

# Installation de pare-feu redondants avec OpenBSD

Matthieu Herrb



# Agenda

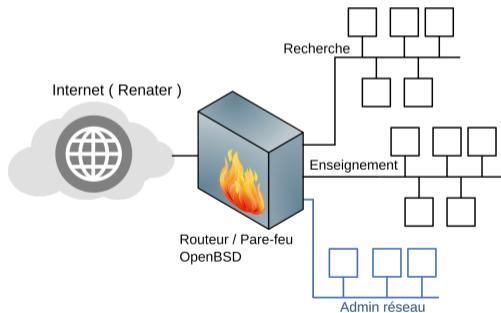
- 1 Introduction
- 2 Installation du système
- 3 Concepts réseau OpenBSD
- 4 PF - Introduction
- 5 Configuration des pare-feux
- 6 PF - tables
- 7 Administration système
- 8 Mises à jour
- 9 Conclusion

# Agenda

- 1 Introduction
- 2 Installation du système
- 3 Concepts réseau OpenBSD
- 4 PF - Introduction
- 5 Configuration des pare-feux
- 6 PF - tables
- 7 Administration système
- 8 Mises à jour
- 9 Conclusion

# Introduction

Ce tutoriel présente le fonctionnement d'un pare-feu basé sur le filtre de paquets **pf** d'OpenBSD et va jusqu'à la configuration complète d'un système redondant, composé de deux machines fonctionnant en routeur en entrée d'un réseau



- Préparatifs :
  - choix et dimensionnement du matériel utilisé,
  - notions réseau OpenBSD : domaines de routage, CARP et PF,
  - installation du système et configuration initiale du réseau en IPv4 et IPv6.
- Configuration et paramétrage :
  - redondance via le protocole CARP + pfsync.
  - utilisation des domaines de routage pour isoler les interfaces d'administration.
  - écriture des règles de filtrage pour implémenter une politique de sécurité.
  - ré-écriture des entêtes des paquets pour réaliser la traduction d'adresses.
- Exploitation au quotidien :
  - gestion des traces via l'interface pflog.
  - outils pour l'analyse des performances et la collecte de métriques.
  - gestion des mises à jour de sécurité et les montées de versions sans interruption grâce à l'architecture mise en place.

- Système d'exploitation dérivé de BSD 4.4, compatible POSIX
- Architectures matérielles : i386, amd64, sparc64, macppc, arm64,...
- Code libre : Licence BSD
- Noyau + userland + documentation maintenus ensemble
- Projets annexes : [OpenSSH](#), [OpenNTPD](#), [OpenSMTPd](#), [OpenKEd](#), [rpki-client](#),...
- Applications tierces disponibles via ports
- Une release tous les 6 mois

### OpenBSD 7.1 sortie le 21 avril 2022.

Nouveautés :

- support plateformes Apple M1
- améliorations parallélisme dans le noyau
- synchronisation DRM avec Linux 5.15
- correction de bugs



# Utilisation d'OpenBSD au LAAS

Plusieurs services :

- pare-feux externes
- serveurs DNS autoritaires (bind)
- serveur NTP
- pare-feux internes
- serveurs DHCP
- serveurs DNS cache/récurifs (unbound)
- cache Web (squid)
- sonde arpwatc



# Ressources

- <http://www.openbsd.org/>
- <http://www.openbsd.org/faq/index.html>
- <http://www.openbsd.org/books.html>
- <http://undeadly.org/>
- <https://webzine.puffy.cafe/>
- Canal IRC #openbsd-fr sur libera.chat

# Agenda

- 1 Introduction
- 2 Installation du système**
- 3 Concepts réseau OpenBSD
- 4 PF - Introduction
- 5 Configuration des pare-feux
- 6 PF - tables
- 7 Administration système
- 8 Mises à jour
- 9 Conclusion

Pour un routeur / pare-feu OpenBSD:

- Processeur x86\_64 (amd64)
  - privilégier la fréquence du CPU au nombre de cœurs (parallélisme limité dans le noyau)
  - éviter la toute dernière génération qui vient de sortir
- 8Go de RAM suffisent pour ce type d'application.
- Disque de 128 Go suffisent. Davantage pour conserver des traces importantes
- Cartes réseau de qualité 1Gb/s ou 10Gb/s
  - Éviter les puces Realtek (pilote *re(4)*)
  - Vérifier la compatibilité dans la [liste des drivers](#)
  - 4 ports réseau pour la configuration de base.

Il est possible de virtualiser les routeurs pare-feux OpenBSD.

Hyperviseurs supportés :

- VMware (pilotes *vmx(4)*, *vic(4)*, *vscsi(4)*)
- Qemu (Proxmox, libvirt,...) (pilotes *vio(4)*, *vioblk(4)*)
- VirtualBox (pilotes *vio(4)*, *vioblk(4)*)
- Xen (pilotes *xnf(4)*, *xbf(4)*, *xen(4)*)
- Hyper-V (pilotes *hvn(4)*, *hvs(4)*)

Pour CARP: autoriser les interfaces hôte en mode promiscuous pour recevoir le trafic sur l'adresse MAC virtuelle

Hyperviseur OpenBSD.

