PF

TP 1

Sommaire

- Présentation de PF
- Prise en main
  - Syntaxe
    - Les variables
    - Les listes
    - Les macros
    - Les tables
  - Filtrage simple
  - NAT
  - Redirection
- Utilisation avancée
  - La normalisation de paquets
  - Haute disponibilité avec CARP et pfsync

Présentation de PF

- PF est un outil de filtrage de paquets initialement développé pour le système d'exploitation OpenBSD.
  - Récemment adopté par le système FreeBSD
    - Ce cours se fonde sur PF sous FreeBSD
- Apporte toutes les fonctionnalités d'un pare-feu moderne à un système OpenBSD / FreeBSD
  - Filtrage à état (stateful inspection)
  - Translation d'adresses (NAT)
  - Redirection de trafic
  - Normalisation et gestion de priorités de trafic
  - Haute disponibilité

Utilisation de base

- Installation
  - PF est intégré au noyau OpenBSD depuis la version 3.0
  - PF est également intégré au noyau FreeBSD depuis la version 5.3
- Démarrage
  - pf=YES dans le fichier /etc/rc.conf (OpenBSD)
  - pf_enable=yes dans le fichier /etc/rc.conf (FreeBSD)
- Configuration
  - Les règles de filtrage et tous les paramètres utilisés par PF sont contenus dans le fichier (texte) /etc/pf.conf.

Démarrage automatique

- Le fichier rc.conf contient les options et commandes lancées au démarrage de la machine
  - Obligatoire : pf_enable="YES"
  - Optionnels
    - pf_rules="/etc/pf.conf" : emplacement du fichier de configuration qui contient les règles
    - pflog_enable="YES" : activation de la journalisation
    - pflog_logfile="/var/log/pflog" : emplacement du fichier journal
    - pf_flags et pflog_flags : paramètres additionnels passés respectivement à PF et pflog.
  - gateway_enable="YES" : obligatoire si la machine est un pare-feu ou si l'on souhaite utiliser la translation d'adresses (NAT)
    - net.inet.ip.forwarding=1 (OpenBSD)

Démarrage manuel

- pfctl
  - Outil de contrôle de PF depuis la ligne de commande
  - Plusieurs utilisations :
    - pfctl –e : active PF (s'il n'est pas activé au démarrage)
    - pfctl –d : désactive PF
    - pfctl –v –nf /path/to/file : parcoure le fichier de règles (test)
    - pfctl –f /path/to./file : charge les règles contenues dans le fichier file.
    - pfctl –sr : affiche les règles actives
    - pfctl –sn : affiche les règles de NAT
    - pfctl –sa: affiche toutes les informations disponibles
    - Note :pfctl est une commande qui ne peut être lancée que par le super utilisateur root.

Syntaxe (1)

- Une règle par ligne
- Une règle décrit les critères qui conditionnent une action effectuée sur un paquet par PF
  - Protocole utilisé, adresses source et destination, ports, etc.
- Syntaxe (simplifiée) :
  - *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*]
  - Quand une option n'est pas précisée, la règle s'applique à toutes ses valeurs :
    - block all est équivalent à
      - block in all
      - block out all

Interprétation des règles (1)

- Les règles sont stockées dans un fichier texte (par défaut /etc/pf.conf).
- Elles sont lues du début à la fin de ce fichier (de bas en haut)
- La règle qui s'applique à un paquet est la dernière du fichier qui en décrit les critères.
  - block in all
    pass in on ne0 proto tcp from any to any port 80
  - Exception : quick
  - Une règle du fichier doit définir la politique par défaut.

Interprétation des règles (2)

Paquet

Protocol: tcp
IP src : 192.168.1.2
IP dst : 10.0.0.2
Port src : 80
Port dst : 1772

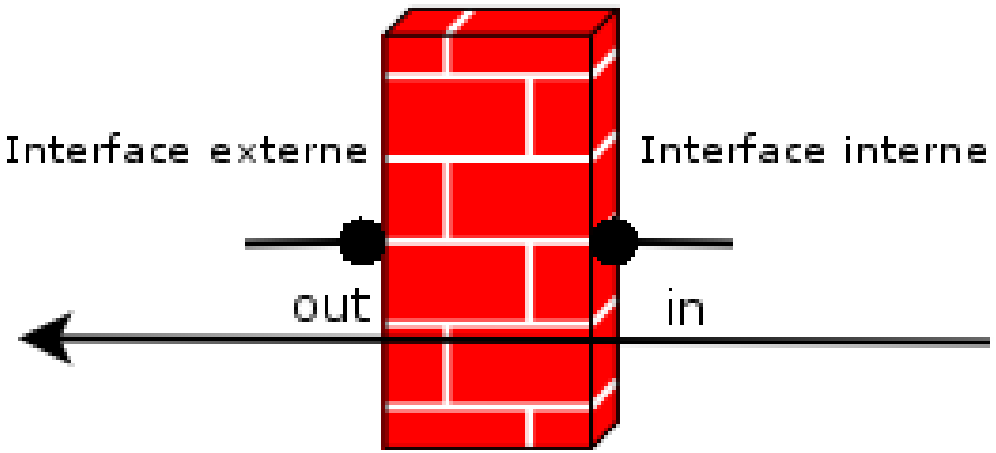
Règles :
1. block all
2. pass in on eth0 inet proto tcp from any port 80 to any port > 1024
3. pass in on eth0 inet proto tcp from any port 22 to any port > 1024
4. pass in on eth0 inet from any to any

Les règles 1, 2 et 4 sont applicables au paquet
mais c'est la règle n°4 qui sera appliquée (pass)

