IPFilter

- ipfilter est un coupe feu à état fournissant aussi des fonctionnalités de traduction d'adresses
- ipfilter est en standard sous FreeBSD, NetBSD et Solaris 10.
- il a été testé sous : solaris 2.3-9,open BSD 2.0-3.5,IRIX,Linux 2.4 et 2.6,HP-UX 11,True 64 unix 5.1a,qnx 6 port.
- doc: ipf-howto: http://www.obfuscation.org/ipf/ipf-howto.html
- syntaxe claire et lisible :-)

IPFilter

- configuration via un (ou deux) fichier(s) de configuration (Ex. : /etc/ipf.conf, /etc/ipnat.conf)
  - une règle par ligne
  - # indique un commentaire
  - **la règle qui s'applique est la DERNIERE auquel le paquet correspond**
  - cas général au début du fichier, cas particuliers à la fin
  - « quick » => action immédiate
  - pas d'outil en ligne de commande pour ajouter/supprimer des régles
  - => générer le fichier de configuration avec des macros processeurs comme m4

IPFilter: chaîne FORWARD ?

- in: paquet arrivant sur l'une des interfaces du firewall, soit à destination du firewall, soit à router
- out: paquet sortant sur l'une des interfaces du firewall, soit provenant du firewall, soit un paquet qui vient d'être routé
- les paquets routés sont traités deux fois :
  - une fois en entrée et une fois en sortie
- bonne pratique: ne pas dupliquer les mêmes tests en entrée et en sortie
  - filtrer en entrée et tout autoriser en sortie
  - tout autoriser en entrée et filtrer en sortie

IPFILTER: IN/OUT/FORWARD ?

- distinguer les paquets de/vers le routeur des paquets routés
  - filtre général concernant les paquets routés
  - filtre particulier concernant les paquets à destination ou venant du routeur (via ip src ou dst)
- Exemple: autoriser les paquets routés provenant du sous-réseau 192.168.10.0/24 et refuser les connexions sur le firewall d'IP 192.168.10.249

```
block in all
pass out all
pass in on vr0 from 192.168.10.0/24 to any
block in on vr0 from any to 192.168.10.249
```

IPFilter: syntaxe de base

- Syntaxe (partielle) :
- action [direction] [log] [quick] [on interface] [proto protocol] [from src_addr [port src_port]] [to dst_addr [port dst_port]] [flag tcp_flags] [keep state]
- action: pass/block
- direction: in/out
- quick: provoque l'exécution immédiate de l'action
- flag: match/testés. ex. S/SA (on regarde Syn et Ack et on veut que seul Syn soit à 1)
- (src|dst)_addr: adresse ip avec notation CIDR ou any

Exemple:

```
 # loopback
pass out quick on lo0 from any to any
pass in quick on lo0 from any to any
#politique par défaut
block in all
block out all
# Enable all outgoing connections
pass out proto tcp from any to any flags S/SA keep state
pass out proto udp from any to any keep state
pass out proto icmp from any to any keep state
#ssh
pass in on vx0 proto tcp from 195.221.162.248 to any port
  = 22 keep state
#paquets avec des options IP
block in log from any to any with ipopts
# paquets trop courts pour avoir un header complet
block in log proto tcp from any to any with short
```

IPFilter: suivi de session

- mot clef keep state
- équivaut à une acceptation implicite de tous les paquets suivants d'une connexion
- Exemple:

```
block in quick on tun0 all
pass out quick on tun0 proto tcp from 20.20.20.1/32 to
  any keep state
```

- keep frags: accepter les fragments suivants

IPFilter: optimisation d'un jeu de règles

- groupes de règles
  - une première règle sert de point d'entrée au groupe
    - si son critère est validé, le groupe est utilisés
    - sinon, on passe à la suite
  - la première règle est repérée par le mot clef head suivi d'un numéro de groupe
  - les autres règles sont repérées par le mot clef group suivi du numéro de groupe

