Pare Feu (*Firewall*)

"Building Internet Firewalls"
E. D. Zwicky, S. Cooper, D. Brent Chapman,
O'Reilly

Olivier ROUSSEL
roussel@iut-lens.univ-artois.fr

Pourquoi ?

- Les actes de piratage se sont largement multipliés sur l'Internet.
- Il n'est plus pensable aujourd'hui de laisser une machine sans protection, y compris les machines individuelles connectées à l'Internet de manière occasionnelle.

Que protège-t-on ?

- Ses données
  - Les secrets qu'elles contiennent
  - Leur intégrité
  - Leur disponibilité
- Ses ressources
  - CPU, espace disque
- Sa réputation
  - Une machine piratée peut servir de relai pour d'autres piratages

Différents types de protections

- Pas de protection (!!)
- Protection par le secret Un système que personne ne connaît n'est pas attaqué (mauvais plan)
- Protection machine par machine
  Verrouiller les machines et mettre à jour leur systèmes régulièrement (indispensable mais ingérable sur un large parc)
- Protection du réseau
  Contrôler l'accès au réseau en plaçant des filtres

Politiques de sécurité

- Tout autoriser sauf ce qui a été identifié comme dangereux Vision naïve ?
- Ne rien autoriser sauf ce qui a été identifié comme inoffensif Vision paranoïaque (trust no one) ?

Divers points sur la sécurité

- Limiter les droits accordés au strict nécessaire
- Avoir plusieurs lignes de défense et plusieurs systèmes de protection (DMZ,...)
- Limiter les points d'accès au système pour mieux les contrôler
- Il ne faut pas qu'une panne d'un composant ouvre le système aux pirates

Le firewall

- Un firewall est un système par lequel devra passer tout le trafic réseau et qui permettra de bloquer les trafics considérés comme indésirables
- Il existe différents types de firewall :
  - Filtrage de paquet
  - Systèmes mandataires (proxy)

Le filtrage de paquet

- Le firewall sert de routeur mais ne laisse passer que les paquets qui répondent aux critères décidés par l'administrateur.
- Pour prendre sa décision, il peut se baser sur
  - Les adresses IP (source et destination)
  - Les numéros de port (source et destination)
  - Le protocole (TCP, UDP, ICMP,...)
  - Les connexions précédentes (stateful firewall)
  - Les flags TCP (SYN,...)

Le filtrage de paquet

- Points forts :
  - Peu gourmand en ressources
  - Complètement transparent
- Points faibles :
  - Le filtrage de paquet n'a pas accès aux informations des couches supérieures. Il ne permet donc pas certains filtrages (nom d'utilisateur,...)
  - Certains protocoles posent problèmes (ftp,...)

Fonctionnement général d'un filtrage de paquet

- Le filtrage est décrit par une séquence de règles de la forme <condition,décision>.
- Quand un datagramme arrive sur une interface, le système parcourt les règles du firewall en séquence.
- Dès qu'une règle décrit une condition qui est vraie pour le datagramme reçu, le système applique la décision de cette règle

Décision pour un paquet

La décision que l'on peut prendre pour un paquet peut être

- laisser passer le datagramme (ACCEPT)
- rejeter le datagramme et renvoyer un message ICMP d'erreur (REJECT)
  permet à l'utilisateur de se rendre compte du filtrage mais fournit des informations précieuses pour un pirate !
- ignorer le datagramme (DROP)
  permet de ne fournir aucune information à un éventuel pirate mais un peu moins "user friendly"
- D'autres décisions sont possibles suivant l'implantation (passage à un programme utilisateur, redirection vers un port local, translation d'adresse,...)

Présentation générale d'un filtrage de paquet

Condition							Décision
Source		Destination					
IP/masque	port	IP/masque	port	interface	protocole	flags	ACCEPT/REJECT/DROP
192.168.0.0/16				eth2			REJECT

iptables

L'exemple de *iptables* sous Linux

- La commande *iptables* permet de manipuler les règles du filtrage de paquet
- Il existe différents ensembles de règles (chaînes)
  - INPUT : permet de laisser entrer un paquet (à destination d'un programme local)
  - OUTPUT : permet de laisser sortir un paquet (en provenance d'un programme local)
  - FORWARD : permet le routage du paquet (depuis une machine extérieure vers une autre machine)
  - L'utilisateur peut créer ses propres chaînes qui vont jouer le rôle de sous-programme pour effectuer une partie des vérifications nécessaires. Dans ce cas, la décision RETURN permet de terminer la vérification de la sous-chaîne et de revenir à la chaîne appelante.

