Filtrage réseau avec netfilter/nftables

Philippe Latu
philippe.latu(at)inetdoc.net

https://www.inetdoc.net

Résumé

Ce support de travaux pratiques est une introduction au filtrage réseau. Il reprend la topologie *Hub & Spoke* du support précédent de la série. Les questions débutent par l'identification des outils et passent à l'application des règles de filtrage avec et sans suivi de communication (*stateful vs stateless inspection*). On introduit aussi les fonctions de traduction d'adresses (NAT).

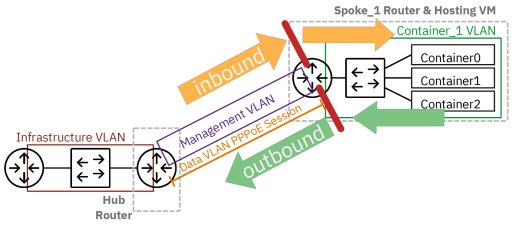


Table des matières

1. Copyright et Licence

Méta-information

Ce document est écrit avec *DocBook* XML sur un système *Debian GNU/Linux*. Il est disponible en version imprimable au format PDF : `netfilter.pdf`.

2. Objectifs

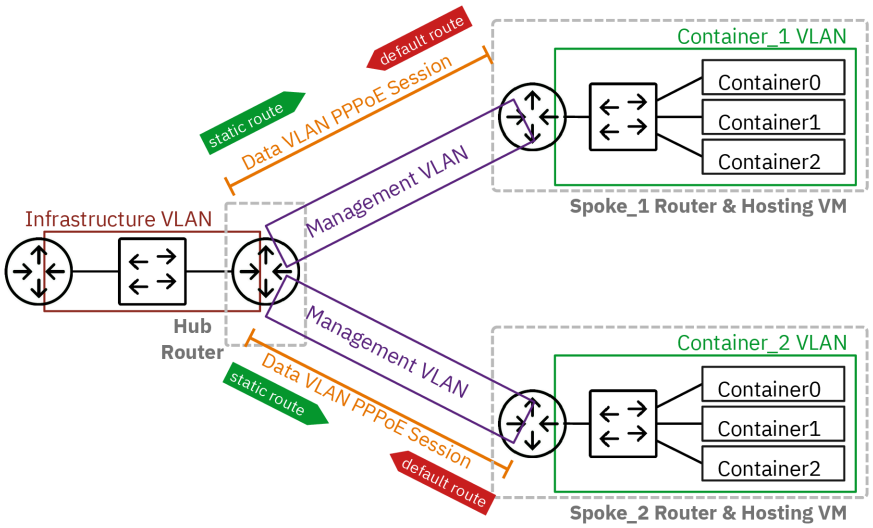
Après avoir réalisé les manipulations présentées dans ce document, les étudiants seront capables de :

1. Configurer et appliquer des règles de filtrage réseau de base sur des routeurs Linux en utilisant nftables.
2. Mettre en place des protections contre l'usurpation d'adresses IP et les attaques par déni de service ICMP.
3. Implémenter un système de protection contre les tentatives d'intrusion sur SSH en utilisant fail2ban.
4. Configurer la traduction d'adresses réseau (NAT) source et destination sur un routeur Linux.
5. Analyser et déboguer les règles de filtrage en utilisant les outils de diagnostic comme conntrack.

3. Architecture réseau étudiée et filtrage existant

Les manipulations sur le système de filtrage réseau présentées ici s'appuient sur la topologie *Hub and Spoke* étudiée dans le support précédent de la série : *Topologie Hub & Spoke avec le protocole PPPoE*.

La topologie étudiée associe trois routeurs qui ont deux rôles distincts.



Topologie entre deux routeurs *Hub* et *Spoke* avec PPPoE

Hub

Traduit mot à mot, le rôle *Hub* correspond à un concentrateur. Il concentre tous les flux réseau des routeurs qui ont le rôle *Spoke*. En effet, les échanges entre deux routeurs *Spoke* doivent passer par le routeur *Hub*.

On lui attribue aussi la fonction de *Broadband Remote Access Server* ou BRAS. Dans notre contexte, cette fonction se caractérise par le fait que ce routeur détient le plan d'adressage. C'est lui qui a la responsabilité de délivrer les adresses IP lors de l'initiation de la session PPP.

Spoke

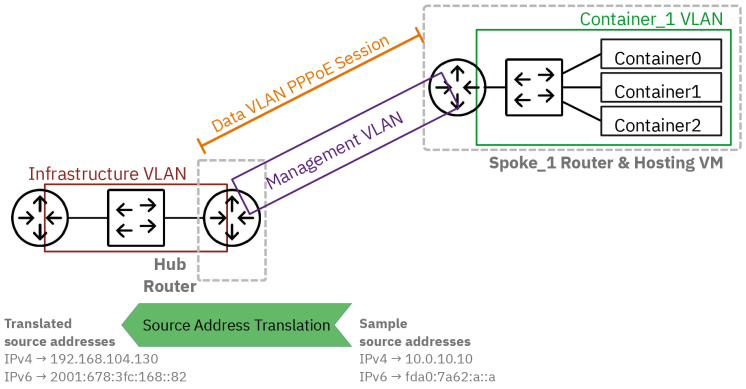
Le rôle *Spoke* correspond à un réseau d'extrémité au delà duquel on ne trouve aucune interconnexion. Le routeur *Spoke* doit s'adresser au routeur Hub dès qu'il veut acheminer un flux réseau. Il s'agit bien d'un routeur d'extrémité qui ne dispose d'aucun chemin alternatif pour joindre l'Internet.

Dans les réseaux domestiques, la «box» correspond bien au rôle *Spoke* dans la mesure où elle se voit attribuer des adresses IPv4 et IPv6 publiques par le fournisseur d'accès Internet. Les seules informations qu'elle détient sont les authentifiants du client de l'opérateur.

Pour commencer, on s'intéresse à l'installation des outils et à la lecture des informations sur le filtrage déjà en place sur le routeur *Hub*.

3.1. Routage et traduction d'adresses sources (situation de départ)

La situation de départ des manipulations suppose que la topologie *Hub & Spoke* est en place et fonctionnelle. On s'appuie sur le support précédent de la série : la *Topologie Hub & Spoke avec le protocole PPPoE* qui comprend déjà un premier niveau de filtrage avec la traduction d'adresses sources de tous les paquets qui sortent vers le réseau d'infrastructure (VLAN rouge).



Traduction d'adresses sources pour les flux sortants du routeur *Hub*.

Q1. Quel est le nom de l'outil de gestion des règles de filtrage réseau ? À quel paquet appartient-il ?

Reprendre la section sur la configuration du routeur *Hub* dans le document *Topologie Hub & Spoke avec le protocole PPPoE*.

Il s'agit de la commande nft fournie avec le paquet `nftables`.

```
apt show nftables
```

```
dpkg -L nftables
```

Q2. Quel est l'état du service `systemd` associé ?

Reprendre la section sur la configuration du routeur *Hub* dans le document *Topologie Hub & Spoke avec le protocole PPPoE* dans laquelle le service a été activé.