Il permet:

- l'exécution de machines virtuelles (para-virtualisation)
- gestion de configuration réseau virtuelles

Utilisé dans ce tutoriel pour grouper la démo sur une seule machine physique:

- les deux pare-feux et leurs interfaces réseau
- une machine virtuelle de test

Mais: support limité de la migration de machines virtuelles entre hôtes :  
peu intéressant en pratique pour la virtualisation de pare-feux redondants en production  
(pour l'instant).

```
viornd0 at virtio0
virtio0: irq 3
virtio1 at pci0 dev 2 function 0 "Qumranet Virtio Network" rev 0x00
vio0 at virtio1: address fe:e1:bb:d1:bf:bb
virtio1: irq 5
virtio2 at pci0 dev 3 function 0 "Qumranet Virtio Network" rev 0x00
vio1 at virtio2: address fe:e1:bb:d2:7c:ea
virtio2: irq 6
virtio3 at pci0 dev 4 function 0 "Qumranet Virtio Network" rev 0x00
vio2 at virtio3: address fe:e1:bb:d3:5a:9e
virtio3: irq 7
virtio4 at pci0 dev 5 function 0 "Qumranet Virtio Network" rev 0x00
vio3 at virtio4: address fe:e1:bb:d4:d7:34
virtio4: irq 9
virtio5 at pci0 dev 6 function 0 "Qumranet Virtio Storage" rev 0x00
vioblk0 at virtio5
scsibus0 at vioblk0: 1 targets
sd0 at scsibus0 targ 0 lun 0: <VirtIO, Block Device, >
sd0: 30720MB, 512 bytes/sector, 62914560 sectors
virtio5: irq 10
virtio6 at pci0 dev 7 function 0 "OpenBSD VMM Control" rev 0x00
vmmci0 at virtio6
virtio6: irq 11
isa0 at mainbus0
com0 at isa0 port 0x3f8/8 irq 4: ns8250, no fifo
com0: console
kbc selftest: 20
softraid0 at root
scsibus1 at softraid0: 256 targets
root on rda swap on rdb dump on rdb
WARNING: CHECK AND RESET THE DATE!
erase ^?, werase ^W, kill ^U, intr ^C, status ^T
```

```
Welcome to the OpenBSD/amd64 7.1 installation program.
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? i
```

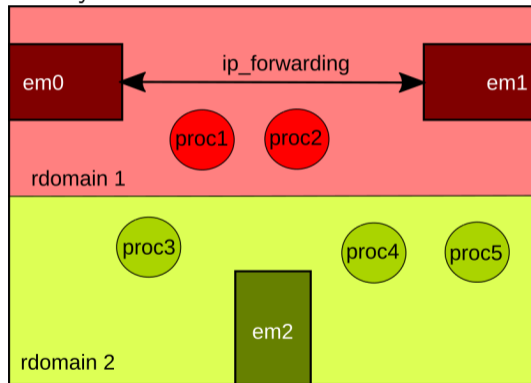
# Agenda

- 1 Introduction
- 2 Installation du système
- 3 Concepts réseau OpenBSD**
- 4 PF - Introduction
- 5 Configuration des pare-feux
- 6 PF - tables
- 7 Administration système
- 8 Mises à jour
- 9 Conclusion

# Domaines de routage

Plusieurs tables de routage indépendantes dans le système.

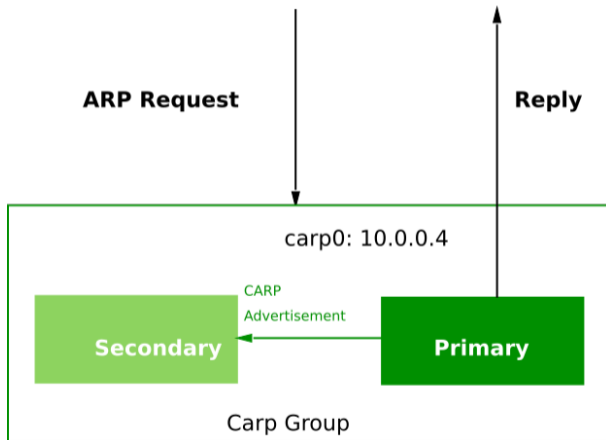
- Chaque processus est attaché à un **domaine de routage** qui contient plusieurs tables de routage.
- Les interfaces réseau sont affectées à un domaine de routage.
- Permet de séparer complètement le trafic réseau.



