARP essaiera de déterminer l'interface à utiliser en cherchant une interface physique qui est dans le même sous-réseau que la combinaison *adresse IP*/*masque* de l'interface carp(4).

advbase
: Ce paramètre optionnel spécifie le nombre de secondes qui s'écoule entre chaque annonce CARP. La valeur par défaut est 1 seconde. Les valeurs acceptables sont de 1 à 255.

advskew
: Ce paramètre optionnel spécifie le biais à introduire au niveau de *advbase* lors de l'envoi d'annonces CARP. En manipulant *advbase*, l'hôte maître CARP peut être choisi. Plus grand est ce nombre, *moindres* sont les chances pour que l'hôte soit retenu lorsqu'un maître est choisi. La valeur par défaut est 0. Les valeurs acceptables sont de 1 à 255.

state
: Permet de forcer l'état de l'interface carp(4). Les états valides sont `init`, `backup`, et `master`.

ipaddress
: Adresse IP partagée entre les membres du groupe de redondance. Cette adresse n'a pas besoin d'être dans le même sous-réseau que l'adresse IP de l'interface physique (si une adresse est configurée sur cette interface). En revanche, elle doit être la même sur tous les membres du groupe.

mask
: Masque de sous-réseau de l'adresse IP partagée.

Vous pouvez contrôler d'avantages de paramètres de CARP à l'aide de sysctl(8).

`net.inet.carp.allow`
: Permet d'accepter ou de refuser les paquets CARP entrants. La valeur par défaut est 1 (accepter).

`net.inet.carp.preempt`
: Permet aux hôtes au sein d'un même groupe de redondance d'avoir de meilleurs `advbase` et `advskew` pour élire le maître. De plus, cette option permet de basculer toutes les interfaces dans le cas où une interface est en panne. Si une des interfaces physiques utilisée pour CARP est indisponible, CARP fixera l'`advskew` de toutes les autres interfaces à 240. Ce qui revient à se déclarer indisponible et forcer un basculement. Cette option est à 0 (désactivation) par défaut.

`net.inet.carp.log`
: Journalise les mauvais paquets CARP. La valeur par défaut est 0 (désactivé).

`net.inet.carp.arpbalance`
: Permet de partager la charge entre plusieurs membres d'un même groupe de redondance. Cette option est à 0 (désactivation) par défaut. Veuillez consulter carp(4) pour plus d'informations.

Exemple de configuration CARP

Voici un exemple de configuration CARP :

```
# sysctl -w net.inet.carp.allow=1
# ifconfig carp1 create
# ifconfig carp1 vhid 1 pass mekmitasdigoat carpdev em0 \
    advskew 100 10.0.0.1 netmask 255.255.255.0
```

Les commandes ci-dessous permettent de faire les opérations suivantes :

- Active la réception des paquets CARP (c'est le comportement par défaut)
- Créé une interface carp(4), `carp1`.
- Configure `carp1` pour l'hôte virtuel #1, met en place un mot de passe, utilise `em0` comme interface physique du groupe, et fais de cette machine un esclave étant donné la valeur de `advskew`, fixée à `100` (en supposant bien entedu que le maître a un `advskew` de moins de 100). L'adresse IP partagée affectée à ce groupe est 10.0.0.1/255.255.255.0.

L'utilisation de `ifconfig` permet de voir l'état de l'interface `carp1`.

```
# ifconfig carp1
carp1: flags=8802<UP,BROADCAST,SIMPLEX,MULTICAST> mtu 1500
     carp: BACKUP carpdev em0 vhid 1 advbase 1 advskew 100
     inet 10.0.0.1 netmask 0xffffff00 broadcast 10.0.0.255
```

Introduction à pfsync

L'interface réseau pfsync(4) expose certains changements effectués au niveau de la table d'état de pf(4). En surveillant cette interface à l'aide de tcpdump(8), les changements de la table d'état peuvent être observés en temps réel. De plus, l'interface pfsync(4) peut envoyer ces messages relatifs aux changements affectant la table d'état sur le réseau afin que d'autres pare-feux PF puissent fusionner ces changements à leurs propres tables d'état. De même, pfsync(4) peut aussi être en écoute des messages entrants.