On affiche l'état du service `systemd` fourni avec le paquet `nftables` et on vérifie qu'il est activé (***enabled***).

```
systemctl status nftables
```

Q3. Dans quel fichier sont enregistrés les jeux de règles à appliquer au lancement du système ?

Rechercher dans les informations affichées à la question précédente, la syntaxe de la commande lancée par le service `systemd`.

On identifie le fichier `/etc/nftables` qui sert à stocker les jeux de règles activés lors de l'initialisation du système.

Q4. Comment afficher la liste des règles de filtrage actives ?

Rechercher dans les pages de manuels de la commande nftables l'option qui permet d'afficher la liste du jeu de règles en cours de traitement.

La lecture des pages de manuels conduit à l'option `list` suivie de `ruleset`.

```
sudo nft list ruleset
```

```
table inet nat {
        chain postrouting {
                type nat hook postrouting priority srcnat; policy accept;
                oifname "enp0s1.360" counter packets 89 bytes 7400 masquerade
        }
}
```

3.2. Lecture des règles de traduction d'adresses sources

Dans cette partie, on s'intéresse à l'identification des éléments qui composent le jeu de règles de traduction d'adresses sources appliqué sur le routeur *Hub* de la topologie étudiée.

Comme le jeu de règles est déjà présent et actif, il faut comparer les champs des règles du fichier `/etc/nftables.conf` avec la représentation graphique générale *Packet Flow in Netfilter*.

Q5. Quelle est la ***table*** utilisée dans les deux jeux de règles de filtrage appliquées sur le routeur *Hub* ?

Il s'agit de la table `nat` que l'on repère en couche liaison de la représentation graphique *Packet Flow in Netfilter*

Q6. Quelle est la ***chaîne*** utilisée dans les deux jeux de règles de filtrage appliquées sur le routeur *Hub* ?

Il s'agit de la chaîne `postrouting` que l'on repère en couche liaison à droite de la représentation graphique *Packet Flow in Netfilter*

Comme son nom l'indique, cette chaîne traite les paquets après que la décision de routage ait été prise.

Q7. Comment l'interface réseau sur laquelle les traitements sont appliqués est-elle identifiée ?

Repérer le mot clé placé avant la chaîne de caractère qui contient le nom de l'interface.

La clé placée avant le nom d'interface est `oifname`. Cette clé correspond à ***output interface name***.

La traduction d'adresses sources a lieu en sortie sur l'interface réseau raccordée au réseau d'infrastructure (VLAN rouge).

Q8. Quel est le nom de la ***cible*** utilisée dans les deux jeux de règles de filtrage appliquées sur le routeur ***Hub*** ?

Identifier le mot clé placé à droite en bout de chaîne de traitement après les compteurs.

Le nom utilisé est ***masquerade***. Il caractérise le fait que l'adresse source de tous les paquets sortants par l'interface `enp0s1.360` est remplacée par celle de cette interface, aussi bien avec IPv4 qu'avec IPv6.

3.3. Comptage des paquets et enregistrements des transactions

La traduction d'adresse source entre dans la catégorie du filtrage ***Stateful***. Il est nécessaire de conserver un enregistrement de la traduction faite sur un paquet sortant pour réaliser l'opération inverse lors de l'arrivée du paquet retour relatif au paquet sortant.

Dans cette partie, on cherche à afficher la liste des enregistrements en cours dans le routeur ***Hub*** pour des flux initiés depuis un conteneur hébergé sur un routeur ***Spoke***.

Q9. Comment caractériser l'utilisation des jeux de règles de filtrage en l'état actuel de la configuration ?

Rechercher dans les résultats de l'affichage des règles en cours sur le routeur ***Hub***, les informations sur le comptage des flux.

La sortie de la commande d'affichage fait apparaître le compte des paquets traités par la chaîne `postrouting` avec l'indication ***counter packets***.

```
sudo nft list ruleset
```

```
table inet nat {
        chain postrouting {
                type nat hook postrouting priority srcnat; policy accept;
                oifname "enp0s1.360" counter packets 89 bytes 7400 masquerade
        }
}
```

Q10. Est-ce que les valeurs de comptage affichées correspondent au volume de trafic vu sur l'interface ?

Rechercher, dans les options de la commande ip, le moyen d'afficher les statistiques de l'interface de sortie sur le réseau d'infrastructure (VLAN rouge). Comparer les valeurs obtenues avec celles de l'affichage des règles de filtrage.

C'est l'option `-s` de la commande ip qui donne les informations sur le volume de trafic qui a transité par une interface.

```
ip -s -h addr ls dev enp0s1.360
4: enp0s1.360@enp0s1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP group default qlen 1000
    link/ether b8:ad:ca:fe:00:05 brd ff:ff:ff:ff:ff:ff
    inet 192.168.104.130/29 brd 192.168.104.135 scope global enp0s1.360
       valid_lft forever preferred_lft forever
    inet6 2001:678:3fc:168:baad:caff:fefe:5/64 scope global dynamic mngtmpaddr noprefixroute
       valid_lft 2591974sec preferred_lft 604774sec
    inet6 2001:678:3fc:168::82/64 scope global
       valid_lft forever preferred_lft forever
    inet6 fe80::baad:caff:fefe:5/64 scope link proto kernel_ll
       valid_lft forever preferred_lft forever
    RX:  bytes packets errors dropped  missed   mcast
         227M   38.1k      0       0       0     137
    TX:  bytes packets errors dropped carrier collsns
         5.26M   71.8k      0       0       0       0
```

On constate que les valeurs ne coïncident pas du tout. En fait, les valeurs annoncées par le système de filtrage correspondent aux ***nouveaux flux enregistrés*** dans le système d'enregistrement et de suivi des transactions appelé ***connection tracking***.

Pour que les valeurs affichées avec la commande nft évoluent, il faut que de nouveaux flux réseaux n'ayant pas encore été enregistrés apparaissent.

Q11. Comment obtenir la liste des enregistrements des transactions traitées par le système de filtrage réseau du routeur ***Hub*** ?

Rechercher le paquet qui contient la commande conntrack puis rechercher les options de cette commande qui permettent d'afficher les états des enregistrements.

On ouvre une console sur le routeur ***Hub*** de la maquette et on installe le paquet `conntrack`.

```
sudo apt -y install conntrack
```

Ensuite, on utilise l'option `-L` pour afficher la liste des enregistrements courants. Voici un exemple ponctuel formaté pour les besoins de la copie d'écran.

```
sudo conntrack -L | fmt -w80
conntrack v1.4.8 (conntrack-tools): 1 flow entries have been shown.
tcp      6 299 ESTABLISHED src=172.16.0.6 dst=192.168.104.130 sport=50376
dport=22 src=192.168.104.130 dst=172.16.0.6 sport=22 dport=50376 [ASSURED]
```

On identifie les paramètres suivants :

tcp

L'identification du protocole permet de savoir si on a affaire à un service orienté connexion ou pas.

Dans le cas d'un service orienté connexion, on utilise les trois phases d'établissement, de maintien et de libération pour savoir si l'enregistrement d'un flux doit être conservé.