Les commandes de base

- `iptables -F [`*NomChaine*`]`
  vider la chaîne nommée (effacer toutes ses règles) ou toutes les chaînes.
- `iptables -P` *NomChaine DécisionParDéfaut*
  indiquer ce que l'on doit faire d'un paquet lorsque toutes les règles ont été examinées et qu'aucune n'était applicable (règle par défaut). La décision doit être DROP ou ACCEPT.
- `iptables -A` *NomChaine Règle*
  ajouter une règle à la fin de la chaîne
- `iptables -I` *NomChaine Numéro Règle*
  ajouter une règle avant celle de numéro *Numéro*
- `iptables -R` *NomChaine Numéro Règle*
  remplace la règle de numéro *Numéro*
- `iptables -D` *NomChaine Numéro*
  détruit la règle de numéro *Numéro*

Les commandes de base (suite)

- `iptables -Z` *NomChaine*
  remettre à zéro les compteurs de paquets
- `iptables -N` *NomChaine*
  créer une nouvelle chaîne utilisateur
- `iptables -X [`*NomChaine*`]`
  détruire la chaîne utilisateur nommée (qui doit être vide et non référencée) ou toutes les chaînes utilisateurs
- `iptables -L [`*NomChaine*]
  liste les règles de la chaîne nommée, ou de toutes les chaînes. Ajouter
  - `-n` pour ne pas faire de résolution inverse des IP,
  - `-v` pour avoir toutes les informations
  - `-line-numbers` pour avoir les numéros de règles.

La description d'une règle

- `[!] -s` *IPSource[/Masque]*
  précise une condition sur l'adresse IP source (! représente la négation)
- `[!] -d` *IPDest[/Masque]*
  précise une condition sur l'adresse IP destination
- `[!] -p` *Protocole*
  précise une condition sur le protocole du paquet (tcp, udp ou icmp)
- `[!] -i` *NomInterface*
  précise une condition sur l'interface d'où provient le paquet
- `[!] -o` *NomInterface*
  précise une condition sur l'interface par laquelle va sortir le paquet
- `-j` *Décision*
  donne la décision pour cette règle (ACCEPT, REJECT, DROP, nom d'une autre chaîne,...)

Pour le protocole TCP

Si `-p tcp` est indiqué, on peut préciser

- `[!] -source-port` *Port[:Port]*
  `[!] -sport` *Port[:Port]*
  précise une condition sur le port source (port isolé ou intervalle de ports).
- `[!] -destination-port` *Port[:Port]*
  `[!] -dport` *Port[:Port]*
  précise une condition sur le port de destination (port isolé ou intervalle de ports).
- `[!] -syn`
  sélectionne les paquets qui ont le flag de synchronisation (SYN) positionné et les flags ACK, RST, FIN à 0
- `[!] -tcp-flags` *Masque Flags*
  sélectionne les paquets dont les flags TCP listé dans `Flags` sont positionnés et ceux qui sont dans `Masque` mais pas dans `Flags` ne sont pas positionnés. `Masque` et `Flags` sont les listes de drapeaux TCP séparés par des virgules (SYN,ACK,FIN,RST,URG,PSH ou bien ALL,NONE).

Pour le protocole UDP

Si `-p udp` est indiqué, on peut préciser

- `[!] -source-port` *Port[:Port]*
  `[!] -sport` *Port[:Port]*
  précise une condition sur le port source (port isolé ou intervalle de ports)
- `[!] -destination-port` *Port[:Port]*
  `[!] -dport` *Port[:Port]*
  précise une condition sur le port de destination (port isolé ou intervalle de ports)

Pour le protocole ICMP

Si `-p icmp` est indiqué, on peut préciser

- `[!] -icmp-type` *TypeICMP*
  précise une condition sur le type icmp (liste par `iptables -p icmp -h`

Modules d'extensions

Avec l'option `-m mac`, on peut préciser

- `[!] -mac-source` *AdresseMAC*
  précise une condition sur l'adresse Ethernet d'où provient le paquet

Modules d'extensions (suite)

Avec l'option `-m limit`, on peut préciser

- `-limit` *TauxMax*
  précise une condition sur le nombre maximum de correspondance (suffixe /second, /minute, /hour, /day). Utile avec la décision LOG pour éviter de saturer l'espace disque.

