Découverte de Packet Filter

Découverte de Packet Filter

Présentation animée le mardi 13 octobre 2009 par Maxime DERCHE, dans le cadre des Mardis du Libre de l'association Chtinux.

Plan de la présentation

I. Introduction

II. Le fichier pf.conf

III. Contre-mesures anti-DoS

Découverte de Packet Filter

I. Introduction

- Généralités
- Histoire
- Fonctionnalités
- Différences par rapport à d'autres produits
- Ce dont on ne parlera pas dans la suite de cette présentation :-)

Découverte de Packet Filter

II. Le fichier pf.conf

- Macros
- Tables
- Options
- Normalisation du trafic (scrub)
- Traduction d'adresse (NAT)
- Règles de filtrage

Découverte de Packet Filter

I. Introduction

- Généralités
- Histoire
- Fonctionnalités
- Différences par rapport à d'autres produits
- Ce dont on ne parlera pas dans la suite de cette présentation :-)

Généralités et historique

- En 2001, Darren Reed décida de modifier la licence de son IPFilter, en interdisant la modification du code.
- Il fut donc retiré d'OpenBSD, et Daniel Hartmeier, qui avait bidouillé quelques bouts de code de son côté, donna à OpenBSD 3.0 un nouveau filtre réseau : Packet Filter.
- L'équipe d'OpenBSD en profita pour réaliser un audit complet des licences des logiciels inclus, afin de régler d'éventuels problèmes de droits...

Fonctionnalités

- Packet Filter est un filtre réseau à suivi d'état (niveaux 3 et 4).
- Il gère la traduction d'adresses, et permet la réécriture de plusieurs champs des en-têtes afin de pacifier un peu nos réseaux. :)
- Il gère la redondance via CARP et pfsync, permettant, sans perte de connexion, qu'une machine secondaire prenne la relève en cas de panne.

Fonctionnalités

- Packet Filter gère également la bande-passante via ALTQ, permettant de créer une politique de qualité de service.
- Un certain nombre d'outils annexes peuvent interagir avec lui pour proposer des services supplémentaires : antispam par liste grise avec spamd, gestion du désastreux protocole FTP via ftp-proxy, passerelle authentifiant via authpf, etc.

Différences par rapport au reste de l'offre existante

- Packet Filter est un filtre de paquets, pas une machine à café.
- Packet Filter ne fait pas de filtrage applicatif.
- Packet Filter ne gère pas naturellement l'ouverture dynamique de ports. Il faut passer par des outils externes, tels que ftp-proxy, qui ont la possibilité d'interagir avec le jeu de règles en passant par un système d'ancres.

Ce dont on ne parlera pas dans la suite de cette présentation :-)