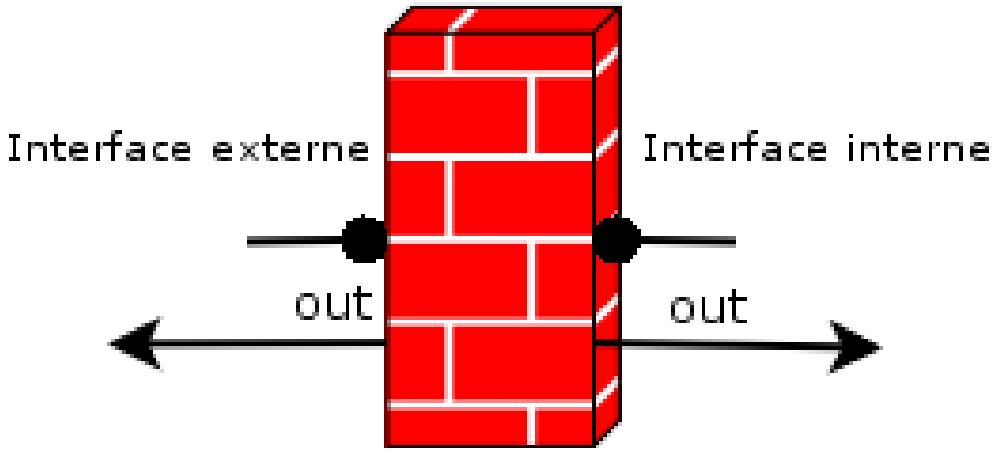
Syntaxe (2)

- action : quelle action doit être effectuée si un paquet remplit les critères de la règle ? (block, pass)
- direction : sens du trafic (in, out)
- interface : nom de l'interface réseau concernée
- af : version du protocole IP (IPv4 ou IPv6)
- From / to : adresses et ports source et destination
- Exemple :
  - pass in on dc0 from 192.168.0.2 port 80 to 192.168.0.1

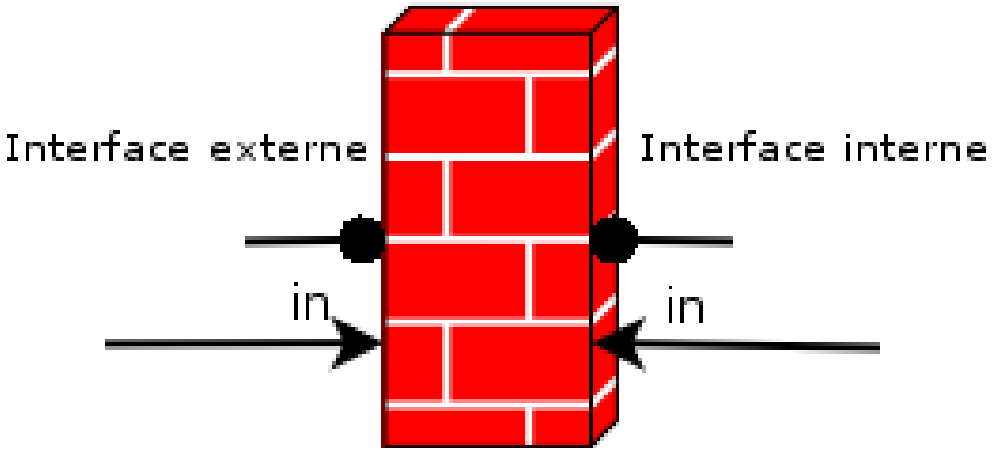
Note sur la direction (1)



Note sur la direction (2)



Note sur la direction (3)



Note sur l'action block

- Le mot-clef block peut être associé à une action complémentaire :
  - Employé seul, block agit comme un « trou noir ». L'expéditeur du paquet bloqué n'est pas averti.
  - Dans la plupart des cas, c'est une bonne décision de ne pas prévenir, mais dans d'autres, il faut le faire.
    - return-icmp : renvoie un message ICMP Unreachable
    - return-rst : renvoie un paquet TCP Reset
    - return
    - set block-policy return

Le mot clef quick

- Par défaut, un paquet est comparé à toutes les règles contenues dans le fichier pf.conf.
- Il est cependant possible de créer des exceptions en utilisant le mot-clef quick dans une règle.
- Si un paquet correspond à une règle qui comporte ce mot-clef, les règles suivantes ne sont pas utilisées.
- Exemple :
  - block in on fxp0 proto tcp from any to any port ssh
    pass in all
  - block in **quick** on fxp0 proto tcp from any to any port ssh
    pass in all

Options de journalisation

- Il est possible de journaliser les paquets qui correspondent à une règle.
  - log et log-all
    - block in log-all on ne0 proto udp port 1434 from any to any
  - Les paquets capturés sont stockés sous forme binaire.
  - Ils peuvent être lus par des outils comme tcpdump, ethereal, pflog.

Syntaxe (3)

- protocol
  - tcp, udp, icmp,icmp6
  - Nom d'un protocole contenu dans /etc/protocols
  - Numéro de protocole (0 à 255)
  - Liste de protocoles
- Adresses IP
  - Adresse IP (v4 ou v6) littérale : 192.168.1.1
  - Bloc CIDR , masque réseau : 192.168.1.0/24
  - Interface : sera alors égal à l'adresse IP de l'interface (PPP)
    - Le nom de l'interface doit être mis entre parenthèses
      - ($ext_if) ou (eth0)
  - Mot-clef any

Syntaxe (4)

- Ports
  - Numéro de port (1 à 65535)
  - Nom d'un service contenu dans /etc/services
  - Une liste de ports
  - Un range de ports
    - Opérateurs
      - != : différent de
      - >, >= : strictement supérieur, supérieur ou égal à
      - <, <= : strictement inférieur, inférieur ou égal à
      - <> : range de ports (1<>1024)
      - >< : range inverse

Syntaxe (5)

- Une règle peut aussi inclure des critères tels que la valeur des drapeaux TCP :
  - **F** : FIN - Finish; end of session
  - **S** : SYN - Synchronize; indicates request to start session
  - **R** : RST - Reset; drop a connection
  - **P** : PUSH - Push; packet is sent immediately
  - **A** : ACK - Acknowledgement
  - **U** : URG - Urgent
  - **E** : ECE - Explicit Congestion Notification Echo
  - **W** : CWR - Congestion Window Reduced
  - Syntaxe : flags check/mask
    - check : valeurs que l'on recherche (bit positionné)
    - mask : bits inspectés
  - Exemple : pass in proto tcp from any to any port ssh flags S/SA
    - Le flag S(yn) doit être positionné et seuls les flags S(yn) et A(ck) sont inspectés.