Fonctionnement de pfsync

Par défaut, pfsync(4) n'envoit pas et ne reçoit pas les mises à jour de la table d'état à travers le réseau. Cependant, les mises à jour peuvent être toujours surveillées en utilisant tcpdump(8) ou d'autres outils similaires en local sur la machine.

Lorsque pfsync(4) est configuré pour envoyer et recevoir des mises à jour à travers le réseau, le comportement par défaut consiste à utiliser le multicast sur le réseau local. Toutes les mises à jour sont envoyées sans authentification. Il faudrait donc utiliser une des deux méthodes suivantes pour sécuriser les échanges :

1. Connectez les deux pare-feux qui devront échanger les mises à jour à l'aide d'un câble croisé. Les interfaces physiques ainsi connectées seront utilisées comme `syncdev` (voir plus bas).
2. Utilisez l'option `syncpeer` de la commande ifconfig(8) (voir plus bas) de telle façon à utiliser des échanges unicast pour envoyer les mises à jour au pair, puis configurez ipsec(4) entre les deux hôtes pour sécuriser le trafic pfsync(4).

Lorsque les mises à jour sont envoyées et reçues à travers le réseau, les paquets pfsync doivent être autorisés par les règles de filtrage :

```
pass on $sync_if proto pfsync
```

`$sync_if` est l'interface physique utilisée pour la communication pfsync(4).

Configurer pfsync

pfsync(4) étant une interface réseau virtuelle, elle est configurée à l'aide de ifconfig(8).

```
ifconfig pfsyncN syncdev syncdev [syncpeer syncpeer]
```

`pfsyncN`
: Le nom de l'interface pfsync(4). Si vous utilisez le noyau `GENERIC`, l'interface `pfsync0` existe par défaut.

`syncdev`
: Le nom de l'interface physique utilisée pour envoyer les mises à jour pfsync.

`syncpeer`
: Ce paramètre optionnel spécifie l'adresse IP d'un hôte avec qui les échanges de mises à jour pfsync auront lieu. Par défaut les mises à jour pfsync sont multicast sur le réseau local. Cette option change ce comportement et utilise des échanges unicast avec le *`syncpeer`* spécifié.

Exemple de configuration pfsync

Voici un exemple de configuration pfsync.

```
# ifconfig pfsync0 syncdev em1
```

Cette commande active pfsync sur l'interface `em1`. Les mises à jour seront envoyées en multicast sur le réseau. Ainsi tout hôte utilisant pfsync pourra les recevoir.

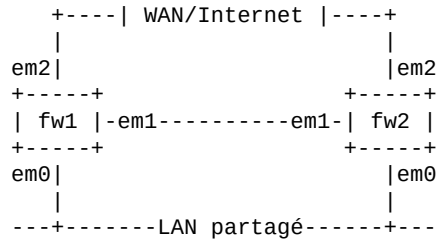
Utilisation combinée de CARP et pfsync pour assurer la haute- disponibilité et la redondance

En combinant les fonctionnalités de CARP et de pfsync, un groupe de deux pare-feux ou plus peut être utilisé pour créer une grappe de pare-feux hautement disponible et entièrement redondante.

CARP:
: Gère le basculement automatique d'un pare-feu à un autre.

pfsync:
: Synchronise la table d'état entre les pare-feux. Dans le cas d'un basculement, le trafic n'est pas interrompu lorsque le nouveau pare-feu maître prend la main.

Voici un scénario typique avec deux pare-feux : `fw1` et `fw2`.



Les interfaces `em1` des pare-feux sont connectées à l'aide d'un câble croisé. Les interfaces `em0` sont connectées au LAN et les interfaces `em2` sont connectées à un réseau WAN ou à Internet. Les adresses IP utilisées sont les suivantes :

- fw1 em0 : 172.16.0.1
- fw1 em1 : 10.10.10.1
- fw1 em2 : 192.0.2.1
- fw2 em0 : 172.16.0.2
- fw2 em1 : 10.10.10.2
- fw2 em2 : 192.0.2.2
- IP partagée sur le LAN : 172.16.0.100
- IP partagée sur WAN/Internet : 192.0.2.100

La politique réseau veut que le pare-feu `fw1` soit maître.