IPFilter: optimisation d'un jeu de règles

- block in quick on xl0 all head 1
- block in quick on xl0 from 192.168.0.0/16 to any group 1
- block in quick on xl0 from 172.16.0.0/12 to any group 1
- block in quick on xl0 from 10.0.0.0/8 to any group 1
- block in quick on xl0 from 127.0.0.0/8 to any group 1
- block in quick on xl0 from 0.0.0.0/8 to any group 1
- block in quick on xl0 from 169.254.0.0/16 to any group 1
- block in quick on xl0 from 192.0.2.0/24 to any group 1
- block in quick on xl0 from 204.152.64.0/23 to any group 1
- block in quick on xl0 from 224.0.0.0/3 to any group 1
- block in log quick on xl0 from 20.20.20.0/24 to any group 1
- block in log quick on xl0 from any to 20.20.20.0/32 group 1
- block in log quick on xl0 from any to 20.20.20.63/32 group 1
- block in log quick on xl0 from any to 20.20.20.64/32 group 1
- block in log quick on xl0 from any to 20.20.20.127/32 group 1
- block in log quick on xl0 from any to 20.20.20.128/32 group 1
- block in log quick on xl0 from any to 20.20.20.255/32 group 1
- pass in on xl0 all group 1

IPFilter: optimisation d'un jeu de règles

- pass out on xl0 all
- block out quick on xl1 all head 10
- pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 80 flags S keep state group 10
- pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 21 flags S keep state group 10
- pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 20 flags S keep state group 10
- pass out quick on xl1 proto tcp from any to 20.20.20.65/32 port = 53 flags S keep state group 10

Packet Filter

- packet filter existe sous OpenBSD, FreeBSD, NetBSD, DragonFlyBSD, ...
- créé en réaction à un changement de licence d'IPFilter
- coupe feu à état ipv4/ipv6
- performant (optimisation automatique des règles)
- in et out mais pas de forward (NetFilter)
- comme IpFilter, pf a peu de modules de suivi de connexion (ALG ou helpers)
- syntaxe proche de celle d'IPFILTER (Attention: sémantique parfois différente)

Packet Filter: différences avec IPFilter

- licence :-)
- facilités (macro, listes, tables)
- extensions:
  - point d'ancrages,
  - authentification (authpf)
  - normalisation de trafic
  - gestion de bande passante
  - haute disponibilité (CARP et pfsync)
  - balisage de paquet (TAG)

Packet Filter : syntaxe

- listes : liste d'éléments (protocoles, ports, adresses, ...) :
  - entre accolades, séparés par des virgules ou des espaces
  - pass out on fxp0 from { 192.168.196.1 192.168.196.246} to {192.168.10.1 192.168.10.2 }
  - est équivalent à
    - pass out on fxp0 from 192.168.196.1 to 192.168.10.2
    - pass out on fxp0 from 192.168.196.246 to 192.168.10.2
    - pass out on fxp0 from 192.168.196.1 to 192.168.10.1
    - pass out on fxp0 from 192.168.196.246 to 192.168.10.1

Packet Filter: syntaxe

- macros: variables utilisateurs
  - nom: commence par une lettre, contient lettres, chiffres, souligné (_). ne doivent pas être des noms réservés
  - définition: nom = valeur. Ex.:ext_if = "fxp0"
  - utilisation: $nom. Ex.:
  - block in on $ext_if from any to any
  - une macro peut représenter une liste
  - une macro peut utiliser la valeur d'autres macro:
  - host1 = "192.168.196.1"
  - host2 = "192.168.196.246"
  - les2 = { $host1 $host2 }

Packet Filter: tables

- tables:
  - regroupe des adresses (ipv4 ou ipv6).
  - s'utilise en général à la place d'une adresse
  - recherche plus rapide que dans une liste
  - deux modes de création : dans pf.conf ou via pfctl
  - les noms des tables sont entourés par < et par >
  - une table peut être modifiée par la suite via pfctl
  - attributs des tables :
    - const : la table ne peut être modifiée une fois créée
    - persist: ne pas détruire la table si aucune règle ne s'y réfère
  - spécification des adresses: ip, nom, interface, self (inclus l'adresse de bouclage)

PF - tables: exemples

- via pf.conf

```
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
```

- via pfctl
  - création/ajout: pfctl -t spammers -T add 218.70.0.0/16
  - lister : pfctl -t spammers -T show
  - suppression d'adresse: pfctl -t spammers -T delete 218.70.0.0/16