Syntaxe (6)

- Macros
  - ext_if = "fxp0"
  - int_if = "dc0"
  - lan_net = "192.168.0.0/24"
- Listes
  - block out on fxp0 from { 192.168.0.1, 10.5.32.6 } to any
  - friends = "{ 192.168.1.1, 10.0.2.5, 192.168.43.53 } »
  - host1 = « 192.168.1.1 »
host2 = « 192.168.2.3 »
my_hosts = « {« $host1, $host2 « } »
pass in on fxp1 from $my_hosts to any

Syntaxe (6)

- Tables
  - Elles sont utilisées pour stocker des groupes d'adresses de manière plus efficace et performante que les listes.
- Syntaxe
  - table <nom> type [{ addr, addr} | file « /path/to/file »]
  - Exemples :
    - table <my_hosts> const {192.168.1.1,192.168.1.2}
    - table <servers> persist file « /etc/servers »
  - Type (optionnel)
    - const : le contenu de la table ne peut pas être modifié.
    - persist : la table est chargée même si aucune règle ne l'utilise.
  - Utilisation dans une règle
    - block in on ne0 from any to <my_hosts> port 25

Antispoofing (1)

- Spoofing : usurpation d'adresse IP.
  - Cas d'un paquet dont l'en-tête IP a été manipulée et dont l'adresse IP de la source n'est pas celle du véritable expéditeur.
  - Technique couramment utilisée pour les opérations de reconnaissance (scan) ou dans les attaques en déni de service.
- Mot clef antispoof
  - antispoof [log] [quick] for *interface* [*af*]
  - antispoof for fxp0 inet

Antispoofing (2)

- Ne pas confondre antispoofing au niveau de l'interface et antispoofing RFC1918
  - RFC 1918 : définit les classes d'adresses IP privées (théoriquement) non routées sur Internet
- Antispoofing RFC1918
  - Définition d'une table contenant les adresses et masques définis dans la RFC 1918
    - table <rfc1918> const { 192.168.0.0/16, 172.16.0.0/12, 10.0.0.0/8 }
  - Ajout d'une règle
    - block in on eth0 from <rfc1918> to any
    - Cette règle ne doit être appliquée que sur l'interface externe (WAN) du pare feu.

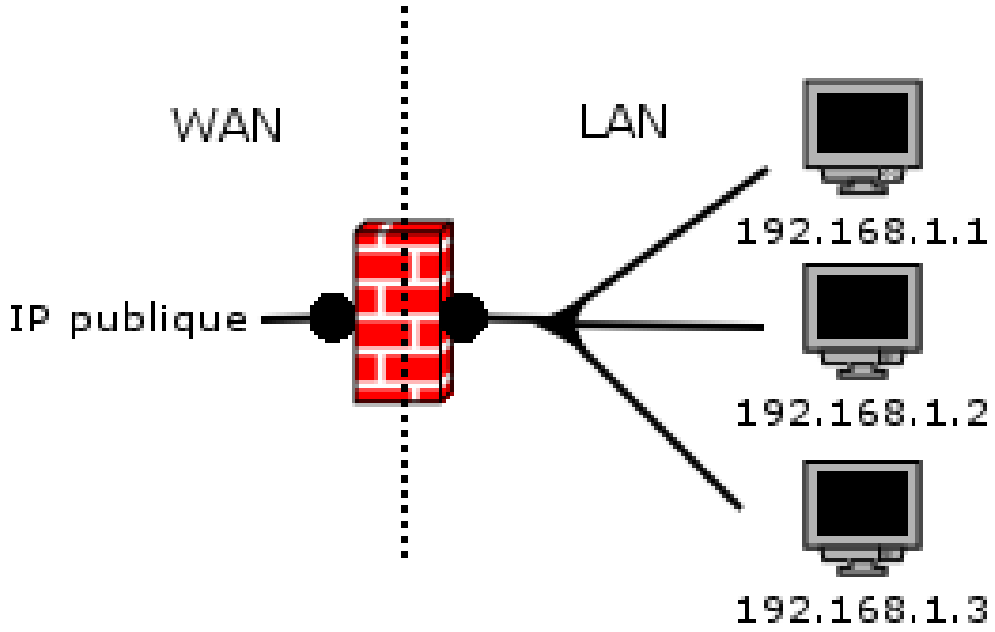
Translation d'adresses (1)

- Action qui consiste à modifier dynamiquement des adresses (et ports) source et destination.
  - Souvent appelée masquerading sous Linux
- Trois types de règles :
  - nat : traduit une adresse ou une groupe d'adresses internes en une adresse externe
  - rdr : opère une redirection d'adresse et de port
  - binat : traduction bidirectionnelle d'une adresse interne en une adresse externe.
- La première règle de type NAT s'applique.