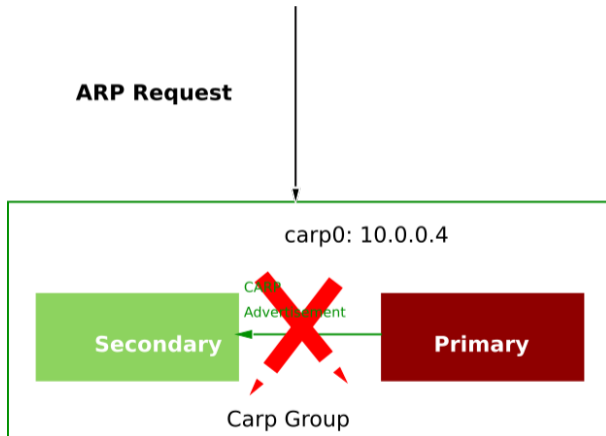


- CARP gère la redondance à la frontière des niveaux 2 et 3.
- **groupe CARP** : une adresse MAC et une adresse IP partagés par plusieurs nœuds
- le nœud actif (*primaire*) répond aux requêtes ARP pour l'adresse IP
- utilise l'adresse MAC du groupe  
→ mise à jour des tables MAC des switches.
- Le nœud primaire envoie périodiquement des annonces CARP (multicast)  
→ indique qu'il est vivant
- Si plus d'annonces reçues, les autres membres envoient leurs annonces, puis élection d'un nouveau nœud primaire.

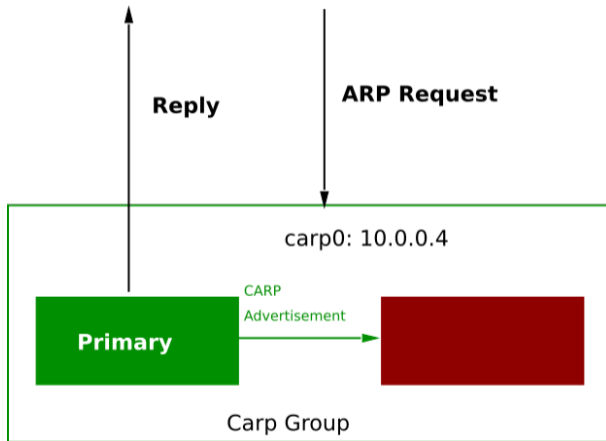
# CARP: illustration



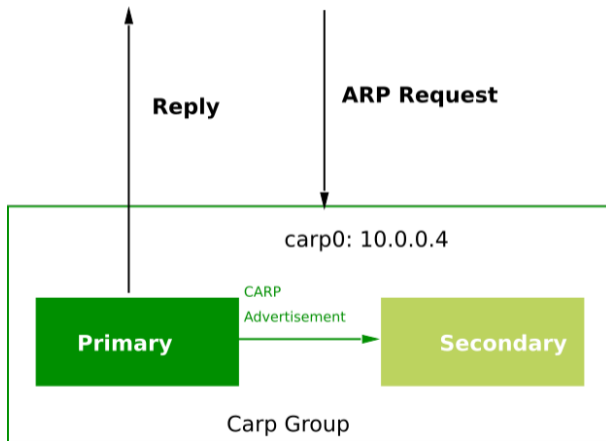
# CARP: illustration



# CARP: illustration



# CARP: illustration



Permet de définir des priorités entre les membres d'un groupe

- celui qui a la plus haute priorité (re)devient le primaire
- répartition de charge à partir des priorités
  - au niveau ARP
  - au niveau IP

# CARP : Spécificités par rapport à VRRP

- indépendant de la version du protocole.  
Supporte IPv4 et IPv6
- fonction arp balance pour faire du partage de charge.
- protection des annonces par HMAC SHA-1

# Agenda

- 1 Introduction
- 2 Installation du système
- 3 Concepts réseau OpenBSD
- 4 PF - Introduction**
- 5 Configuration des pare-feux
- 6 PF - tables
- 7 Administration système
- 8 Mises à jour
- 9 Conclusion



# Qu'est-ce que PF ?

- Filtre de Paquets réseau niveau 3
- Intervient sur le trafic entrant ou sortant d'une interface
- Permet de :
  - bloquer/autoriser le paquet à continuer son chemin
  - placer un label sur le paquet
  - modifier le paquet (NAT, ...)
  - re-diriger le paquet vers une autre destination

# Filtrage à état (stateful)

- Les protocoles réseau sont gérés par des automates (exemple: TCP)
- PF suit l'état de l'automate pour les protocoles connus
- Cela permet de :
  - détecter les paquets qui violent le protocole
  - limiter la complexité des règles de filtrage en laissant passer automatiquement les paquets valides du protocole.
  - contrôler les ressources utilisées par le filtre

- Stockées dans `/etc/pf.conf`
- S'appliquent au niveau 3 (IP, ICMP, TCP, UDP, ...)
- Lues dans l'ordre où elles apparaissent
- La **dernière** règle qui matche un paquet s'applique.  
(Attention : différent de Cisco / Linux ...)
- Si une règle modifie un paquet, le paquet modifié est utilisé en entrée des règles suivantes.
- Les tables permettent de traiter des listes dynamiques d'adresses IP

- réécriture des adresses source : NAT/PAT vers
  - une adresse unique
  - un pool d'adresses
- réécriture des adresses destination : redirection vers
  - un relais applicatif (hors du noyau)
  - équilibrage de charge
- traductions en cours stockées dans la table d'états des connexions.