À l'inverse, si le service n'est pas orienté connexion, on utilise des temporisations prédéfinies pour savoir si un enregistrement est périmé.

ESTABLISHEDASSURED

Ces indicateurs d'état nous informent sur le fait que la communication est dans la phase de maintien et que les échangent sont en cours.

sportdport

Les couples de numéros de ports source et destination en couche transport ainsi que les couples d'adresses IPv4 en couche réseau nous informent sur les permutations réalisées si le trafic réseau est sortant ou entrant.

Q12. Comment provoquer de nouveaux enregistrements IPv4 et IPv4 dans le système de suivi des transactions réseau ?

Pour que me scénario soit plus réaliste, on initie de nouvelles transactions depuis les conteneurs hébergés sur un routeur ***Spoke*** et on observe les résultats sur le routeur ***Hub***.

On commence par installer le paquet qui contient la commande wget sur chaque conteneur du routeur ***Spoke*** utilisé pour les tests.

```
for i in {0..2}
do
   echo ">>>>>>>>>>>>>>>> c$i"
   incus exec c$i -- apt -y install wget
done
```

Toujours sur le même routeur ***Spoke***, on lance une boucle de génération de trafic web. Dans l'exemple ci-dessous, on lance cinq consultations de page web dans chacun des trois conteneurs hébergés.

```
for i in {0..2}
do
   echo ">>>>>>>>>>>>>>>> c$i"
   for j in {0..4}
   do
      incus exec c$i -- wget -4 -O /dev/null https://www.iana.org
      sleep 1
   done
done
```

Sur le routeur ***Hub***, on affiche la liste des enregistrements du système de suivi des transactions. Voici un extrait IPv4 ponctuel non formaté pour afficher correctement les débuts de lignes.

```
sudo conntrack -f ipv4 -L
```

```
tcp      6 16 TIME_WAIT src=10.0.10.10 dst=192.0.46.8 sport=37766 dport=443 src=192.0.46.8 dst=192.168.104.130 sport=443 dp
tcp      6 21 TIME_WAIT src=10.0.10.10 dst=192.0.46.8 sport=49580 dport=443 src=192.0.46.8 dst=192.168.104.130 sport=443 dp
udp      17 24 src=10.0.10.10 dst=172.16.0.2 sport=35685 dport=53 src=172.16.0.2 dst=192.168.104.130 sport=53 dport=35685 m
tcp      6 34 TIME_WAIT src=10.0.10.11 dst=192.0.46.8 sport=56480 dport=443 src=192.0.46.8 dst=192.168.104.130 sport=443 dp
tcp      6 55 TIME_WAIT src=10.0.10.12 dst=192.0.46.8 sport=41652 dport=443 src=192.0.46.8 dst=192.168.104.130 sport=443 dp
tcp      6 45 TIME_WAIT src=10.0.10.12 dst=192.0.46.8 sport=55804 dport=443 src=192.0.46.8 dst=192.168.104.130 sport=443 dp
tcp      6 48 TIME_WAIT src=10.0.10.12 dst=192.0.46.8 sport=55820 dport=443 src=192.0.46.8 dst=192.168.104.130 sport=443 dp
```

Dans l'extrait ci-dessus on retrouve :

- Les adresses IPv4 des trois conteneurs : `10.0.10.10`, `10.0.10.11` et `10.0.10.12`.
- L'adresse du site web sur l'Internet : `192.0.46.8`
- L'adresse du routeur ***Hub*** sur le réseau d'infrastructure : `192.168.104.130`

On lance la même boucle en IPv6 pour afficher les enregistrements propres à ce protocole réseau.

```
for i in {0..2}
do
   echo ">>>>>>>>>>>>>>>> c$i"
   for j in {0..4}
   do
      incus exec c$i -- wget -6 -O /dev/null https://www.iana.org
      sleep 1
   done
done
```

Voici un nouvel extrait de la liste des enregistrements qui montre les trois adresses IPv6 des conteneurs, l'adresse IPv6 du serveur et enfin celle du routeur ***Hub***.

```
sudo conntrack -f ipv6 -L
```

```
sudo conntrack -f ipv6 -L
tcp      6 108 TIME_WAIT src=fda0:7a62:a:0:216:3eff:fe6a:395c dst=2620:0:2830:200::b:8 sport=56654 dport=443 src=2620:0:283
tcp      6 113 TIME_WAIT src=fda0:7a62:a:0:216:3eff:fe6a:395c dst=2620:0:2830:200::b:8 sport=56678 dport=443 src=2620:0:283
tcp      6 110 TIME_WAIT src=fda0:7a62:a:0:216:3eff:fe6a:395c dst=2620:0:2830:200::b:8 sport=56660 dport=443 src=2620:0:283
tcp      6 116 TIME_WAIT src=fda0:7a62:a:0:216:3eff:fe37:f12a dst=2620:0:2830:200::b:8 sport=36964 dport=443 src=2620:0:283
tcp      6 111 TIME_WAIT src=fda0:7a62:a:0:216:3eff:fe6a:395c dst=2620:0:2830:200::b:8 sport=56666 dport=443 src=2620:0:283
tcp      6 98 TIME_WAIT src=fda0:7a62:a:0:216:3eff:fed9:a25b dst=2620:0:2d0:200::b:8 sport=50388 dport=443 src=2620:0:2d0:2
tcp      6 115 TIME_WAIT src=fda0:7a62:a:0:216:3eff:fe6a:395c dst=2620:0:2830:200::b:8 sport=56684 dport=443 src=2620:0:283
tcp      6 96 TIME_WAIT src=fda0:7a62:a:0:216:3eff:fed9:a25b dst=2620:0:2d0:200::b:8 sport=50384 dport=443 src=2620:0:2d0:2
tcp      6 119 FIN_WAIT src=fda0:7a62:a:0:216:3eff:fe37:f12a dst=2620:0:2830:200::b:8 sport=36986 dport=443 src=2620:0:2830
tcp      6 118 TIME_WAIT src=fda0:7a62:a:0:216:3eff:fe37:f12a dst=2620:0:2830:200::b:8 sport=36978 dport=443 src=2620:0:283
```

4. Protection de base des routeurs Hub et Spoke

Du point de vue conception, cette partie est consacrée au filtrage réseau sans état. Les trois sections proposent des règles qui demandent un traitement le plus rapide possible et le moins coûteux possible en ressources.

Le but de cette partie est de mettre en place les fonctions de filtrage de base communes à tous les routeurs de la topologie. Ceci implique que le paquet `nftables` soit installé et que le service `systemd` soit activé sur tous routeurs *Spoke*

```
sudo apt -y install nftables
```

```
sudo systemctl enable --now nftables
```

Voici une description des fonctions à mettre en œuvre dans cette section.

Protection contre l'usurpation des adresses sources

Pour bloquer tous les paquets provenant d'un réseau ***extérieur*** avec des adresses IP sources appartenant à un réseau ***intérieur***, on implante une chaîne `rpfilter` dans la table `raw` qui assure un filtrage sans état.