Translation d'adresses (2)

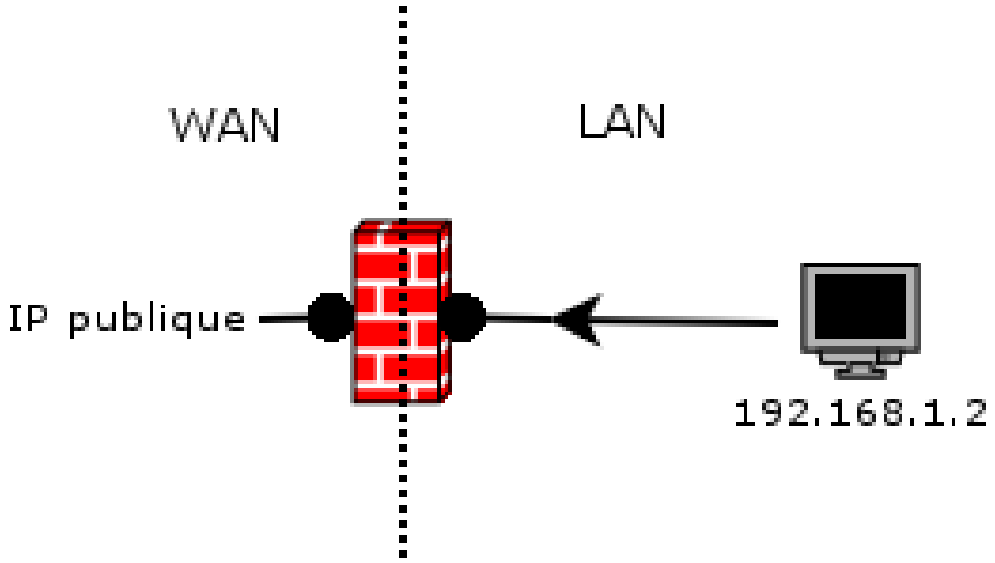
- Règle nat
  - nat [pass] on *interface* [*af*] from *src_addr* [port *src_port*] to \ *dst_addr* [port *dst_port*] -> *ext_addr* [*pool_type*] [static-port]
  - pool_type : désigne une liste d'adresses utilisées par la règle
    - nat on $ext_if inet from any to any -> { 192.0.2.5, 192.0.2.10 }
  - static-port : ne pas modifier les ports TCP ni UDP d'origine
- Règle binat
  - Même syntaxe.
    - binat on tl0 from $web_serv_int to any -> $web_serv_ext
- Mot clef no : désactive la translation d'adresses
- Visualisation des translations actives : pfctl -s state

Translation d’adresses (3)



nat : les 3 IPs LAN seront translatées
en 1 seule IP publique

Translation d’adresses (4)



binat : l'IP LAN sera translatée en 1 seule IP publique ce qui permet un trafic entrant.
La NAT de ce type évite d'avoir à router du trafic Internet jusque dans la DMZ.

Redirection

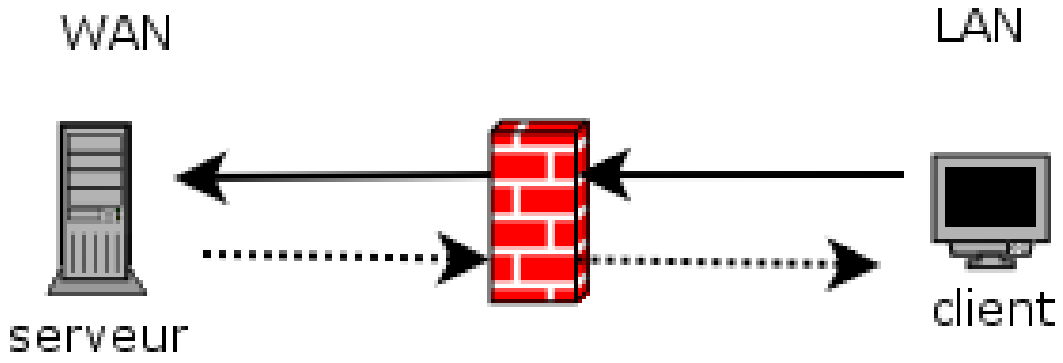
- C’est une forme particulière de NAT qui permet de modifier dynamiquement non seulement les adresses source et destination, mais aussi les ports.
- Utile pour forcer le passage d’un type de flux par un proxy
  - Exemple le plus courant : redirection des flux HTTP vers un proxy Squid
  - rdr on tl0 proto tcp from any to any port 80 -> $squid port 8080

Stateful inspection (1)

- Permet de spécifier l'état d'un paquet dans une session.
- Facilite l'écriture des règles, augmente leur pertinence et améliore les performances du pare feu : si un paquet appartient à une session autorisée établie, il est transmis sans passage par les règles.
  - PF utilise une table des sessions en cours. Le contenu de cette table est utilisé plutôt que les règles pour déterminer l'action (block ou pass) qu'il faut appliquer à un paquet.

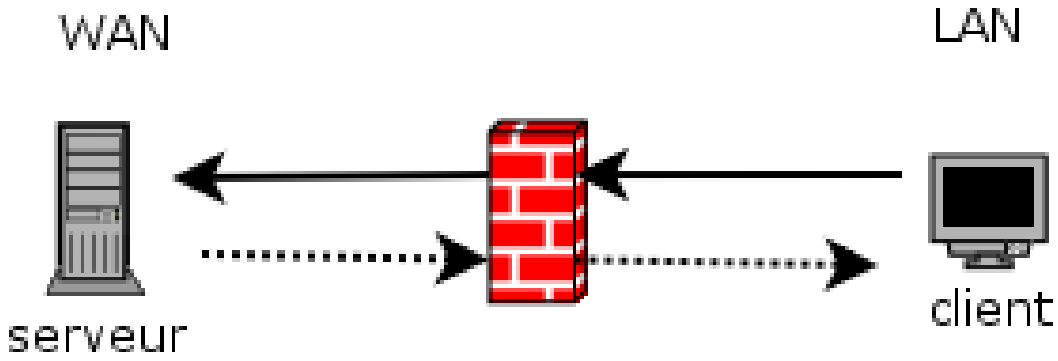
Stateful inspection (2)

Exemple sans stateful inspection :



Règle autorisant l'établissement d'un connexion HTTP du LAN -> WAN
1. pass proto tcp from <lan> port > 1024 flags S/SA to any port 80
Règle autorisant les autres paquets de la connexion
2. pass proto tcp from <lan> port > 1024 to any port 80
Règle autorisant les flux retour
3. pass proto tcp from any port 80 to <lan> port > 1024
Règle interdisant les ouvertures ilicites
4. block protoc tcp from any port 80 flags S/SA to <lan>

Stateful inspection (3)



Règle autorisant l'établissement d'un connexion HTTP du LAN -> WAN
1. pass proto tcp from <lan> port > 1024 flags S/SA to any port 80 keep state
Et c'est tout !
PF maintient une table des sessions ouvertes et autorisera les flux retour correspondants.
Avantages :
- fichier de règles plus léger
- meilleure performance du filtrage (PF parcourt une table et non toutes les règles)
- la règle de retour sera mieux renseignée puisqu'elle prendra en compte les adresses IP du serveur et du client.