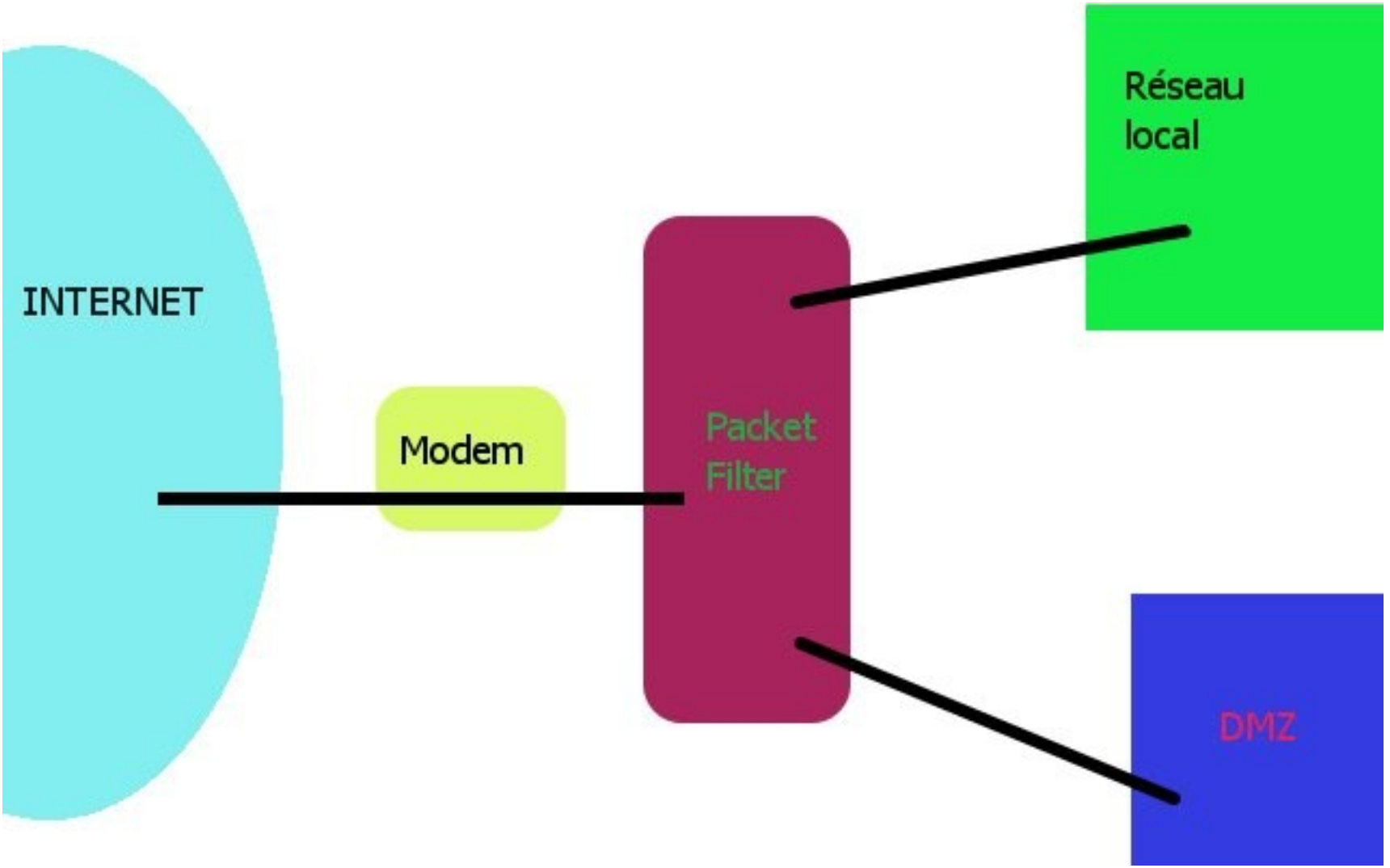
- Gestion de la bande passante (ALTQ)
- Redondance (CARP/pfsync)
- IPv6
- FTP (ftp-proxy)
- Passerelle authentifiante (authpf)

Découverte de Packet Filter

II. Le fichier pf.conf

- Macros
- Tables
- Options
- Normalisation du trafic (scrub)
- Traduction d'adresse (NAT)
- Règles de filtrage

Schéma



Avertissement

- Il y a eu beaucoup de travail sur Packet Filter entre les versions 4.5 (1er mai 2009) et 4.6 (1er novembre 2009) d'OpenBSD.
- La principale différence concerne la normalisation des paquets (`scrub`), nous en reparlerons plus tard.
- À partir d'OpenBSD 4.6, l'ordre des règles n'a plus à être respecté de manière stricte (l'option `require-order` est désormais à `no` par défaut).

Les macros

- Les macros sont des « variables ».
- Elles sont définies dans la première section du fichier de règles PF, et sont utilisées partout ailleurs.
- Après avoir défini la macro `macro`, on l'utilise en préfixant son nom par un `$` : `$macro`.

Les macros

- À chaque référence d'une macro, son contenu est développé au sein de la règle.
- Si la macro est une liste, alors ce sont en fait plusieurs règles qui sont créées : une pour chaque élément de la liste.

Les macros

```
localhost = "127.0.0.1"

ext_if = "fxp0"
int_if = "tl1"
dmz_if = "tl0"

int_net = "192.168.2.0/24"
dmz_net = "192.168.1.0/24"
```

Les macros

```
# arbeit is my main workstation
arbeit = "192.168.2.1"
arbeit_services_tcp = "4665" # Hello HADOPI !
arbeit_services_udp = "4675"


# espoir is my home server
espoir = "192.168.1.1"
espoir_services_tcp = "{ ssh, smtp, www,
https, smtps, submission, silc, pop3s }"


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

Les macros : exemple

Si on a