Les motivations de ces méthodes de filtrage réseau sont présentées dans les documents de référence RFC2827 *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing* et RFC 3704.

Les tests de validation de ces mécanismes sont faciles à réaliser sur les routeurs ***Spoke***. Tout paquet qui arrive via l'interface PPP et dont l'adresse source IPv4 ou IPv6 appartient au réseau d'hébergement (VLAN vert) doit être jeté. On protège ainsi les routeurs contre les dénis de services.

Protection contre les dénis de services ICMP

Les routeurs doivent s'assurer que le volume de trafic qui est présenté en entrée est compatible avec un fonctionnement nominal des services.

Protection contre les robots de connexion au service SSH

Les routeurs ont besoin d'un accès d'administration à distance via SSH. Pour autant, cet accès doit être protégé contre les tentatives d'intrusion par dictionnaire de couples d'authentifiants.

L'outil fail2ban fourni avec le paquet du même nom introduit une chaîne de filtrage dédiée à ces tentatives d'intrusion.

4.1. Protection contre l'usurpation d'adresse source

Voici le jeu de règles à implanter dans le fichier `/etc/nftables.conf` sur les deux routeurs ***Spoke*** pour activer la protection contre l'usurpation d'adresses réseau source.

```
#!/usr/sbin/nft -f

flush ruleset

table inet raw {
    chain rpfilter {
        type filter hook prerouting priority raw; policy accept;
        iifname "ppp0" fib saddr . iif oif 0 counter packets 0 bytes 0 drop
    }
}
```

> **Important**
>
> Dans le but de faciliter les tests de validation, le trafic “malveillant” est émis depuis le routeur ***Hub***. Sur ce routeur, nous avons la possibilité d'installer tous les outils et de faire toutes les manipulations possibles pour falsifier les adresses sources. De plus, on contrôle le plan d'adressage. Si ce n'était pas le cas, il faudrait utiliser une autre machine virtuelle et les erreurs auraient des conséquences plus beaucoup plus graves.

Q13. Comment définir le rôle de la table `raw` dans le système de filtrage du noyau Linux ?

Rechercher la présentation de cette table dans la documentation `netfilter`.

La table `raw` du système de filtrage Linux `netfilter` est principalement utilisée pour marquer les paquets qui doivent éviter le suivi des connexions. Elle intervient au tout début du traitement des paquets, ce qui permet de traiter certains flux réseau en évitant des opérations qui consomment beaucoup de ressources, comme le suivi des connexions.

Q14. Comment expliquer l'utilisation de la correspondance `fib` dans la règle de filtrage des paquets entrant par l'interface `ppp0` ?

Rechercher le mot clé `fib` dans les pages de manuels de la commande nft.

La règle `fib saddr . iif oif missing` vérifie si l'adresse source correspond à l'interface entrante selon la FIB (***Forwarding Information Base***). Cette règle met en œuvre le filtrage du chemin inverse.

Q15. Comment afficher la liste des règles de filtrage de la table `raw` dédiée au filtrage sans état (***stateless***) ?

Rechercher dans les pages de manuels de la commande nft les options relatives aux listes.

C'est l'option `list` qui permet l'affichage des règles implantées dans les différentes tables.

Voici un exemple dans le contexte de la maquette sur un routeur ***Spoke***. Un jeu de règles a déjà été inséré dans la table `raw`. Elle permet de visualiser les compteurs de correspondance qui montrent que la règle a bien été utilisée.

```
sudo nft list table inet raw
```

```
table inet raw {
    chain rpfilter {
        type filter hook prerouting priority raw; policy accept;
        iifname "ppp0" fib saddr . iif oif 0 counter packets 10 bytes 280 drop
    }
}
```

Avertissement

Si aucune règle n'a été implantée dans une table, le résultat de la commande produit une erreur.

Q16. Comment valider la fonction de blocage des tentatives d'usurpation d'adresses entre le routeur ***Hub*** et les routeurs ***Spoke*** ?

Pour falsifier les adresses réseau source, nous devons distinguer les deux protocoles.

- Pour IPv4, on installe le paquet `hping3`.

  Rechercher dans les pages de manuels de la commande hping3 les options qui permettent de générer du trafic ICMP avec des adresses source aléatoires à destination d'un conteneur hébergé sur un routeur ***Spoke***.

- Pour IPv6, on doit créer une interface réseau de type `dummy` à laquelle on attribue une adresse du réseau d'hébergement de conteneur.

  Rechercher les instructions de création d'une interface `dummy`.

On débute les tests avec le protocole IPv4 et la commande hping3.

Voici un exemple de test effectué sur le routeur ***Hub*** dans lequel l'option `-a` désigne l'adresse IPv4 source usurpée tandis que l'adresse en bout de ligne désigne la destination. Ici, on cherche à contacter un conteneur avec l'adresse source d'un conteneur voisin en étant placé “à l'extérieur” du VLAN vert.

```
sudo hping3 -1 -a 10.0.10.12 --fast -c 10 10.0.10.10
```

```
HPING 10.0.10.10 (ppp0 10.0.10.10): icmp mode set, 28 headers + 0 data bytes

--- 10.0.10.10 hping statistic ---
10 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Côté routeur ***Hub***, on constate qu'aucune réponse n'a été reçue.

Côté routeur ***Spoke***, on affiche le jeu des règles de filtrage actif et on relève les valeurs des compteurs de paquets jetés.

```
sudo nft list table inet raw
```

```
table inet raw {
    chain rpfilter {
        type filter hook prerouting priority raw; policy accept;
        iifname "ppp0" fib saddr . iif oif 0 counter packets 20 bytes 560 drop
    }
}
```

Dans l'exemple ci-dessus, 20 paquets ont été jetés.

Pour le protocole IPv6, il n'existe pas de solution équivalente à l'utilisation de la commande hping3. De plus, le fait d'utiliser une session PPP pour acheminer le trafic ne facilite pas les tests.

C'est la raison pour laquelle on utilise un autre artifice : une interface de type `dummy`. Cette interface est factice et nous permet d'émettre des requêtes ICMP6 avec son adresse IPv6.

Voici comment créer une interface et lui ajouter une adresse. On commence par charger le module `dummy` en précisant que nous besoin d'une seule interface.

```
sudo modprobe -v dummy numdummies=1
```

```
sudo ip link set dev dummy0 up
```

On peut maintenant ajouter une adresse IPv6 appartenant au réseau d'hébergement situé au-delà du routeur ***Spoke*** à cette interface.

```
sudo ip -6 addr add fda0:7a62:a::e/64 dev dummy0
```

On modifie la table de routage pour s'assurer que les paquets partiront en direction de la session PPP.

```
sudo ip -6 route del fda0:7a62:a::/64 dev dummy0
sudo ip -6 route add fda0:7a62:a::/64 dev dummy0 metric 2048
```

On vérifie la solution d'acheminement du trafic à destination du réseau d'hébergement des conteneurs.