Configurons d'abord fw1 :

```
! activation de la préemption et le basculement de toutes les interfaces
# sysctl -w net.inet.carp.preempt=1

! configuration de pfsync
# ifconfig em1 10.10.10.1 netmask 255.255.255.0
# ifconfig pfsync0 syncdev em1
# ifconfig pfsync0 up

! configuration de CARP côté LAN
# ifconfig carp1 create
# ifconfig carp1 vhid 1 carpdev em0 pass lanpasswd \
    172.16.0.100 netmask 255.255.255.0

! configuration de CARP côté WAN/Internet
# ifconfig carp2 create
# ifconfig carp2 vhid 2 carpdev em2 pass netpasswd \
   192.0.2.100 netmask 255.255.255.0
```

Configurons maintenant fw2 :

```
! activation de la préemption et le basculement de toutes les interfaces
# sysctl -w net.inet.carp.preempt=1

! configuration de pfsync
# ifconfig em1 10.10.10.2 netmask 255.255.255.0
# ifconfig pfsync0 syncdev em1
# ifconfig pfsync0 up

! configuration de CARP côté LAN
# ifconfig carp1 create
# ifconfig carp1 vhid 1 carpdev em0 pass lanpasswd \
     advskew 128 172.16.0.100 netmask 255.255.255.0

! configuration de CARP côté WAN/Internet
# ifconfig carp2 create
# ifconfig carp2 vhid 2 carpdev em2 pass netpasswd \
    advskew 128 192.0.2.100 netmask 255.255.255.0
```

Problèmes opérationnels

Voici quelques problèmes opérationnels communément rencontrés avec CARP/pfsync.

Configuration de CARP et pfsync au démarrage

Etant donné que carp(4) et pfsync(4) sont des interfaces réseau, elles peuvent être configurées au démarrage en créant des fichiers hostname.if(5). Le script de démarrage netstart prendra soin de la création et de la configuration des interfaces.

Exemples :

/etc/hostname.carp1
```
inet 172.16.0.100 255.255.255.0 172.16.0.255 vhid 1 carpdev em0 \
   pass lanpasswd
```

/etc/hostname.pfsync0
```
up syncdev em1
```

Comment forcer le basculement vers ou depuis le noeud maître

De temps à autre, il peut y avoir besoin de forcer le basculement du maître intentionnellement. Il peut s'agir par exemple de la nécessité d'arrêter le maître pour des besoins de maintenance ou de diagnostic suite à des problèmes de fonctionnements. L'objectif est de faire basculer le trafic vers un des hôtes esclaves afin que les utilisateurs ne soient pas affectés par ces opérations.

Pour forcer le basculement, il vous suffit d'arrêter l'interface carp(4) sur le maître. Ceci aura pour effet d'amener le maître à annoncer des valeurs `advbase` et `advskew` "infinies". Les autres membres du groupe de redondance le remarqueront immédiatement et l'un d'eux prendra le rôle de maître.

```
# ifconfig carp1 down
```

Astuces pour la création de règles

Filtrer l'interface physique. Pour PF, le trafic réseau arrive à partir de l'interface physique, pas de l'interface virtuelle CARP (i.e., `carp0`). N'oubliez pas que, sous PF, le nom d'une interface dans une règle correspond au nom de l'interface physique ou à l'adresse réseau qui lui est associée. Tenez-en compte lors de l'écriture de vos règles. Par exemple, la règle suivante pourrait être correcte :

```
pass in on fxp0 inet proto tcp from any to carp0 port 22
```

mais le remplacement de `fxp0` par `carp0` ne permettrait pas de la faire fonctionner comme on le souhaite.

N'oubliez PAS d'autoriser `proto carp` et `proto pfsync`!

