SÉCURITÉ DES RÉSEAUX
PARE-FEU NETFILTER (IPTABLES ET CONNTRACK)

Jonathan CERTES - Benoît MORGAN

1 Environnement de travail

1.1 Travail dans une machine virtuelle

Télécharger Oracle VirtualBox :
`https://www.virtualbox.org/wiki/Downloads`
Installer VirtualBox sur sa machine personnelle :
`https://www.virtualbox.org/manual/UserManual.html#installation`
Télécharger la machine virtuelle fournie par l'enseignant.
Importer la machine virtuelle dans VirtualBox :
`https://www.virtualbox.org/manual/UserManual.html#ovf-import-appliance`
Démarrer la machine virtuelle :
`https://www.virtualbox.org/manual/UserManual.html#intro-starting`
Tout le TP sera réalisé dans la machine virtuelle.

1.2 Environnement

La machine virtuelle contient un conteneur `lxc` par machine du réseau virtuel pour ce TP (hôte ou routeur). Ces conteneurs ne possèdent pas d'interface graphique. Il s'agit de lancer un terminal dans la machine virtuelle et de s'y attacher autant de fois que nécessaire, pour tous les conteneurs.

Le lancement des conteneurs est automatisé par un script présent dans la machine virtuelle. Pour procéder à leur lancement, ouvrir un terminal et exécuter les commandes suivantes :

```
debian@myhostname:~$ cd tp-firewall/
debian@myhostname:~/tp-firewall$ ./start.sh
```

Le mot de passe pour le compte utilisateur est : `debian`.

1.3 S'associer aux conteneurs

Dans 4 terminaux, répéter ces actions pour les 4 conteneurs `dmz`, `firewall`, `interne` et `internet` :

```
debian@myhostname:~$ sudo xhost +
debian@myhostname:~$ sudo lxc-attach dmz
root@dmz:/$
```

1.4 Ouvrir Wireshark sur la machine virtuelle

La dernière étape de préparation de ce TP est le lancement de Wireshark (3 fois) sur les bridges (connexions virtuelles) de notre réseau. Il n'est nécessaire d'observer le trafic que sur les 3 bridges `br-dmz`, `br-interne` et `br-internet`.

Dans 3 terminaux :

```
debian@myhostname:~$ sudo wireshark
```

Réduire les terminaux **sans les fermer**. Dans chacun des 3 Wireshark, sélectionner la capture sur le bridge `br-dmz`, `br-interne` ou `br-internet`. Une fois la capture lancée, appliquer le filtre `!dhcp && !arp`.

2 Objectifs

L'objectif de ces séances est de se familiariser avec les principes de fonctionnement et la configuration d'un pare-feu dans un réseau TCP/IP. L'architecture que nous utilisons est similaire à celles que nous retrouvons dans les réseaux d'entreprises.

C'est parti !

3 Introduction

Netfilter est un cadriciel (framework) implémentant un pare-feu au sein du noyau Linux à partir de la version 2.4 de ce dernier. Il prévoit des accroches (hooks) dans le noyau pour l'interception et la manipulation des paquets réseau lors des appels des routines de réception ou d'émission des paquets des interfaces réseau.

La version 1.4.2 a reçu un Certificat de Sécurité de Premier Niveau (CSPN) par l'Agence nationale de la sécurité des systèmes d'information.

Source : `https://fr.wikipedia.org/wiki/Netfilter`

Les modules du noyau nommés `ip_tables`, `ip6_tables`, `arp_tables` (les soulignements font partie du nom) et `ebtables` sont les systèmes des accroches de Netfilter. Ils fournissent un système basé sur des tableaux pour définir des règles de pare-feu qui filtrent les paquets ou les transforment. Les tableaux peuvent être administrés par les outils utilisateur **iptables**, ip6tables, arptables et ebtables, respectivement.

Source : `https://fr.wikipedia.org/wiki/Netfilter`

Une des caractéristiques importantes construites sur le framework Netfilter est **Connection Tracking**. CT permet au noyau de garder la trace de toutes les connexions réseau logiques ou de sessions, et, par conséquent, porte tous les paquets qui composent cette connexion. NAT s'appuie sur cette information pour traduire tous les paquets de la même manière, et iptables peut utiliser cette information pour agir comme un pare-feu avec persistance.

Source : `https://fr.wikipedia.org/wiki/Netfilter`

Wireshark est un analyseur de paquets libre et gratuit. Il est utilisé dans le dépannage et l'analyse de réseaux informatiques, le développement de protocoles, l'éducation et la rétro-ingénierie.

Wireshark utilise la bibliothèque logicielle Qt pour l'implémentation de son interface utilisateur et pcap pour la capture des paquets ; il fonctionne sur de nombreux environnements compatibles UNIX comme GNU/Linux, FreeBSD, NetBSD, OpenBSD ou Mac OSX, mais également sur Microsoft Windows. Il existe aussi entre autres une version en ligne de commande nommé TShark. Ces programmes sont distribués gratuitement sous la licence GNU General Public License.

Wireshark reconnaît 1 515 protocoles.

Source : `https://fr.wikipedia.org/wiki/Wireshark`

Nmap est un scanner de ports libre créé par Fyodor et distribué par Insecure.org. Il est conçu pour détecter les ports ouverts, identifier les services hébergés et obtenir des informations sur le système d'exploitation d'un ordinateur distant. Ce logiciel est devenu une référence pour les administrateurs réseaux car l'audit des résultats de Nmap fournit des indications sur la sécurité d'un réseau. Il est disponible sous Windows, Mac OS X, Linux, BSD et Solaris.

Source : `https://fr.wikipedia.org/wiki/Nmap`

Scapy est un logiciel libre de manipulation de paquets réseau écrit en python. Il est capable, entre autres, d'intercepter le trafic sur un segment réseau, de générer des paquets dans un nombre important de protocoles, de réaliser une prise d'empreinte de la pile TCP/IP, de faire un traceroute, d'analyser le réseau informatiques...

Scapy n'est pas un outil clé en main (comme Nmap ou autre) mais un Framework basé sur Python fournissant un ensemble de fonctions pour interagir avec le réseau. Il a donc les avantages suivants :

— Une grande liberté d'action car chaque paramètre peut être modifié
— Il décode mais n'interprète pas les paquets reçus

Source : `https://fr.wikipedia.org/wiki/Scapy`

4 Configuration réseau

Procéder au lancement des conteneurs comme décrit dans la section 1.2. L'architecture du réseau virtuel créée est représentée sur la figure 1. Chaque machine possède, sur ce réseau, une adresse IPv4 et un nom (`dmz`, `interne` ou `internet`).

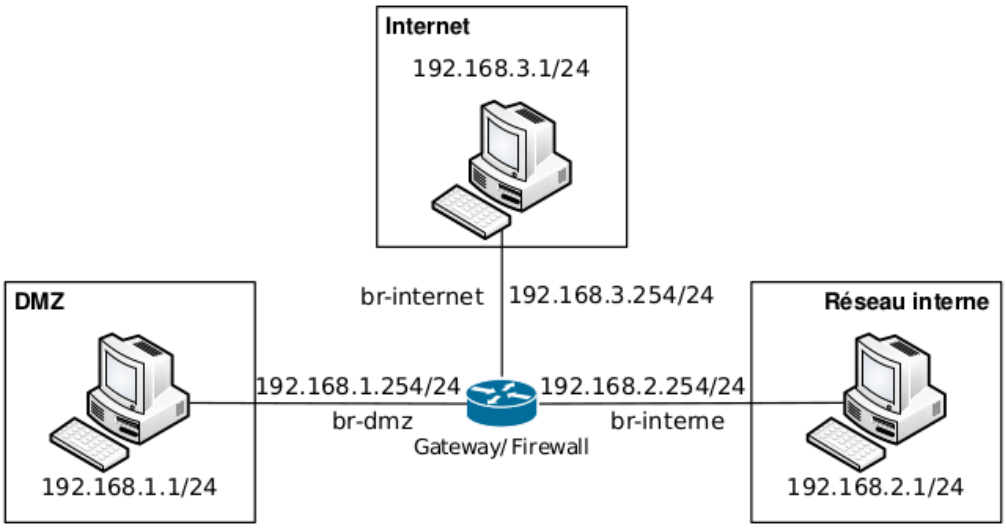

FIGURE 1 – Architecture du réseau

Dans ce TP, nous incarnons l'administrateur système et réseau d'une entreprise. Son objectif est de sécuriser les connexions aux machines du réseau interne de l'entreprise tout en laissant accessibles des services publiques (par exemple, le serveur HTTP hébergeant le site web de l'entreprise).

À ces fins, le réseau de l'entreprise est séparé en deux zones :

— le sous-réseau **interne**, qui contient les machines hébergeant des données sensibles/confidentielles.

— le sous-réseau **DMZ**, pour *demilitarized zone* (en français, zone démilitarisée), qui contient les machines hébergeant des données accessibles au public.

Également, le réseau de l'entreprise est connecté à **internet**, un ensemble de machines contrôlées par des utilisateurs potentiellement malveillants. La difficulté de l'administration réseau est la suivante :

— Les machines présentes sur **internet** doivent pouvoir accéder aux services hébergées sur la **DMZ** mais de doivent pas pouvoir accéder aux machines présentes sur le sous-réseau **interne**.

— Les machines présentes sur sous-réseau **interne** doivent pouvoir accéder aux services hébergées sur **internet** et sur la **DMZ**.

Les différents réseaux sont connectés par un routeur, la *gateway*, sur lequel un pare-feu doit être configuré. Durant ce TP, nous allons donc configurer Netfilter sur ce routeur, à l'aide de la commande `iptables` et du module *Connection Tracking* (aussi appelé *conntrack*). Nous pouvons prendre le contrôle du routeur contenant le pare-feu avec la commande suivante :

```
debian@myhostname:~$ sudo lxc-attach firewall
root@firewall:/$
```

Question 1 Vérifier le fonctionnement du réseau décrit sur la figure 1. Pour chacune des 3 machines qui simule les sous-réseaux, (`dmz`, `interne` et `internet`), tenter d'envoyer une requête *ping*.
Vérifier qu'une réponse est reçue. Vérifier que les deux paquets transitent bien sur le réseau à l'aide de 3 *wireshark*.

```
root@dmz:/$ ping -c 1 interne
```

Répéter cette opération en effectuant un *ping* vers `internet`.
Répéter ces deux opérations depuis la machine `interne` (vers `dmz` puis `internet`).
Répéter ces deux opérations depuis la machine `internet` (vers `dmz` puis `interne`).
À chaque opération, redémarrer la capture sur *wireshark* pour bien observer les nouveaux paquets.

Contactez l'enseignant si vous n'observez pas de réponse sur une de ces communications !

5 Mise en place des services

Pour tester les règles du pare-feu, différents services doivent être installés sur la DMZ : serveur HTTP, FTP, SMTP, DNS, etc. La mise en place et la configuration de ces services est fastidieuse et peu pertinente pour notre besoin. Nous allons donc simuler ces serveurs à l'aide de *netcat*.

La commande suivante permet de lancer un serveur qui écoute (option `-l` de *listen*) sur un port (`-p`) en TCP, pour une adresse IP source (`-s`) donnée. L'option `-v` (*verbose*) indique d'afficher plus d'informations que normalement.

```
netcat -v -l -s <ip> -p <port>
```

Penser à utiliser l'adresse IP de l'interface qui nous intéresse. Par exemple, si nous choisissons `127.0.0.1`, l'écoute ne se fera que sur l'interface `lo` (en *loopback*).

De la même manière, la commande suivante permet de lancer un serveur qui écoute sur un port en UDP (option `-u`), toujours pour une adresse IP source et un port donnés.

```
netcat -v -l -u -s <ip> -p <port>
```

5.1 Tester si un service est accessible

Lorsque *netcat* est en écoute sur un port pour une adresse IP donnée, nous pouvons lui envoyer de la donnée à travers le réseau. Si le protocole utilisé est TCP, *netcat* se terminera après la fin de session.

Pour transmettre de la donnée à *netcat* depuis une autre machine et fermer la session, nous pouvons utiliser *telnet*. La commande suivante établit une session TCP vers une adresse IP sur un port donné et la termine.

```
echo "quit" | telnet <ip> <port>
```

Question 2 Simuler un serveur HTTP sur la DMZ à l'aide d'une écoute de *netcat* en TCP sur le port 80.

```
root@dmz:/$ netcat -v -l -s $(hostname --ip-address) -p 80
Listening on dmz 80
```

Question 3 Depuis internet, simuler une communication avec le serveur HTTP de la DMZ avec *telnet*, en TCP sur le port 80. Vérifier que *netcat* se termine sur la DMZ.

```
root@internet:/$ echo "quit" | telnet dmz 80
Trying 192.168.1.1...
Connected to dmz.
Escape character is '^]'.
Connection closed by foreign host.
```

Question 4 À l'aide des 3 *wireshark* qui observent les bridges `br-dmz`, `br-interne` et `br-internet`, vérifier que la communication TCP a bien transité sur les bridges `br-dmz` et `br-internet`.
À chaque simulation, redémarrer la capture sur *wireshark* pour bien observer les nouveaux paquets.

Si le protocole utilisé est UDP, il n'y a pas de session à proprement parler, *netcat* ne se terminera donc pas après l'envoi de donnée. De plus, nous ne pouvons pas utiliser *telnet* qui utilise uniquement le protocole TCP.

Pour transmettre de la donnée à *netcat* depuis une autre machine, nous allons donc utiliser un autre *netcat* et terminer la session manuellement. La commande suivante établit une connexion UDP vers une adresse IP sur un port donné (noter l'absence de l'option `-l` car ce n'est pas une écoute).

```
netcat -v -u <ip> <port>
```

Question 5 Simuler un résolveur DNS sur la DMZ à l'aide d'une écoute de *netcat* en UDP sur le port 53.

```
root@dmz:/$ netcat -v -l -u -s $(hostname --ip-address) -p 53
Bound on dmz 53
```

Question 6 Depuis internet, simuler une communication avec le résolveur DNS de la DMZ, avec *netcat*, en UDP sur le port 53. Terminer la connexion avec la combinaison de touches `Ctrl` + `C` sur la machine internet. Sur la DMZ, appuyer sur `Entrée` pour vérifier que la session de *netcat* s'est bien terminée.

```
root@internet:/$ netcat -v -u dmz 53
Connection to dmz (192.168.1.1) 53 port [udp/domain] succeeded!

^C
```

Question 7 À l'aide des 3 *wireshark* qui observent les bridges `br-dmz`, `br-interne` et `br-internet`, vérifier que la communication UDP a bien transité sur les bridges `br-dmz` et `br-internet`.
À chaque simulation, redémarrer la capture sur *wireshark* pour bien observer les nouveaux paquets.

5.2 Pour le reste du TP

Dans la suite de ce TP, nous configurons notre pare-feu et vérifions le bon fonctionnement des règles que nous implémentons. Afin d'effectuer les vérifications, nous testons si un service est accessible (dans le cas où le pare-feu doit le rendre accessible) ou si celui-ci est inaccessible (dans le cas où le pare-feu doit en bloquer l'accès). À chaque vérification, nous observons le transit des paquets sur le réseau à l'aide des 3 *wireshark*.

Pour ne pas avoir à lancer plusieurs fois *netcat* (pour chaque service sur chaque machine) à chaque vérification, il est possible d'exécuter celui-ci en arrière plan, dans une boucle infinie. Par exemple, la commande suivante permet de simuler, pour la DMZ, un serveur HTTP qui écoute en permanence sur le port 80 :

```
while true ; do netcat -v -l -s 192.168.1.1 -p 80 ; done &
```

Sur chacune des 3 machines (`dmz`, `interne`, `internet`), un script `/rw/machinerc` est disponible et exécute plusieurs fois *netcat* dans des boucles infinies pour simuler les services en écoute via le protocole TCP. Pour le protocole UDP, les vérifications doivent être réalisées à la main.

Question 8 Sur chacune des 3 machines, exécuter le script `/rw/machinerc` pour mettre en place la simulation des services.

```
root@dmz:/$ /rw/machinerc
```

```
root@interne:/$ /rw/machinerc
```

```
root@internet:/$ /rw/machinerc
```

Le programme `lsof` (pour *list open files*) permet de visualiser les processus qui ont ouvert une lecture sur un fichier ou un *socket*. Il nous permet de vérifier les ports qui sont en écoute sur une machine donnée. Des options de filtrages permettent de limiter la liste.