```
ip -6 route get fda0:7a62:a::a
fda0:7a62:a::a from :: dev ppp0 src fda0:7a62:a::e metric 1024 pref medium
```

Il ne reste plus qu'à lancer des requêtes ICMP6 avec cette adresse source.

```
sudo ping6 -c 10 fda0:7a62:a::a
```

```
PING fda0:7a62:a::a (fda0:7a62:a::a) from fda0:7a62:a::e dummy0: 56 data bytes

--- fda0:7a62:a::a ping statistics ---
10 packets transmitted, 0 received, 100% packet loss, time 9210ms
```

Là encore, aucune réponse n'est revenue et c'est heureux !

Si on relève le compte des paquets jetés côté routeur ***Spoke***, on voit que toutes les requêtes sont tombées dans la règle de la table `raw`.

```
sudo nft list table inet raw
```

```
table inet raw {
    chain rpfilter {
        type filter hook prerouting priority raw; policy accept;
        iifname "ppp0" fib saddr . iif oif 0 counter packets 40 bytes 4160 drop
    }
}
```

Attention

N'oubliez pas de supprimer l'interface `dummy` après les tests.

```
sudo ip link set dev dummy0 down
sudo modprobe -r dummy
```

Q17. Comment utiliser la fonction `rp_filter` du sous-système réseau du noyau Linux pour le protocole IPv4 ?

Rechercher la clé `rp_filter` dans la documentation du noyau Linux : *Kernel IP sysctl*

Il existe un réglage du sous-système réseau du noyau avec la clé `rp_filter`. Voici un extrait de la documentation du noyau.

```
rp_filter - INTEGER
0 - No source validation.
1 - Strict mode as defined in RFC3704 Strict Reverse Path
    Each incoming packet is tested against the FIB and if the interface
    is not the best reverse path the packet check will fail.
    By default failed packets are discarded.
2 - Loose mode as defined in RFC3704 Loose Reverse Path
    Each incoming packet's source address is also tested against the FIB
    and if the source address is not reachable via any interface
    the packet check will fail.

Current recommended practice in RFC3704 is to enable strict mode
to prevent IP spoofing from DDos attacks. If using asymmetric routing
or other complicated routing, then loose mode is recommended.

The max value from conf/{all,interface}/rp_filter is used
when doing source validation on the {interface}.

Default value is 0. Note that some distributions enable it
in startup scripts.
```

On peut fixer la clé à 1 et lancer un test avec une série d'adresses IPv4 source aléatoires.

```
sudo sysctl -w net.ipv4.conf.all.rp_filter=1
```

Voici un exemple de commande qui provoquera un nombre de blocages aléatoire en fonction des correspondances.

```
sudo hping3 -1 --rand-source --fast -c 100 10.0.10.10
```

Dans ce cas, c'est la fonction de protection du noyau qui a détecté des paquets “martiens” pour lesquels il n'existe aucune solution de routage.

```
journalctl -n 500 -f --grep martian
```

```
spoke1 kernel: IPv4: martian source 10.0.10.10 from 239.9.60.83, on dev ppp0
spoke1 kernel: IPv4: martian source 10.0.10.10 from 225.61.235.216, on dev ppp0
spoke1 kernel: IPv4: martian source 10.0.10.10 from 234.137.124.42, on dev ppp0
spoke1 kernel: IPv4: martian source 10.0.10.10 from 225.151.17.117, on dev ppp0
spoke1 kernel: IPv4: martian source 10.0.10.10 from 227.163.239.170, on dev ppp0
spoke1 kernel: IPv4: martian source 10.0.10.10 from 227.27.22.34, on dev ppp0
spoke1 kernel: IPv4: martian source 10.0.10.10 from 233.200.216.31, on dev ppp0
```

4.2. Protection contre les dénis de service ICMP

Dans cette section, on reprend le jeu de règles précédentes et on le complète avec la limitation du nombre de requêtes ICMP entrantes.

Le trafic “malveillant“ est toujours généré sur le routeur ***Hub*** à destination des conteneurs des réseaux d'hébergement des sites distants.

Voici une nouvelle copie du fichier `/etc/nftables.conf` qui contient le jeu de règles à ajouter sur les deux routeurs ***Spoke*** pour assurer la protection contre les dénis de service par saturation de requêtes ICMP.

```
#!/usr/sbin/nft -f

flush ruleset

table inet raw {
    # BCP38 Rules
    chain rpfilter {
        type filter hook prerouting priority raw; policy accept;
        iifname "ppp0" fib saddr . iif oif 0 counter packets 0 bytes 0 drop
    }

    # ICMP Rate Limiting Rules
    chain icmpfilter {
        type filter hook prerouting priority raw; policy accept;
        icmp type echo-request limit rate 10/second burst 5 packets counter accept
        icmp type echo-request counter drop
    }
}
```

Q18. À quelle chaîne prédéfinie de la table `raw` font appels les chaînes personnalisées appelées `icmpfilter` ?

Consulter la représentation graphique *Packet Flow in Netfilter* et repérer les chaînes prédéfinies de la table `raw`.

Il s'agit de la chaîne `prerouting` qui traite les flux réseau “au plus tôt”, avant qu'une décision de routage soit prise.

Q19. Comment qualifier le fonctionnement des règles de limitation du nombre de nouvelles requêtes ICMP avec IPv4 ?

Rechercher les options de la commande hping3 qui permettent de générer un envoi de requêtes ICMP en grand nombre.

Voici un exemple d'envoi de requêtes ICMP en nombre à destination du deuxième conteneur hébergé sur le premier routeur ***Spoke***.

```
sudo hping3 -1 --flood -c 10 10.0.10.11
```

Si le temps d'exécution de ces émission paraît trop long, il ne faut pas hésiter à interrompre l'émission avec Ctrl+C.

```
HPING 10.0.10.11 (ppp0 10.0.10.11): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown

--- 10.0.10.11 hping statistic ---
61492309 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
```

Les résultats de cette commande montrent que le mécanisme de protection a bien fonctionné. On doit aussi vérifier qu'une émission “raisonnable“ de requêtes ICMP donne des résultats corrects.

```
ping -qc 10 10.0.10.11
```

```
PING 10.0.10.11 (10.0.10.11) 56(84) bytes of data.

--- 10.0.10.11 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 1.056/1.279/2.034/0.268 ms
```

Côté routeur ***Spoke*** "cible", on peut relever les compteurs des règles de traitement ICMP et constater qu'un grand nombre de paquets ont été jetés.

```
sudo nft list table inet raw
```

```
table inet raw {
    chain rpfilter {
        type filter hook prerouting priority raw; policy accept;
        iifname "ppp0" fib saddr . iif oif 0 counter packets 0 bytes 0 drop
    }

    chain icmpfilter {
        type filter hook prerouting priority raw; policy accept;
        icmp type echo-request limit rate 10/second burst 5 packets counter packets 5378 bytes 150584 accept
        icmp type echo-request counter packets 90460174 bytes 2532884872 drop
    }
}
```

Q20. Comment qualifier le fonctionnement des règles de limitation du nombre de nouvelles requêtes ICMP avec IPv6 ?

Rechercher les options de la commande ping qui permettent de générer un flux de saturation ICMPv6.

On s'intéresse plus particulièrement aux options `-f` et `-i`.

Voici un exemple de tentative de saturation à destination du deuxième conteneur du réseau d'hébergement du premier routeur ***Spoke***.