Références supplémentaires

Pour plus d'informations, veuillez consulter les sources suivantes :

- carp(4)
- pfsync(4)
- ifconfig(8)
- hostname.if(5)
- pf.conf(5)
- ifstated(8)
- ifstated.conf(5)

[Précédent : Authpf : Shell Utilisateur pour les Passerelles d'Authentifications] [Index] [Suivant : pare-feu pour réseau domestique ou petite société]


www@openbsd.org
$OpenBSD: carp.html,v 1.9 2006/10/29 10:58:52 jufi Exp $

[Précédent : Haute-disponibilité des pare-feux avec CARP et pfsync] [Index]

PF : Exemple : Pare-feu pour réseau domestique ou petite société

Table des Matières

Le Scenario

Dans cet exemple, PF fonctionne sur une machine OpenBSD jouant le rôle de pare-feu et de passerelle NAT pour un réseau SoHo. L'objectif global est de fournir un accès Internet au réseau local, de fournir un accès limité au pare-feu depuis Internet et d'exposer un serveur web interne à l'Internet. Ce document a pour but de vous montrer comment on construit un jeu de règles correspondant à l'objectif précité.

Le Réseau

Le réseau est conçu de la manière suivante :

```
[ COMP1 ]    [ COMP3 ]
    |            |
 ---+------+-----+------- xl0 [ OpenBSD ] fxp0 -------- ( Internet )
           |
       [ COMP2 ]
```

Il y a un certain nombre de machines sur le réseau interne. Le diagramme en montre trois mais le vrai nombre n'est pas une donnée utile. Ces machines sont des stations de travail normales servant à surfer sur le web, écrire des messages électroniques, participer à des forums de discussion en ligne, etc à l'exception de COMP3 qui sert aussi de serveur Web. Le réseau interne utilise le bloc de réseau 192.168.0.0 / 255.255.255.0.

Le pare-feu OpenBSD est une machine dotée d'un Celeron 300 et de deux cartes réseau : une 3Com 3c905B (`xl0`) et d'une Intel EtherExpress Pro/100 (`fxp0`). Le pare-feu a une connexion câble vers Internet et utilise la NAT pour partager cette connexion avec le réseau interne. L'adresse IP de l'interface externe est attribuée dynamiquement par le Fournisseur d'Accès Internet.

Les Objectifs

Les objectifs sont les suivants :

- Fournir un accès Internet sans restriction pour chaque machine du réseau interne.
- Utiliser un jeu de règles bloquant tout flux par défaut.
- Autoriser les flux entrants suivants sur le pare-feu à partir d'Internet :
  - SSH (port TCP 22) : utilisé pour la maintenance externe du pare-feu.
  - Auth/Ident (port TCP 113): utilisé par quelques services tels que SMTP et IRC.
  - Messages ICMP "Echo Request" : Le type de paquet ICMP utilisé par ping(8).
- Rediriger les tentatives de connexion sur le port TCP 80 (qui sont des tentatives d'accès à un serveur Web) vers la machine COMP3. Et autoriser aussi le trafic vers le port TCP 80 destiné à COMP3 à travers le pare-feu.
- Enregistrer des statistiques de filtrage pour l'interface externe.
- Par défaut, renvoyer un RST TCP ou un message ICMP "Unreachable" pour les paquets bloqués.
- S'assurer que le jeu de règles est aussi simple et facile à maintenir que possible.

Préparation

Ce document suppose que le hôte OpenBSD a été correctement configuré pour fonctionner comme routeur : configuration réseau, connexion Internet et positionnement à "`1`" des variables `net.inet.ip.forwarding` et `net.inet6.ip6.forwarding` de sysctl(3).

Le Jeu de Règles

Les sections ci-après détaillent la manière dont le jeu de règles répondra aux objectifs précités.