Stateful inspection (4)

- Mots clefs keep state et modulate state
  - keep state
    - pass out on fxp0 proto tcp from any to any keep state
    - Le trafic TCP sortant est autorisé ainsi que toutes les réponses (trafic entrant) aux paquets émis depuis l'intérieur.
  - modulate state
    - Identique à keep state mais ne s'applique qu'au trafic TCP.
    - Modifie les ISN pour les rendre aléatoires
      - Permet de contrer certaines attaques basées sur le caractère trop prédictible des ISNs.

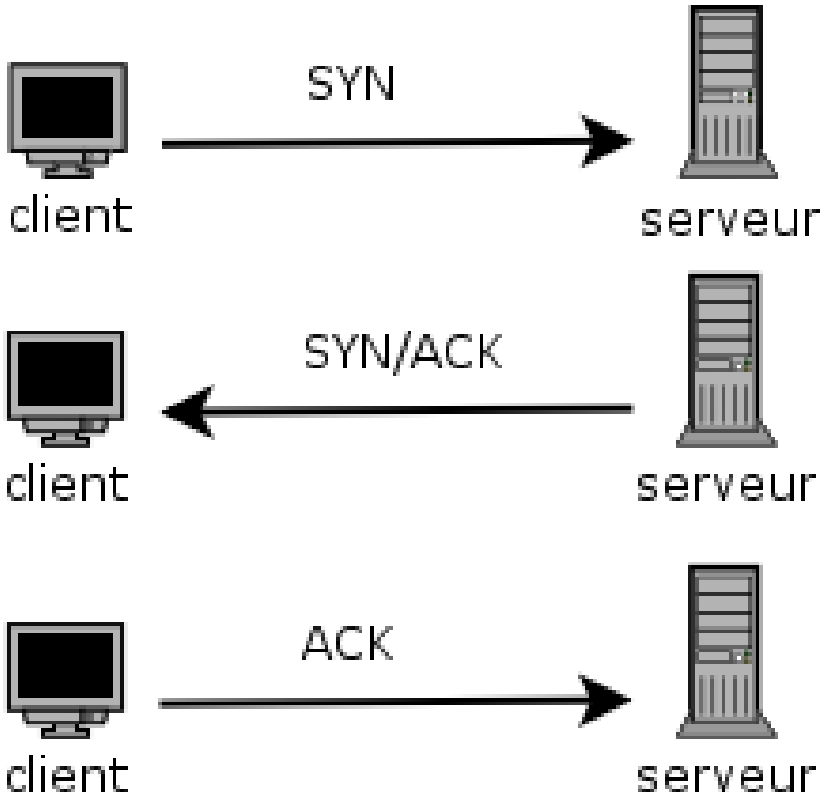
Stateful inspection (5)

- Cas du protocole UDP
  - Protocole sans connexion, pas de notion de début ni de fin de session contrairement à TCP.
  - PF maintient une table pour les paquets UDP. Cette table contient l'historique des sessions UDP en cours.
  - PF utilise aussi un timeout : dès qu'une session UDP référencée dans la table atteint ce timeout, la session est considérée comme terminée.
- La même méthode est utilisée pour le protocole ICMP.

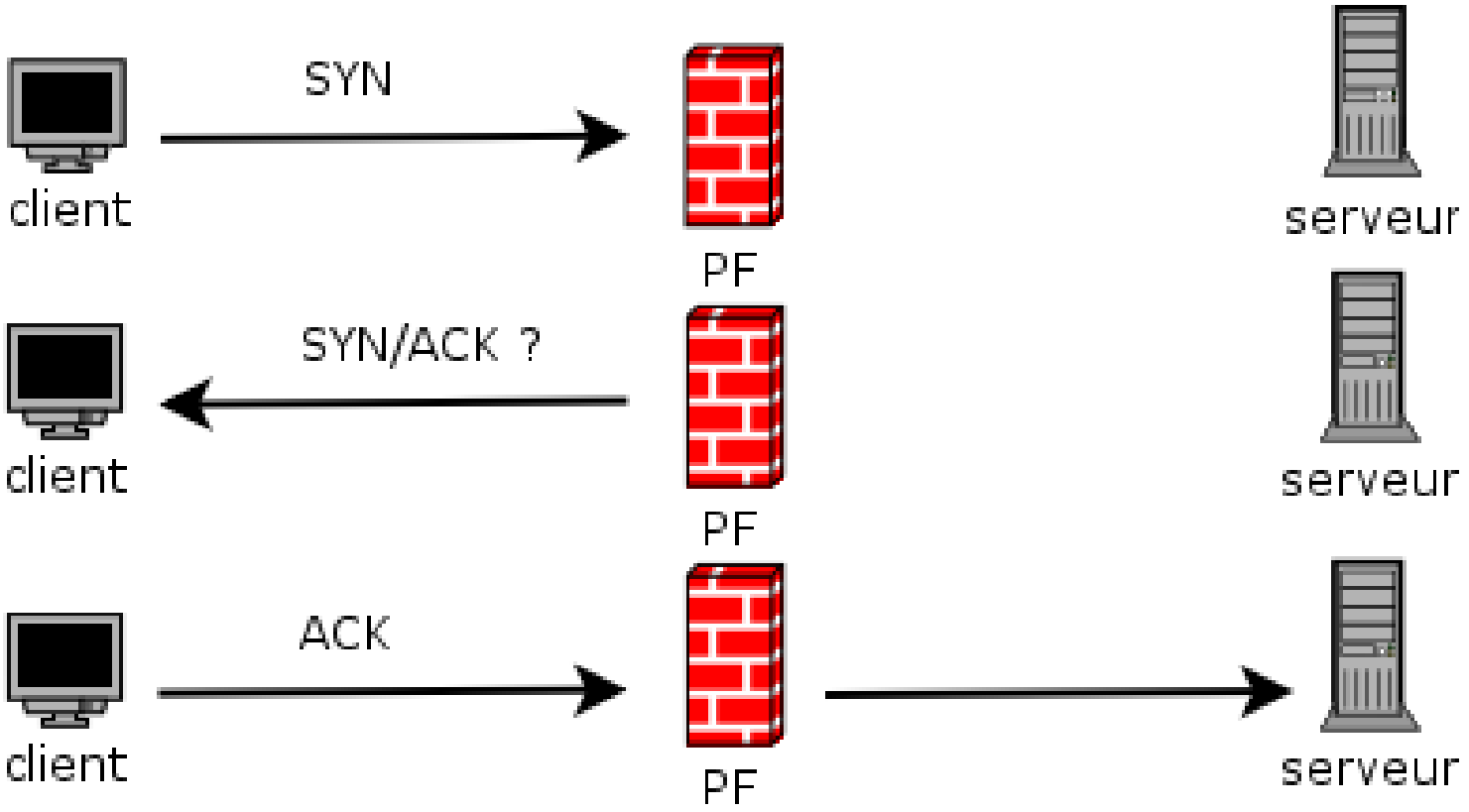
TCP Syn Proxy (1)