```
sudo ping -6 -c100 -i 0.0005 -f fda0:7a62:a::b
```

En réponse à cette commande, on voit que le taux de perte de paquets est important.

```
PING fda0:7a62:a::b (fda0:7a62:a::b) 56 data bytes
.....................................................................
--- fda0:7a62:a::b ping statistics ---
100 packets transmitted, 18 received, 82% packet loss, time 1315ms
rtt min/avg/max/mdev = 0.316/0.845/2.057/0.334 ms, ipg/ewma 13.281/0.931 ms
```

Côte routeur "cible", on relève à nouveau un grand nombre de paquets jetés.

```
sudo nft list table inet raw
```

```
table inet raw {
        chain rpfilter {
                type filter hook prerouting priority raw; policy accept;
                iifname "ppp0" fib saddr . iif oif 0 counter packets 0 bytes 0 drop
        }

        chain icmpfilter {
                type filter hook prerouting priority raw; policy accept;
                icmpv6 type echo-request limit rate 10/second burst 5 packets counter packets 93 bytes 9672 accept
                icmpv6 type echo-request counter packets 485 bytes 50440 drop
        }
}
```

Comme on l'a fait pour le protocole IPv4, on vérifie qu'on obtient un retour correct suite à des requêtes émises "à un rythme normal".

```
ping -qc10 fda0:7a62:a::b
```

```
PING fda0:7a62:a::b (fda0:7a62:a::b) 56 data bytes

--- fda0:7a62:a::b ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9018ms
rtt min/avg/max/mdev = 0.839/1.055/1.191/0.097 ms
```

4.3. Protection contre les robots de connexion au service SSH

Comme dans les deux sections précédentes, l'évaluation du mécanisme de protection se joue entre le routeur ***Hub*** et un routeur ***Spoke***. On verra que le cas d'une protection contre les robots SSH, celle-ci doit s'appliquer sur tous les systèmes qui autorisent une connexion par mot de passe.

Le sujet de ce document étant l'étude du filtrage réseau, on s'intéresse ici à la génération automatique de règles suite à la violation des critères définis dans l'application de gestion des tentatives de connexion : fail2ban.

Q21. Quel est la fonction de ***fail2ban*** ?

Afficher la description du paquet ***fail2ban*** après l'avoir installé.

```
sudo apt -y install fail2ban
```

```
apt show fail2ban | grep  Description
```

```
Description : ban hosts that cause multiple authentication errors Fail2ban
```

Fail2ban est un outil de sécurité qui analyse les journaux système pour détecter et empêcher les attaques par force brute sur les serveurs. Il bloque automatiquement les adresses IP suspectes après plusieurs tentatives de connexion infructueuses au cours d'une période donnée en mettant à jour les règles du pare-feu. Cette application prend en charge différents services, dont SSH, et offre des paramètres personnalisables pour améliorer la protection des serveurs contre les attaques automatisées.

Q22. Quel est le numéro de port utilisé par le service SSH sur les routeurs ?

Il est important de connaître les caractéristiques du service qui doit être surveillé par fail2ban. Rechercher dans la liste des ports réseau ouverts celui qui concerne le service SSH.

Dans le contexte de la maquette, le service SSH a été paramétré pour utiliser le port numéro 2222. On obtient la liste des ports en écoute avec les commandes lsof ou ss.

```
sudo lsof -i tcp:2222 -sTCP:listen
```

```
COMMAND PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd    672 root    7u  IPv4   7601      0t0  TCP *:2222 (LISTEN)
sshd    672 root    8u  IPv6   7603      0t0  TCP *:2222 (LISTEN)
```

```
ss -tapl '( sport = :2222 )' | fmt -t -w80
```

```
State  Recv-Q Send-Q Local Address:Port Peer Address:PortProcess
LISTEN 0      128          0.0.0.0:2222      0.0.0.0:*
LISTEN 0      128             [::]:2222         [::]:*
```

Ce sont donc les tentatives de connexion au service SSH sur le port numéro ***2222*** que le service fail2ban doit surveiller.

Q23. Comment créer un fichier de configuration des paramètres de gestion du service SSH ?

Rechercher dans la documentation de fail2ban les paramètres de réglage suivants.

- Le numéro de port
- Le service à filtrer
- La règle à appliquer en cas de violation du nombre de tentatives de connexion autorisées
- Le nombre de tentatives de connexion autorisées
- La durée de mise en quarantaine de l'adresse IPv4 ou IPv6
- La durée pendant laquelle on comptabilise les tentatives de connexion

Pour traiter la demande, on créé un fichier spécifique dans le répertoire dédié : `/etc/fail2ban/jail.d/custom-sshd.conf` qui contient tous les paramètres utiles.

```
cat << EOF | sudo tee /etc/fail2ban/jail.d/custom-sshd.conf
[sshd]
enabled = true
port = 2222
filter = sshd
backend = systemd
banaction = nftables-multiport
maxretry = 3
bantime = 1h
findtime = 10m
EOF
```

De manière classique, il convient de redémarrer le service pour que le nouveau fichier de configuration soit pris en compte.

```
sudo systemctl restart fail2ban
```

Q24. Quels sont les outils qui permettent de connaître l'état du service et le fonctionnement du nouveau filtrage des tentatives de connexion ?

Relever l'état du service pour commencer. Ensuite, rechercher dans les outils fournis avec le paquet fail2ban, celui qui affiche les informations sur le filtrage SSH.

L'affichage de l'état du service n'a rien de spécifique à fail2ban, mais c'est le point de départ obligatoire.

```
systemctl status fail2ban.service
```

Les mots clé recherchés sont : ***enabled*** et ***active (running)***.

Pour la partie plus spécifique à fail2ban, on liste les fichiers contenus dans le paquet du même nom.

```
dpkg -L fail2ban | grep bin
```

```
/usr/bin
/usr/bin/fail2ban-client
/usr/bin/fail2ban-regex
/usr/bin/fail2ban-server
/usr/bin/fail2ban-testcases
/usr/bin/fail2ban-python
```

C'est la commande fail2ban-client qui nous intéresse. La recherche du mot clé `status` dans les pages de manuels donne le mode opératoire pour accéder au statut des services traités.

```
sudo fail2ban-client status
```

```
Status
|- Number of jail:      1
`- Jail list:   sshd
```

```
sudo fail2ban-client status sshd
```

```
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     0
|  `- Journal matches:  _SYSTEMD_UNIT=ssh.service + _COMM=sshd
`- Actions
   |- Currently banned: 0
   |- Total banned:     0
   `- Banned IP list:
```

Q25. Comment caractériser le fonctionnement du service fail2ban ?

Si le service a été installé et configuré sur un routeur ***Spoke***, il est possible de lancer plusieurs tentatives de connexion SSH depuis le routeur ***Hub*** en se trompant de mot de passe.

On peut alors afficher les règles de filtrage nftables et obtenir la liste des adresses bannies par fail2ban.

On commence par lancer plusieurs tentatives (au moins 3) de connexion SSH à partir du routeur ***Hub***.