```
ext_if = "fxp0"
int_if = "tl1"
```

alors

```
block on $ext_if
block on $int_if
```

deviennent, respectivement :

```
block on fxp0
block on tl1
```

Les macros : exemple

Si on a

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

alors

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

devient

```
pass inet proto icmp all icmp-type echoreq
pass inet proto icmp all icmp-type unreach
```

Les tables

- Les tables sont analogues aux listes, mais la recherche d’un élément dans une table est optimisée (en temps et en mémoire).
- La FAQ de PF stipule 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 ».
- Elles sont donc idéales pour regrouper des adresses IP (listes noires/blanches, etc.).

Les tables

- La syntaxe pour nommer une table est la même que la syntaxe permettant de la référencer par la suite : `<table>`.
- Les tables doivent être définies dans la deuxième section du fichier pf.conf, après les macros mais avant les options.

Les tables : exemples

```
table <spamd-white> persist file "/etc/mail/spamd-white"

table <bruteforce> persist

# Non-routable addresses from elsewhere...
table <martians> const { 127.0.0.0/8, \
192.168.0.0/16, 172.16.0.0/12, \
10.0.0.0/8, 169.254.0.0/16, 192.0.2.0/24, \
0.0.0.0/8, 240.0.0.0/4 }
```

Les tables : pfctl(8)

Et avec `pfctl(8)` :

- visualisation du contenu d'une table :

```
pfctl -t table -T show
```

- ajout d'un élément :

```
pfctl -t table -T add 66.102.9.99
```

- suppression d'un élément :

```
pfctl -t table -T delete 66.102.9.99
```

Les options

- Il s'agit d'options générales affectant le comportement global de Packet Filter.
- Elles se configurent par le biais de la directive `set`.
- Elles sont à indiquer après la définition des tables mais avant les règles de normalisation de paquets (troisième section).

Les options : block-policy

Quand une connexion se présente (paquet SYN), et qu'on décide de la bloquer, on a deux choix : soit on répond explicitement non (paquet TCP RST ou ICMP unreach), soit on ne fait rien.

L'option `block-policy` donne un comportement par défaut : `return` pour répondre, `drop` pour abandonner silencieusement la connexion.

```
set block-policy return
```

Les options : loginterface

Définit l'interface depuis laquelle les données statistiques doivent être récupérées. Cette option est par défaut positionnée sur `none` (aucune interface), mais dans l'immense majorité des cas ce devrait être `$ext_if`. On ne peut récupérer des informations que depuis une seule interface à la fois.

```
set loginterface $ext_if
```

Les options : debug

Permet de paramétrer le niveau de débogage de pf.

Par défaut, pf ne journalise qu'à partir du niveau `urgent` (messages de débogage générés pour les erreurs sérieuses), mais le niveau `misc` permet d'avoir l'état du sous-système `scrub`.

```
set debug misc
```

Les options : skip

Permet de désactiver *tous* les traitements sur une interface donnée. Particulièrement utile pour les interfaces de bouclage (loopback).

```
set skip on lo0
```

Normalisation du trafic : scrub

L'idée de la normalisation du trafic est d'empêcher que des paquets mal formés ou des combinaisons de paquets incohérentes (`nmap -O`) puissent atteindre le réseau que l'on souhaite protéger. Le réassemblage des paquet fragmentés entre également dans ce cadre.

Dans Packet Filter, tout cela se fait **très** simplement, au moyen de la directive `scrub`.

Avertissement

- Il y a eu beaucoup de travail sur Packet Filter entre les versions 4.5 (1er mai 2009) et 4.6 (1er novembre 2009) d'OpenBSD.
- À partir d'OpenBSD 4.6, la directive `scrub` disparait, et elle devient intégrée à la logique des règles de filtrage (nous en parlerons juste après).
- Ce changement de comportement sera probablement intégré plus tard dans les implémentations de PF des autres systèmes (FreeBSD, NetBSD, Dragonfly BSD, etc.).

Normalisation du trafic : scrub

Basiquement, il suffit de déclarer (après les options mais avant les règles de traduction) :