Par exemple, la commande suivante affiche la liste des processus qui lisent les paquets reçus en IPv4 (`-i4`) et (`-a`) dont le protocole est TCP (`-itcp`) ou UDP (`-iudp`) ; un filtre sur le nom du processus est appliqué (`-c`) pour afficher seulement *netcat*.

```
lsof -i4 -a -itcp -iudp -c netcat
```

Question 9 Sur chacune des 3 machines, lister les processus en écoute des paquets reçus en IPv4 et dont le protocole est TCP ou UDP.
Vérifier que `sshd` écoute sur le port 22 (*ssh*) sur chaque machine.
Vérifier que `netcat` écoute sur les ports 53 (*domain*), 25 (*smtp*), 443 (*https*) et 80 (*http*) sur la DMZ.
Vérifier que `netcat` écoute sur les ports 443 (*https*) et 80 (*http*) sur internet.

Le résultat obtenu doit ressembler à ceci :

```
root@dmz:/$ lsof -i4 -a -itcp -iudp
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dhclient  89 root    9u  IPv4  19404      0t0  UDP *:bootpc
sshd     122 root    3u  IPv4  20427      0t0  TCP *:ssh (LISTEN)
netcat   140 root    3u  IPv4  24612      0t0  TCP dmz:domain (LISTEN)
netcat   141 root    3u  IPv4  24615      0t0  TCP dmz:smtp (LISTEN)
netcat   142 root    3u  IPv4  24619      0t0  TCP dmz:https (LISTEN)
netcat   144 root    3u  IPv4  24622      0t0  TCP dmz:http (LISTEN)
```

```
root@interne:/$ lsof -i4 -a -itcp -iudp
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dhclient  90 root    9u  IPv4  21371      0t0  UDP *:bootpc
sshd     122 root    3u  IPv4  22368      0t0  TCP *:ssh (LISTEN)
netcat   140 root    3u  IPv4  22470      0t0  TCP interne:smtp (LISTEN)
```

```
root@internet:/$ lsof -i4 -a -itcp -iudp
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
dhclient   91 root    9u  IPv4  23309      0t0  UDP *:bootpc
sshd      122 root    3u  IPv4  24452      0t0  TCP *:ssh (LISTEN)
netcat   6161 root    3u  IPv4  26976      0t0  TCP internet:http (LISTEN)
netcat   6163 root    3u  IPv4  26999      0t0  TCP internet:https (LISTEN)
```

Contactez l'enseignant si vous n'observez pas d'écoute sur ces ports !

Question 10 Vérifier que tous les services simulés par `netcat` sont accessibles sur le réseau. Par exemple, vérifier que nous pouvons accéder aux services hébergés sur la DMZ et sur internet depuis internet.

```
root@internet:/$ echo "quit" | telnet dmz 53
root@internet:/$ echo "quit" | telnet dmz 25
root@internet:/$ echo "quit" | telnet dmz 443
root@internet:/$ echo "quit" | telnet dmz 80
root@internet:/$ echo "quit" | telnet internet 443
root@internet:/$ echo "quit" | telnet internet 80
```

Question 11 Vérifier qu'il est possible de réaliser une connexion `ssh` vers le réseau interne depuis internet.
Note : les conteneurs ne possèdent qu'un compte *root* et n'ont pas de mot de passe. Il n'est donc pas possible d'aboutir à une connexion. Une simple demande de mot de passe suffit à vérifier que le pare-feu rend la connexion possible.

```
root@internet:/$ ssh root@interne
The authenticity of host 'interne (192.168.2.1)' can`t be established.
ECDSA key fingerprint is SHA256:ecM40Soiss//Lx5QP0WUnTHvrA/9WIrt4eUxjoIVSbA.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'interne,192.168.2.1' (ECDSA) to the list of known hosts.
root@interne`s password:
```

La configuration de notre réseau semble terminée. Néanmoins la sécurité de celui-ci laisse à désirer :

- il est possible de prendre le contrôle des machines de notre réseau interne depuis n'importe où sur internet.
- il est possible d'utiliser le serveur SMTP (port 25) de la DMZ depuis n'importe où sur internet.
- des paquets peuvent être transmis en grand nombre sur notre réseau interne pour réaliser une attaque par déni de service depuis n'importe où sur internet.

Afin de se protéger, nous allons donc configurer un pare-feu sur la *gateway* de notre DMZ et de notre réseau interne. L'objectif est que le trafic malveillant soit bloqué mais que les services restent accessibles lorsque le trafic est légitime.

Pour cela, nous allons nous appuyer sur les fonctionnalités de **Netfilter**, que nous allons configurer à l'aide de la commande `iptables`.

6 Rappels techniques : fonctionnement de `iptables`

Iptables est utilisé pour mettre en place, maintenir, et inspecter les tables des règles de filtrage des paquets IP du noyau Linux. Plusieurs tables différentes peuvent être définies. Chaque table contient un nombre de chaînes pré-définies, et peut aussi contenir des chaînes définies par l'utilisateur.

Chaque chaîne est une liste de règles auxquelles peuvent correspondre un ensemble de paquets. Chaque règle spécifie ce qui doit être fait avec un paquet qui correspond. Cela s'appelle une "cible", qui peut être un saut vers une chaîne définie par l'utilisateur dans la même table.

Source : `man iptables`

6.1 Traversée des paquets

Lorsque l'on crée une règle pour `iptables`, il faut préciser où l'insérer et à quelle moment celle-ci doit être appliquée. Comme son nom l'indique, `iptables` applique des règles que l'on insère dans des **tables**. Selon la destination du paquet et l'instant auquel nous appliquons les règles, l'enchaînement des règles est différent. Nous appelons cela une **chaîne**.

Les règles peuvent être insérées dans les **tables** suivantes :
- **`filter`**, c'est la table par défaut, elle contient les règles de filtrage. C'est-à-dire qu'elle permet de déterminer si un paquet est accepté ou si nous devons l'abandonner (*drop*).
- **`nat`**, la table qui contient des règles qui mèneront à des modifications de type NAT (*Network Address Translation*) sur les paquets.
- **`mangle`**, la table qui contient des règles qui mèneront à des modifications arbitraires sur les paquets.

La table **`conntrack`** contient les règles qui déterminent si plusieurs paquets correspondent à une même connexion ou pas, enregistrent quelles connexions sont établies, etc. Cette table n'est pas modifiable et elle sera ignorée tout au long du TP.

La chaîne détermine l'instant d'application de la règle. Il y a cinq **chaînes** par défaut dans *iptables* :
- **`PREROUTING`**, qui concerne les paquets entrants avant les décisions de routage.
- **`FORWARD`**, qui concerne les paquets qui doivent être routés, c'est-à-dire qui sont destinés à une autre machine que celle sur laquelle s'exécute Netfilter.
- **`INPUT`**, qui concerne les paquets destinés aux *sockets* locaux, c'est-à-dire qui sont destinés à la machine sur laquelle s'exécute Netfilter.
- **`OUTPUT`**, qui concerne les paquets générés localement, par la machine sur laquelle s'exécute Netfilter.
- **`POSTROUTING`**, qui concerne les paquets sur le point de partir, après routage ou génération locale.

Le schéma 2 montre les instants de traitement des paquets lors de leur traversée du pare-feu. Notons que les règles ne peuvent être ajoutées que lorsque un disque de couleur est présent sur ce schéma. En conséquence :
- la table `nat` ne possède pas de chaîne `FORWARD` ou `INPUT`.
- la table `filter` ne possède pas de chaîne `PREROUTING` ou `POSTROUTING`.

Pour ajouter une règle, il est donc nécessaire de préciser la table (si on ne la précise pas, `filter` est utilisée par défaut) et une chaîne. Lorsqu'un paquet traverse le pare-feu, il parcourt les règles de chaque chaîne dans l'ordre.

Le chemin emprunté par le paquet varie selon si celui-ci est destiné à la machine sur laquelle s'exécute Netfilter ou si celui-ci est destiné à une autre machine. L'ordre de parcourt des chaînes de chaque table varie donc en conséquence. Ici aussi, le schéma 2 montre les instants de traitement des paquets.

6.2 Fonctionnement des chaînes

Lorsqu'un paquet passe par une chaîne d'une table donnée dans le traitement, chaque règle de cette chaîne est étudiée :
- Si le paquet ne correspond pas à la règle qui est définie, alors la règle suivante est examinée.
- Si le paquet correspond, une action est réalisée sur le paquet. Cette action est appelée "**cible**". La suite dépend alors de la cible de cette règle mais en général, les règles suivantes ne sont pas traitées.

Lors de la définition d'une règle, les **cibles** suivantes peuvent être appliquées et entraînent une fin de traitement du paquet dans la chaîne courante :
- **`ACCEPT`** : laisse passer le paquet et passe à la chaîne suivante.
- **`DROP`** : élimine le paquet. Fin complète du traitement.

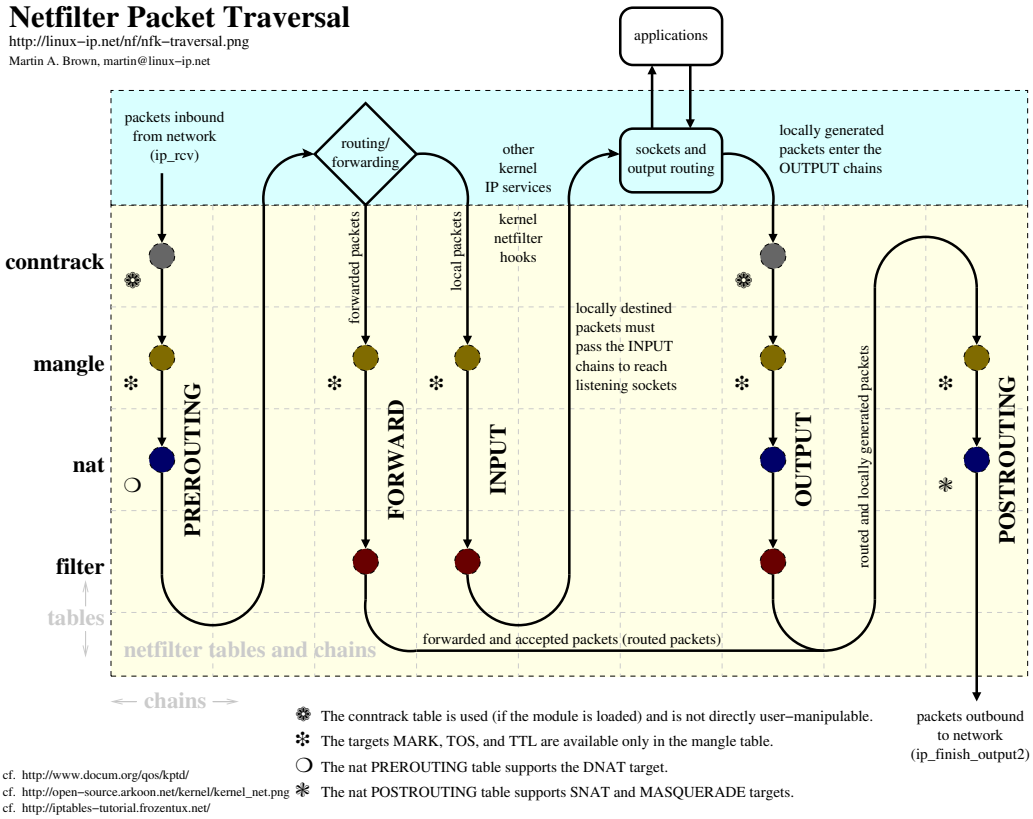

FIGURE 2 – Traversée des tables et chaînes dans Netfilter

— **`REJECT`** : élimine le paquet mais, contrairement à `DROP`, envoie un message à l'émetteur en indiquant la raison. Fin complète du traitement.
Dans le cas d'un `REJECT`, nous devons stipuler l'option `--reject-with` qui stipule la raison. Par exemple : `icmp-port-unreachable` ou `tcp-reset`.

D'autres cibles sont possibles, comme par exemple `DNAT`, `SNAT` et `MASQUERADE` pour faire de la traduction d'adresse, ou `NFLOG` pour faire de la journalisation. Contrairement à `ACCEPT`, `DROP` et `REJECT`, ces cibles n'entraînent pas l'arrêt du traitement de la chaîne. Après leur application, la règle suivante de la chaîne est alors étudiée.

Lorsque toutes les règles de la chaîne ont été étudiées, si aucune cible n'a mis fin au traitement, alors une action par défaut est appliquée sur le paquet. Cette action est définie par la **politique par défaut**. Nous pouvons attribuer la cible `ACCEPT` ou `DROP` à cette politique par défaut.

6.3 Quelques commandes pour `iptables`

Dans cette section sont listées quelques commandes pour `iptables`, utiles à l'ajout de règles dans les tables/chaînes de Netfilter et à la définition des cibles.

6.3.1 Interaction avec les tables/chaînes

`-t, --table <table>` : Sélectionne la table à modifier. Si la table n'est pas spécifiée, alors la table par défaut est `filter`.

```
iptables -t filter ...
```

`-j, --jump <cible>` : Exécute l'action définie par `<cible>`.

```
iptables ... -j ACCEPT
```

`-A, --append <chaine>` : Ajoute la règle à la fin de la chaîne spécifiée.

```
iptables -A INPUT ...
```

`-I, --insert <chaine> [<index>]` : Permet d'ajouter une règle dans un endroit spécifié de la chaîne. Si aucun chiffre n'est spécifié à la suite, la règle est ajoutée au début de chaîne.

```
iptables -I INPUT 1 --dport 80 -j ACCEPT
```

`-D, --delete <chaine> [<index>|<regle>]` : Permet de supprimer une chaîne ou une simple règle. On peut l'utiliser de 2 manières, soit en spécifiant le numéro de la chaîne a supprimer, soit en spécifiant la règle à retirer.

```
iptables -D INPUT 1
iptables -D INPUT --dport 80 -j DROP
```

`-F, --flush <chaine>` : Permet de vider toutes les règles d'une chaîne. Si chaîne n'est pas précisé, cela efface toutes les chaînes de la table.

```
iptables -F INPUT
```

`-X, --delete-chain <chaine>` : Permet d'effacer une chaîne. Si chaîne n'est pas précisé, cela supprime toutes les chaînes de la table.

```
iptables -X
```

`-L, --list <chaine>` : Permet d'afficher les règles.

```
iptables -L       # Affiche toutes les regles des chaines de la table filter
iptables -L INPUT # Affiche toutes les regles de la chaine INPUT (table filter)
```

`-P, --policy <cible>` : Permet de spécifier au noyau la politique par défaut d'une chaîne : `ACCEPT` ou `DROP`.

```
iptables -P INPUT DROP
```

6.3.2 Définition des règles

`-s, --source <adresse>[/masque]` :
Adresse source du paquet. Peut être une plage d'adresses si `/masque` est précisé.

`-d, --destination <adresse>[/masque]` :
Adresse destination du paquet. Peut être une plage d'adresses si `/masque` est précisé.

`-i, --in-interface <interface>` :
Interface d'entrée du paquet.

`-o, --out-interface <interface>` :
Interface de sortie du paquet.

`-p, --protocol <protocole>` :
Protocole du paquet : `icmp`, `udp`, `tcp`, etc.

`-p, --protocol tcp|udp --sport <ports>` :
Ports source. Valeur unique ou plage au format `start:end`.

`-p, --protocol tcp|udp --dport <ports>` :
Ports destination. Valeur unique ou plage au format `start:end`.

`-p, --protocol tcp --tcp-flags <masque> <flags>` :
Flags qui doivent être présents dans le paquet TCP. La première chaîne correspond au masque : la liste de *flags* à examiner. La deuxième chaîne précise lesquels doivent être présents.

Exemple d'utilisation :

```
iptables -t filter -A FORWARD -i eth1 -s 192.168.1.0/24 -d 192.168.2.0/24 -p tcp --dport 23 -j ACCEPT
```

Dans cette exemple, nous ajoutons une règle dans la chaîne `FORWARD` (`-A`) de la table (`-t`) `filter`.
Cette règle s'applique à tous les paquets TCP (`-p tcp`) qui sont routés vers une autre machine (`FORWARD`), qui arrivent par l'interface (`-i`) `eth1`, dont l'adresse IP source (`-s`) est une adresse du sous-réseau 192.168.1.0/24, dont l'adresse IP de destination (`-d`) est une adresse du sous-réseau 192.168.2.0/24 et dont le port de destination (`--dport`) est 23.
Si un paquet traverse la chaîne `FORWARD` de la table `filter` et correspond à ces critères, alors il est accepté (cible `-j ACCEPT`).