- PF propose une fonction de proxyfication des ouvertures de sessions TCP.
- Le pare feu s'assure que le handshake TCP est bien terminé par le client avant de transmettre le flux au serveur.
- Protection contre les DoS de type Syn Flood
- Mot clef synproxy state
  - pass in on $ext_if proto tcp from any to $web_server port www \ flags S/SA synproxy state

TCP Syn Proxy (2)



TCP Syn Proxy (3)

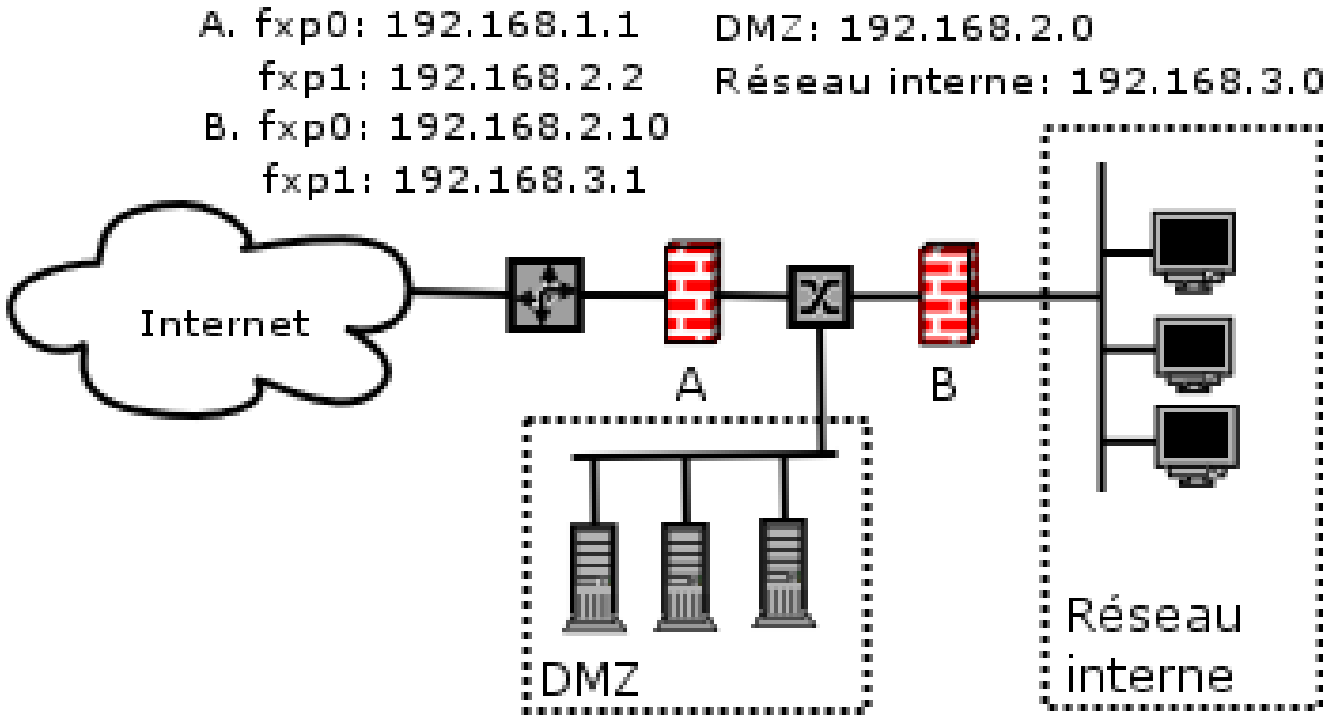


Note sur les DoS UDP

- Quand un paquet est émis en direction d'un port UDP fermé sur un serveur, ce dernier doit émettre un paquet ICMP Port Unreachable
- Dans le cas d'une attaque DoS par saturation, l'émission de ces paquets ICMP par le serveur attaqué va provoquer la saturation et donc le déni de service.
- Bloquer silencieusement (block/drop) les flux UDP inutiles au niveau du pare feu PF évite cela.

Exemple simple

Note: on suppose que les pare feux A et B sont tous deux des machines BSD/PF.