Modules d'extensions (suite)

Un pare-feu "stateful" conserve des informations sur les paquets précédents et permet de prendre des décisions plus intelligentes. Avec l'option `-m state`, on peut préciser

- `-state` *Etat*
  Etat peut valoir
  - NEW : ce paquet correspond à une nouvelle connexion
  - RELATED : ce paquet est une nouvelle connexion mais est associé à une connexion déjà établie (ex. de FTP ou erreur ICMP)
  - ESTABLISHED : le paquet fait partie d'une connexion déjà établie
  - INVALID : le paquet ne fait pas partie d'une connexion connue

Décision LOG

Ne décide pas du sort du datagramme. mais permet de conserver une trace des connexions dans un journal.

- `-log-level` *Niveau*
  donne la priorité du message
- `-log-prefix` *Préfixe*
  ajoute une chaîne devant le message
- `-log-tcp-options`
- `-log-ip-options`

Divers

- Avec la décision REJECT, on peut préciser le type du message ICMP qu'on renvoie avec `-reject-with` *Type* où *Type* peut être icmp-net-unreachable, icmp-host-unreachable, icmp-port-unreachable, icmp-proto-unreachable, icmp-net-prohibited ou icmp-host-prohibited.
- Ne pas oublier d'activer le routage quand on le veut (`sysctl -w net.ipv4.ip_forward=1`)

L'exemple de FTP

Rappel : il y a deux connexions :

- Une connexion de contrôle (pour les commandes) sur le port 21 du serveur
- Une connexion de données : 2 cas possibles
  - Soit depuis le port 20 du serveur vers un port désigné par le client en mode actif (commande PORT)
  - Soit vers un port >1024 alloué par le serveur et sur lequel il se met en écoute. Le numéro du port alloué est transmis sur la connexion de contrôle. Le client initie la connexion de données (mode passif, commande PASV)

Le cas FTP

- Si le firewall autorise toutes les connexions TCP externes, le mode passif convient bien.
- Si l'administrateur ne souhaite pas autoriser toutes les connexions extérieures :
  - Autoriser les connexions extérieures depuis le port 20 pour le mode actif (mais un pirate peut alors utiliser ce port 20 pour s'introduire)
  - Utiliser pour le mode passif un filtrage stateful qui va autoriser la connexion sortante pour peu que l'on ait vu passer la commande PASV

IPv6

- iptables ne gère que les datagrammes IPv4
- les datagrammes IPv6 sont gérés par ip6tables (même principe que iptables)

FirewallBuilder

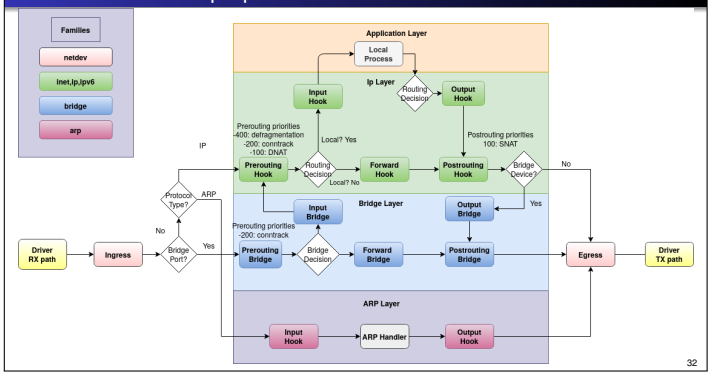
- http ://www.fwbuilder.org (paquetage fwbuilder)
- interface graphique pour mettre en place un filtrage par paquets
- peut générer des règles dans différentes syntaxes (iptables, Cisco, ...)

nftables

nftables

- nftables remplace iptables
- Il permet de regrouper la gestion de plusieurs filtrages (IPv4, IPv6, ARP, bridge).
- La commande pour gérer nftables s'appelle `nft`. Elle remplace différentes commandes :
  - iptables
  - ip6tables
  - arptables
  - ebtables
- Il n'y a plus de chaîne par défaut (INPUT, OUTPUT, FORWARD). C'est à l'utilisateur de créer ses propres chaînes, et de les greffer dans les points de filtrage du noyau.