Autre exemple d'utilisation :

```
iptables -t filter -L
```

Cette commande vous permet de visualiser toutes les règles que vous avez configurées dans la table `filter`.

7 Configuration de base du pare-feu

Dans cette section, nous configurons le pare-feu de manière basique pour protéger le sous-réseau interne. Dans toutes les questions, l'objectif est de se référer au manuel de `iptables` ou à la section 6.3 pour ajouter/supprimer des règles de pare-feu. Le manuel de `iptables` est accessible avec la commande suivante :

```
man iptables
```

Pour éviter de devoir retaper de nombreuses commandes à chaque fois que nous faisons une erreur et pour enregistrer notre travail au fur et à mesure, nous créons un fichier contenant les commandes `iptables` à exécuter. Nous avons seulement besoin d'exécuter le contenu de ce fichier pour appliquer les nouvelles règles, et ce à chaque modification. Ce fichier est accessible sur le `firewall` avec le chemin `/rw/firewallrc`.

Ensuite, à chaque nouvelle question, nous ajoutons de nouvelles commandes `iptables` à la fin de ce fichier et nous l'exécutons à nouveau.

7.1 Restauration de la configuration initiale

Question 12 Dans un nouveau terminal, prendre le contrôle du `firewall` et éditer le fichier `/rw/firewallrc` avec votre éditeur de texte préféré.
Observer son contenu.

```
debian@myhostname:~$ sudo lxc-attach firewall
root@firewall:/$ nano /rw/firewallrc
```

Question 13 Dans un nouveau terminal, prendre de nouveau le contrôle du `firewall` et exécuter le contenu du fichier `/rw/firewallrc` dans votre *shell* courant.
Vérifier qu'aucune erreur n'est retournée.

```
debian@myhostname:~$ sudo lxc-attach firewall
root@firewall:/$ source /rw/firewallrc
```

À partir de maintenant, nous éditons le fichier `/rw/firewallrc` dans le premier terminal et nous l'évaluons dans le second terminal. À chaque question, lorsqu'une règle est ajoutée/modifiée, nous **vérifions l'effet de la règle** en utilisant les machines `dmz`, `interne` et `internet` (avec *netcat* ou *telnet*). Les 3 *wireshark* permettent de voir passer les paquets (ou pas, s'ils sont bloqués) et nous **redémarrons la capture** à chaque vérification.

Question 14 Afin de ne pas avoir à taper les adresses IP et masques de chaque sous-réseau à chaque nouvelle commande, définir des variables `DMZ`, `INTNET` et `EXTNET`, contenant respectivement les masques de sous-réseau de la DMZ, du réseau interne et du réseau externe (internet).
Se référer au schéma de la figure 1 pour obtenir les valeurs.

```
DMZ="192.168.1.0/24"
INTNET="192.168.2.0/24"
EXTNET="192.168.3.0/24"
```

Question 15 De la même manière, pour ne pas avoir à réfléchir sur quelle interface du `firewall` est connectée à quel sous-réseau, définir des variables `DMZIF`, `INTIF` et `EXTIF`, contenant les interfaces `eth0`, `eth1` et `eth2`.
Se référer au schéma de la figure 1 pour obtenir leurs adresses IP et exécuter la commande `ip address` sur le `firewall` pour obtenir les correspondances.

```
root@firewall:/$ ip address
```

```
DMZIF=eth0
INTIF=eth1
EXTIF=eth2
```

Dans la suite du TP, nous utiliserons ces variables dans les commandes `iptables` que nous ajouterons au fichier `/rw/firewallrc`.

Question 16 Ajouter au fichier `/rw/firewallrc` les commandes pour supprimer toutes les chaînes autres que celles par défaut. Cette suppression doit être réalisée dans les tables `filter`, `nat` et `mangle`.
Ajouter également les commandes pour vider les chaînes restantes.

```
iptables -t filter -F
iptables -t filter -X
iptables -t nat -F
iptables -t nat -X
iptables -t mangle -F
iptables -t mangle -X
```

Question 17 Tester ce fichier en ajoutant une règle à la main et en exécutant son contenu par la suite.
Pour chacune des tables `filter`, `nat` et `mangle`, utiliser la commande `iptables -t <table> -L` pour vérifier que les chaînes sont bien vides.

```
root@firewall:/$ iptables -t filter -A FORWARD -i eth1 -s 192.168.1.0/24 -p tcp --dport 23 -j ACCEPT
root@firewall:/$ iptables -t filter -L
root@firewall:/$ source /rw/firewallrc
root@firewall:/$ iptables -t filter -L
```

7.2 Politique par défaut

L'option `-P` (politique, ou *policy*) permet de dire quelle est la cible (c'est-à-dire l'action) par défaut pour une chaîne et une table donnée si un paquet n'a été ni accepté ni éliminé en fin de chaîne.

Il y a donc deux philosophie pour configurer un pare-feu :

— soit par liste noire : la politique par défaut est d'accepter tous les paquets qui ne correspondent à aucune règle, puis d'ajouter une règle pour chaque ensemble de paquets que nous souhaitons éliminer.

— soit par liste blanche : la politique par défaut est de d'éliminer tous les paquets qui ne correspondent à aucune règle, puis d'ajouter une règle pour chaque ensemble de paquets que nous souhaitons accepter.

Dans ce TP, nous procédons à l'établissement d'une liste blanche pour renforcer la sécurité. Nous n'autorisons les accès qu'à des services particuliers, que nous avons identifiés.

Question 18 Changez la cible par défaut de `ACCEPT` à `DROP` pour la chaine `FORWARD` de la table `filter`. Testez à nouveau les communications par *ping* et *ssh*. Vous ne devriez plus pouvoir communiquer entre les réseaux mais tous les ordinateurs devraient pouvoir échanger avec le routeur.

```
iptables -t filter -P FORWARD DROP
```

Question 19 Changez aussi la politique à `DROP` pour les chaînes `INPUT` et `OUTPUT` de la table `filter`. Testez à nouveau. Tout devrait être coupé.

```
iptables -t filter -P INPUT   DROP
iptables -t filter -P OUTPUT  DROP
```

Question 20 Depuis le `firewall`, essayer d'échanger des paquets avec soi-même. Que peut-on observer ?

```
root@firewall:/$ ping localhost
root@firewall:/$ ssh root@localhost
```

Question 21 En éliminant par défaut tous les paquets, le `firewall` ne peut plus communiquer avec lui-même. Ajoutez une règle à la table `filter` pour que tous les paquets de la boucle locale (*loopback*) soient acceptés dans les deux sens. Tester. *Ping* et *ssh* devraient passer en local mais ne pas passer depuis les autres machines vers le `firewall`. Il ne devrait pas être non plus possible d'envoyer des paquets depuis le `firewall` vers l'extérieur.

```
iptables -t filter -A INPUT  -i lo -j ACCEPT
iptables -t filter -A OUTPUT -o lo -j ACCEPT
```

7.3 Serveur HTTP de la DMZ

Un serveur HTTP est hébergé sur la DMZ. Celui-ci doit pouvoir être accessible publiquement. Nous devons donc configurer notre pare-feu pour que n'importe quelle machine, depuis le réseau interne ou depuis internet, puisse y accéder.

Dans un premier temps, nous suivons une approche simpliste et pas très sécurisée pour réaliser cette tâche, ensuite nous introduisons la bonne manière de faire.

Question 22 Ajouter des règles à la table `filter` pour autoriser les paquets uniquement allant vers la DMZ en HTTP (port de destination 80).

```
iptables -t filter -A FORWARD -d ${DMZ} -p tcp --dport 80 -j ACCEPT
```

Question 23 Essayer d'initier une connexion depuis un autre réseau. Cela ne devrait pas fonctionner (les paquets sortants ne sont pas autorisés).

```
root@internet:/$ echo "quit" | telnet dmz 80
```

Question 24 Ajouter des règles à la table `filter` pour autoriser les paquets dans l'autre sens (venant de la DMZ et ayant pour port source le port 80).

```
iptables -t filter -A FORWARD -s ${DMZ} -p tcp --sport 80 -j ACCEPT
```

Question 25 Essayer de nouveau d'initier une connexion depuis un autre réseau. Cela devrait désormais fonctionner.

```
root@internet:/$ echo "quit" | telnet dmz 80
```

Attaque du réseau interne

Rappelons que, par définition, la DMZ est démilitarisée, donc peu voir pas sécurisée. Ce sous-réseau doit donc être considéré comme potentiellement malveillant vis-à-vis du réseau interne.
Imaginons désormais que nous soyons un adversaire placé sur une machine de la DMZ et tentons de tirer profit du fait que le pare-feu autorise la transmission des paquets sortants depuis le port 80.
Pour cela, nous allons commencer par utiliser un outil de manipulation de paquets : *scapy* [1].

Question 26 Prendre le contrôle de la DMZ et exécuter *scapy*. Construire un paquet TCP/IP avec de la donnée quelconque. Définir le contenu du paquet comme suit :
- adresse IP source : adresse de la machine de la DMZ
- adresse IP destination : adresse de la machine sur le réseau interne
- port source : port 80
- port destination : port *ssh* (22)

Terminer par envoyer le paquet sur le réseau.

```
debian@myhostname:~$ sudo lxc-attach dmz
root@dmz:/$ scapy
# ...
>>> thePacket = IP() / TCP() / "the content of my payload"
>>> thePacket.src = "192.168.1.1"
>>> thePacket.dst = "192.168.2.1"
>>> thePacket.sport = 80
>>> thePacket.dport = 22
>>> send(thePacket)
```

Question 27 Observer sur le réseau le transit du paquet à l'aide de *wireshark*.
Pouvez vous dire pourquoi ce comportement est extrêmement problématique ? Si ce n'est pas le cas, faites appel à l'enseignant.

Question 28 La dernière règle, que nous venons d'ajouter au pare-feu, compromet la sécurité du réseau interne dès qu'un adversaire réussit à s'introduire dans la DMZ.
Pour se convaincre de cette compromission, prendre le contrôle de la DMZ et tenter de faire un *ssh* sur la machine du réseau interne.
Que peut on observer à l'aide de *wireshark* sur le bridge `br-dmz` ? Quel est le port source utilisé ?
Que peut on observer sur le bridge `br-interne` ? Pourquoi ?

Dans *openssh*, les ports source TCP sont tirés au hasard par *linux* au delà de 1024. Or notre pare-feu n'autorise les paquets sortants de la DMZ que si ceux-ci ont comme port source 80. Nous avons donc l'impression qu'il n'est pas possible de faire un *ssh* sur la machine du réseau interne depuis la DMZ car le port source ne peut **jamais** être 80.

Cependant, si la DMZ est compromise, un adversaire peut instrumenter Netfilter pour que les paquets des connexions *ssh* soient modifiés et utilisent le port source 80.

Question 29 Prendre le contrôle de la DMZ et ajouter une règle à la chaîne `POSTROUTING` de la table `nat` : pour tous les paquets TCP avec pour IP de destination une adresse sur le réseau interne et comme port de destination le port 22 (*ssh*), modifier le port source vers 80.

```
root@dmz:/$
  iptables -t nat -A POSTROUTING -d 192.168.2.1/24 -p tcp --dport 22 -j MASQUERADE --to-ports 80
```

Question 30 Tenter de nouveau de réaliser un *ssh* depuis la DMZ vers le réseau interne.
Que peut on observer à l'aide de *wireshark* sur le bridge `br-dmz` ? Quel est le port source utilisé ?

1. `https://scapy.net/`

Question 31 Sur la DMZ, réinitialiser la chaîne `POSTROUTING` de la table `nat` en effaçant toutes les règles. Vérifier que nous obtenons bien le comportement précédent en tentant de réaliser un *ssh* : le port source doit être supérieur à 1024.

```
root@dmz:/$ iptables -t nat -F
```

8 Suivi des connexions

Le suivi des connexions est la capacité de créer et maintenir des informations sur les connexions, comme les adresses, ports, types de protocole, l'état et la durée de la connexion, etc. Le suivi de connexion est, bien sûr, propre aux pare-feux à états (*stateful*).

Dans Netfilter, c'est le module du noyau linux ***conntrack*** qui embarque les fonctionnalités de suivi de connexion et de pare-feu à état. Dans les distributions GNU/Linux modernes, ce module est chargé par défaut dans le noyau linux.

Si *conntrack* n'est pas chargé dans le noyau Linux, les commandes suivantes permettent de l'ajouter :

```
modprobe ip_conntrack
modprobe ip_conntrack_ftp
```

La commande `iptables` interagit directement avec le module *conntrack* lorsque le nom de son extension est précisé dans l'option `-m`, `--match`. Ensuite, l'état du pare-feu à état peut être sélectionné avec l'option spécifique à *conntrack* : `--ctstate`. Plusieurs états peuvent être stipulés lorsqu'ils sont séparés par une virgule. Les différents états possibles sont :

- **`NEW`**, le paquet doit être le premier d'une nouvelle connexion.
- **`ESTABLISHED`**, le paquet doit appartenir à une connexion déjà établie.
- **`RELATED`**, le paquet ne doit pas appartenir à une connexion déjà établie mais doit avoir une relation avec une autre connexion déjà établie (ICMP, communication de données FTP après une connexion de contrôle établie, etc.).

D'autres états sont possibles (`INVALID`, `UNTRACKED`) mais nous ne les utiliserons pas dans ce TP.

La notion de connexion dans `iptables` est assez large et existe pour tout protocole où il y a une réponse. C'est au moment de recevoir la première réponse à un paquet `NEW` que l'état du pare-feu passe à `ESTABLISHED`. Comme le module *conntrack* agit avant toute chaîne (voir la figure 2), les paquets sont traités comme `ESTABLISHED` à partir du premier paquet de réponse inclus.

Le suivi de connexions nous permet d'autoriser des connexions dans un sens donné de façon beaucoup plus simple et sûre qu'avec des règles dites *stateless* (sans considérer l'état du pare-feu) comme nous l'avons fait dans la section précédente.

L'idée est d'autoriser par défaut tous les paquet lorsque le pare-feu est dans l'état `RELATED` ou `ESTABLISHED`, quelque soit le réseau, les adresses, les port source et destination. Ensuite, nous appliquons les règles de filtrages sur les paquets lorsque le pare-feu est dans l'état `NEW`. Ces règles vont autoriser des connexions dans un sens entre deux points pour les paramètres souhaités (adresses IP et ports source et destination, protocole, etc.). Elles permettront de déterminer quelles initialisations nous autorisons, et la règle générale assurera que ce qui découle de notre autorisation est accepté tandis que le reste est abandonné.

Question 32 Ajouter deux commandes `iptables` pour supprimer les deux règles qui autorisent les connexions entrantes et sortantes vers/depuis la DMZ sur le port 80.

```
iptables -t filter -D FORWARD -d ${DMZ} -p tcp --dport 80 -j ACCEPT
iptables -t filter -D FORWARD -s ${DMZ} -p tcp --sport 80 -j ACCEPT
```

Tester et vérifier que nous ne pouvons plus joindre la DMZ sur le port 80 depuis internet ou le réseau interne.

```
root@internet:/$ echo "quit" | telnet dmz 80
```

Question 33 Ajouter une règle à la chaîne `FORWARD` de la table `filter` qui accepte tous les paquets lorsque le pare-feu se situe dans l'état `RELATED` ou `ESTABLISHED`.

```
iptables -t filter -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
```

Tester et vérifier que nous ne pouvons toujours pas joindre la DMZ sur le port 80 depuis internet ou le réseau interne.

Question 34 Ajouter une règle à la chaîne `FORWARD` de la table `filter` qui accepte les paquets à destination de la DMZ sur le port 80 lorsque le pare-feu se situe dans l'état `NEW`.

```
iptables -t filter -A FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport 80 -j ACCEPT
```

Tester et vérifier que nous pouvons de nouveau joindre la DMZ sur le port 80 depuis internet ou le réseau interne.

Question 35 Reproduire l'attaque précédente du réseau interne depuis la DMZ (à l'aide de *scapy* ou d'une instrumentation de Netfilter) et vérifier que nous sommes protégé contre celle-ci.

8.1 Trafic réseau vers la DMZ

Maintenant que nous avons un moyen sécurisé d'accepter les paquets ayant une relation avec une connexion déjà établie, nous pouvons configurer notre pare-feu pour que tous les services publiques de la DMZ soient accessibles.