- **but** : éliminer les ambiguïtés / erreurs d'interprétation des états et protection des piles IP vulnérables.
- ré-assemble les fragments des paquets IP avant filtrage
- modification (réécriture) des numéros de séquence TCP

- Utilise le format **pcap** ( tcpdump )
- Permet analyse ultérieure avec nombreux outils
- Outils pour redirection vers syslog disponible

# Configuration minimale

`/etc/pf.conf :`

```
pass
```

Charger les règles dans le noyau : `pfctl -f /etc/pf.conf`

Voir les règles créées : `pfctl -vs rules`

Voir les états créés : `pfctl -vs states`

# Protection de la machine locale

```
set skip on lo0
block in
pass out
pass in on egress proto tcp port ssh
```

**egress** : groupe d'interfaces vers l'extérieur (défini automatiquement lors de la configuration réseau par `/etc/netstart`).



# Un pare-feu simple

```
intif = "em0"  
dmz = "em1"  
extif = "em2"  
  
set skip on lo, $intif, $dmz  
block in on $extif  
pass out on $extif  
pass in on $extif proto icmp  
pass in on $extif to ($dmz:network) \  
    port {ssh, smtp, domain, www, https, smtps, imaps}
```

# Translation d'adresses et redirection

## *nat-to adresse*

réécrit l'adresse source du paquet,  
et éventuellement le numéro de port source.

### Exemple :

```
match out on $ext_if from 192.168.1.0/24 nat-to ($ext_if)
```

## *rdr-to adresse*

réécrit l'adresse destination et le port destination

### Exemple :

```
pass in quick on $ext_if proto tcp to port ssh rdr-to 192.168.1.22
```

# Routeur NAT

```
extif = "em0"
intif = "em1"
bastion = "192.168.1.100"
set skip on {lo0, $intif}

# NAT en sortie
match on $extif inet from $intif:network nat-to ($extif)

block in log all
pass out on $extif
# ping
pass in on $extif proto icmp all icmp-type echoreq

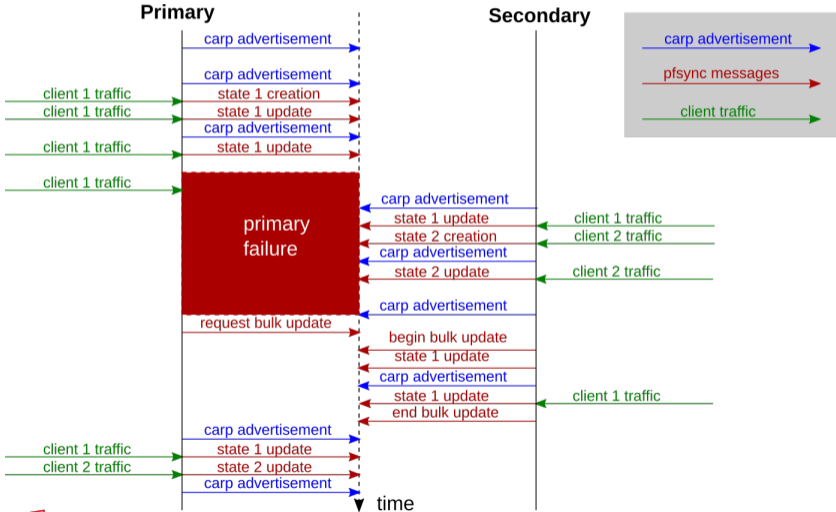
# SSH sur port 2222 vers bastion
pass in on $extif proto tcp to ($extif) port 2222 rdr-to $bastion port 22
```

# Synchronisation de l'état entre les pare-feu.

## pfsync

- protocole multicast 240
- diffuse les créations, mises à jour et destructions d'états
- écoute les mises à jour des voisins
- met à jour la table locale des états

# Chronogramme CARP + pfsync



```
router-a# ifconfig carp | egrep 'carp([01]:|:)'
carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> rdomain
 1 mtu 1500
   carp: MASTER carpdev vio0 vhid 10 advbase 1 advskew 0
carp1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> rdomain
 1 mtu 1500
   carp: MASTER carpdev vio1 vhid 20 advbase 1 advskew 0
router-a# █
```