```
scrub in all
```

C’est tout. ;-)

scrub : les options

- `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.

- `reassemble tcp`

  Ajoute des normalisations supplémentaires afin de limiter encore les fuites d’informations (impossibilité de réduire le TTL IP, aléa sur les timestamps TCP).

scrub : les options

- `fragment reassemble`

  Comportement par défaut : les paquets fragmentés sont réassemblés avant d'être transmis au moteur de filtrage. C'est la seule option `fragment` qui fonctionne avec la NAT.

- `no-df`

  Supprime le bit "don't fragment" de l'en-tête du paquet IP. Cela permet de gérer certaines incompatibilités introduites par la normalisation.

Rappel

Tout ceci n’est vrai que pour les implémentations de Packet Filter correspondant aux versions antérieures à OpenBSD 4.6 !

scrub pour OpenBSD 4.6

- Entre OpenBSD 4.5 et 4.6, la directive `scrub` a fait l'objet d'une refonte, et elle est désormais intégrée à la logique des règles de filtrage.
- L'activation de la normalisation de paquet se fait donc désormais soit :
  - en ajoutant le paramètre `scrub (`*`options`*`)` aux règles de filtrage ;
  - via une règle `match`.

scrub pour OpenBSD 4.6

Cela ajoute un degré de souplesse : on peut toujours gérer la normalisation de trafic au niveau global (règle `match`), mais on peut désormais aussi la gérer au cas par cas en ajoutant `scrub` à une règle `pass`.

scrub pour OpenBSD 4.6

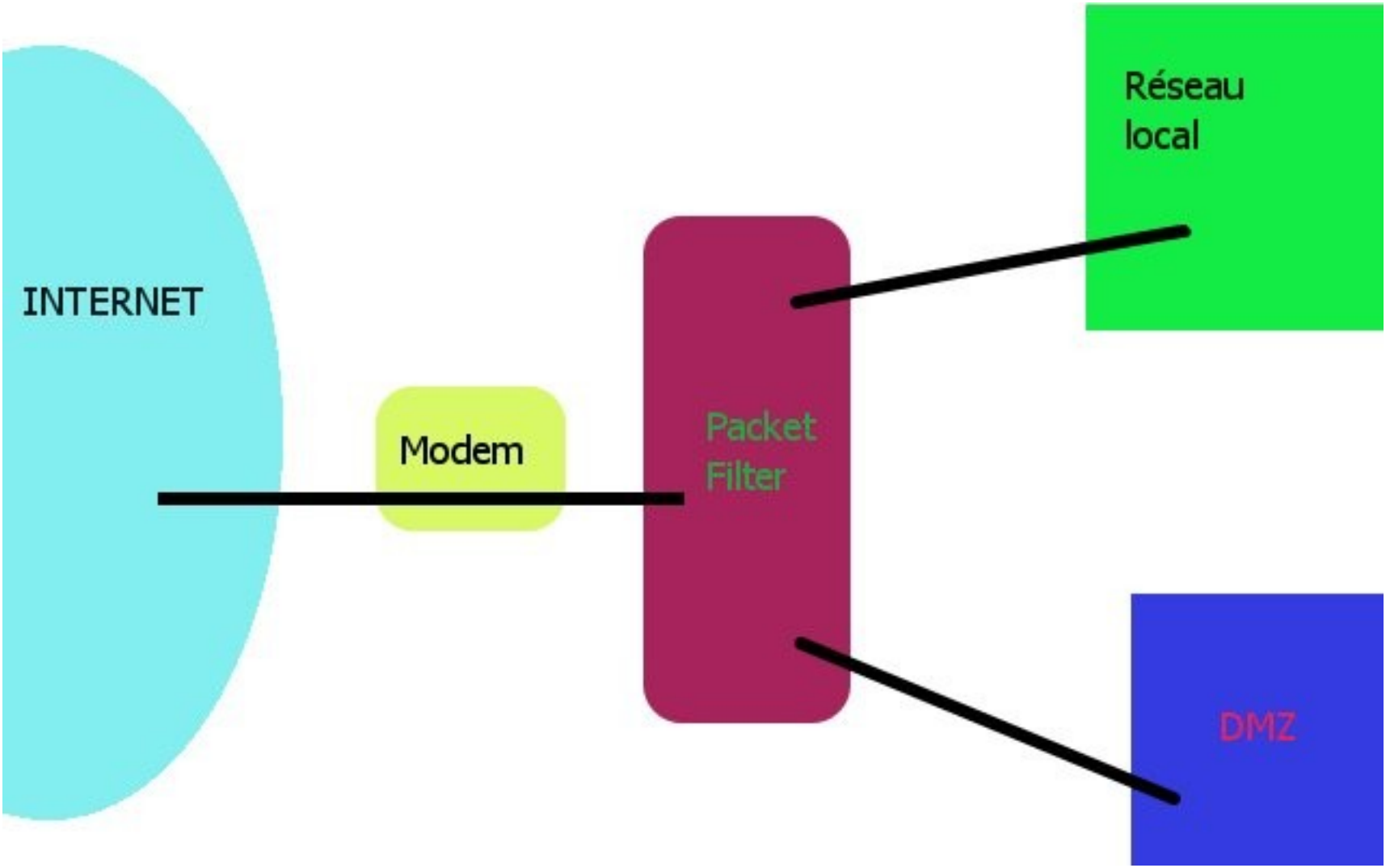
Concrètement, pour remplacer la règle