Question 36 Ajoutez une règle à la table `filter` pour autoriser les connexions vers la DMZ en HTTPS (TCP port 443).
Tester et vérifier que nous pouvons joindre la DMZ depuis internet ou le réseau interne.

```
iptables -t filter -A FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport 443 -j ACCEPT
```

Reproduire l'attaque précédente du réseau interne depuis la DMZ (à l'aide de *scapy* ou d'une instrumentation de Netfilter), en utilisant le port 443, et vérifier que nous sommes protégé contre celle-ci.

Question 37 Ajoutez une règle à la table `filter` pour autoriser les connexions vers la DMZ en SMTP (TCP port 25).
Tester et vérifier que nous pouvons joindre la DMZ depuis internet ou le réseau interne.

```
iptables -t filter -A FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport 25 -j ACCEPT
```

Question 38 Ajoutez des règles à la table `filter` pour autoriser les connexions vers la DMZ en DNS (port 53), et ce en TCP et en UDP.
Tester et vérifier que nous pouvons joindre la DMZ depuis internet ou le réseau interne.
Rappel : en UDP, nous devons simuler manuellement le résolveur DNS à l'aide d'une écoute de *netcat*. Voir la section 5.1 pour plus de détails.

```
iptables -t filter -A FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport 53 -j ACCEPT
iptables -t filter -A FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p udp --dport 53 -j ACCEPT
```

Question 39 Autoriser le *ping* (protocole ICMP) vers la DMZ et sa réponse.
Tester et vérifier le fonctionnement depuis internet et le réseau interne.

```
iptables -t filter -A FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p icmp -j ACCEPT
```

8.2 Trafic réseau vers le réseau interne

Sur le réseau interne, un serveur SMTP est également disponible. Celui-ci doit pouvoir être contacté pour envoyer des mails seulement depuis les machines de la DMZ (pas depuis internet).

Question 40 Ajoutez une règle à la table `filter` pour autoriser les connexions vers le réseau interne en SMTP (TCP port 25) seulement depuis la DMZ.
Tester et vérifier que nous pouvons joindre le réseau interne depuis la DMZ.
Tester et vérifier que nous ne pouvons pas joindre le réseau interne depuis internet.

```
iptables -t filter -A FORWARD -m conntrack --ctstate NEW -s ${DMZ} -d ${INTNET} \
  -p tcp --dport 25 -j ACCEPT
```

8.3 Filtrage *ingress/egress* (règles anti-*spoofing*)

Jusqu'à présent, nous avons écrit nos règles de filtrage en nous basant sur les adresses IP source et destination contenus dans les paquets, ainsi que les ports source et destination. Or, il est possible pour un adversaire de modifier son adresse IP source pour tromper le pare-feu, voir même de forger des paquets personnalisés (comme nous l'avons fait avec *scapy*).

Pour les services publiques, tels que ceux qui s'exécutent sur la DMZ, il ne semble pas y avoir de problème. Par contre, pour les services à accès restreint, comme ceux qui s'exécutent sur le réseau interne, il est possible pour un adversaire de transmettre des paquets depuis internet.

Question 41 Prendre le contrôle de la machine sur internet et modifier son adresse IP pour obtenir une adresse dans le sous-réseau de la DMZ.

```
debian@myhostname:~$ sudo lxc-attach internet
root@internet:/$ ip address add 192.168.1.2 dev eth0
root@internet:/$ ip address del 192.168.3.1/24 dev eth0
```

Vérifier que la nouvelle adresse est correcte.

```
root@internet:/$ ip address
```

Question 42 Tenter une connexion vers le réseau interne en SMTP (TCP port 25) depuis internet.
Que peut on observer ? À quelle classe d'attaques sommes nous vulnérable ?

```
root@internet:/$ echo "quit" | telnet interne 25
```

Question 43 Restaurer la configuration d'origine de la machine sur internet : modifier son adresse IP et son masque sous-réseau tel qu'ils étaient avant.

```
root@internet:/$ ip address add 192.168.3.1/24 dev eth0
root@internet:/$ ip address del 192.168.1.2/32 dev eth0
```

Vérifier le fonctionnement en établissant une connexion vers la DMZ.

```
root@internet:/$ echo "quit" | telnet dmz 25
```

Durant cette expérience, nous observons que les paquets transmis depuis internet ne sont pas filtrés par le pare-feu. En effet l'adresse source correspond à une adresse dans le sous-réseau de la DMZ, l'adresse destination est celle du réseau interne et le port destination est le port 25. Le port source, en revanche, est choisi aléatoirement.

Le réseau interne répond à ce paquet. Dans sa réponse, les adresse source et adresse destination sont inversées par rapport à la requête ; les port source et port destination sont également inversés.

La machine sur internet ne reçoit pas de réponse. La raison est que l'adresse destination de la réponse est une adresse sur le sous réseau de la DMZ. Le routeur doit donc rediriger le paquet vers la DMZ, pas vers internet. Il ne sait d'ailleurs pas à qui l'adresser car, sur ce sous-réseau, aucune machine ne répond à l'adresse IP 192.168.1.2. Si nous retirons les filtres dans *wireshark*, nous pouvons observer sur le bridge `br-dmz` des paquets ARP comme ceci : `Who has 192.168.1.2? Tell 192.168.1.254`.

Si nous avions choisi une adresse IP existante (192.168.1.1), alors nous aurions vu passer le paquet réponse à destination de la DMZ. En effet, même si le port source de ce paquet est celui qui a précédemment été choisi aléatoirement, le paquet est une réponse à une connexion qui s'établit, le pare-feu est donc dans l'état `ESTABLISHED`. Quoi qu'il advienne, la machine sur internet n'aurait pas non plus reçu de réponse. En revanche, la machine de la DMZ (192.168.1.1) aurait répondu à celle du réseau interne pour lui demander d'annuler sa connexion (présence du *flag* `RST` dans le paquet).

Avec un tel comportement, un adversaire peut donc, depuis internet, réaliser une attaque par déni de service en forçant les machines de la DMZ et du réseau interne à communiquer et à annuler leurs connexions.

Question 44 Prendre le contrôle de la machine sur internet et utiliser *scapy* pour forger des paquets TCP avec comme adresse IP source celle de la DMZ, comme adresse IP destination celle du réseau interne et comme port destination le port 25 (SMTP).
Envoyer 10 paquets en changeant le port source à chaque fois et observer le résultat.

```
>>> thePacket = IP() / TCP() / "please talk to each other"
>>> thePacket.src = "192.168.1.1"
>>> thePacket.dst = "192.168.2.1"
>>> thePacket.dport = 25
>>> for i in range(0, 10):
...:     thePacket.sport = 1024 + i
...:     send(thePacket)
...:
>>>
```

Afin de se protéger contre cette classe d'attaque, nous pouvons ajouter des règles de pare-feu qui consistent à se baser sur l'interface utilisée en plus de l'adresse IP.

Nous pouvons donc demander à notre pare-feu d'abandonner (`DROP`) tous les paquets dont l'adresse IP correspond au sous-réseau de la DMZ mais dont l'interface d'entrée n'est pas `eth0` (interface sur laquelle est connectée ce sous-réseau).

Question 45 Ajouter une règle à la chaîne `FORWARD` de la table `filter` qui abandonne tous les paquets dont l'adresse IP provient de la DMZ mais dont l'interface d'entrée n'est pas celle connectée à la DMZ.
Nous pouvons utiliser l'opérateur négation (`!`) devant l'option qui permet de sélection l'interface d'entrée pour inverser son comportement.

```
iptables -t filter -A FORWARD -s ${DMZ} ! -i ${DMZIF} -j DROP
```

Question 46 Tester en envoyant de nouveau un paquet forgé depuis internet (avec *scapy*) et observer le résultat.
Notre mesure de sécurité a-t-elle fonctionné ? Pourquoi ?

Indice : la réponse se cache dans le résultat de la commande suivante.

```
iptables -t filter -L FORWARD -v
```

Si vous n'avez transmis qu'un seul paquet malveillant avec *scapy* après avoir modifié les règles de pare-feu, alors la liste des règles dans la chaîne `FORWARD` de la table `filter` doit ressembler à quelque-chose comme ceci :

```
Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target prot opt in     out  source          destination
    2    84 ACCEPT all  --  any    any  anywhere        anywhere        ctstate RELATED,ESTABLISHED
# [...]
    1    65 ACCEPT tcp  --  any    any  192.168.1.0/24  192.168.2.0/24  ctstate NEW tcp dpt:smtp
    0     0 DROP   all  --  !eth0  any  192.168.1.0/24  anywhere
```

La première colonne indique le nombre de paquets qui ont correspondu à la règle définie sur une ligne donnée. Dans cet exemple, 2 paquets ont été acceptés car le pare-feu était dans un état `RELATED` ou `ESTABLISHED` et 1 paquet a été accepté car son adresse source provenait de la DMZ et son adresse destination était sur le réseau interne. Nous voyons également qu'aucun paquet n'a été abandonné par la règle que nous venons d'ajouter.

La raison d'un tel comportement est que, pour rappel, lorsqu'un paquet passe par une chaîne d'une table donnée dans le traitement, chaque règle de cette chaîne est étudiée l'une après l'autre (**l'ordre est important**). Si le paquet correspond à la règle, alors une action est réalisée, telle que définie par la cible.

Si la cible est `ACCEPT` ou `DROP`, alors les règles suivantes ne sont pas examinées. Dans le cas d'un `ACCEPT`, Netfilter laisse passer le paquet et passe à la chaîne suivante.

Dans notre exemple, le premier paquet transmis (celui envoyé à l'aide de *scapy*) correspond à l'avant dernière règle. Il est donc accepté et la dernière règle n'est pas traitée car Netfilter passe à la chaîne suivante.

Les deux autres paquets acceptés (car le pare-feu est dans un état `RELATED` ou `ESTABLISHED`) correspondent à la réponse de la machine sur le réseau interne (transmise à la DMZ) et la demande de la DMZ de *reset* la connexion.

D'un point de vue du pare-feu, afin de se protéger contre les attaques par *spoofing* d'adresses IP, il est nécessaire de placer les règles qui abandonnent (`DROP`) les paquets avant les règles les acceptent.

De manière générale, même avec une politique par défaut qui abandonne tout paquet ne correspondant à aucune règle, il est conseillé de placer en premières positions les règles d'abandon supplémentaires.

Question 47 Sur le pare-feu, supprimer la dernière règle que nous venons d'ajouter et insérer une règle identique en première position.

```
iptables -t filter -D FORWARD   -s ${DMZ} ! -i ${DMZIF} -j DROP
iptables -t filter -I FORWARD 1 -s ${DMZ} ! -i ${DMZIF} -j DROP
```

Question 48 Tester de nouveau en envoyant un paquet forgé depuis internet (avec *scapy*) et observer le résultat.
Notre mesure de sécurité a-t-elle fonctionné cette fois-ci ?
Afficher la liste des règles dans la chaîne `FORWARD` de la table `filter` pour s'assurer que c'est bien notre pare-feu qui a bloqué le paquet.

```
iptables -t filter -L FORWARD -v
```

Question 49 Pour sécuriser tout notre réseau contre les utilisateurs malveillants sur tous les sous-réseaux, ajouter deux règles à la chaîne `FORWARD` de la table `filter` qui abandonnent tous les paquets dont l'adresse IP provient d'un sous réseau mais dont l'interface d'entrée n'est pas la sienne.

```
iptables -t filter -I FORWARD 1 -s ${INTNET} ! -i ${INTIF} -j DROP
iptables -t filter -I FORWARD 1 -s ${EXTNET} ! -i ${EXTIF} -j DROP
```

Question 50 Vérifier que nous pouvons toujours réaliser nos accès légitimes :
- depuis internet ou le réseau interne vers la DMZ : ports 25, 53, 80 et 443
- seulement depuis la DMZ vers le réseau interne : port 25

8.4 Administration du routeur/pare-feu

Depuis le début de ce TP, nous prenons le contrôle du pare-feu/routeur à l'aide de `lxc-attach`. C'est-à-dire que nous sous-entendons avoir un accès physique à cette machine. Dans un réseau d'entreprise, le pare-feu/routeur est généralement une machine dépourvue d'entrée/sortie standard (écran, clavier, souris).

L'administrateur réseau prend donc le contrôle du pare-feu/routeur via un accès distant (`ssh`). Dans cette section, nous configurons le pare-feu pour n'autoriser qu'une seule machine à prendre le contrôle du routeur.

Question 51 Autoriser les nouvelles connexions `ssh` (TCP port 22) entrantes au niveau du routeur si elles ont pour origine une machine donnée du réseau interne (par exemple 192.168.2.1).
Attention : les paquets sont à destination du pare-feu/routeur lui-même et ne sont pas transférés à une autre machine. Il ne faut donc pas ajouter cette règle à la chaîne `FORWARD`.

```
iptables -t filter -A INPUT  -m conntrack --ctstate NEW -s 192.168.2.1 -p tcp --dport 22 -j ACCEPT
```

Question 52 Prendre le contrôle de la machine du réseau interne et tenter d'établir une connexion `ssh` sur le pare-feu/routeur.
Que peut on observer ? Pourquoi ?

```
root@interne:/$ ssh root@firewall
```

Nous avons autorisé l'établissement de nouvelles connexions `ssh` entrantes au niveau du pare-feu/routeur si elles ont pour origine une machine du réseau interne. Par contre, nous n'avons pas autorisé le pare-feu/routeur à lui répondre. La connexion `ssh` ne peut donc pas aboutir.

Question 53 Autoriser les paquets sortant du pare-feu/routeur lorsque le pare-feu est placé dans l'état `RELATED` ou `ESTABLISHED`.
Attention : les paquets ont pour source le pare-feu/routeur lui-même et ne sont pas transférés depuis une autre machine. Ici non plus, il ne faut donc pas ajouter cette règle à la chaîne `FORWARD`.

```
iptables -t filter -A OUTPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
```

Question 54 Prendre le contrôle de la machine du réseau interne et tenter de nouveau d'établir une connexion `ssh` sur le pare-feu/routeur.
Que peut on observer ? Pourquoi ?

```
root@interne:/$ ssh root@firewall
```

Pour s'aider dans notre enquête, nous pouvons observer les *flags*[2] tels que `SYN`, `ACK` ou `PSH` :
- `SYN` (*synchronisation*) : établissement d'une nouvelle connexion TCP/IP
- `ACK` (*acknowledge*) : accusé de réception
- `PSH` (*push*) : la donnée doit être transmise à l'application.

La connexion `ssh` n'aboutit toujours pas. Observons les paquets qui transitent sur le réseau entre le pare-feu/routeur et le réseau interne (bridge `br-interne`). Nous pouvons voir, au début, deux paquets (numérotés respectivement 1 et 2 par *wireshark*) :

```
No. Time    Source        Destination   Protocol  Length  Info
1   0.0000  192.168.2.1   192.168.2.254 TCP       74      37040 → 22 [SYN]      #...
2   0.0001  192.168.2.254 192.168.2.1   TCP       74      22 → 37040 [SYN, ACK] #...
```

- un paquet provenant du réseau interne et à destination du routeur. Le port source peut varier aléatoirement. Ce paquet a le *flag* `SYN` levé. Ceci indique une demande de *synchronisation* avec la destination.
- un paquet provenant du routeur à destination du réseau interne. Le port destination est identique au port source du paquet précédent. Ce paquet a également le *flag* `SYN` levé, ainsi que le *flag* `ACK`. Ceci indique un accusé de réception pour que la machine qui communique puisse continuer l'établissement de la connexion (transmettre les paquets suivants).

Nos règles de pare-feu ont autorisé la transmission de ces paquets. Ces deux paquets sont le reflet d'un comportement normal lors de l'établissement d'une connexion (`ssh` ou autre). Observons les paquets suivants. La machine du réseau interne transmet deux nouveaux paquets (numérotés respectivement 3 et 4 par *wireshark*) :

```
3  0.0001  192.168.2.1  192.168.2.254  TCP    66  37040 → 22 [ACK] #...
4  0.0007  192.168.2.1  192.168.2.254  SSHv2  98  Client: Protocol (SSH-2.0-OpenSSH_8.4p1 Debian-5)
```

- un paquet ayant le *flag* `ACK`, qui accuse réception de l'accusé de réception (envoyé par le pare-feu/routeur).
- un paquet dont la *payload*[3] est destinée à l'application `ssh`. Si nous regardons le détail de l'entête TCP, nous pouvons voir que les *flags* `PSH` (*push*) et `ACK` sont présents. La présence du *flag* `PSH` témoigne que la donnée doit être transmise à l'application.