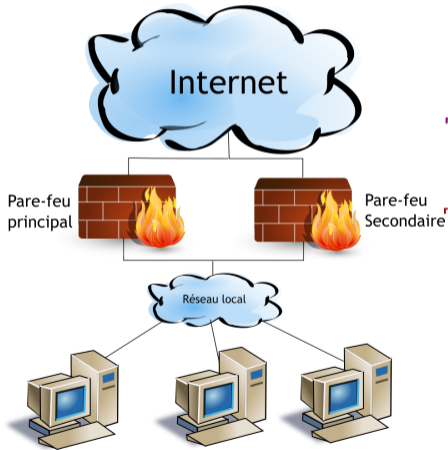
```
router-b# ifconfig carp | egrep 'carp([01]:|:)'
carp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> rdomain
 1 mtu 1500
   carp: BACKUP carpdev vio0 vhid 10 advbase 1 advskew 200
carp1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> rdomain
 1 mtu 1500
   carp: BACKUP carpdev vio1 vhid 20 advbase 1 advskew 200
router-b# █
```

I

```
64 bytes from 194.57.3.79: icmp_seq=4 ttl=241 time=13.664 ms
64 bytes from 194.57.3.79: icmp_seq=5 ttl=241 time=14.209 ms
64 bytes from 194.57.3.79: icmp_seq=6 ttl=241 time=16.766 ms
64 bytes from 194.57.3.79: icmp_seq=7 ttl=241 time=14.770 ms
64 bytes from 194.57.3.79: icmp_seq=8 ttl=241 time=14.892 ms
64 bytes from 194.57.3.79: icmp_seq=9 ttl=241 time=14.864 ms
64 bytes from 194.57.3.79: icmp_seq=10 ttl=241 time=14.599 ms
64 bytes from 194.57.3.79: icmp_seq=11 ttl=241 time=13.577 ms
█
```

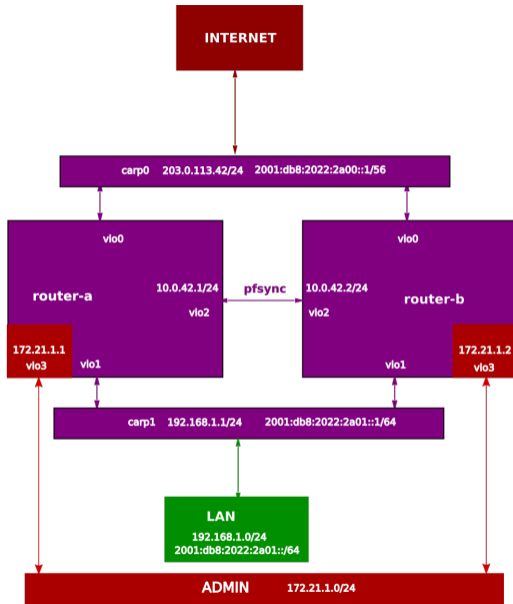
# Agenda

- 1 Introduction
- 2 Installation du système
- 3 Concepts réseau OpenBSD
- 4 PF - Introduction
- 5 Configuration des pare-feux**
- 6 PF - tables
- 7 Administration système
- 8 Mises à jour
- 9 Conclusion



routing domain 1

routing domain 0





## Interfaces physiques

/etc/hostname.vio0 et /etc/hostname.vio1 sur les 2 routeurs

```
up
rdomain 1
```

routeur A : /etc/hostname.vio2

```
rdomain 1
up
inet 10.0.42.1 255.255.255.0
```

routeur B : /etc/hostname.vio2

```
rdomain 1
up
inet 10.0.42.2 255.255.255.0
```

(le transport de pfsync c'est IPv4 uniquement)

## CARP: routeur A

/etc/hostname.carp0

```
rdomain 1
vhid 10 carpdev vio0 pass SuperSecret10
inet 203.0.113.42 255.255.255.255
inet6 2001:db8:2022:2a00::1/56
! route -T1 add default 203.0.113.1
! route -T1 add -inet6 default fe80::1%carp0
```

/etc/hostname.carp1

```
rdomain 1
vhid 20 carpdev vio1 pass SuperSecret20
inet 192.168.1.1 255.255.255.0
inet6 2001:db8:2022:2a01::1/64
```

## CARP: routeur B

/etc/hostname.carp0

```
rdomain 1
vhid 10 advskew 200 carpdev vio0 pass SuperSecret10
inet 203.0.113.42 255.255.255.255
inet6 2001:db8:2022:2a00::1/56
! route -T1 add default 203.0.113.1
! route -T1 add -inet6 default fe80::1%carp0
```

/etc/hostname.carp1

```
rdomain 1
vhid 20 advskew 200 carpdev vio1 pass SuperSecret20
inet 192.168.1.1 255.255.255.0
inet6 2001:db8:2022:2a01::1/64
```

/etc/hostname.pfsync0 sur les 2 routeurs :

```
rdomain 1  
up syncdev vio2
```

Sur les 2 routeurs :

```
ext=carp0
ext_if=vio0
int=carp1
int_if=vio1
sync=pfsync0
sync_if=vio2
adm=vio3

set skip on { lo $int_if $adm }

set loginterface $ext_if
```

```
block log

# Laisse passer pfsync
pass quick on $sync_if proto pfsync \
    keep state (no-sync)
pass quick on $sync_if proto icmp

# Laisse passer carp
pass quick on $ext_if proto carp \
    keep state (no-sync)

pass out on $ext_if inet nat-to ($ext)
pass out on $ext_if inet6
```