PF: balisage de paquets

- permet de marquer un paquet avec un identifiant interne (mot clef tag)
- l'identifiant pourra ensuite être utilisé dans les critères (mot clef tagged)
- permet de créer des paquets de confiance et évite de répéter des tests
- Exemple classique :
  - pass in on $int_if all tag INTERNAL_NET keep state
  - pass out on $ext_if all tagged INTERNAL_NET keep state

Points d'ancrage:

- jeux de règles secondaires (idem chaîne NetFilter) chargés/déchargés dynamiquement
- utilisable pour filtrage, NAT, redirection
- utilisé par authpf

Authpf

- permet de charger dynamiquement des règles lorsqu'un utilisateur s'authentifie
- chaque utilisateur peut possèder des règles spécifiques
- fonctionnement:
  - l'utilisateur se connecte via ssh sur le firewall où authpf est son shell de connexion défini dans /etc/passwd
  - des règles spécifiques sont chargées
  - à la fermeture de la session ssh, ces règles sont déchargées
  - ouverture et fermeture de session sont transmises à syslogd

Antispoofing/OS finger print

- antispoofing:
  - idem rp_filter noyau linux
  - compare ip src avec les réseaux des autres interfaces de l'hôtes
- sélection selon l'OS source
  - s'appuie sur les particularités des paquets tcp SYN pour reconnaître l'OS de la machine source du paquet
  - Exemple:

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

Pf: Partage de charge

- dans la prochaine version de ce document

Pf: suivi de session

- keep state : idem NetFilter. Exemple:
  - pass in proto tcp from 192.168.196.0/24 to any port 80 flags S/SA keep state
- modulate state: idem mais pf modifiera les « ISN » de façon à être réellement aléatoires
- synproxy state: c'est pf qui gère l'ouverture de session tcp
  - il ne passe les paquets à la destination interne qu'une fois le 3way handshake réalisé
  - objectif: protéger un serveur interne d'un syn flood en laissant le fw la gérer (s'il peut :-))
  - principe: le firewall est plus résistant qu'un serveur interne

Pf: suivi de session

- pf fourni un ensemble d'options permettant de contrôler/réagir au suivi de session
- Exemple: interdire l'accès à un hôte qui dépasse un certain nombre de connexions par seconde
  - `block quick from <evil_hosts>`
  - `pass in proto tcp from any to $sshserver poirt 22 flags S/SA kepp state (max-src-conn-rate 100/10, overload <evil_hosts> flush)`
  - tout hôte tentant plus de 100 connexion ssh en moins de 10 secondes :
    - se verra ajouté à la table evil_host
    - verra ses connexions concernées par cette régles effacées de la table des états
- Attention au denis de service !

Balisage des paquets

- marquer les paquets avec un identifiant qui peut ensuite servir comme critère de filtrage
- une seule balise à un instant donné par paquet
- évite de dupliquer des critères de filtrage
- Exemple:

```
pass in on $int_if proto tcp to port 80 tag webOut keep
  state
pass out in $ext_if tagged webOut keep state
```

- remarque : avec pf, keep state marche pour les paquets dans le même sens ou pour les paquets réponse. cela explique l'intérêt de la seconde ligne.

atténuer les attaques par dénis de service

- les attaques par saturation du lien réseau ne peuvent être combattues
- synproxy
- options suivi de session
  - max-src-conn-rate & Co (vu précédemment)
  - max-src-states: limite le nombre d'état par source
  - max-src-nodes: limite le nombre de sources (lutter contre l'usurpation d'ip source)
- adaptatives timeout
  - adapter les timetous au nombre total d'états
  - les états inutilisés meurent plus vite

atténuer les attaques par dénis de service

- ALTQ: gestion de la qualité de service
- gestion de la congestion de la queue d'entrée :
  - quand la queue d'entrée est pleine, le cpu devient surchargé, la machine en répond plus
  - solution (de situation de crise) :
    - les paquets correspondant à un état sont gérés
    - les autres sont détruits
  - avantage:
    - ces paquets auraient été mal gérés ou détruits vu l'état de la machine
    - la machine gère les connexions existantes