Macros

Les macros suivantes sont définies pour rendre la maintenance et la lecture du jeu de règles plus faciles :

```
ext_if="fxp0"
int_if="xl0"

tcp_services = "{ 22, 113 }"
icmp_types = "echoreq"

comp3 = "192.168.0.3"
```

Les deux premières lignes définissent les interfaces réseau sur lesquelles le filtrage sera effectué. En les définissant ici, si nous devons déplacer ce système vers une autre machine avec un matériel différent, nous pourrons alors ne changer que ces deux lignes et le reste des règles restera valable. Les troisièmes et quatrièmes lignes listent les numéros de port TCP des services ouverts depuis Internet (SSH et ident/auth) et les types de paquets ICMP qui sont autorisés à parvenir jusqu'aù pare-feu. Enfin, la dernière ligne définit l'adresse IP de COMP3.

Remarque : Si la connexion Internet nécessite l'utilisation de PPPoE, le filtrage et la NAT s'effectueront sur l'interface `tun0` *au lieu de* `fxp0`.

Options

Les deux options suivantes spécifient la réponse par défaut fournie par les règles de filtrage `block` et activent la collecte de statistiques sur l'interface externe :

```
set block-policy return
set loginterface $ext_if
```

Chaque système Unix possède une interface de "loopback". C'est une interface réseau virtuelle utilisée par les applications pour conversée avec les autres au sein du même système. Sur OpenBSD, l'interface de bouclage ("loopback") est lo(4). Le filtrage est communément désactivé sur ces interfaces. L'utilisation de set skip permet d'accomplir cela.

```
set skip on lo
```

Veuillez prendre note que nous n'effectuons pas de filtrage sur le groupe d'interfaces `lo` entier, de cette manière nous pourrions rajouter des interfaces de loopback supplémentaire sans avoir à nous soucier de mettre à jour cette partie de la configuration.

Scrub

Il n'y a aucune raison pour ne pas utiliser la normalisation de paquets recommandée pour tous les paquets entrants. Il suffit d'utiliser la ligne suivante :

```
scrub in
```

Traduction d'Adresses Réseau

Pour effectuer la NAT du réseau interne, la règle de `nat` suivante est utilisée :

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

Dans ce cas, "`!($ext_if)`" pourrait être aisément remplacé par "`$int_if`" mais si vous ajoutez plusieurs interfaces internes vous devrez ajouter de nouvelles règles NAT, alors qu'avec cette structure, NAT sera activé sur toutes les interfaces protégées.

Vu que l'adresse IP de l'interface externe est attribuée dynamiquement, des parenthèses sont utilisés autour de l'interface de traduction afin que PF tienne compte automatiquement des changements d'adresse IP sur cette interface.

Puisque nous souhaitons que le proxy FTP fonctionne, nous ajouterons également l'ancre NAT :

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

Redirection

La première redirection est pour ftp-proxy(8) afin de permettre aux clients FTP sur le réseau interne de se connecter à des serveurs FTP sur Internet.

```
rdr-anchor "ftp-proxy/*"
rdr on $int_if proto tcp from any to any port 21 -> 127.0.0.1 port 8021
```

Il est à noter que cette règle ne fonctionnera que pour les connexions FTP au port 21. Si les utilisateurs se connectent de manière régulière à des serveurs FTP sur d'autres ports, une liste devra être utilisée pour spécifier le port de destination, par exemple : `from any to any port { 21, 2121 }`.

La dernière règle de redirection est utilisée pour toutes les tentatives de connexion sur le port TCP 80 du pare-feu en provenance d'Internet. Les tentatives légitimes d'accès à ce port seront générées par des utilisateurs qui essaient d'accèder au serveur web du réseau. Ces tentatives de connexion doivent être redirigées vers COMP3 :

```
rdr on $ext_if proto tcp from any to any port 80 -> $comp3
```

Règles de Filtrage

Détaillons maintenant les règles de filtrage. Commencez par l'interdiction par défaut de tout trafic :

```
block in
```

A ce point, tout le trafic entrant sera bloqué même celui en provenance du réseau interne. Des règles suivantes vont ouvrir un certain nombre de flux sur le pare-feu afin de répondre aux objectifs précités et d'ouvrir toutes les interfaces virtuelles nécessaires.