Description

- Pare feu A
  - Interface externe : fxp0
  - Interface interne : fxp1
  - Adresse externe : 192.168.1.1
  - Adresse interne : 192.168.2.1
- Pare feu B
  - Interface externe : ne0
  - Interface interne : ne1
  - Adresse externe : 192.168.2.10
  - Adresse interne : 192.168.3.1
- DMZ
  - Réseau 192.168.2.0/24
  - DNS : 192.168.2.20
  - Proxy : 192.168.2.21
  - Mail : 192.168.2.22
- Réseau interne
  - Réseau 192.168.3.0/24
  - NFS/SMB: 192.169.3.30
  - Backup : 192.169.3.31

Remarques

- Pour construire son fichier de règles :
  - Commencer par choisir quelle sera la politique de filtrage par défaut pour chaque direction (in et out) et la positionner dans le fichier pf.conf
  - Choisir une politique générale pour les règles drop et state
    - set block-policy
    - set state-policy
  - Identifier tous les paramètres qui peuvent être passés sous forme de variables.

Politique de sécurité

- Flux autorisés
  - Tous les flux depuis le réseau interne vers la DMZ
  - Tous les flux TCP depuis le réseau interne vers Internet.
  - Les flux SMTP vers le serveur de messagerie en DMZ
  - Les flux DNS vers le resolver en DMZ
  - Les flux HTTP et HTTPS vers Internet depuis le proxy en DMZ
- Flux interdits
  - Tous les flux entrants qui ne sont pas des réponses à des connexions ouvertes depuis la DMZ ou le réseau Interne
  - Les connexions directes HTTP et HTTPS depuis le réseau interne vers Internet
  - L'envoi de mail par SMTP depuis le réseau interne vers Internet
  - Les flux entrants d'Internet vers le réseau interne
  - Le trafic usurpé

Exemple de fichier de règles (1)

- # Interfaces
- ext_if="ne0"
- priv_if="ne1"
- dmz_if="ne"
- # Liste d'hôtes
- prv_hosts = "{192.168.1.1, 192.168.1.10, 10.3.1.5}"
- dmz_hosts = "{192.168.2.1/32, 192.168.2.2/32, 192.168.2.3}"
- dmz_www = "192.168.2.1/32"
- dmz_smtp = "192.168.2.2/32"
- dmz_dns = "192.168.2.3/32"
- # Options
- set block-policy drop

Exemple de fichier de règles (2)

- # Normalisation
- scrub in all
- scrub out all
- # NAT
- nat on $ext_if inet from $prv_hosts to any -> ($ext_if)
- nat on $ext_if inet from $dmz_hosts to any -> ($ext_if)
- rdr on $ext_if inet proto tcp from any to ($ext_if) \
- port 80 -> $dmz_www
- rdr on $ext_if inet proto tcp from any to ($ext_if) \
- port 25 -> $dmz_smtp
- rdr on $ext_if inet proto {tcp, udp} from any to ($ext_if) \
- port 53 (DNS) -> $dmz_dns

Exemple de fichier de règles (3)

- block in log all
- pass out quick on $ext_if from ($ext_if) to any flags S/SA \
- modulate state
- pass in quick on $prv_if from $prv_hosts to any flags S/SA
- pass in quick on $dmz_if from $dmz_hosts to any flags S/SA
- pass in on $ext_if from any to $dmz_www port 80 flags S/SA \
- synproxy state
- pass in on $ext_if from any to $dmz_smtp port 25 flags S/SA \
- synproxy state
- pass in on $ext_if from any to $dmz_dns port 53 flags S/SA \
- keep state
- antispoof for $ext_if
- antispoof for $prv_if

Utilisation avancée

- Normalisation de trafic
- Haute disponibilité avec CARP

Normalisation de trafic (1)

- Mot clef scrub
- Cette fonctionnalité permet de nettoyer le trafic :
  - Ré assembler les paquets fragmentés
    - Protection contre les attaques DoS fondées sur la fragmentation de paquet
  - Rejette tous les paquets mal formés ou anormaux
    - Exemple : flags TCP incompatibles
  - Comme pour les règles de type nat, c'est la première règle de type scrub qui s'applique.
  - scrub in all

Normalisation de trafic (2)

- Options de scrub
  - no-df : remet à zéro le bit DF (Don't Fragment) de l'en-tête IP. Dans certains cas, des paquets fragmentés sont émis avec ce bit positionné (NFS).
  - random-id : remplace le champ ID de l'en-tête IP par une valeur aléatoire. Ne s'applique qu'aux paquets sortants.
  - fragment reassemble : réassemble les paquets fragmentés avant de les envoyer aux filtres. Opération gourmande en mémoire.
  - fragment crop : élimine les fragments dupliqués et les recouvrements (overlapping). Plus performant que reassemble
  - fragment drop-ovl : élimine les fragments dupliqués ou les recouvrement ainsi que tous les fragments suivants de la session.

Haute dispo. avec CARP (1)

- CARP : Common Address Redundancy Protocol
  - Équivalent à HSRP (CISCO) et VRRP (Alteon Websystems)
  - Principe : partage d'une adresse IP par plusieurs équipements (ici des pare feux) d'un même groupe.
  - L'IP partagée est appelée adresse IP virtuelle (VIP).
  - Dans le groupe, un seul équipement (master) est actif et détient cette VIP. Les autres (backup) sont en marche mais ne reçoivent aucun trafic (standby).
  - L'hôte master envoie régulièrement des avis aux hôtes backup pour les informer de son état.
  - Quand l'hôte master n'envoie plus d'avis, il est considéré hors ligne (down) et un hôte backup prend sa place (up).
- Pfsync : outil de synchronisation des tables PF entre plusieurs pare feux.
- Haute disponibilité de pare feux : CARP + Pfsync