# Agenda

- 1 Introduction
- 2 Installation du système
- 3 Concepts réseau OpenBSD
- 4 PF - Introduction
- 5 Configuration des pare-feux
- 6 PF - tables**
- 7 Administration système
- 8 Mises à jour
- 9 Conclusion

# Tables

- stocker de façon efficace et compacte des listes d'adresses IP
- initialisées dans `pf.conf` ou un fichier dédié
- modifiables dynamiquement avec `pfctl`
- utilisable presque partout où pf attend une adresse.
- mot clé `table`

## Exemple :

```
table <rfc1918> const { 192.168.0.0/16, \  
                      172.16.0.0/12, 10.0.0/8}  
table <spammers> persist file "/etc/pf/spammers"  
  
block from <spammers> to any port { smtp, smtps, submission }  
block to <rfc1918>
```

# Tables - manipulations

```
pfctl -t table -T opération [-f fichier]
```

*table* nom de la table à manipuler

*opération* commande à appliquer

**add** *adresse* ajoute une adresse

**delete** *adresse* supprime une adresse

**load** recharge la définition de la table

**zero** vide la table

**expire** *secondes* supprime les entrées sans trafic depuis plus que *secondes*

**replace** remplace le contenu de la table à partir d'un fichier

**show** liste le contenu de la table



## Négations - attention danger !

! **adresse** matche toutes les adresses sauf celle-ci.

Comportement non intuitif dans une liste !

```
block from { 192.168.2.0/24, !192.168.2.1 }
```

devient :

```
block from 192.168.2.0/24  
block from !192.168.2.1
```

→ bloque tout !

Mais avec une table :

```
table <badguys> { 192.168.2.0/24, !192.168.2.1 }  
block from <badguys>
```

# Agenda

- 1 Introduction
- 2 Installation du système
- 3 Concepts réseau OpenBSD
- 4 PF - Introduction
- 5 Configuration des pare-feux
- 6 PF - tables
- 7 Administration système**
- 8 Mises à jour
- 9 Conclusion

- pf logue des paquets bruts marqués `log` dans les règles
- pf crée une interface dédiée : `pfllog0`
- pf utilise le format **pcap** (tcpdump)
- le démon `pfllogd(8)` permet de stocker ces paquets dans `/var/log/pfllog`
- `tcpdump(8)` peut être utilisé pour visualiser les paquets.  
Options supplémentaires pour examiner l'état de pf associé.

## Logs : exemple

### tcpdump -e -n -i pflog0

```
19:09:09.719965 rule 17/(match) block out on em2: 140.93.1.176 > 192.168.3.1: icmp: echo request
19:09:09.963940 rule 43/(match) block out on em2: 140.93.65.166.1343 > 218.61.22.27.25: \
  S 1926308279:1926308279(0) win 65535 <mss 1460,nop,nop,sackOK> (DF)
19:09:10.459138 rule 43/(match) block out on em2: 140.93.65.166.1343 > 218.61.22.27.25: \
  S 1926308279:1926308279(0) win 65535 <mss 1460,nop,nop,sackOK> (DF)
19:09:10.895738 rule 17/(match) block out on em2: 140.93.0.15.60086 > 172.31.32.250.25: \
  S 192965518:192965518(0) win 49640 <mss 1460,nop,wscale 0,nop,nop,sackOK> (DF)
19:09:10.958700 rule 0/(match) block out on em2: 140.93.254.76.1230 > 169.254.56.138.123: \
  v1 client strat 0 poll 0 prec 0
19:09:10.960826 rule 43/(match) block out on em2: 140.93.65.166.1343 > 218.61.22.27.25: \
  S 1926308279:1926308279(0) win 65535 <mss 1460,nop,nop,sackOK> (DF)
19:09:13.118242 rule 0/(match) block out on em2: 140.93.254.232.1230 > 169.254.56.138.123:\
  v1 client strat 0 poll 0 prec 0
19:09:15.297774 rule 17/(match) block out on em2: 195.83.132.135.123 > 10.1.1.199.123: v3 \
  sym_pas strat 3 poll 4 prec -6 [tos 0x10]
19:09:17.565871 rule 17/(match) block out on em2: 140.93.2.193.58642 > 172.31.32.250.25: \
  S 2821787999:2821787999(0) win 49640 <mss 1460,nop,wscale 0,nop,nop,sackOK> (DF)
19:09:19.884189 rule 0/(match) block out on em2: 140.93.254.107.1230 > 169.254.56.138.123: \
  v1 client strat 0 poll 0 prec 0
```

# Informations sur l'état du système

**pfctl(8)** permet de voir :

- l'état général de pf : **pfctl -si**
- les règles actives : **pfctl -vvsr**
- la table des états : **pfctl -vvss**
- les tables : **pfctl -vvsT**
- ...