Ici aussi, ces deux paquets sont le reflet d'un comportement normal lors de l'établissement d'une connexion. Dans ce cas, d'une connexion `ssh`.
Les paquets suivants, en revanche, ne reflètent pas un un comportement normal. Ceux-ci apparaissent plusieurs fois sur le réseau et sont des retransmissions ou des paquets dupliqués.

2. *flag* = *drapeau* : information contenue dans l'en-tête TCP qui indique le rôle du paquet.
3. *payload* = *chargement*, le contenu du paquet destiné à l'application. Couche supérieure du modèle OSI. Ceci est indépendant du protocole TCP/IP.

```
No. Time    Source        Destination   Protocol  Length Info
X   X.XXXX  192.168.2.254 192.168.2.1   TCP       74     [TCP Retransmission] 22 → 37040 [SYN, ACK]
X   X.XXXX  192.168.2.1   192.168.2.254 TCP       66     [TCP Dup ACK 4#X]    37040 → 22 [ACK]
X   X.XXXX  192.168.2.1   192.168.2.254 TCP       98     [TCP Retransmission] 37040 → 22 [PSH, ACK]
```

— le pare-feu/routeur envoie des retransmissions du paquet numéroté 2 par *wireshark*.
— la machine du réseau interne envoie des paquets dupliqués depuis le paquet numéroté 3 par *wireshark*.
— la machine du réseau interne envoie également des retransmissions du paquet numéroté 4 par *wireshark*.

Dans cette expérience, le paquet numéroté 1 par *wireshark* correspond à la première règle de pare-feu que nous avons ajoutée dans cette section : acceptation des paquets liés à une nouvelle connexion `ssh` (TCP port 22) entrants au niveau du routeur.
Le paquet numéroté 2 par *wireshark* correspond à la seconde règle de pare-feu que nous avons ajoutée dans cette section : acceptation des paquets sortant du pare-feu/routeur lorsque le pare-feu est placé dans l'état `RELATED` ou `ESTABLISHED`.
Les paquets numérotés 3 et 4 par *wireshark*, en revanche, ne correspondent pas à la première règle car ils sont entrants au niveau du routeur mais ne sont pas liés à une nouvelle connexion. C'est donc la politique par défaut qui décide alors l'abandon de ces paquets.
Toutes les retransmissions et tous les envois des paquets dupliqués sont donc une conséquence que le par-feu ne reçoit pas les paquets numérotés 3 et 4.

Question 55 Proposer une solution pour rendre possible les connexions `ssh` (TCP port 22) sur le routeur sans ouvrir de port en sortie sur celui-ci.

Question 56 Autoriser les paquets entrant dans le pare-feu/routeur lorsque le pare-feu est placé dans l'état `RELATED` ou `ESTABLISHED`.

```
iptables -t filter -A INPUT  -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
```

Question 57 Vérifier qu'il est possible de se connecter en `ssh` sur le pare-feu/routeur depuis la machine du réseau interne ayant comme adresse IP 192.168.2.1.

```
root@interne:/$ ssh root@firewall
```

Question 58 Choisir une adresse IP différente pour la machine du réseau interne et tenter de nouveau d'établir une connexion `ssh`.
Vérifier que la machine ne reçoive aucune réponse.

```
root@interne:/$ ip address add 192.168.2.2/32 dev eth0
root@interne:/$ ip address del 192.168.2.1/24 dev eth0
root@interne:/$ ssh root@firewall
```

Question 59 Restaurer la configuration d'origine de la machine du réseau interne : modifier son adresse IP et son masque sous-réseau tel qu'ils étaient avant.

```
root@interne:/$ ip address add 192.168.2.1/24 dev eth0
root@interne:/$ ip address del 192.168.2.2/32 dev eth0
```

9 Protections contre les attaques classiques

Notre pare-feu peut désormais être administré à l'aide d'un accès distant (`ssh`) depuis une machine sur le réseau interne. Nos règles de filtrage assurent *a priori* des accès aux services hébergés seulement depuis les sous-réseaux légitimes.

Dans cette section, nous partons du principe que ces règles sont suffisantes et nous nous intéressons aux attaques classiques sur les réseaux, ne visant pas d'application en particulier.

9.1 Scans de ports

En informatique, le balayage de ports (*port scanning* en anglais) est une technique servant à rechercher les ports ouverts sur un serveur de réseau.

Cette technique est utilisée par les administrateurs des systèmes informatiques pour contrôler la sécurité des serveurs de leurs réseaux. La même technique est aussi utilisée par les pirates informatiques pour tenter de trouver des failles dans des systèmes informatiques. Un balayage de ports (*port scan* ou *portscan* en anglais) effectué sur un système tiers est généralement considéré comme une tentative d'intrusion, car un balayage de ports sert souvent à préparer une intrusion.

Le balayage de ports est une des activités considérées comme suspectes par un système de détection d'intrusion. Un système de détection d'intrusion peut être réglé à différents niveaux de sensibilité. Un niveau de sensibilité élevé génèrera plus de fausses alertes, un niveau de sensibilité bas risque de laisser passer les balayages effectués par des systèmes sophistiqués comme `nmap` qui disposent de diverses options pour camoufler leurs balayages.

Source : `https://fr.wikipedia.org/wiki/Scanner_de_ports`

`nmap` est un scanner de ports libre créé par Fyodor et distribué par Insecure.org. Il est conçu pour détecter les ports ouverts, identifier les services hébergés et obtenir des informations sur le système d'exploitation d'un ordinateur distant. Ce logiciel est devenu une référence pour les administrateurs réseaux car l'audit des résultats de Nmap fournit des indications sur la sécurité d'un réseau. Il est disponible sous Windows, Mac OS X, Linux, BSD et Solaris.

Source : `https://fr.wikipedia.org/wiki/Nmap`

Question 60 Depuis le sous-réseau internet, utiliser `nmap` pour réaliser un scan de ports sur la DMZ. Nous utiliserons l'option `-r`, qui provoque un scan linéaire, pour faciliter notre étude.
Vérifier que les ports ouverts sont bien détectés.
En combien de temps le scan de ports est réalisé ?

```
root@internet:/$ nmap -r dmz
```

Question 61 Utiliser *wireshark* pour observer le trafic sur le bridge `br-internet` et ré-itérer le scan de ports.
Que peut on observer ? Que fait `nmap` et quels sont les *flags* TCP utilisés ?

Le scan de ports se traduit par un envoi de paquets TCP pour initialiser une connexion (*flag* `SYN`) et par une analyse de la réponse. Le scan est souvent rapide et chaque port est testé un par un.

Une bonne pratique pour limiter les scans de ports est donc de réduire le nombre de paquets reçus. Pour cela, nous limitons dans le temps les paquets TCP ayant un certain *flag*.

Attention, cette pratique ne bloque pas les paquets en fonction de certaines caractéristiques comme nous l'avons fait jusqu'à présent : elle permet d'accepter un certain nombre de paquets ayant ces caractéristiques avant d'abandonner les suivants. Pour cela, nous n'utilisons pas la table `filter`, qui contient les règles de filtrage, mais la table `mangle` qui contient des règles menant à des modifications arbitraires sur les paquets. Nous allons appliquer nos modifications dès l'arrivée des paquets, avant les règles de filtrage : c'est-à-dire dans la chaîne `PREROUTING`.
Note : la table `filter` ne contient d'ailleurs pas de chaîne `PREROUTING` car ceci n'a pas de sens.

Question 62 Avant de modifier les règles de la table `mangle`, ajouter des commandes pour restaurer d'origine les règles et la politique par défaut dans la chaîne `PREROUTING`.
De préférence, placer ces commandes au début du fichier `/rw/firewallrc`.

```
iptables -t mangle -P PREROUTING ACCEPT
iptables -t mangle -F PREROUTING
```

Le module d'extension de Netfilter nommé `limit` permet de limiter dans le temps les paquets selon des caractéristiques choisies, par exemple les paquets TCP ayant un certain *flag*. Il fera correspondre seulement un certain nombre de paquets par seconde, par minute, par heure ou par jour. Une réserve permet de faire correspondre un nombre choisi de paquets avant d'appliquer la limite. Le module d'extension `limit` possède deux arguments optionnels :

- `--limit` suivi d'un nombre, pour spécifier (en moyenne) le nombre maximum de correspondances acceptées par seconde. On peut spécifier son unité explicitement, en utilisant `/second`, `/minute`, `/hour` ou `/day`, ou en abrégé (ainsi `5/second` est identique à `5/s`).
- `--limit-burst` suivi d'un nombre, indique la réserve maximale (ou la salve) avant que la limite ci-dessus n'entre en jeu.

Le comportement par défaut est d'avoir 3 correspondances par heure, avec une réserve de 5.

Question 63 L'option `--tcp-flags` nous permet de faire correspondre les paquets TCP à partir des *flags* qui sont présents dans l'entête. Nous pouvons appliquer un masque (ne regarder que certains *flags*) et vérifier que des *flags* sont présents dans ce que l'on regarde.
Dans la chaîne `PREROUTING` de la table `mangle`, pour les paquets TCP contenant le *flag* `SYN` et entrants par l'interface connectée à internet, appliquer une limite de 5 paquets à accepter par seconde.

```
iptables -t mangle -A PREROUTING -i ${EXTIF} -p tcp --tcp-flags SYN SYN -m limit --limit 5/s -j ACCEPT
```

Question 64 Ré-itérer le scan de ports à l'aide de `nmap`. Combien de temps prend le scan ? Vérifier avec *wireshark* la cohérence.
Notre limitation du nombre de paquets a-t-elle fonctionné ? Pourquoi ?
Indice : la réponse se cache dans le résultat de la commande suivante.

```
iptables -t mangle -L PREROUTING -v
```

Nous avons accepté un certain nombre de paquets par seconde. Pour les paquets qui ne correspondent pas à la règle, Netfilter tente de les faire correspondre avec les règles suivantes. Dans notre cas, c'est la politique par défaut qui s'applique : soit accepter le paquet.

En bref, nous acceptons un certain nombre de paquets par seconde puis nous acceptons les autres. Cela ne limite donc pas grand chose.

Question 65 Modifier la politique par défaut de la chaîne `PREROUTING` de la table `mangle` pour abandonner les paquets qui ne correspondent pas à la règle.

```
iptables -t mangle -P PREROUTING DROP
```

Question 66 Ré-itérer le scan de ports à l'aide de `nmap`. Vérifier que la limite est bien appliquée cette fois ci.

Question 67 Utiliser 2 fois *wireshark* pour observer les bridges `br-internet` et `br-dmz`. Tenter un accès légitime au serveur HTTP de la DMZ depuis internet.
L'accès légitime fonctionne-t-il ? Pourquoi ?

```
root@internet:/$ echo "quit" | telnet dmz 80
```

En changeant la politique par défaut de la table `mangle`, nous abandonnons tous les paquets qui ne correspondent pas à la règle. La DMZ ne peut donc plus répondre car ses paquets n'entrent pas le pare-feu par l'interface connectée à internet : ils ne peuvent pas correspondre à la règle. C'est donc la politique par défaut qui s'applique sur tous les paquets en provenance de la DMZ. Ils sont donc tous abandonnés.

Question 68 Dans la chaîne `PREROUTING` de la table `mangle`, ajouter deux règles pour accepter tous les paquets entrant par les interfaces connectées à la DMZ et au réseau interne.

```
iptables -t mangle -A PREROUTING -i ${INTIF} -j ACCEPT
iptables -t mangle -A PREROUTING -i ${DMZIF} -j ACCEPT
```

Question 69 Vérifier que nous avons corrigé le problème en tentant de nouveau de faire un accès légitime au serveur HTTP de la DMZ depuis internet.

```
root@internet:/$ echo "quit" | telnet dmz 80
```

Question 70 Vérifier que le réseau interne puisse faire des accès légitimes au serveur HTTP de la DMZ.

```
root@interne:/$ echo "quit" | telnet dmz 80
```

Question 71 Utiliser 2 fois *wireshark* pour observer les bridges `br-internet` et `br-dmz`. Tenter un accès légitime au résolveur DNS de la DMZ, en UDP, depuis internet.
L'accès légitime fonctionne-t-il ? Pourquoi ?

```
root@dmz:/$ netcat -v -l -u -s $(hostname --ip-address) -p 53
Bound on dmz 53
```

```
root@internet:/$ netcat -v -u dmz 53
Connection to dmz (192.168.1.1) 53 port [udp/domain] succeeded!

^C
```

En appliquant notre limite, nous n'acceptons, en provenance depuis internet, que les paquets TCP ayant le *flag* `SYN`. Tous les paquets UDP en provenance d'internet sont donc abandonnés.

En réalité, le problème ne se pose pas qu'en UDP, des accès légitimes depuis internet peuvent nécessiter des paquets TCP n'ayant pas le *flag* `SYN`. Également, les paquets `ICMP` (*ping*) sont concernés.

Question 72 Proposer une solution pour toujours limiter le nombre paquets TCP ayant le *flag* `SYN` à 5 par secondes, mais également accepter tous les autres paquets.

Afin de ne pas abandonner tous les paquets différents de ceux que nous souhaitons limiter, une solution consiste à ne pas utiliser une seule règle pour limiter les paquets (comme nous l'avons fait) mais deux règles :

— la première règle est identique à celle que nous avons écrite, elle accepte les paquets dans une certaine limite.

— la seconde règle fait correspondre les mêmes paquets mais a pour cible `DROP`, elle les abandonne.

Nous pouvons ajouter autant de paires de règles que nous souhaitons pour limiter les paquets selon certains critères.

Après avoir ajouté nos paires de règles, nous pouvons ensuite ajouter une dernière règle, qui va accepter tous les paquets que nous ne souhaitons pas limiter. Une autre solution est de modifier la politique par défaut pour accepter les paquets.

Question 73 Afficher la liste des règles présentes dans la chaîne `PREROUTING` de la table `mangle`. Vérifier que la règle qui accepte les paquets avec une limite est la première de la chaîne.

```
iptables -t mangle -L PREROUTING -v
```

Question 74 Toujours dans la chaîne `PREROUTING` de la table `mangle`, insérer en deuxième position une nouvelle règle pour abandonner les paquets qui correspondent à ceux que nous souhaitons limiter.

```
iptables -t mangle -I PREROUTING 2 -i ${EXTIF} -p tcp --tcp-flags SYN SYN -j DROP
```

Question 75 Ajouter une règle pour accepter tous les autres paquets en provenance de l'interface connectée à internet.

```
iptables -t mangle -A PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 76 Tenter de nouveau un accès légitime au résolveur DNS de la DMZ, en UDP, depuis internet. Vérifier que celui-ci fonctionne.

```
root@dmz:/$ netcat -v -l -u -s $(hostname --ip-address) -p 53
Bound on dmz 53
```

```
root@internet:/$ netcat -v -u dmz 53
Connection to dmz (192.168.1.1) 53 port [udp/domain] succeeded!

^C
```

Question 77 Ré-itérer le scan de ports à l'aide de `nmap`. Vérifier que la limite est toujours appliquée.

```
root@internet:/$ nmap -r dmz
```

Si nous avons utilisé `nmap` pour envoyer des paquets TCP ayant le *flag* `SYN`, dans la pratique, il est possible d'envoyer des paquets différents pour réaliser des scans de ports. Par exemple, il est possible d'envoyer des paquets TCP ayant le *flag* `ACK`, `FIN` ou `RST`. Les options `-sA`, `-sF` de `nmap` permettent respectivement de faire un scan avec des paquets TCP ayant le *flag* `ACK` ou `FIN`.

Dans ce cas, l'hôte répond en général un paquet TCP contenant le *flag* `RST` et `nmap` peut en déduire que le pare-feu laisse bien passer les paquets. Il est donc préférable d'ajouter également des paires de règles pour limiter ces paquets.

Question 78 Réaliser un scan de ports avec `nmap` en lui demandant d'envoyer des paquets TCP ayant le *flag* `ACK`. Observer les paquets envoyés sur le bridge `br-internet`.

```
root@internet:/$ nmap -r -sA dmz
```

Question 79 Insérer 3 paires de règles pour limiter les paquets contenant les *flags* `ACK`, `FIN` et `RST`. Attention à bien insérer ces règles avant la règle qui accepte les paquets restants.