```
scrub in all no-df max-mss 1440
```

on peut utiliser ceci :

```
match in all scrub (no-df max-mss 1440)
```

Schéma



Traduction d’adresses (NAT)

Le mécanisme de NAT (Network Address Translation, soit *Traduction d’Adresse Réseau*) se compose de deux sous-mécanismes, l’un étant utile pour le trafic sortant, l’autre pour le trafic entrant.

Traduction d’adresses (NAT)

- Pour le trafic sortant, on utilise des adresses IP privées, non routables sur Internet, et la passerelle doit réécrire les en-têtes des paquets afin de modifier les adresses.
- Pour le trafic entrant, on utilise des redirections d’adresses et de ports ; la passerelle doit également réécrire les en-têtes des paquets pour modifier les adresses et/ou les ports.

Traduction d’adresses (NAT)

Ces deux mécanismes sont donc bien entendu gérés de manière indépendantes dans Packet Filter.

Mais, avant d’aborder le premier d’entre eux, il est nécessaire d’introduire le mécanisme de balisage (`tag`)...

Balisage des paquets : `tag`

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.

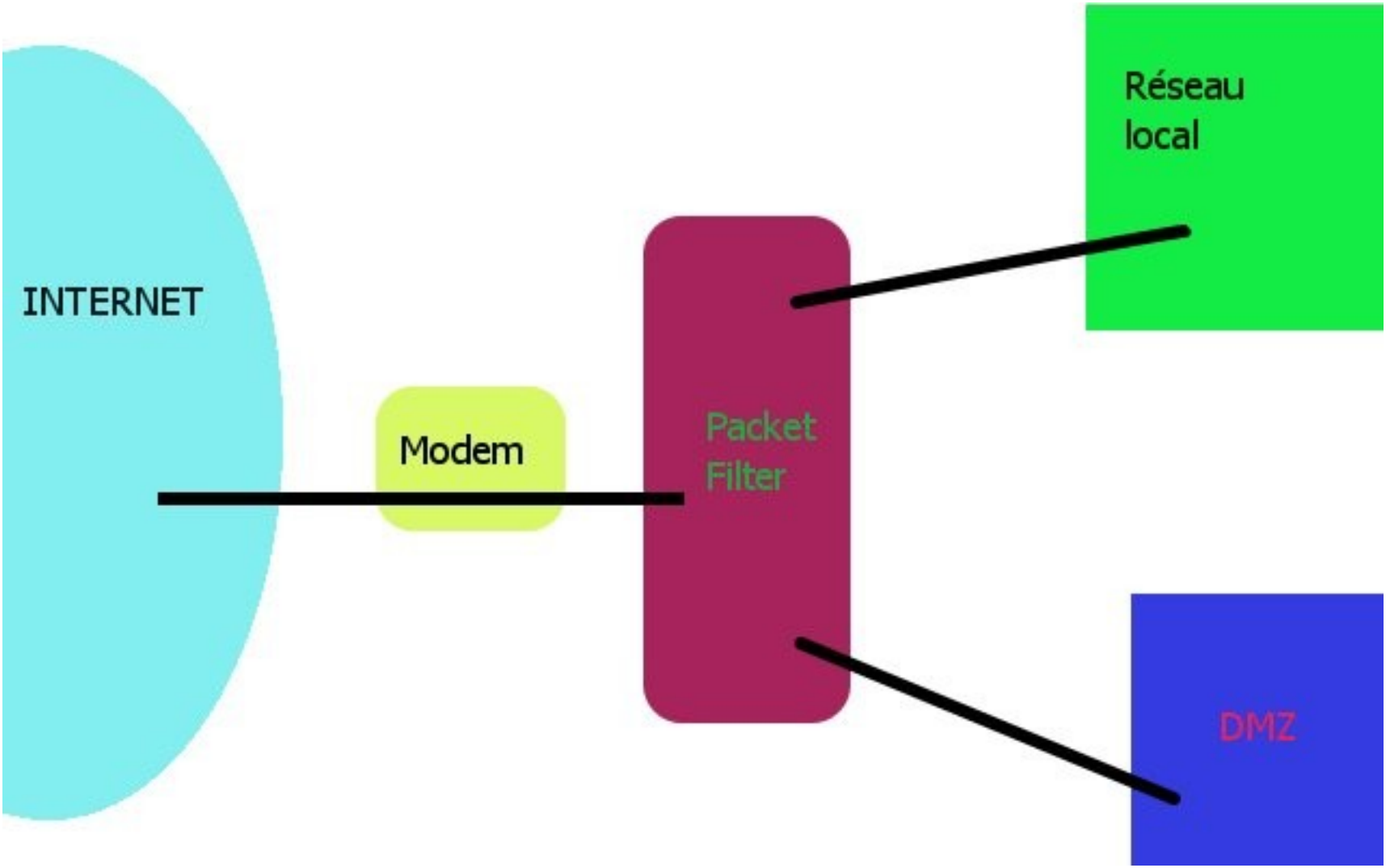
Grâce au balisage, il est possible de déterminer si des paquets ont été traités par les règles de traduction d'adresses. Il est aussi possible de faire du filtrage suivant une politique au lieu de faire du filtrage par règle.

Balisage des paquets : `tag`

Concrètement, dans une règle (de filtrage ou de traduction), on ajoute l'expression `tag` *`TAG`* pour appliquer une balise, et l'expression `tagged` *`TAG`* pour vérifier si une balise a été appliquée au paquet.

Nous n'allons pas ici nous contenter de faire du filtrage par règle : nous allons construire une politique de filtrage !

Schéma



Les règles `nat`