**systat(8)** permet de voir en temps réel :

- l'état général de pf : **systat pf**
- les règles actives : **systat rules**
- le début de la table des états : **systat states**

Nombreuses briques de base :

- `snmpd(8)` permet d'exporter les compteurs par interface.  
(Y compris `pflog0`).
- `pfstat(1)` dans les ports permet de faire des graphiques avec `rrdtool`
- `pflow(4)` permet d'exporter des données *netflow* v5.



# Agenda

- 1 Introduction
- 2 Installation du système
- 3 Concepts réseau OpenBSD
- 4 PF - Introduction
- 5 Configuration des pare-feux
- 6 PF - tables
- 7 Administration système
- 8 Mises à jour**
- 9 Conclusion



# Les branches d'OpenBSD

- release** la version distribuée, ne change pas après la mise à disposition initiale.  
Une release tous les six mois.
- stable** la version distribuée avec des patches de sécurité ou des corrections de bugs importants.  
Suivi via **syspatch**
- current** la version en cours de développement, compilée à partir des sources de CVS à une date donnée.  
distribution pré-compilée : snapshots.  
*sysupgrade(8)* pour mise à jour non interactive à partir des snapshots.

# Mise à jour de sécurité (-stable)

## Commande `syspatch(8)`

- recherche les patches disponibles
- télécharge les derniers patches pas encore appliqués
- applique les patches
- suggère de rebooter si nécessaire
- permet de revenir en arrière en cas de problème (option `-r`)

# Mise à jour majeure à partir d'une distribution binaire

Commande **sysupgrade(8)** :

- télécharge la mise à jour dans `/home/_sysupgrade/`
- reboote sur `bsd.rd`
- exécute la mise à jour en mode automatique
- redémarre
- exécute automatiquement **sysmerge(8)**

Procédure simple et rapide...

# Mise à jour des fichiers de configuration

- Fusionner les modifs locales dans la configuration d'une nouvelle version
- L'upgrade d'OpenBSD ne touche pas les fichiers de configuration (n'installe pas le nouveau etc).
- **sysmerge(8)**
  - compare les fichiers du nouvel etc avec les fichiers installés,
  - montre les différences
  - permet de faire la fusion manuellement

Utilisation : **sysmerge**

# Mise à jour des paquets

- 3 branches comme pour le système de base
- paquets binaires pour -stable (uniquement la plus récente)

Mise à jour automatique des paquets binaires :

- `pkg_add -u`
- pose quelques questions en cas de conflit ou d'ambiguïté

Mise à jour d'un paquet par les sources :

- 1 `cd /usr/ports/....`
- 2 `make update`
- 3 `make clean-depends`

Ou mieux : utiliser `dpb(1)` distributed package build

# Agenda

- 1 Introduction
- 2 Installation du système
- 3 Concepts réseau OpenBSD
- 4 PF - Introduction
- 5 Configuration des pare-feux
- 6 PF - tables
- 7 Administration système
- 8 Mises à jour
- 9 Conclusion**

# Conclusion

Nous avons vu :

- l'installation du système,
- la configuration redondante avec CARP et pfsync
- l'utilisation d'un domaine de routage séparé pour l'administration
- les règles de filtrage avec PF
- l'administration quotidienne du pare-feu
- des outils pour l'accès aux données du pare-feu (traces, statistiques)
- les procédures de mise à jour

Après ce tuto, vous pourrez étendre cette configuration pour fournir d'autres services (serveur DHCP, cache DNS récursif, passerelle VPN, proxy web, ...).

Questions ?



## Annexes

## Plusieurs types de définitions :

**macros** remplacements textuels

**tables** structures efficaces pour des listes d'adresses

**options** configuration du comportement de PF

**files d'attente** contrôle du débit et des priorités entre flux

**règles de filtrage** bloque ou laisse passer les paquets, avec réécriture si nécessaire.

**points d'ancrage** pour l'ajout dynamique de règles

## Règles syntaxiques :

- Les lignes blanches sont ignorées
- Les lignes commençant par # sont des commentaires
- Utiliser \ pour continuer sur plusieurs lignes

Une liste permet de spécifier plusieurs règles similaires.  
Définie par des éléments entre { }

```
block in on vr0 from { 192.168.0.1, 10.5.32.6 } to any
```

est expansée en :

```
block in on vr0 from 192.168.0.1 to any  
block in on vr0 from 10.5.32.6 to any
```

Variables définies par l'utilisateur.

Permettent de réduire la complexité et d'augmenter la lisibilité.