```
iptables -t mangle -I PREROUTING 3 -i ${EXTIF} -p tcp --tcp-flags ACK ACK -m limit --limit 5/s -j ACCEPT
iptables -t mangle -I PREROUTING 4 -i ${EXTIF} -p tcp --tcp-flags ACK ACK -j DROP
iptables -t mangle -I PREROUTING 5 -i ${EXTIF} -p tcp --tcp-flags FIN FIN -m limit --limit 5/s -j ACCEPT
iptables -t mangle -I PREROUTING 6 -i ${EXTIF} -p tcp --tcp-flags FIN FIN -j DROP
iptables -t mangle -I PREROUTING 7 -i ${EXTIF} -p tcp --tcp-flags RST RST -m limit --limit 5/s -j ACCEPT
iptables -t mangle -I PREROUTING 8 -i ${EXTIF} -p tcp --tcp-flags RST RST -j DROP
```

Question 80 Vérifier la cohérence de la chaîne `PREROUTING` de la table `mangle`.

```
iptables -t mangle -L PREROUTING -v
```

Le résultat doit ressembler à ceci :

```
Chain PREROUTING (policy DROP 0 packets, 0 bytes)
 pkts bytes target prot opt in   out source   destination
    0     0 ACCEPT tcp  --  eth2 any anywhere anywhere    tcp flags:SYN/SYN limit: avg 5/sec burst 5
    0     0 DROP   tcp  --  eth2 any anywhere anywhere    tcp flags:SYN/SYN
    0     0 ACCEPT tcp  --  eth2 any anywhere anywhere    tcp flags:ACK/ACK limit: avg 5/sec burst 5
    0     0 DROP   tcp  --  eth2 any anywhere anywhere    tcp flags:ACK/ACK
    0     0 ACCEPT tcp  --  eth2 any anywhere anywhere    tcp flags:FIN/FIN limit: avg 5/sec burst 5
    0     0 DROP   tcp  --  eth2 any anywhere anywhere    tcp flags:FIN/FIN
    0     0 ACCEPT tcp  --  eth2 any anywhere anywhere    tcp flags:RST/RST limit: avg 5/sec burst 5
    0     0 DROP   tcp  --  eth2 any anywhere anywhere    tcp flags:RST/RST
    0     0 ACCEPT all  --  eth1 any anywhere anywhere
    0     0 ACCEPT all  --  eth0 any anywhere anywhere
    0     0 ACCEPT all  --  eth2 any anywhere anywhere
```

9.2 Inondations (*flood*) et attaques par déni de service

En informatique, le *flood* est une action généralement malveillante qui consiste à envoyer une grande quantité de données inutiles dans un réseau afin de le rendre inutilisable, par exemple en saturant sa bande passante ou en provoquant le plantage des machines du réseau dont le déni de service est la conséquence possible. *Flood* est un anglicisme désignant une inondation, illustrant ainsi le flux de données ininterrompu et excessif transitant sur le réseau, provoquant alors des dégâts.

Sur IRC ou des chats, le *flood* consiste à envoyer de nombreux messages avec le même mot ou les mêmes lettres sur un groupe ou un canal de discussion sur une courte période de temps, ce qui rend la lecture très difficile, voire impossible, pour les autres participants. Cette technique peut aussi être utilisée contre un serveur Internet, en envoyant un grand nombre de requêtes pour provoquer un trafic important. Le service est alors dégradé pour les autres utilisateurs, dans les cas extrêmes le serveur peut aussi interrompre totalement le service sous l'effet de la charge du *flood*.

De la même manière que contre les scans de ports, nous pouvons réduire l'impact des inondations en utilisant le module `limit` de Netfilter. Les règles que nous avons implémentées précédemment protègent d'ailleurs contre les inondations par paquets TCP contenant les *flags* `SYN`, `ACK`, `RST` ou `FIN`.

Question 81 Par défaut, `scapy` envoie les paquets TCP avec le *flag* `SYN`. Utiliser `scapy` pour envoyer une centaine de paquets TCP vers la DMZ depuis internet.
Vérifier que nos règles précédentes limitent bien le nombre de paquets transmis à seulement 5 par seconde.

```
>>> thePacket = IP() / TCP() / "flood me I'm famous"
>>> thePacket.src = "192.168.3.1"
>>> thePacket.dst = "192.168.1.1"
>>> thePacket.dport = 80
>>> for i in range(0, 100):
...:     thePacket.sport = i
...:     send(thePacket)
...:
>>>
```

Question 82 Nos règles de pare-feu ne limitent pas la transmission de paquets ICMP. Utiliser `scapy` pour envoyer une centaine de paquets ICMP vers la DMZ depuis internet.
Observer le nombre de réponses envoyées par la DMZ.

```
>>> thePacket = IP() / ICMP()
>>> thePacket.src = "192.168.3.1"
>>> thePacket.dst = "192.168.1.1"
>>> for i in range(0, 100):
...:     send(thePacket)
...:
>>>
```

Question 83 La commande `ping`, utilisée pour envoyer des paquets ICMP et traiter leur réponse, possède une option spécifique pour réaliser une inondation : `-f`, pour *flood*.
Utiliser l'option `-f` de `ping` pour inonder la DMZ de paquets ICMP depuis internet (utiliser **ctrl + C** pour stopper l'attaque).

```
root@internet:/$ ping -f dmz
```

Question 84 Modifier les règles de pare-feu pour limiter le nombre de paquets ICMP en provenance d'internet à 5 par seconde.
Attention : il est nécessaire de placer les règles qui limitent avant celle qui accepte les paquets restants.

```
iptables -t mangle -D PREROUTING -i ${EXTIF} -j ACCEPT
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -m limit --limit 5/s -j ACCEPT
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -j DROP
iptables -t mangle -A PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 85 Réaliser, de nouveau, une inondation de la DMZ depuis internet avec la commande `ping`. Observer le nombre de paquets limités par le pare-feu.
Note : la commande `ping` avec l'option `-f` affiche un point lorsqu'un paquet ICMP de type *echo request* ne reçoit pas de réponse.

Des règles similaires peuvent être ajoutées pour tous les types de paquets et ainsi se protéger contre les inondations. Nous en avons donc terminé avec le module `limit` de Netfilter.

Notons un effet de bord : lors de l'affichage de la liste des règles, Netfilter simule une transmission de paquets pour lister les règles qui s'appliquent. En abandonnant des paquets depuis la chaîne `PREROUTING` de la table *mangle*, nous empêchons le bon fonctionnement de l'affichage de la liste.

Question 86 Constater le problème d'affichage de la liste des règles dans la table *filter*.
Attendre pour voir apparaitre les règles petit à petit.

```
root@firewall:/$ iptables -t filter -L
```

Question 87 Nous pouvons résoudre ce problème en acceptant tous les paquets en provenance de l'interface *loopback* dans la chaîne `PREROUTING` de la table *mangle*.
Ajouter une règle pour réaliser cette action.

```
iptables -t mangle -A PREROUTING -i lo -j ACCEPT
```

Question 88 Vérifier que nous avons résolu le problème d'affichage de la liste des règles dans la table *filter*.

```
root@firewall:/$ iptables -t filter -L
```

10 Journalisation : garder une trace

En informatique, le concept d'**historique des événements** ou de **journalisation** désigne l'enregistrement séquentiel dans un fichier ou une base de données de tous les événements affectant un processus particulier (application, activité d'un réseau informatique...). Le journal (en anglais *log file* ou plus simplement *log*), désigne alors le fichier contenant ces enregistrements. Généralement datés et classés par ordre chronologique, ces derniers permettent d'analyser pas à pas l'activité interne du processus et ses interactions avec son environnement. Un fichier texte peut être la structure sous-jacente d'un fichier journal.

La journalisation est une technique utilisée entre autres en sécurité informatique et dans les systèmes de comptabilité et paiement. Dans le cadre de la sécurité, les événements enregistrés seront les accès au système, les modifications de fichiers, etc. On consacre typiquement une ligne par événement, en commençant par le moment exact (date, heure, minute, seconde) où il a eu lieu. La journalisation permet d'effectuer des analyses diverses, généralement statistiques ; de faire des hypothèses sur les dysfonctionnements ou les pertes de performance d'un système. L'accès aux journaux peut contrevenir à certaines exigences de confidentialité, voire de sécurité.

10.1 La cible `NFLOG`

La cible `NFLOG` permet de demander à Netfilter d'envoyer des messages au démon `ulogd` via le noyau Linux. Les messages reçus par `ulogd` depuis le noyau sont journalisés dans un fichier dont le chemin est défini par la configuration de `/etc/ulogd.conf`. Par défaut, ce chemin est `/var/log/ulog/syslogemu.log`.

Pour les distributions GNU/Linux où le démon `ulogd` n'est pas configuré par défaut, il est nécessaire de l'installer et de l'exécuter. Par exemple, sous Debian, les commandes suivantes permettent de réaliser ces tâches :

```
root@firewall:/$ sudo apt install --yes ulogd2
root@firewall:/$ sudo mkdir -p /run/ulog
root@firewall:/$ sudo systemctl start ulogd
```

Dans ce TP, le démon `ulogd` a déjà été configuré pour que les messages soient disponibles dans le fichier suivant :`/var/log/ulog/syslogemu.log`. Si aucun message n'a été transmis, alors ce fichier est vide.

Question 89 Utiliser la commande `tail` pour observer les messages reçus par `ulogd` depuis le noyau sur le pare-feu. Vérifier que le fichier est vide.

```
root@firewall:/$ tail /var/log/ulog/syslogemu.log
```

Note : si le fichier n'est pas vide, nous pouvons exécuter la commande suivante pour vider son contenu :

```
root@firewall:/$ echo "" > /var/log/ulog/syslogemu.log
```

Question 90 L'option `-f` de la commande `tail` permet de suivre (*follow*) les modifications du fichier : au lieu de rendre la main, `tail` montre alors les dernières lignes du fichier et se met en attente de modifications. Dès que quelque chose s'ajoute à la fin du fichier, `tail` affiche les nouvelles lignes.
Utiliser l'option `-f` de la commande `tail` pour suivre les prochains messages reçus par `ulogd` depuis le noyau sur le pare-feu.
Vérifier que `tail` ne nous rende pas la main.

```
root@firewall:/$ tail -f /var/log/ulog/syslogemu.log
```

Question 91 Afin de réaliser un premier test, insérer une règle pour envoyer un message à chaque paquet ICMP reçu : c'est-à-dire dans la chaîne `PREROUTING` de la table `mangle`.
Notons que contrairement aux cibles `ACCEPT` ou `DROP`, la cible `NFLOG` n'entraîne pas l'arrêt du traitement de la chaîne. Après son application, les règles suivantes de la chaîne sont alors toujours étudiées.

```
iptables -t mangle -I PREROUTING 1 -p icmp -j NFLOG
```

Question 92 Depuis internet, envoyer un paquet ICMP à la machine de la DMZ à l'aide de `ping`. Vérifier que le journal est mis à jour avec deux messages :

- un pour la requête ICMP, émise depuis internet,
- l'autre pour la réponse ICMP, émise depuis la DMZ.

```
root@internet:/$ ping -c 1 dmz
```

Question 93 Si le test est concluant, supprimer la règle pour envoyer un message à chaque paquet ICMP reçu. Nous allons désormais nous intéresser à journaliser des évènements spécifiques.

```
iptables -t mangle -D PREROUTING -p icmp -j NFLOG
```

Notre objectif est de garder une trace des évènements importants afin de pouvoir réaliser une étude *a posteriori*. Nous souhaitons donc journaliser les tentatives de connexions `ssh` sur le pare-feu ; mais aussi les *spoofing* d'adresses IP, les inondations et les scans de ports.

10.2 Journaliser les connexions `ssh`

Nous avons précédemment configuré le pare-feu pour permettre une connexion `ssh` vers celui-ci depuis une adresse IP donnée (192.168.2.1). Nous souhaitons ici journaliser les tentatives d'établissement de nouvelles connexions.

Question 94 Ajouter une règle au pare-feu pour journaliser les nouvelles connexions TCP sur le port 22 du pare-feu, depuis l'adresse IP 192.168.2.1.

```
iptables -t filter -A INPUT  -m conntrack --ctstate NEW -s 192.168.2.1 -p tcp --dport 22 -j NFLOG
```

Question 95 Depuis le réseau interne, établir une connexion `ssh` sur le pare-feu.

```
root@interne:/$ ssh root@firewall
```

La tentative de connexion est elle journalisée ? Pourquoi ?

L'ordre des règles est important ! Si le paquet TCP servant à établir une nouvelle connexion correspond à la règle qui dit que celui-ci est accepté, alors les règles suivantes ne sont pas étudiées.

Pour que la nouvelle connexion soit journalisée, il est préférable (voire nécessaire) que les deux règles auxquelles correspond le paquet (celle pour journaliser et celle pour l'accepter) se suivent, et que la règle demandant de journaliser soit étudiée en premier.

Question 96 Déplacer la règle stipulant que nous acceptons les paquets établissant une nouvelle connexion `ssh` sur le pare-feu en dernière position.

```
iptables -t filter -D INPUT -m conntrack --ctstate NEW -s 192.168.2.1 -p tcp --dport 22 -j ACCEPT
iptables -t filter -A INPUT -m conntrack --ctstate NEW -s 192.168.2.1 -p tcp --dport 22 -j ACCEPT
```

Question 97 Tenter de nouveau une connexion `ssh` sur le pare-feu depuis le réseau interne et vérifier que la tentative est bien journalisée.

Question 98 Afficher la liste des règles de la chaîne `INPUT` dans la table `filter`, vérifier que les deux règles discutées précédemment se suivent.
Le résultat obtenu doit ressembler à ceci :

```
root@firewall:/$ iptables -t filter -L INPUT -v
Chain INPUT (policy DROP 9 packets, 3612 bytes)
 pkts bytes target prot opt in  out source   destination
   72  6048 ACCEPT all  --  lo  any any anywhere anywhere
   11  2284 ACCEPT all  --  any any anywhere anywhere    ctstate RELATED,ESTABLISHED
    1    60 NFLOG  tcp  --  any any interne  anywhere    ctstate NEW tcp dpt:ssh
    1    60 ACCEPT tcp  --  any any interne  anywhere    ctstate NEW tcp dpt:ssh
```

10.3 Garder une trace des attaques

Dans la section précédente, nous avons modifié nos règles pour journaliser les tentatives de connexions `ssh` depuis une machine légitime (sur le réseau interne). Ceci se présente sous la forme de deux règles qui se suivent : une dont la cible est `NFLOG` et une dont la cible est `ACCEPT`.

Or, durant tout ce TP, nous avons identifié différentes attaques et avons abandonné (`DROP`) les paquets qui correspondaient à celles-ci. Dans cette section, nous proposons de garder une trace des attaques, nous allons donc ajouter une journalisation avant chaque règle dont la cible est `DROP`.

Inondations ICMP

Dans un premier temps, nous nous intéressons à la journalisation des inondations. Notre premier objectif est de journaliser l'abandon des paquets ICMP après une acceptation d'un nombre limité de ceux-ci.

Question 99 Nous allons donc modifier nos règles dans la chaîne `PREROUTING` de la table `mangle`. Nous allons désormais utiliser 4 règles pour limiter les paquets ICMP :

- une pour accepter les paquets qui ne sont pas des paquets ICMP,
- une pour journaliser les abandons de paquets ICMP,
- une pour accepter des paquets ICMP dans une certaine limite,
- une pour abandonner les paquets ICMP qui excèdent cette limite.

Dans quel ordre ces règles doivent être ajoutées à la chaîne concernée ? Pourquoi ?

Question 100 Commencer par supprimer les règles permettant de limiter les paquets ICMP en provenance d'internet et d'accepter tout paquet dont le trafic n'est pas limité.

```
iptables -t mangle -D PREROUTING -i ${EXTIF} -p icmp -m limit --limit 5/s -j ACCEPT
iptables -t mangle -D PREROUTING -i ${EXTIF} -p icmp -j DROP
iptables -t mangle -D PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 101 À cet instant, le pare-feu abandonne donc tous les paquets ICMP. Pourquoi ?
Vérifier en tentant d'inonder la DMZ depuis internet.

```
root@internet:/$ ping -f -c 20 dmz
```

Nous allons ajouter 4 règles dans cet ordre :

1. nous allons accepter des paquets ICMP dans une certaine limite, cette règle entraîne une fin de traitement du paquet dans la chaîne courante.
2. pour les paquets qui ne correspondent pas à la règle précédente, nous journalisons son trafic si et seulement si c'est un paquet ICMP. La règle suivante de la chaîne sera toujours analysée.
3. nous abandonnons le paquet si et seulement si c'est un paquet ICMP, cette règle entraîne une fin de traitement du paquet dans la chaîne courante.
4. nous acceptons le paquet. Cette règle n'est analysée que si le paquet ne correspond à aucune des règles précédentes. C'est-à-dire si ce n'est pas un paquet ICMP.

Question 102 Ajouter ces 4 règles au pare-feu.