```
# NAT on LAN -> INET (but not on LAN -> DMZ)
nat on $ext_if tag LAN_INET_NAT tagged LAN_INET ->
($ext_if)

# NAT on DMZ -> INET (but not on DMZ -> LAN)
nat on $ext_if tag DMZ_INET_NAT tagged DMZ_INET ->
($ext_if)
```

Ceci fonctionne parce la logique de NAT se place entre `$int_if`/`$dmz_if` et `$ext_if` !

Redirections de ports

```
# arbeit
rdr on $ext_if proto tcp from any to any port
$arbeit_services_tcp -> $arbeit
rdr on $ext_if proto udp from any to any port
$arbeit_services_udp -> $arbeit

# espoir
rdr on $ext_if proto tcp from any to any port
$espoir_services_tcp -> $espoir
```

Redirections de ports

La redirection de ports se fait après l'entrée via l'interface extérieure, mais avant la sortie via l'interface LAN ou l'interface DMZ.
Il ne faut donc pas oublier de l'autoriser à sortir !

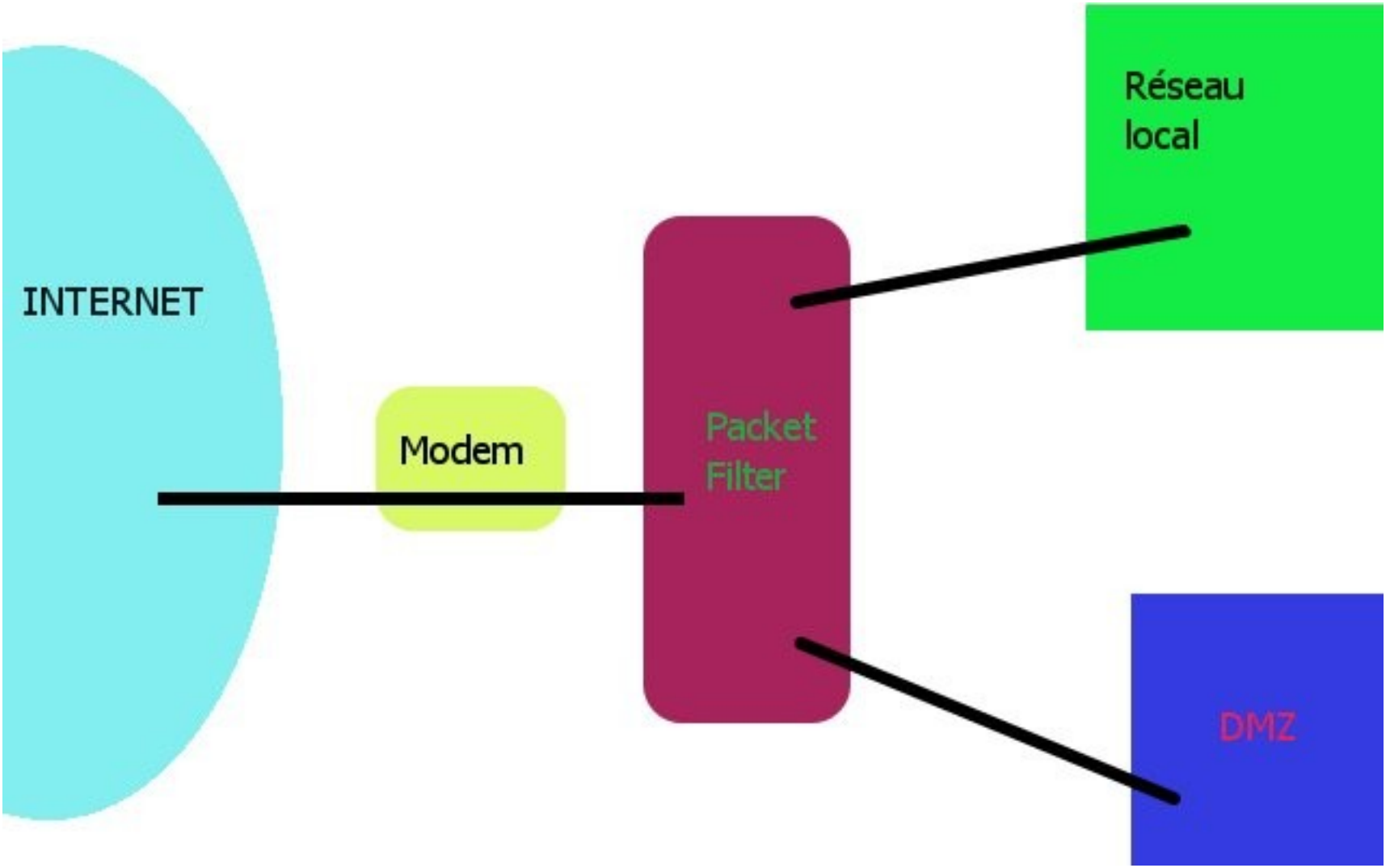
On peut le faire soit en spécifiant `rdr pass` dans la règle de redirection, soit en créant une règle de filtrage, qui pourra prendre en compte le `tag` éventuellement indiqué au niveau NAT...

Règles de filtrage

Nous avons choisi de ne pas nous contenter de faire du filtrage par règle, mais d'établir une véritable politique de filtrage pour la totalité de notre réseau.

Nous allons donc créer notre jeu de règles de filtrage en séparant notre trafic en différents cas, et en les traitant **tous** un par un.

Schéma



Énumération des cas

- Le trafic entrant et sortant du réseau local : vers Internet (deux cas) ou vers la DMZ (deux cas).
- Le trafic entrant et sortant de la DMZ : vers Internet (deux cas) ou vers le réseau local (deux cas déjà énumérés).
- Le trafic propre à la passerelle : vers Internet (deux cas), vers le réseau local (deux cas) et vers la DMZ (deux cas).
- Cela nous fait donc 12 cas à traiter, répartis en 6 paires « réciproques ».

Politique de refus par défaut

Par défaut, nous bloquons tout le trafic, puis nous autorisons au cas par cas :