Cheminement des paquets



Familles d'adresses

Lors de la création d'une table, on devra préciser à quelle famille d'adresse elle se rattache. Les familles possibles sont :

- ip : IPv4
- ip6 : IPv6
- inet : IPv4 et IPv6
- arp : ARP
- bridge : passerelle de niveau 2

Dans notre cas, on n'utilisera que inet. En l'absence de précision, ip est la famille par défaut.

Interception des paquets pour la famille inet

- prerouting : paquets à leur arrivée avant le routage
- **input :** paquets avant leur transmission à un programme local
- **forward :** paquets transitant par la machine mais qui ne lui sont pas destinés (routeur)
- **output :** paquets émis par un programme local
- postrouting : paquets avant leur sortie de la machine

Dans notre cas, on ne travaillera que sur input, output et forward.

Règles, chaînes, tables

- Une règle est une condition de la forme `if <condition> then <decision>`
  - La condition d'une règle va porter sur différents éléments d'un paquet
  - La décision d'une règle va indiquer s'il faut bloquer ou laisser passer le paquet.
  - Les conditions d'une règle sont reliées par un ET logique
- Les règles sont regroupées dans des chaînes (chain). Ces chaînes jouent le rôle de sous-programmes (fonctions).
- Les chaînes sont regroupées dans des tables. On peut voir ces tables comme l'équivalent de paquetages.
- Le mot-clef ruleset désigne l'ensemble des tables, chaînes et règles.

Squelette type iptables

Les identifiants en majuscules sont modifiables. À charger avec `nft -f fichier`

```
flush ruleset

table inet FILTER {
  chain INPUT {
    type filter hook input priority filter; policy drop;
    # les règles de filtrage pour INPUT
  }
  chain FORWARD {
    type filter hook forward priority filter; policy drop;
    # les règles de filtrage pour FORWARD
  }
  chain OUTPUT {
    type filter hook output priority filter; policy accept;
    # les règles de filtrage pour OUTPUT
  }
}
```


Exemple de filtrage INPUT

```
flush ruleset

table inet FILTER {
  chain INPUT {
    type filter hook input priority filter; policy drop;
    # accepter tout ce qui vient de lo (i.e. local à la machine)
    iif lo accept
    # accepter les réponses aux datagrammes qu'on envoie
    ct state { established, related } accept
    # accepter une connexion ssh depuis l'adresse 1.2.3.4
    ip saddr 1.2.3.4 tcp dport ssh accept
    # bloquer tous les autres paquets en renvoyant un message ICMP
    reject
  }
}
```


Décision d'une règle

- `accept` : laisser passer le paquet (arrête l'examen des règles)
- `drop` : bloquer le paquet sans renvoyer de message ICMP (arrête l'examen des règles)
- `reject` : bloquer le paquet et renvoyer un message ICMP d'erreur (arrête l'examen des règles). On peut choisir le message d'erreur en ajoutant :
  - `with icmp` *code*
  - `with icmpv6` *code*
  - `with icmpx` *code*
  - `with tcp reset`
- `jump` *chain* : examiner les règles de la chaîne et revenir à la règle qui suit après un return où à la fin de la chaîne
- `return` : fin de l'examen des règles de la chaîne courante et retour à la chaîne appelante
- `goto` *chain* : examiner les règles de la chaîne (sans retour)

Conditions IPv4 et IPv6

- `ip saddr` *ListeDAdresses*
- `ip daddr` *ListeDAdresses*
- `ip6 saddr` *ListeDAdresses*
- `ip6 daddr` *ListeDAdresses*
- `ip protocol` *tcp/udp/icmp/...*
- `ip6 nexthdr` *tcp/udp/icmp/...*

La liste d'adresses peut être :

- une adresse individuelle (/32) : 1.0.0.1
- une adresse de réseaux : 1.0.0.0/8
- un ensemble d'adresses : {1.0.0.0/8,2.0.0.1}
- un opérateur relationnel (==,! =,<,<=,>,>=) suivi d'une adresse