Gardez à l'esprit que pf peut bloquer le trafic entrant ou sortant d'une interface? Cela peut vous simplifier la tâche de ne filtrer que dans une direction plutôt que d'essayer de rester consistant en filtrant le trafic sortant et entrant. Dans notre cas, nous choisirons de filtrer le trafic entrant uniquement, mais lorsque celui-ci sera permit sur une interface nous ne l'empêcherons pas d'en sortir. Pour se faire, nous utiliserons :

```
pass out keep state
```

Nous avons besoin d'une ancre pour ftp-proxy(8) :

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

Il est utile d'utiliser la protection contre le spoofing d'adresses :

```
antispoof quick for { lo $int_if }
```

Maintenant, il faut ouvrir les ports utilisés par les services réseau disponibles depuis Internet. Tout d'abord, le trafic destiné au pare-feu lui-même :

```
pass in on $ext_if inet proto tcp from any to ($ext_if) \
   port $tcp_services flags S/SA keep state
```

La spécification des ports réseau par le biais de la macro `$tcp_services` rend l'ouverture de nouveaux services pour des connexions provenant d'Internet plus facile dans la mesure où il suffira de modifier la macro et recharger le jeu de règles. Des services UDP peuvent aussi être mis à disposition en créant la macro `$udp_services` et en ajoutant une règle de filtrage adéquate similaire à la règle de filtrage ci-dessus en spécifiant `proto udp`.

En plus de la règle `rdr` qui fait suivre tout le trafic web vers COMP3, nous DEVONS aussi autoriser ce trafic à travers le pare-feu :

```
pass in on $ext_if inet proto tcp from any to $comp3 port 80 \
    flags S/SA synproxy state
```

Pour un niveau de sécurité plus élevé, nous utilisons TCP SYN Proxy pour améliorer la protection du web serveur.

Le trafic ICMP doit être permis :

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

Comme pour la macro `$tcp_services`, la macro `$icmp_types` peut facilement être modifiée pour changer les types des paquets ICMP qui doivent être autorisés à atteindre le pare- feu. Notez que cette règle s'applique à toutes les interfaces réseau.

Maintenant, le trafic en provenance du réseau interne doit être autorisé. Nous supposerons que les utilisateurs du réseau interne savent ce qu'ils font et ne provoqueront pas de problème sur Internet. Ce n'est pas nécessairement une bonne supposition pour de nombreux environnements.

```
pass in quick on $int_if
```

Le trafic TCP, UDP, et ICMP à destination d'Internet est autorisé sortir du pare-feu grâce à la règle précédente "`pass out keep state`". L'information sur l'état des connexions est sauvegardée pour permettre aux paquets de retour de passer à leur tour la barrière que constitue le pare-feu.

Le Jeu de Règles Complet

```
# macros
ext_if="fxp0"
int_if="xl0"

tcp_services = "{ 22, 113 }"
icmp_types = "echoreq"

comp3 = "192.168.0.3"

# options
set block-policy return
set loginterface $ext_if

set skip on lo

# scrub
scrub in

# nat/rdr
nat on $ext_if from !($ext_if) -> ($ext_if:0)
nat-anchor "ftp-proxy/*"
rdr-anchor "ftp-proxy/*"
rdr pass on $int_if proto tcp to port ftp -> 127.0.0.1 port 8021
rdr on $ext_if proto tcp from any to any port 80 -> $comp3

# règles de filtrage
block in

pass out keep state

anchor "ftp-proxy/*"
antispoof quick for { lo $int_if }
```

```
pass in on $ext_if inet proto tcp from any to ($ext_if) \
   port $tcp_services flags S/SA keep state

pass in on $ext_if inet proto tcp from any to $comp3 port 80 \
   flags S/SA synproxy state

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

pass quick on $int_if
```

[Précédent : Haute-disponibilité des pare-feux avec CARP et pfsync] [Index]


www@openbsd.org

$OpenBSD: example1.html,v 1.21 2006/10/29 10:58:52 jufi Exp $