```
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -m limit --limit 5/s -j ACCEPT
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -j NFLOG
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -j DROP
iptables -t mangle -A PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 103 Tenter d'inonder la DMZ depuis internet avec 20 paquets.

```
root@internet:/$ ping -f -c 20 dmz
```

Vérifier qu'un nombre limité de paquets est transmis à la DMZ.
Vérifier que des messages sont ajoutés au journal pour chaque paquet qui a été abandonné. La somme entre le nombre de paquets est transmis à la DMZ et le nombre de messages ajoutés au journal doit être égale à 20.

Inondations TCP avec le *flag* `SYN` / scans de ports

Notre système de journalisation nous permet de garder une trace des inondations avec des paquets ICMP. Or, comme nous l'avons vu précédemment, les inondations peuvent être réalisées avec d'autres types de paquets. Notre objectif suivant est de journaliser l'abandon des paquets TCP avec le *flag* `SYN`, après une acceptation d'un nombre limité de ceux-ci.

Le raisonnement est similaire à la journalisation des paquets ICMP : nous devons glisser une règle qui journalise de trafic des paquets entre celle qui les accepte avec un nombre limité et celle qui les abandonne. Nous devons également veiller à garder en dernière position la règle qui accepte tous les paquets qui ne correspondent à aucune règle (dans la chaîne `PREROUTING` de la table `mangle`, car c'est ici que nous traitons les inondations).

Question 104 Commencer par supprimer 3 règles : celles qui permettent de limiter les paquets TCP avec le *flag* `SYN` en provenance d'internet et celle qui permet d'accepter tout paquet dont le trafic n'est pas limité.

```
iptables -t mangle -D PREROUTING -i ${EXTIF} -p tcp --tcp-flags SYN SYN -m limit --limit 5/s -j ACCEPT
iptables -t mangle -D PREROUTING -i ${EXTIF} -p tcp --tcp-flags SYN SYN -j DROP
iptables -t mangle -D PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 105 Ici aussi, après suppression de ces règles, nous ne pouvons plus communiquer depuis internet, car la politique par défaut dans la chaîne `PREROUTING` de la table `mangle` est d'abandonner les paquets. Nous avons donc cassé l'accès au réseau.
Vérifier en tentant un accès légitime au serveur HTTP de la DMZ depuis internet.

```
root@internet:/$ echo "quit" | telnet dmz 80
```

Question 106 Ajouter 4 règles : celles qui permettent de limiter les paquets TCP avec le *flag* `SYN` en provenance d'internet, celle qui permet de journaliser ceux qui sont abandonnés et celle qui permet d'accepter tout paquet dont le trafic n'est pas limité.

```
iptables -t mangle -A PREROUTING -i ${EXTIF} -p tcp --tcp-flags SYN SYN -m limit --limit 5/s -j ACCEPT
iptables -t mangle -A PREROUTING -i ${EXTIF} -p tcp --tcp-flags SYN SYN -j NFLOG
iptables -t mangle -A PREROUTING -i ${EXTIF} -p tcp --tcp-flags SYN SYN -j DROP
iptables -t mangle -A PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 107 Vérifier que nous avons rétablit l'accès au réseau en tentant un accès légitime au serveur HTTP de la DMZ depuis internet.
Vérifier qu'aucun message n'est ajouté au journal dans ce cas là.

```
root@internet:/$ echo "quit" | telnet dmz 80
```

Question 108 Tenter un scan de ports vers la DMZ depuis internet avec `nmap`, tester seulement les ports compris entre 1 et 100. L'option `-p` permet de spécifier un intervalle de ports à scanner, nous pouvons spécifier le premier port et le dernier séparés par un tiret.
Vérifier à l'aide de *wireshark* que certains paquets sont acceptés dans une certaine limite.
Vérifier que des messages sont ajoutés au journal dans ce cas où les paquets sont abandonnés.

```
root@internet:/$ nmap -p 1-100 -r dmz
```

Nous avons terminé avec la journalisation de l'abandon des paquets TCP avec le *flag* `SYN`. Pour bien faire, nous devrions répéter cette tâche pour tous les paquets que nous avons identifiés et qui peuvent être utilisés pour les inondations ou les scans de ports.

Nous ne réaliserons pas ces tâches ici mais celles-ci sont similaire à ce que nous venons de faire ; juste, nous devons remplacer le *flag* `SYN` par le *flag* `ACK`, puis `RST`, etc.

Attaques bloquées par le filtrage

Notre système de journalisation nous permet de garder une trace des inondations que nous avons identifiées. Nous utilisons la chaîne `PREROUTING` de la table `mangle` pour les traiter.

Lorsque nous avons écrit nos règles de pare-feu, nous avons choisi une philosophie par liste blanche : la politique par défaut est de d'éliminer tous les paquets qui ne correspondent à aucune règle, puis d'ajouter une règle pour chaque ensemble de paquets que nous souhaitons accepter. De ce fait, nous nous sommes protégés contre beaucoup d'attaques que nous n'avons pas explicitement identifiées. Par exemple : nous empêchons les connexions `ssh` vers le réseau interne depuis internet.

Ce sont les chaînes `FORWARD`, `INPUT` et `OUTPUT` de la table `filter` qui nous protègent. Ici aussi, avant d'abandonner les paquets, il est souhaitable de garder une trace en journalisant cet abandon.

Question 109 Tenter une connexion `ssh` vers le réseau interne depuis internet. Cela ne doit pas fonctionner.
Vérifier qu'aucun message n'est ajouté au journal.

```
root@internet:/$ ssh root@interne
```

Question 110 Comme c'est la politique par défaut qui se charge d'abandonner les paquets, il nous suffit simplement de rajouter une règle pour journaliser en fin de chaîne.
Ajouter une règle à la chaîne `FORWARD` de la table `filter` pour journaliser.

```
iptables -t filter -A FORWARD -j NFLOG
```

Question 111 Tenter de nouveau un `ssh` vers le réseau interne depuis internet.
Vérifier qu'un message est ajouté au journal à chaque tentative (le client `ssh` envoie plusieurs requêtes s'il n'obtient pas de réponse).

Question 112 Cette nouvelle règle nous permet de journaliser toutes les attaques auxquelles nous n'avons pas pensées. Tenter une connexion TCP sur un port fermé de la DMZ depuis internet.
Vérifier qu'un message est ajouté au journal à chaque tentative.

```
root@internet:/$ echo "quit" | telnet dmz 999
```

Nous journalisons désormais tous les paquets abandonnés par le pare-feu à destination des sous-réseaux (interne, DMZ, internet). Une dernière étape consiste à journaliser les paquets abandonnés à destination du routeur lui même.

Par exemple, nous souhaitons pouvoir journaliser les tentatives de connexions `ssh` vers le routeur depuis internet.

Question 113 Tenter une connexion `ssh` vers le routeur depuis internet. Cela ne doit pas fonctionner.
Vérifier qu'aucun message n'est ajouté au journal.

```
root@internet:/$ ssh root@192.168.3.254
```

Question 114 Ici aussi, comme c'est la politique par défaut qui se charge d'abandonner les paquets, il nous suffit simplement de rajouter une règle pour journaliser en fin de chaîne.
Ajouter une règle à la chaîne `INPUT` de la table `filter` pour journaliser.

```
iptables -t filter -A INPUT -j NFLOG
```

Question 115 Tenter de nouveau un `ssh` vers le routeur depuis internet.
Vérifier qu'un message est ajouté au journal à chaque tentative.

```
root@internet:/$ ssh root@192.168.3.254
```

Note : cette nouvelle règle nous permet également de voir journaliser des abandons de paquets DHCP (protocole UDP sur les ports source 68 et destination 67, avec pour IP source 0.0.0.0 et IP de destination 255.255.255.255). Ce n'est pas l'objet de ce TP, nous donc ignorons ces paquets.

Nous pouvons donc s'assurer que ces message n'apparaissent pas en les ignorant lors de la journalisation. Par exemple, nous pouvons modifier la dernière règle comme suit :

```
iptables -t filter -D INPUT -j NFLOG
iptables -t filter -A INPUT ! -d 255.255.255.255 -j NFLOG
```

Dernières journalisations : les attaques identifiées lors du filtrage

Un dernier point à noter est que nous avons identifié des attaques particulières dont nous avons filtré les paquets : il s'agit des attaques de *spoofing* d'adresses IP. Les règles qui permettent ce filtrage sont les 3 premières règles de la chaîne `FORWARD` dans la table `filter`.

Pour journaliser l'abandon des paquets réalisé par ces règles, nous devons simplement ajouter une règle à laquelle correspondent les mêmes paquets, dont la cible est `NFLOG`. Ici aussi, la règle qui permet la journalisation doit être placée avant la règle qui permet d'abandonner le paquet.

Question 116 Observer les règles de la chaîne `FORWARD` dans la table `filter`. Vérifier que les 3 premières règles sont les règles anti-*spoofing*.

```
root@firewall:/$ iptables -t filter -L FORWARD -v
```

Question 117 Insérer 3 règles en premières positions où les mêmes paquets vont correspondre et dont la cible est `NFLOG`.

```
iptables -t filter -I FORWARD 1 -s ${DMZ}    ! -i ${DMZIF} -j NFLOG
iptables -t filter -I FORWARD 1 -s ${INTNET} ! -i ${INTIF} -j NFLOG
iptables -t filter -I FORWARD 1 -s ${EXTNET} ! -i ${EXTIF} -j NFLOG
```

Question 118 Prendre le contrôle de la machine sur internet et utiliser *scapy* pour forger un paquets TCP avec comme adresse IP source celle de la DMZ, comme adresse IP destination celle du réseau interne et comme port destination le port 25 (SMTP).
Vérifier qu'un message est ajouté au journal lors de l'envoi du paquet.

```
>>> thePacket = IP() / TCP() / "do you see me?"
>>> thePacket.src = "192.168.1.1"
>>> thePacket.dst = "192.168.2.1"
>>> thePacket.dport = 25
>>> thePacket.sport = 1024
>>> send(thePacket)
```

Idéalement, pour chaque attaque identifiée, c'est-à-dire pour chaque règle dont la cible est `DROP`, la règle précédente doit être identique mais avec pour cible `NFLOG`. Cette tâche doit donc être répétée au besoin dans toutes les chaînes de toutes les tables.

10.4 Déni de service du routeur

Désormais, nous partons du principe que nous journalisons tous les abandons de paquets réalisés par le pare-feu. Cela nous permet de réaliser une étude *a posteriori* des attaques subies par nos sous-réseaux. Le journal est donc stocké dans le fichier `/var/log/ulog/syslogemu.log`.

Si cette pratique de journalisation est efficace, notons qu'elle introduit une faille de sécurité. En effet, notre routeur est désormais vulnérables à des attaques par déni de service. Dans cette section, nous allons voir comment.

Question 119 Effacer le contenu du journal sur le routeur.

```
root@firewall:/$ echo "" > /var/log/ulog/syslogemu.log
```

Question 120 Quelle place occupe le fichier journal sur le disque dur du routeur ?

```
root@firewall:/$ du -h /var/log/ulog/syslogemu.log
```

Question 121 Depuis le réseau internet, tenter une inondation avec 1000 paquets ICMP vers la DMZ.

```
root@internet:/$ ping -f -c 1000 dmz
```

Question 122 Désormais, quelle place occupe le fichier journal sur le disque dur du routeur ?
Quelle place approximative occuperait il si nous avions envoyé 1 milliard de paquets ICMP vers la DMZ ?

```
root@firewall:/$ du -h /var/log/ulog/syslogemu.log
```

Voici une des problématiques de la journalisation : stocker des informations sur le disque dur de la machine implique que celui-ci peut saturer dans le cas d'une journalisation excessive. De ce fait, il est possible que la machine s'arrête de fonctionner. Dans ce cas, le routeur ne remplit plus ses tâches et les sous-réseaux (DMZ et réseau interne) ne peuvent plus communiquer.

Une solution simple consiste à archiver, voire supprimer, les journaux après une certaine durée. C'est d'ailleurs une tâche qui peut être automatisée avec le concept de rotation des journaux. Sous GNU/Linux, la commande `logrotate` permet cette automatisation.

Une solution complémentaire (bien qu'elle ne soit pas suffisante) consiste à limiter l'impact de la journalisation en réduisant leur nombre. De ce fait, un nombre de paquets plus important est nécessaire pour atteindre la saturation du disque.

Question 123 Proposer une solution pour limiter l'impact de la journalisation en termes d'occupation sur le disque.

Pour réduire le nombre de messages dans le journal, nous allons combiner l'utilisation du module `limit` de Netfilter avec celle de la cible `NFLOG`.

Nous allons appliquer ce raisonnement aux règles permettant la journalisation des inondations par paquets ICMP, mais celui-ci doit être appliqué à toutes les règles de journalisation.

Question 124 Commencer par supprimer les 4 règles permettant de limiter et journaliser les paquets ICMP en provenance d'internet et d'accepter tout paquet dont le trafic n'est pas limité.

```
iptables -t mangle -D PREROUTING -i ${EXTIF} -p icmp -m limit --limit 5/s -j ACCEPT
iptables -t mangle -D PREROUTING -i ${EXTIF} -p icmp -j NFLOG
iptables -t mangle -D PREROUTING -i ${EXTIF} -p icmp -j DROP
iptables -t mangle -D PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 125 Ajouter 4 règles identiques en modifiant la journalisation pour que celle-ci ne soit réalisée que 2 fois par seconde (sans compter la réserve).

```
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -m limit --limit 5/s -j ACCEPT
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -m limit --limit 2/s -j NFLOG
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -j DROP
iptables -t mangle -A PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 126 De nouveau, effacer le contenu du journal sur le routeur et observer quelle place occupe le fichier sur le disque.

```
root@firewall:/$ echo "" > /var/log/ulog/syslogemu.log
root@firewall:/$ du -h /var/log/ulog/syslogemu.log
```

Question 127 Ré-itérer l'inondation avec 1000 paquets ICMP depuis le réseau internet vers la DMZ.

```
root@internet:/$ ping -f -c 1000 dmz
```

Question 128 Vérifier que des messages sont toujours présents (en quantité réduite) dans le journal. Désormais, quelle place occupe le fichier journal sur le disque dur du routeur ?

```
root@firewall:/$ du -h /var/log/ulog/syslogemu.log
```

Combiner l'utilisation du module `limit` avec la cible `NFLOG` permet de réduire le nombre de messages dans le journal. Cette méthode vient donc en complément de la rotation des journaux.

Pour que celle-ci soit efficace, elle doit donc être appliquée à toutes les règles dont la cible est `NFLOG`. En particulier, son utilisation est nécessaire lorsque l'on souhaite journaliser des attaques par déni de service.

Nous n'allons pas le faire dans ce TP, mais toutes les règles de journalisation que nous avons précédemment écrites doivent être modifiées en ce sens.

10.5 Niveau d'alerte personnalisé : préfixer les messages

Désormais, nous partons du principe que nous journalisons en faible quantité tous les abandons de paquets réalisés par le pare-feu. Cela nous permet de réaliser une étude *a posteriori* des attaques subies par nos sous-réseaux.

Cependant, malgré la quantité plus ou moins réduite de messages dans le journal, une telle étude reste fastidieuse. En effet, en fonction des informations sur les paquets (protocole, adresses IP source et destination, ports source et destination, *flags*, etc.), il est nécessaire de déterminer la classe d'attaque.

Une fonctionnalité qui serait appréciée serait de rajouter une information dans chaque message pour indiquer cette classe d'attaque lorsque celle-ci est identifiée. C'est dans cette optique que la cible `NFLOG` est accompagnée d'une option `--nflog-prefix`, qui permet de stipuler une information avec laquelle préfixer le message.

Dans cette section, nous allons utiliser cette option pour préfixer certains de nos messages. Nous proposons de préfixer les messages qui journalisent les tentatives de connexion `ssh` légitimes sur le routeur et les inondations par paquets ICMP. Ici aussi, il est vivement conseiller d'appliquer cette méthode à toutes les règles dont la cible est `NFLOG`.