Conditions UDP

- `udp sport` *ListeDePorts*
- `udp dport` *ListeDePorts*

La liste de ports peut être :

- un port individuel (entier ou nom du port)
- un intervalle de ports : 1000-2000
- un ensemble de ports : {ssh,smtp,domain}
- un opérateur relationnel (==,! =,<,<=,>,>=) suivi d'un port

Conditions TCP

- `tcp sport` *ListeDePorts*
- `tcp dport` *ListeDePorts*
- `tcp flags`
  donne la liste des flags TCP actifs, à utiliser dans une expression
  Exemple : `tcp flags & (syn | ack | fin) == (syn | ack)`

Connection tracking

- Le système conserve un “résumé” des paquets précédents, ce qui lui permet de classer les paquets dans plusieurs états :
  - new : aucune relation avec un paquet précédent
  - established : une réponse à un paquet précédent
  - related : un paquet qui débute une nouvelle connexion, négociée dans une communication existante.
  - invalid : un paquet mal formé
- Les tests les plus utiles :
  - `ct state {new}`
    vrai pour un “nouveau” paquet (qui ne correspond à aucun dialogue existant)
  - `ct state {established,related}`
    vrai pour un paquet qui correspond à un dialogue existant. Permet d’autoriser les réponses à des paquets autorisés sans rien autoriser d’autre.

log

- `log`
  provoque l’enregistrement des information sur le paquet dans le fichier journal. **Doit apparaître après toutes les conditions.**
- `log prefix "texte"`
  idem mais en ajoutant le texte dans le fichier journal

Divers

- `counter`
  à ajouter à la fin d’une condition, permet de compter le nombre de paquets qui satisfont la condition et de sommer leur taille
- `iif|iifname|oif|oifname` *nomDInterface*
  vrai si l’interface d’entrée (iif|iifname) ou de sortie (oif|oifname) est égale à *nomDInterface*. *if est évalué à la lecture de la règle (donc problème si l’interface disparaît ou change de nom), *ifname est évalué à l’exécution.
- `ether saddr|daddr` *adresseMAC*
  condition sur l’adresse source ou destination de la trame ethernet
- `icmp|icmpv6 type` *type*
  vrai si le paquet icmp a le type indiqué
- `icmp|icmpv6 code` *code*
  vrai si le paquet icmp a le code indiqué

limit

- `limit rate 10/second`
  la règle ne pourra pas s’appliquer plus de 10 fois par seconde
- `limit rate over 10/second`
  la règle ne pourra s’appliquer qu’au delà de 10 correspondances par seconde

Divers

- `include` *fichier*
  inclusion d’un fichier
- `define` *var=valeur*
  définition d’une variable
- *$var*
  valeur de la variable
- .
  Le point est l’opérateur de concaténation

Commandes sur les ruleset

Le mot-clef ruleset désigne l’ensemble des tables, chaînes et règles.

- Tout lister :
  `nft list ruleset`
- Lister uniquement une famille donnée :
  `nft list ruleset` *famille*
- Tout effacer :
  `nft flush ruleset`
- Effacer uniquement une famille donnée :
  `nft flush ruleset` *famille*
- Tout sauvegarder
  `nft list ruleset > backupFile`
- Tout restaurer
  `nft flush ruleset ; nft -f backupFile`

Commandes sur les tables

- Lister les tables existantes :
  `nft list tables`
- Il n’y a aucune chaîne ou table définie par défaut.
- Création d’une table : `nft add table` *famille nomDeLaTable*
  Exemple : `nft add table inet MyFW`
- Lister le contenu d’une table :
  `nft list table` *famille nomDeLaTable*

Commandes sur les chaînes

- Créer une chaîne dans une table :
  `nft add chain` *famille nomDeLaTable nomDeLaChaîne*
  Exemple : `nft add chain MyFW MyInput`
- Créer une chaîne dans une table et la greffer au point de filtrage *Hook* :
  `nft add chain` *famille nomDeLaTable nomDeLaChaîne* `"{type filter hook` *Hook* `priority 0 ;}"`
  Exemple : `nft add chain inet MyFW MyInput "{type filter hook input priority 0 ;}"`
- Créer une chaîne dans une table, la greffer au point de filtrage *Hook* et choisir la politique par défaut *Policy* (accept ou drop) :
  `nft add chain` *famille nomDeLaTable nomDeLaChaîne* `"{type filter hook` *Hook* `priority 0 ; policy` *Policy}"*
  Exemple : `nft add chain inet MyFW MyInput "{type filter hook input priority 0; policy drop;}"`
- Pour nommer une chaîne, il faudra toujours préciser le triplet *famille nomDeLaTable nomDeLaChaîne*.