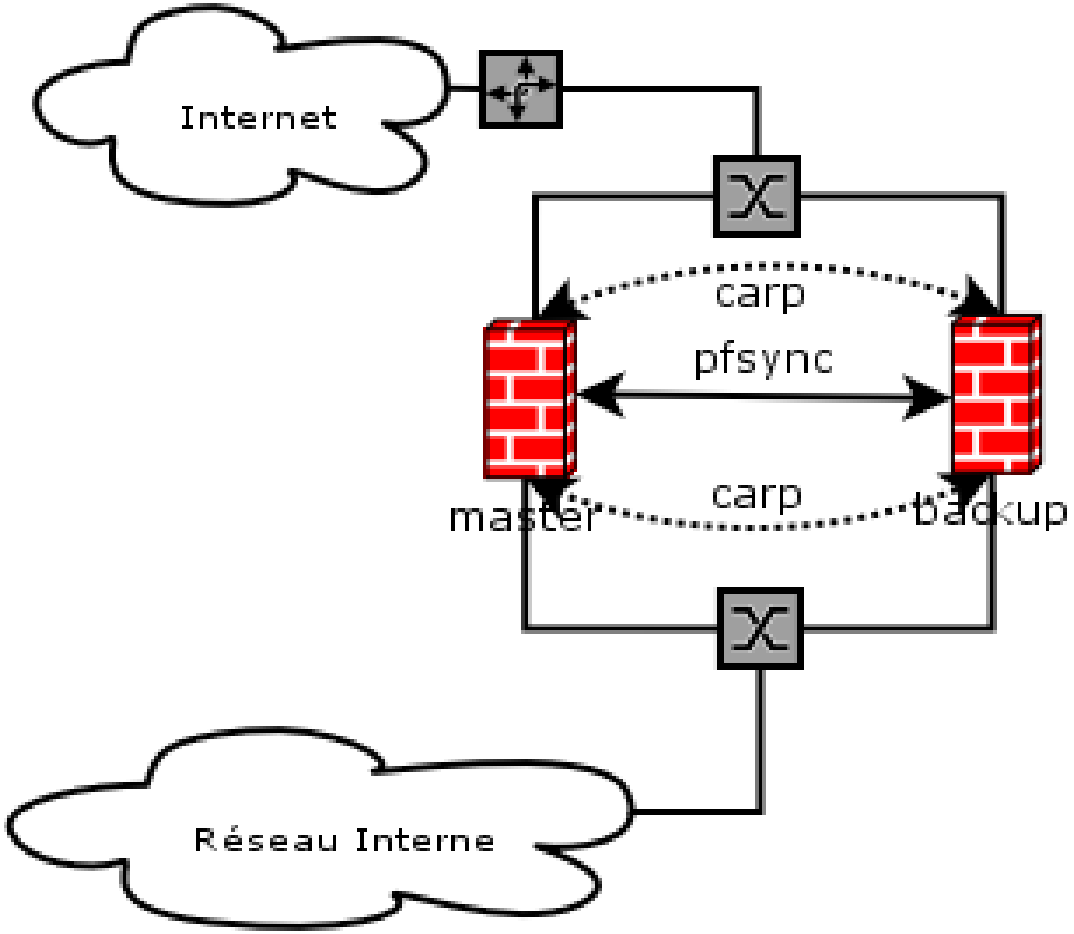
Haute dispo. avec CARP (2)

- Configuration de CARP
  - Elle se fait à l'aide de la commande ifconfig
  - Cette commande permet de créer une interface réseau virtuel qui sera utilisée pour le partage d'adresse virtuelle par les hôtes d'un même groupe.
  - ifconfig *carpN* create
    ifconfig *carpN* vhid *vhid* [pass *password*] [carpdev *carpdev*] \
    [advbase *advbase*] [state *state*] *ipaddress mask*
    - vhid : identifiant de l'hôte dans le groupe
    - advbase : temps entre les avis
    - state : état dans le groupe (master, backup)

Haute dispo. avec CARP (3)

- Pfsync
  - Synchronise les tables de session entre plusieurs pare feux PF.
    - Si on ne synchronise pas ces tables, il y a rupture de la fonction stateful inspection en cas de bascule.
      - Il n'y a pas arrêt du filtrage ; seules les sessions en cours sont remises à zéro (= coupées !).
      - Si on n'utilise pas la fonction stateful inspection, l'utilisation de pfsync est inutile et seul CARP suffit.
    - Les pare feux concernés sont généralement reliés par un câble direct et s'échange les informations par ce moyen.
    - Création d'un device virtuel
      - ifconfig *pfsyncN* syncdev *syncdev*

Haute dispo. avec CARP (4)



Haute dispo. avec CARP (5)

- Mise en œuvre sur le pare feu 1
  - ! enable preemption and group interface failover
    # sysctl -w net.inet.carp.preempt=1
    ! configure pfsync
    # ifconfig em1 10.10.10.1 netmask 255.255.255.0
    # ifconfig pfsync0 syncdev em1
    # ifconfig pfsync0 up
    ! configure CARP on the LAN side
    # ifconfig carp1 create
    # ifconfig carp1 vhid 1 carpdev em0 password lanpasswd \
    172.16.0.100 255.255.255.0
    ! configure CARP on the WAN/Internet side
    # ifconfig carp2 create
    # ifconfig carp2 vhid 2 carpdev em2 password netpasswd \
    192.0.2.100 255.255.255.0

Haute dispo. avec CARP (6)

- Mise en œuvre sur le pare feu 2
  - ! enable preemption and group interface failover
    # sysctl -w net.inet.carp.preempt=1
    ! configure pfsync
    # ifconfig em1 10.10.10.2 netmask 255.255.255.0
    # ifconfig pfsync0 syncdev em1
    # ifconfig pfsync0 up
    ! configure CARP on the LAN side
  - # ifconfig carp1 create
    # ifconfig carp1 vhid 1 carpdev em0 password lanpasswd \
    advskew 128 172.16.0.100 255.255.255.0
    ! configure CARP on the WAN/Internet side
    # ifconfig carp2 create
    # ifconfig carp2 vhid 2 carpdev em2 password netpasswd \
    advskew 128 192.0.2.100 255.255.255.0

Références

- FAQ PF
  - OpenBSD - http://www.openbsd.org/faq/pf/index.html
- FreeBSD
  - Projet FreeBSD - http://www.freebsd.org
- OpenBSD / CARP
  - Projet OpenBSD - http://www.openbsd.org