Question 129 Commencer par supprimer 2 règles, celles qui permettent de journaliser et d'accepter des nouvelles tentatives de connexion `ssh` vers le routeur.

```
iptables -t filter -D INPUT -m conntrack --ctstate NEW -s 192.168.2.1 -p tcp --dport 22 -j NFLOG
iptables -t filter -D INPUT -m conntrack --ctstate NEW -s 192.168.2.1 -p tcp --dport 22 -j ACCEPT
```

Question 130 Ajouter 2 règles identiques en modifiant la journalisation pour que celle-ci ajoute un préfixe au message informant que le paquet est une connexion `ssh`.
Penser à ajouter une limite pour se protéger contre les attaques par déni de service sur le routeur.

```
iptables -t filter -A INPUT -m conntrack --ctstate NEW -s 192.168.2.1 -p tcp --dport 22 \
  -m limit --limit 2/s -j NFLOG --nflog-prefix "[INFO ssh]"
iptables -t filter -A INPUT -m conntrack --ctstate NEW -s 192.168.2.1 -p tcp --dport 22 -j ACCEPT
```

Question 131 Vérifier qu'il est possible de se connecter en `ssh` sur le pare-feu/routeur depuis la machine du réseau interne. Observer le journal et constater qu'un message est ajouté, contenant le préfixe `[INFO ssh]`.

```
root@interne:/$ ssh root@firewall
```

Question 132 Nous allons désormais faire de même avec les abandons de paquets lors des inondations par paquets ICMP.
Supprimer les 4 règles permettant de limiter et journaliser les paquets ICMP en provenance d'internet et d'accepter tout paquet dont le trafic n'est pas limité.

```
iptables -t mangle -D PREROUTING -i ${EXTIF} -p icmp -m limit --limit 5/s -j ACCEPT
iptables -t mangle -D PREROUTING -i ${EXTIF} -p icmp -m limit --limit 2/s -j NFLOG
iptables -t mangle -D PREROUTING -i ${EXTIF} -p icmp -j DROP
iptables -t mangle -D PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 133 Ajouter 4 règles identiques en modifiant la journalisation pour que celle-ci ajoute un préfixe au message informant que le paquet est potentiellement issu d'une attaque par inondation.

```
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -m limit --limit 5/s -j ACCEPT
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp \
  -m limit --limit 2/s -j NFLOG --nflog-prefix "[WARNING flood ICMP]"
iptables -t mangle -A PREROUTING -i ${EXTIF} -p icmp -j DROP
iptables -t mangle -A PREROUTING -i ${EXTIF} -j ACCEPT
```

Question 134 Tenter d'inonder la DMZ depuis internet avec 100 paquets.

```
root@internet:/$ ping -f -c 100 dmz
```

Vérifier qu'un nombre limité de paquets est transmis à la DMZ.
Vérifier qu'un nombre limité de messages est ajouté au journal. Vérifier également que ces messages contiennent le préfixe `[WARNING flood ICMP]`.

À l'aide des préfixe dans les messages, nous pouvons désormais simplifier l'étude *a posteriori* des attaques en consultant les journaux. Un simple `grep` permet par exemple d'isoler les messages que nous attribuons à une utilisation légitime ou à une potentielle attaque.

```
root@firewall:/$ grep WARNING /var/log/ulog/syslogemu.log
```

```
root@firewall:/$ grep INFO /var/log/ulog/syslogemu.log
```

Bien évidemment, pour que la simplification de l'étude soit efficace, un préfixe doit être ajouté à chaque message. Nous ne le ferons pas ici mais il est donc nécessaire de modifier toutes les règles dont la cible est `NFLOG` pour utiliser l'option `--nflog-prefix`.

11 Les chaînes utilisateur

Un dernier point important sur le fonctionnement de Netfilter est la possibilité de construire des chaînes utilisateur. Dans la section précédente, nous avons modifié plusieurs fois nos règles, par exemple pour limiter le nombre de messages dans le journal ou ajouter des préfixes. Dans cette section, nous allons voir comment nous pouvons réduire cette quantité de travail et rendre le traitement des paquets plus modulaire.

11.1 Présentation

Il est possible de construire d'autres chaînes que celles par défaut et de mettre dedans des règles comme nous l'avons fait jusqu'à maintenant. Nous pouvons utiliser toute chaîne utilisateur comme cible d'une règle. Lorsque l'on fait cela, le traitement dans la chaîne actuelle s'arrête, Netfilter saute à la chaîne cible et traite toutes les règles. Si aucune de ces règles ne provoque un arrêt du traitement, alors Netfilter retourne à la chaîne initiale et on continue le traitement. Ces chaînes sont généralement utilisées pour deux choses.

Premièrement, cela permet de définir des enchaînements d'actions comme une chaîne et, lorsque l'on souhaite réaliser cet enchaînement d'actions, on appelle la chaîne comme cible. Il y a une cible supplémentaire dont nous n'avons pas parlé jusqu'à maintenant mais qui est particulièrement utile dans les chaînes utilisateur : c'est la cible `RETURN`. La cible `RETURN` fait que le traitement dans la chaîne utilisateur actuelle s'arrête et que Netfilter retourne immédiatement à la chaîne qui l'a appelée (comme en programmation avec les fonctions), même si la chaîne utilisateur n'est pas complètement traitée. L'utilisation des chaînes utilisateurs et de `RETURN` permet donc de faire des fichiers de configuration modulaires, comme dans n'importe quel langage de programmation.

Le deuxième cas d'usage est l'amélioration des performances. Si vous avez 100 règles dans la chaîne `FORWARD` d'une table, chaque paquet qui passe par le routeur va subir 100 tests ; ce qui va ralentir considérablement la machine, mais aussi le transfert du paquet sur le réseau. Les chaînes utilisateurs nous permettent de rendre le traitement arborescent et de réduire ainsi le nombre de règles traitées. Supposons que dans la chaîne `FORWARD`, nous avons des règles du type "*si le protocole est TCP, alors aller à la cible* `regles-pour-tcp`" et dix autres règles comme celle-ci envoyant vers dix chaînes utilisateur qui regroupent par paquets de dix (dans un cas idéal) les cent règles initiales. Pour chaque paquet on ne vérifiera donc que onze règles au total et nous aurons considérablement amélioré les performances.

11.2 Un cas pratique

Dans la section 10, nous avons souhaité combiner deux actions : `NLOG` et `ACCEPT` ou `NLOG` et `DROP`. Dans d'autres cas, nous voudrons combiner plus d'actions. Or il est relativement embêtant de devoir reproduire des ensembles d'actions (deux règles à la suite) dans le code, et de devoir les modifier tous à chaque fois que nous souhaitons faire un changement sur le principe de fonctionnement (comme, par exemple, en modifiant le préfixe).

En réalité, l'action de faire une journalisation et une acceptation d'un paquet est un enchaînement que nous pourrions appeler `LOG-AND-ACCEPT` ; de même, l'action de faire une journalisation et un abandon d'un paquet est un enchaînement que nous pourrions appeler `LOG-AND-DROP`. Nous pourrions définir ces enchaînements et les appeler autant de fois que nous le souhaitons, comme on appelle une fonction dans un langage de programmation. Ceci est une des fonctionnalités que nous permettent de faire les chaînes utilisateur.

11.2.1 Commandes pour manipuler les chaînes utilisateur

`-N, --new-chain <chaine>` : Crée une nouvelle chaîne définie par l'utilisateur avec le nom indiqué. Il ne doit pas déjà exister de cible de même nom.

```
iptables -N MY_USER_CHAIN
```

`-X, --delete-chain [<chaine>]` : Efface la chaîne désignée définie par l'utilisateur. Si aucune chaîne n'est précisée, la commande essaye d'effacer toutes les chaînes utilisateur. Attention : une chaîne utilisateur ne peut être effacée que si elle est vide (utilisez `-F` pour la vider) et qu'aucune règle ne l'utilise.

```
iptables -X MY_USER_CHAIN
```

11.2.2 Une chaîne LOG-AND-DROP

Nous allons créer une chaîne qui permet de faire une journalisation et un abandon d'un paquet. Ensuite, nous allons utiliser cette chaîne pour réduire le nombre de règles utilisées pour filtrer les paquets qui correspondent à des attaques de *spoofing* d'adresses IP.

Question 135 Commencer par créer une chaîne utilisateur nommée `LOG_N_DROP`.

```
iptables -N LOG_N_DROP
```

Question 136 Ajouter deux règles à la chaîne utilisateur : une pour ajouter un message au journal, dans la limite de 2 par secondes avec une réserve de 1, et une pour abandonner le paquet.

```
iptables -A LOG_N_DROP -m limit --limit 2/s --limit-burst 1 -j NFLOG --nflog-prefix "[WARNING]"
iptables -A LOG_N_DROP -j DROP
```

Question 137 Les chaînes utilisateur sont créées dans la table `filter`. Vérifier que les deux règles ajoutées précédemment sont bien présentes dans la chaîne `LOG_N_DROP`.

```
iptables -t filter -L LOG_N_DROP -v
```

Question 138 Supprimer les 6 règles utilisées pour journaliser et abandonner les paquets qui correspondent à des attaques de *spoofing* d'adresses IP.

```
iptables -t filter -D FORWARD -s ${DMZ}    ! -i ${DMZIF} -j NFLOG
iptables -t filter -D FORWARD -s ${INTNET} ! -i ${INTIF} -j NFLOG
iptables -t filter -D FORWARD -s ${EXTNET} ! -i ${EXTIF} -j NFLOG
iptables -t filter -D FORWARD -s ${DMZ}    ! -i ${DMZIF} -j DROP
iptables -t filter -D FORWARD -s ${INTNET} ! -i ${INTIF} -j DROP
iptables -t filter -D FORWARD -s ${EXTNET} ! -i ${EXTIF} -j DROP
```

Question 139 Insérer 3 règles en premières positions de la chaîne `FORWARD` dans la table `filter` pour que les paquets qui correspondent à des attaques de *spoofing* d'adresses IP soient transmis à la chaine `LOG_N_DROP`.

```
iptables -t filter -I FORWARD 1 -s ${DMZ}    ! -i ${DMZIF} -j LOG_N_DROP
iptables -t filter -I FORWARD 1 -s ${INTNET} ! -i ${INTIF} -j LOG_N_DROP
iptables -t filter -I FORWARD 1 -s ${EXTNET} ! -i ${EXTIF} -j LOG_N_DROP
```

Question 140 Prendre le contrôle de la machine sur internet et utiliser *scapy* pour forger un paquets TCP avec comme adresse IP source celle de la DMZ, comme adresse IP destination celle du réseau interne et comme port destination le port 25 (SMTP).
Vérifier qu'un message est ajouté au journal lors de l'envoi du paquet et que le paquet est abandonné.

```
>>> thePacket = IP() / TCP() / "do you log and drop me?"
>>> thePacket.src = "192.168.1.1"
>>> thePacket.dst = "192.168.2.1"
>>> thePacket.dport = 25
>>> thePacket.sport = 1024
>>> send(thePacket)
```

```
root@firewall:/$ tail /var/log/ulog/syslogemu.log
```

Question 141 Vérifier que le paquet a bien été transmis à la chaîne utilisateur `LOG_N_DROP`, même s'il ne correspondait pas à la première chaîne évaluée.

```
root@firewall:/$ iptables -t filter -L FORWARD -v
```

11.2.3 Améliorer les performances

Dans la chaîne utilisateur précédente, nous avons réduit de 3 le nombre de règles évaluées par paquet dans la chaîne `FORWARD` de la table `filter`. Nous pouvons encore réduire le nombre de règles en factorisant toutes les règles destinées aux paquets TCP. Dans cette chaîne, nous ajoutons des règles pour accepter les paquets que nous considérons légitimes et nous laissons d'autres règles s'évaluer pour les paquets qui ne correspondent pas.

Question 142 Créer une nouvelle chaîne utilisateur que nous appelons `TCP_FILTER`.

```
iptables -N TCP_FILTER
```

Question 143 Ajouter 5 règles à cette chaîne utilisateur : 4 pour accepter les paquets TCP à destination de la DMZ sur les ports 80, 443, 25 et 53 ; 1 pour accepter les les paquets TCP à destination du réseau interne et en provenance de la DMZ sur le port 25.

```
iptables -t filter -A TCP_FILTER -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport  80 -j ACCEPT
iptables -t filter -A TCP_FILTER -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport 443 -j ACCEPT
iptables -t filter -A TCP_FILTER -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport  25 -j ACCEPT
iptables -t filter -A TCP_FILTER -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport  53 -j ACCEPT
iptables -t filter -A TCP_FILTER -m conntrack --ctstate NEW -d ${INTNET} -s ${DMZ} -p tcp --dport 25 -j ACCEPT
```

Question 144 Nous allons utiliser cette chaîne utilisateur pour filtrer tous les paquets TCP dans la chaîne `FORWARD` de la table `filter`. Si un paquet ne correspond à aucune des règles que nous avons ajoutées, alors il doit être de nouveau évalué dans la chaîne `FORWARD` de la table `filter`.
Ajouter une règle à la chaîne `TCP_FILTER` pour que le traitement de tous les paquets restants retourne dans la chaîne appelante.

```
iptables -t filter -A TCP_FILTER -j RETURN
```

Question 145 Vérifier que les règles ajoutées précédemment sont bien présentes dans la chaîne `TCP_FILTER`.

```
iptables -t filter -L TCP_FILTER -v
```

Question 146 Dans la chaîne `FORWARD` de la table `filter`, supprimer les 5 règles qui permettent d'accepter les paquets TCP.

```
iptables -t filter -D FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport  80 -j ACCEPT
iptables -t filter -D FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport 443 -j ACCEPT
iptables -t filter -D FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport  25 -j ACCEPT
iptables -t filter -D FORWARD -m conntrack --ctstate NEW -d ${DMZ} -p tcp --dport  53 -j ACCEPT
iptables -t filter -D FORWARD -m conntrack --ctstate NEW -d ${INTNET} -s ${DMZ} -p tcp --dport 25 -j ACCEPT
```

Question 147 Remplacer ces 5 règles par un renvoi à la chaîne `TCP_FILTER` pour tous les paquets TCP.

```
iptables -t filter -I FORWARD 5 -p tcp -j TCP_FILTER
```

Question 148 Vérifier la cohérence de la chaîne `FORWARD` de la table `filter`.

```
iptables -t filter -L FORWARD -v
```

Le résultat doit ressembler à ceci :

```
Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out source         destination
    0     0 LOG_N_DROP  all  -- !eth2  any 192.168.3.0/24 anywhere
    0     0 LOG_N_DROP  all  -- !eth1  any 192.168.2.0/24 anywhere
    0     0 LOG_N_DROP  all  -- !eth0  any 192.168.1.0/24 anywhere
    0     0 ACCEPT      all  -- any    any anywhere       anywhere       ctstate RELATED,ESTABLISHED
    0     0 TCP_FILTER  tcp  -- any    any anywhere       anywhere
    0     0 ACCEPT      udp  -- any    any anywhere       192.168.1.0/24 ctstate NEW udp dpt:domain
    0     0 ACCEPT     icmp  -- any    any anywhere       192.168.1.0/24 ctstate NEW
    0     0 NFLOG       all  -- any    any anywhere       anywhere
```

Question 149 Envoyer un paquet ICMP depuis internet vers la DMZ. Vérifier que la chaîne `TCP_FILTER` n'a pas été traversée.

```
root@internet:/$ ping -c 1 dmz
```

```
root@firewall:/$ iptables -t filter -L FORWARD -v
```

Question 150 Tenter une connexion TCP sur un port fermé de la DMZ depuis internet.
Vérifier que la chaîne `TCP_FILTER` a été traversée 1 fois et que le paquet a continué de traverser la chaine `FORWARD` jusqu'à être journalisé puis abandonné par la politique par défaut.

```
root@internet:/$ echo "quit" | telnet dmz 999
```

```
root@firewall:/$ iptables -t filter -L FORWARD -v
```

Vérifier aussi qu'un message est ajouté au journal pour garder une trace de cette action.

```
root@firewall:/$ tail /var/log/ulog/syslogemu.log
```

Question 151 Vérifier que les communications TCP fonctionnent.

```
root@internet:/$ echo "quit" | telnet dmz 80
root@internet:/$ echo "quit" | telnet dmz 443
root@internet:/$ echo "quit" | telnet dmz 25
root@internet:/$ echo "quit" | telnet dmz 53
```

```
root@dmz:/$ echo "quit" | telnet interne 25
```

Question 152 Si vous en voulez encore, essayez de rendre votre traitement un peu plus arborescent pour que, quand vous ajouterez des nouvelles règles, tout ne se fasse pas de façon linéaire.
Sinon, reposez vous, vous l'avez mérité.