Les règles

- Lister les règles et leurs identifiants (handle)
  `nft -a list chain` *famille nomDeLaTable nomDeLaChaîne*
- Ajout d’une règle à la fin d’une chaîne (append)
  `nft add rule` *famille nomDeLaTable nomDeLaChaîne ListeDeConditions Décision*
- Ajout d’une règle après la règle d’identifiant *Handle*
  `nft add rule` *famille nomDeLaTable nomDeLaChaîne* `handle` *Handle ListeDeConditions Décision*
- Ajout d’une règle au début d’une chaîne
  `nft insert rule` *famille nomDeLaTable nomDeLaChaîne ListeDeConditions Décision*
- Ajout d’une règle avant la règle d’identifiant *Handle*
  `nft insert rule` *famille nomDeLaTable nomDeLaChaîne* `handle` *Handle ListeDeConditions Décision*

Les règles (suite)

- Supprimer une règle d'identifiant *Handle*
  `nft delete rule` *famille nomDeLaTable nomDeLaChaîne* `handle` *Handle*
- Effacer toutes les règles d'une chaîne
  `nft flush chain` *famille nomDeLaTable nomDeLaChaîne*
- Remplacer une règle d'identifiant *Handle*
  `nft replace rule` *famille nomDeLaTable nomDeLaChaîne* `handle` *Handle ListeDeConditions Décision*

Proxy

Les systèmes mandataires

- Un système mandataire est une application qui sert de relai. Le client au lieu de contacter directement le serveur contacte le système mandataire (proxy) qui se charge alors de contacter le serveur.
- Le système mandataire peut choisir de ne pas relayer la requête
- Le système mandataire doit être un point de passage obligé (imposé par un filtrage de paquets ou l'absence de route)
- Exemples : Squid, TIS FWTK, SOCKS

Les systèmes mandataires

- Points forts :
  - Ils sont conçus pour un protocole spécifique (ex : telnet, ftp) et peuvent utiliser des informations spécifiques à ce protocole
  - Il permettent de maintenir une trace des connexions
  - Il permettent de gérer un cache (ex : Squid)
  - Ils peuvent authentifier l'utilisateur
- Points faibles
  - Plus gourmands en resources
  - Il faut avoir un proxy pour chacun des protocoles
  - Le client doit être configuré pour contacter le proxy (mais il existe dans certains cas des systèmes transparents)

Exemple de configuration de Squid (/etc/squid/squid.conf)

```
# port et adresses du proxy
http_port IPDuProxy:3128
icp_port 3130

access_log /var/log/squid/access.log squid
error_directory /usr/share/squid/errors/French
coredump_dir /var/spool/squid

cache_mgr       webmaster@VotreDomaine.fr

# contacter le proxy (par defaut), pas de requete ICP,
# pas de mise en cache
# cache_peer UnProxyEnAMont parent 3128 3130 proxy-only no-query

# les differents reseaux
acl reseau_local src 10.0.0.0/8
# localhost
acl localhost src 127.0.0.1/32

# serveur web locaux
acl local-web dstdomain .VotreDomaine.fr
```


Exemple de configuration de Squid (suite)

```
# definitions standards
acl manager proto cache_object
acl SSL_ports port 443
acl Safe_ports port 80          # http
acl Safe_ports port 21          # ftp
acl Safe_ports port 443         # https
acl Safe_ports port 70          # gopher
acl Safe_ports port 210         # wais
acl Safe_ports port 1025-65535  # unregistered ports
acl Safe_ports port 280         # http-mgmt
acl Safe_ports port 488         # gss-http
acl Safe_ports port 591         # filemaker
acl Safe_ports port 777         # multiling http
acl CONNECT method CONNECT

# tout doit passer par le proxy parent, sauf le web local
# never_direct deny local-web
# never_direct allow all
```