```
ssh -p 2222 etu@10.44.1.2
etu@10.44.1.2's password:
Permission denied, please try again.
etu@10.44.1.2's password:
Permission denied, please try again.
etu@10.44.1.2's password:
etu@10.44.1.2: Permission denied (publickey,password)
```

Une fois le mécanisme de blocage activé, il est impossible de tenter une nouvelle connexion.

```
ssh -p 2222 etu@10.44.1.2
ssh: connect to host 10.44.1.2 port 2222: Connection refused
```

On relève ensuite les résultats côté routeur ***Spoke***.

L'état de la quarantaine montre que l'adresse IPv4 du routeur ***Hub*** sur le lien PPP est en quarantaine.

```
sudo fail2ban-client status sshd
```

```
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     3
|  `- Journal matches:  _SYSTEMD_UNIT=ssh.service + _COMM=sshd
`- Actions
   |- Currently banned: 1
   |- Total banned:     1
   `- Banned IP list:   10.44.1.1
```

La liste des règles de filtrage montre qu'une nouvelle chaîne a été ajoutée. Dans cette chaîne, on reconnaît l'adresse IPv4 du lien PPP côté ***Hub***.

```
sudo nft list ruleset
```

En limitant l'affichage à la table ajoutée par le service fail2ban, on obtient le jeu de règles suivant.

```
sudo nft list table inet f2b-table
```

```
table inet f2b-table {
    set addr-set-sshd {
        type ipv4_addr
        elements = { 10.44.1.1 }
    }

    chain f2b-chain {
        type filter hook input priority filter - 1; policy accept;
        tcp dport 2222 ip saddr @addr-set-sshd reject with icmp port-unreachable
    }
}
```

Enfin, on répète l'opération avec l'adresse IPv6 du routeur ***Spoke*** sur le lien PPP.

```
ssh -p 2222 etu@fe80::c59:5d57:7476:13cc%ppp0
etu@fe80::c59:5d57:7476:13cc%ppp0's password:
Permission denied, please try again.
etu@fe80::c59:5d57:7476:13cc%ppp0's password:
Permission denied, please try again.
etu@fe80::c59:5d57:7476:13cc%ppp0's password:
etu@fe80::c59:5d57:7476:13cc%ppp0: Permission denied (publickey,password).
```

```
ssh -p 2222 etu@fe80::c59:5d57:7476:13cc%ppp0
ssh: connect to host fe80::c59:5d57:7476:13cc%ppp0 port 2222: Connection refused
```

On voit apparaître une nouvelle adresse dans la liste sur le routeur ***Spoke***.

```
sudo fail2ban-client status sshd
```

```
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed:     6
|  `- Journal matches:  _SYSTEMD_UNIT=ssh.service + _COMM=sshd
`- Actions
   |- Currently banned: 2
   |- Total banned:     2
   `- Banned IP list:   10.44.1.1 fe80::f153:4881:c7c4:f371
```

Les règles de filtrage pour le protocole IPv6 ont aussi été complétées.

```
sudo nft list table inet f2b-table
```

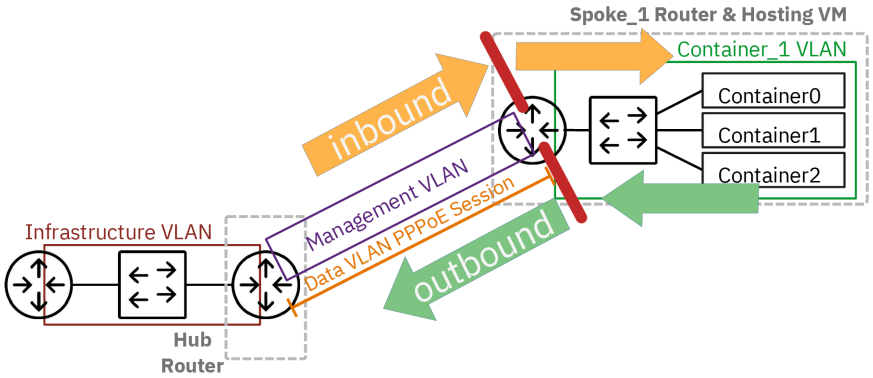
```
table inet f2b-table {
    set addr-set-sshd {
        type ipv4_addr
        elements = { 10.44.1.1 }
    }

    set addr6-set-sshd {
        type ipv6_addr
        elements = { fe80::f153:4881:c7c4:f371 }
    }

    chain f2b-chain {
        type filter hook input priority filter - 1; policy accept;
        tcp dport 2222 ip saddr @addr-set-sshd reject with icmp port-unreachable
        tcp dport 2222 ip6 saddr @addr6-set-sshd reject with icmpv6 port-unreachable
    }
}
```

En appliquant cette configuration du service fail2ban sur les trois routeurs, on dispose d'une base de protection contre les attaques de type ***brute force*** sur l'authentification au service SSH.

5. Filtrage des flux réseaux traversant les routeurs Spoke

Pour rappel, la mise en place du filtrage réseau sur les équipements doit répondre à deux principes.

- On considère que les équipements d'interconnexion mis en œuvre dans ces travaux pratiques délimitent des périmètres de dimension moyenne. Par conséquent, on a une connaissance exhaustive des flux réseaux sur le système. On adopte donc la règle suivante : ***tout trafic réseau non autorisé est interdit***.
- Le filtrage est basé sur le suivi de communication (***stateful inspection***). On cherche donc à écrire des règles qui ***décrivent le plus précisément possible le premier paquet qui doit être enregistré dans la table de suivi de communication***. Ces règles de description du premier paquet doivent être placées après celles qui laissent passer le trafic correspondant ou relatif à une communication déjà enregistrée dans les tables.
- Afin de simplifier l'étude du filtrage, on fait le choix d'autoriser tous les flux sortants émis par les routeurs *Hub* et *Spoke*.



Caractérisation des flux réseau traversant un routeur *Spoke* avec identification du sens entrant ou sortant

On commence par afficher les règles actives sur un routeur *Spoke* à l'issue des questions de la section précédente : Section 4, « Protection de base des routeurs Hub et Spoke ».

> ⚠ Attention
>
> Les noms d'interfaces correspondent à la maquette de test.