Pf: normalisation

- motivations
  - en sortie:
    - limiter l'information sur les OS, applications et architecture du réseau;
    - palier certains OS faibles (ex: W98 et No de séquence prévisibles).
  - en entrée: supprimer toute ambiguïté dans l'interprétation d'un paquet par le destinataire ou un NIDS intermédiaire :
    - protéger les OS d'attaques utilisant des paquets mal formés, la fragmentation et le recouvrement de fragments, ...
    - faciliter le travail des IDS

Pf: normalisation IP

- les paquets incorrects sont rejetés. Par exemple:
  - version IP incorrecte
  - champ longueur d'entête trop petit ou trop lont
  - checksum incorrect
- certains champs sont modifiés :
  - remise à zéro des options IP
  - ajustement du champ TTL
    - éviter les abus classique (ttl faible pour éviter une machine ids située plus loin par ex.)

Pf: normalisation UDP et TCP, exemples

- UDP:
  - rejet des paquets avec un mauvais checksum ou une taille différente de la taille IP
- ICMP: rejet
  - des message echo request avec une adresse destination diffusée (multicast, broadcast)
  - des messages dont le checksum est incorrect
- TCP:
  - correction les anomalies des drapeaux tcp
  - rejet des paquets à checksum incorrect

routeurs et redondances

- but: permettre à un routeur de secours de prendre le relais d'un routeur HS de façon transparente pour les hôtes du réseau et les connexions en cours
- Protocoles:
  - HSRP (cisco): Hot Standby Routing Protocol
  - VRRP: virtual router redundancy protocole (RFC 2338)
- impliquent:
  - des annonces entre routeurs actifs et de secours
  - une utilisation des adresses MAC et IP du routeur actif HS par le routeur de secours qui prend le relais

VRRP (rfc 2338): Virtual Router Redondancy Protocol

- permet la migration des informations niveau 2 (MAC) et 3 (IP) d'un routeur HS vers un autre OK. Les éléments utilisés :
  - Virtual router ID: tous les routeurs d'un groupe ont le même
  - master virtuel routeur: routeur actif (1 seul)
  - backup virtual routeur: les routeurs de secours
- fonctionnement (succinct) :
  - le routeur actif diffuse des annonces via multicast
  - en cas d'interruption de ces annonces, les autres routeurs élisent un nouveau routeur maître qui utilise les adresses MAC et IP du groupe

CARP (rfc 3040): Common Address Redundancy Protocol)

- pourquoi CARP ? HSRP et VRRP sont couverts par une licence cisco (qui la revendique) => non utilisables dans des produits libres
- différence entre CARP et VRRP/HSRP:
  - CARP est indépendant du type d'adresse IP (v4/v6)
  - arpbalance (ràf: sera traité dans la prochaine version de ce document)
  - les annonces sont sécurités (SHA-1 HMAC)
  - CARP est un protocole libre

CARP : protocole

- chaque groupe a :
  - une adresse MAC virtuelle
  - une ou plusieurs adresses IP virtuelles
- les membres du groupe répondent aux requêtes ARP de cette adresse MAC commune
- les annonces CARP ont cette adresse comme source
- Le maître d'une adresse envoie régulièrement des annonces via multicast en utilisant le protocole CARP (IP protocole 112)
- les membres du groupe écoutent ces annonces
  - interruption des annonces => l'un prend le relais

Coupe-feu et redondance

- La redondance suppose :
  - qu'un coupe feu prenne le relais en cas de panne du coupe feu actif
  - que le nouveau coupe feu se comporte comme l'ancien
    - adresse MAC et IP identiques: info statique, facile
    - configuration statique identique: info statique, facile
    - configuration dynamique identique: suppose communication en temps réel entre les coupes feux
- VRRP/HSRV & Co :
  - suffisant en cas de filtre de paquet sans état (pas de configuration dynamique)
  - insuffisance en cas de suivi de session: table des états

une solution: pfsync

- pfsync: protocole permettant le transfert des informations de suivi de connexion entre coupes-feux
-

une solution: psync