Exemple de configuration de Squid (suite)

```
# restrictions standards
http_access allow manager localhost
http_access deny manager
http_access deny !Safe_ports
http_access deny CONNECT !SSL_ports

http_access allow reseau_local
http_access allow localhost
# par defaut, on interdit l'acces
http_access deny all

icp_access deny all

# defaults
hierarchy_stoplist cgi-bin ?

refresh_pattern ^ftp:           1440    20%     10080
refresh_pattern ^gopher:        1440    0%      1440
refresh_pattern (cgi-bin|\?)    0       0%      0
refresh_pattern .               0       20%     4320
```


Problèmes non résolus par les pare-feux

- Le pare-feu ne protège pas des dangers venant de l'intérieur
- Le pare-feu ne protège guère contre les failles des applications qui sont autorisées à passer
- Un tunnel peut être utilisé pour contourner le filtrage du pare-feu.
- Le pare-feu ne peut pas surveiller les communications chiffrées (SSL,...)

Exemples de tunnel avec SSH

- `ssh -X user@remoteHost`
  L'affichage graphique (X11) de la commande distante sera transmise via la connexion sécurisée et s'affichera sur notre écran.
- `ssh -L[bindAddr:]localPort:remoteAddr:remotePort user@remoteHost`
  Un proxy TCP se mettra à l'écoute (optionnellement uniquement sur l'adresse *bindAddr*) sur le port *localPort* de la machine locale. Dès qu'un client se connecte, la communication sera transmise via le canal sécurisé et le serveur SSH contactera depuis la machine *remoteHost* la machine *remoteAddr* sur le port *remotePort*.

Exemples de tunnel avec SSH (suite)

- `ssh -R[bindAddr:]port:remoteAddr:remotePort user@remoteHost`
  Un proxy TCP se mettra à l'écoute (optionnellement uniquement sur l'adresse *bindAddr*) sur le port *port* de la machine *remoteHost*. Dès qu'un client se connecte, la communication sera transmise via le canal sécurisé et le serveur SSH contactera depuis la machine locale la machine *remoteAddr* sur le port *remotePort*.

La translation d'adresse

- NAT (Network Address Translation)
- Permet d'utiliser en interne un ensemble d'adresse IP (ex : adresses de réseau privé) et d'utiliser en externe un autre ensemble d'adresse IP (adresses du firewall). Permet une "compression" d'adresses.
- La translation d'adresse modifie les paquets pour remplacer les adresses internes par les adresses externes ou vice versa. Le firewall maintient une table pour savoir à qui renvoyer quoi.
- La translation n'est pas aussi évidente qu'il n'y paraît : certains protocoles envoient l'adresse IP du client à l'intérieur des paquets. Problème avec les protocoles non connectés : pendant combien de temps faut-il assurer la translation ?

Translation d'adresse (suite)

Vocabulaire :

- NAPT (Network Address and Port Translation), PAT (Port Address Translation) : translation d'adresse utilisant l'adresse IP et le port TCP/UDP
- Masquerade : remplacer l'adresse IP source par l'adresse du routeur
- DNAT : on modidie l'adresse IP/le port destination
- SNAT : on modidie l'adresse IP/le port source
- Port forwarding : on renvoie une communication vers un autre port

Translation d'adresse sous Linux

Sous Linux,

- iptables gère plusieurs tables. La table par défaut est "filter". C'est elle qui contient les chaînes INPUT, FORWARD et OUTPUT.
- Le table qui gère le NAT s'appelle "nat". On doit fournir l'option "-t nat" à iptables. Cette table contient les chaînes PREROUTING, OUTPUT et POSTROUTING. De nouvelles décisions sont possibles : DNAT, SNAT, MASQUERADE,...
- Exemple pour remplacer les adresses de notre réseau local par l'adresse du pare-feu.

```
iptables -t nat -A POSTROUTING -s ${localnet} -j MASQUERADE
```


DMZ

- Une DMZ (DeMilitarized Zone) est un réseau intermédiaire où l'on place les machines exposées (i.e. visibles de l'Internet). Si jamais elles sont piratées, le hacker n'aura pas un accès direct au réseau interne.