```
block in quick inet6 all
block on $ext_if
block on $int_if
block in on $dmz_if
block out log on $dmz_if
```

PF ne prend en compte que la dernière règle correspondante !

Suivi de l’état des connexions

- Packet Filter est un filtre à suivi d’état (*stateful*).
- Depuis OpenBSD 4.1, `keep state` (conservation de l’état) est activée par défaut pour chaque règle.
- Packet Filter gère la modulation de l’état via la directive `modulate state`. Elle ne s'applique qu'aux paquets TCP sortants, en renforçant le caractère aléatoire de leurs numéros de séquence initiaux (ISN).

Suivi des connexions : synproxy

- La directive `synproxy state` permet à PF de s'occuper lui-même de l'établissement d'une connexion TCP (la fameuse triple poignée de main), pour rendre par la suite le contrôle de la connexion à la machine concernée.
- `synproxy state` apporte les mêmes avantages que les options keep state et modulate state.

Filtrage : LAN -> INTERNET

```
# LAN -> INET
# Remember: those rules are NATed...
pass in quick on $int_if from $int_net to !$dmz_net tag
LAN_INET keep state
pass out quick on $ext_if tagged LAN_INET_NAT keep state
```

La logique de NAT s'applique entre ces deux règles, d'où le changement de balise.

Le mot-clé `quick` permet de sortir immédiatement de la logique de filtrage.

Filtrage : INTERNET -> LAN

```
# INET -> LAN
# INET -> arbeit
# e-mule (Hi HADOPI !)
pass in on $ext_if proto tcp to $arbeit port
$arbeit_services_tcp tag INET_ARBEIT_TCP keep state
pass in on $ext_if proto udp to $arbeit port
$arbeit_services_udp tag INET_ARBEIT_UDP keep state

pass out on $int_if tagged INET_ARBEIT_TCP keep state
pass out on $int_if tagged INET_ARBEIT_UDP keep state
```

Filtrage : DMZ -> INTERNET

```
# DMZ -> INET
# espoir -> INET
# Remember: thoses rules are NATed...
pass in log quick on $dmz_if from $espoir to !$int_net tag
DMZ_INET keep state
pass out quick on $ext_if tagged DMZ_INET_NAT keep state
```

C’est exactement la même chose que pour le trafic qui va du LAN vers Internet...

Filtrage : INTERNET -> DMZ

```
# INET -> DMZ
# INET -> espoir
pass in log on $ext_if proto tcp to $espoir port
$espoir_services_tcp tag INET_ESPOIR flags S/SFRA synproxy
state
pass out quick on $dmz_if tagged INET_ESPOIR keep state
```

L'option flags se rapporte aux drapeaux trouvés dans l'en-tête du paquet TCP. Ici, pour que la règle corresponde, il faut que seul le drapeau SYN soit activé parmi les drapeaux SYN, FIN, RST et ACK.

antispoof

La directive `antispoof` permet de se prémunir, au niveau d'une interface réseau, contre l'usurpation d'adresse IP (*spoofing*), en vérifiant que l'adresse source d'un paquet appartient bien au réseau rattaché à l'interface.

Concrètement, il nous suffit de placer ces deux règles juste avant les règles `block` :

```
antispoof log quick for $int_if
antispoof log quick for $dmz_if
```

Les adresses venues d'ailleurs

Nous avions déclaré la tables <martians> pour présenter les tables, voici venu le moment de s'en servir :

```
# Non-routable addresses from elsewhere...
block in quick on $ext_if from <martians> to any
block out quick on $ext_if from any to <martians>
```

III. Contre-mesures anti-DoS

L'idée est de placer en liste noire les adresses qui dépassent certaines limites déclarées au niveau d'une règle.

On déclare une table `<bruteforce>` persistante, et on peut alors utiliser des règles comme celles-ci :

```
pass in log quick on $ext_if proto tcp to $espoir port ssh
tag INET_ESPOIR flags S/SA synproxy state \
      (max-src-conn 25, max-src-conn-rate 10/5, overload
<bruteforce> flush global)
```

FIN :)

Les exemples de cette présentation étaient tous tirés de mon script pf.conf personnel :
https://www.mouet-mouet.net/doku.php?id=mouet-mouet:routeur

On trouvera plus d'informations en lisant l'excellente documentation officielle (pf.conf(5) et la FAQ PF trouvable sur http://www.openbsd.org/faq/pf/fr/) ainsi que l'ouvrage « Le Livre de Packet Filter » de Peter N. M. HANSTEEN (Éditions Eyrolles).