Pas expansées entre "..."

```
extif = "fxp0"
```

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

Peuvent contenir des listes :

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

```
host1 = "192.168.1.1"
```

```
host2 = "192.168.1.2"
```

```
my_hosts = "{" $host1 $host2 "}"
```

# Tables

- stocker de façon efficace et compacte des listes d'adresses IP
- initialisées dans `pf.conf` ou un fichier dédié
- modifiables dynamiquement avec `pfctl`
- utilisable presque partout où pf attend une adresse.
- mot clé `table`

## Exemple :

```
table <rfc1918> const { 192.168.0.0/16, \  
                      172.16.0.0/12, 10.0.0/8}  
table <spammers> persist file "/etc/pf/spammers"  
  
block from <spammers> to any port { smtp, smtps, submission }  
block to <rfc1918>
```

# Tables - manipulations

```
pfctl -t table -T opération [-f fichier]
```

*table* nom de la table à manipuler

*opération* commande à appliquer

**add** *adresse* ajoute une adresse

**delete** *adresse* supprime une adresse

**load** recharge la définition de la table

**zero** vide la table

**expire** *secondes* supprime les entrées sans trafic depuis plus que *secondes*

**replace** remplace le contenu de la table à partir d'un fichier

**show** liste le contenu de la table

## Négations - attention danger !

! **adresse** matche toutes les adresses sauf celle-ci.

Comportement non intuitif dans une liste !

```
block from { 192.168.2.0/24, !192.168.2.1 }
```

devient :

```
block from 192.168.2.0/24  
block from !192.168.2.1
```

→ bloque tout !

Mais avec une table :

```
table <badguys> { 192.168.2.0/24, !192.168.2.1 }  
block from <badguys>
```

Syntaxe **simplifiée** :

```
block|match|pass [in|out] [log] [quick] \  
  [on interface] \  
  [inet|inet6] [protocole proto] \  
  [from adr_src] [port port_src] \  
  [to adr_dest] [port port_dst] \  
  [flags tcp_flags] [paramètres]
```



**block** bloque le paquet.

Paramètre global *block-policy* :

drop	paquet ignoré en silence
return	répond RST pour TCP et ICMP Unreachable pour les autres
return-icmp	répond tjs ICMP Unreachable

**match** la règle est évaluée sans changer la décision,  
par ex. pour **scrub** ou NAT.

**pass** le paquet passe avec création d'état sauf si **no state** est spécifié

**In/Out** permet de spécifier le sens (entrant ou sortant). Par défaut la règle s'applique dans les 2 sens.

**log** permet de tracer le paquet via l'interface `pfllog0`.

**quick** permet d'arrêter l'exploration des règles pour ce paquet si il matche la règle. (Rappel : c'est la *dernière* règle matchant un paquet qui s'applique).

**on interface** spécifie l'interface concernée.

Par défaut la règle s'applique à toutes les interfaces actives

## Famille d'adresse et protocole

- La famille d'adresse peut limiter aux paquets IPv4 (**inet**) ou IPv6 (**inet6**) seuls. Par défaut les 2 versions du protocole sont considérées.
- Le protocole limite la règle au protocole (défini dans le RFC 1340 et dans `/etc/protocols`)

La suite de la règle dépend du protocole.

`pfctl` génère une erreur si incohérence entre protocole et le reste.

## Adresses source et destination

- une adresse IPv4 ou IPv6 seule
- un bloc d'adresses en notation CIDR
- un nom résolu dans DNS, remplacé par toutes les adresses contenues dans la réponse
- le nom d'une interface remplacé par ses adresses.

Modificateurs :

:network adresse du réseau

:peer le correspondant d'une liaison point-à-point

:0 l'adresse principale seulement (sans aliases)

- une table
- une liste d'adresses ci-dessus
- une adresse précédée de ! (négation)
- le mot clé **any**

# Numéros de ports (TCP & UDP)

- un nombre entre 1 et 65535
- un nom de `/etc/services`
- un ensemble de ports défini par une liste
- une expression définissant un ensemble:
  - != différent
  - <, >, <=, >= inférieur, supérieur (resp ou égal)
  - >< entre les deux (exclusif)
  - : entre les deux (inclusif)
  - <> inférieur au premier ou supérieur au second

## Exemples :

port {80, 443}

port >= 1024

port 6881:6889

Sous la forme *test/masque* ou bien **any**

Matche si les drapeaux de *test* parmi *masque* sont positionnés

**Abréviations :** (F)IN, (S)YN, (R)ST, (P)USH, (A)CK, (U)RG, (E)CE, C(W)R

**Exemples :**

**flags S/S** le flag SYN est actif. Les autres sont ignorés

**flags S/SA** le flag SYN est actif, mais pas ACK. Les autres sont ignorés

**flags /SFRA** aucun des flags SYN, FIN, RST ou ACK n'est positionné.

Par défaut un état est créé pour tous les paquets TCP qui passent avec les flags S/SA.

**no state** permet de ne pas créer d'état.

**flags any** permet de créer un état avec n'importe quel paquet, pour « attraper » une connexion existante.

(Rappel : la table des états est consultée avant les règles et si un paquet match un état existant il passe directement)