```
sudo nft list ruleset
```

```
table inet raw {
        chain rpfilter {
                type filter hook prerouting priority raw; policy accept;
                iifname "ppp0" fib saddr . iif oif 0 counter packets 0 bytes 0 drop
        }

        chain icmpfilter {
                type filter hook prerouting priority raw; policy accept;
                icmp type echo-request limit rate 10/second burst 5 packets counter packets 6821 bytes 193228 accept
                icmp type echo-request counter packets 116414622 bytes 3259609416 drop
        }
}
table inet f2b-table {
        set addr-set-sshd {
                type ipv4_addr
                elements = { 10.44.1.1 }
        }

        set addr6-set-sshd {
                type ipv6_addr
                elements = { fe80::f153:4881:c7c4:f371 }
        }

        chain f2b-chain {
                type filter hook input priority filter - 1; policy accept;
                tcp dport 2222 ip saddr @addr-set-sshd reject with icmp port-unreachable
                tcp dport 2222 ip6 saddr @addr6-set-sshd reject with icmpv6 port-unreachable
        }
}
```

Q26. Quel est le nom de la table du système `netfilter` utilisée par défaut pour le traitement des flux traversant un routeur ?

Consulter la représentation graphique *Packet Flow in Netfilter* et repérer le nom des tables concernées par le transfert des flux réseau.

On commence par identifier la colonne ***FORWARD PATH*** au centre de la représentation.

Dans cette colonne, on trouve les tables `mangle` et `filter`.

Une recherche sur les différences entre ces deux tables nous donne les indications suivantes.

Table filter

Cette table est principalement utilisée pour filtrer les paquets et prendre des décisions concernant l'autorisation ou le blocage du trafic. Elle sert à contrôler quels paquets sont autorisés à entrer dans le système, à le quitter ou à le traverser.

Table mangle

La table d'altération est utilisée pour la modification de paquets. Les altérations principales portent sur les en-têtes, les métadonnées et le marquage, pour permettre un traitement ultérieur par d'autres tables ou les outils de gestion de la qualité de service.

Q27. Quel est le nom de la ***chaîne*** de traitement des flux traversant les routeurs ?

Consulter la représentation graphique *Packet Flow in Netfilter* et repérer les noms des chaînes de la table `filter`.

Comme on se concentre sur la colonne ***FORWARD PATH*** de la représentation graphique, on repère facilement la chaîne `forward`.

Voici une proposition de jeu de règles de filtrage à implanter dans la chaîne `forward` de la table `filter` des routeurs *Spoke*.

```
table inet filter {
    chain forward {
        type filter hook forward priority 0; policy drop;

        # Allow outbound new connections
        oifname "ppp0" ct state new counter accept

        # Allow established and related connections
        ct state established,related accept

        # Allow specific inbound traffic
        # ICMP IPv4 + IPv6
        iifname "ppp0" meta l4proto {icmp, ipv6-icmp} ct state new counter accept
        # SSH
        iifname "ppp0" tcp dport 2222 ct state new counter accept
        # HTTP(S)
        iifname "ppp0" meta l4proto {tcp, udp} th dport {80, 443} ct state new counter accept

        # Count dropped packets
        counter comment "count dropped packets"
    }
}
```

Q28. Quelle est la politique par défaut appliquée par ce nouveau jeu de règles?

Rechercher le mot clé ***policy*** dans la copie ci-dessus.

La politique appliquée à tous les paquets qui passent par la chaîne `forward` est `drop`.

Tous les flux qui ne sont pas explicitement autorisés dans les règles de la chaîne `forward` sont jetés.

Q29. Comment identifier le sens du flux qui traverse le routeur dans le jeu de règles proposées ?

Rechercher les mots clé associés à l'interface réseau qui raccorde le routeur ***Spoke*** au ***Hub***

Dans la topologie étudiée, c'est l'interface `ppp0` du site distant qui donne accès à tous les autres réseaux. Dans les règles proposées, cette interface est précédées des clés `iifname` ou `oifname`.

iifname

La clé se lit ***Input InterFace NAME*** et désigne les flux entrants par l'interface `ppp0`. Vue du routeur, on emploie l'expression ***inbound traffic*** en anglais.

oifname

La clé se lit ***Output InterFace NAME*** et désigne les flux sortants par l'interface `ppp0`. Vue du routeur, on emploie l'expression ***outbound traffic*** en anglais.

Q30. Comment est géré l'enregistrement des états de flux sortants ?

Rechercher les règles dans lesquelles les états `new`, `established` et `related` apparaissent.

Dans le cas des flux sortants, on voit que tous les nouveaux paquets sont autorisés à passer par l'interface `ppp0` en sortie. La règle associe l'état `new` à l'autorisation `accept`.

```
oifname "ppp0" ct state new counter accept
```

Q31. Comment est géré l'enregistrement des états de flux entrants ?

Rechercher à nouveau les règles dans lesquelles l'état `new` est associé à la clé `iifname`.

Dans le cas des flux entrants, on voit qu'une liste fermée de protocoles sont acceptés. Tous les nouveaux flux qui ne correspondent pas à l'une des règles avec la décision `accept` sont jetés.

Prenons l'exemple du protocole HTTP, une règle autorise l'accès aux ports `80` et `443`.

```
iifname "ppp0" meta l4proto {tcp, udp} th dport {80, 443} ct state new counter accept
```

Q32. Comment sont gérés les flux déjà admis dans le système d'enregistrement ?

Rechercher la règle qui contient les états `established` et `related`.

Cette question touche au cœur du filtrage ***stateful***. Tout flux retour relatif à une première transaction admise ou enregistrée est accepté. C'est la raison pour laquelle on doit insister encore et encore sur le fait qu'il faut décrire le premier paquet admis dans le système d'enregistrement de la façon la plus exhaustive possible.

Voici la règle qui laisse passer tous les flux relatifs à un enregistrement déjà effectué.

```
ct state established,related accept
```

Q33. Comment appliquer le jeu de règles proposé sur les routeurs ***Spoke*** ?

On dispose de plusieurs réponse à cette question. Dans le but de rendre l'application des règles de filtrage pérenne, on complète le fichier `/etc/nftables.conf` et on relance le service.

On a une fois de plus recours au ***heredoc***.

```
cat << 'EOF' | sudo tee -a /etc/nftables.conf

table inet filter {
    chain forward {
        type filter hook forward priority 0; policy drop;

        # Allow outbound new connections
        oifname "ppp0" ct state new counter accept

        # Allow established and related connections
        ct state established,related accept

        # Allow specific inbound traffic
        # ICMP IPv4 + IPv6
        iifname "ppp0" meta l4proto {icmp, ipv6-icmp} ct state new counter accept
        # SSH
        iifname "ppp0" tcp dport 2222 ct state new counter accept
        # HTTP(S)
        iifname "ppp0" meta l4proto {tcp, udp} th dport {80, 443} ct state new counter accept

        # Count dropped packets
        counter comment "count dropped packets"
    }
}
EOF
```

Ensuite, on relance le service et on vérifie que le jeu de règles actives est complet.

```
sudo systemctl restart nftables.service
```

```
sudo nft list ruleset
```

Q34. Comment tester les règles relatives aux flux sortants ?

Pour tester la chaîne `forward` dans le sens réseau d'hébergement vers le routeur ***Hub***, il faut initier du trafic depuis les conteneurs.

Voici un exemple qui passe par la couche application : la mise à jour du catalogue de paquets de chaque conteneur.

```
for i in {0..2}
do
    echo ">>>>>>>>>>>>>>>> c$i"
    incus exec c$i -- apt update
done
```

Une fois la mise à jour effectuée dans les conteneurs, on peut relever les compteurs associés aux règles de filtrage.

```
sudo nft list table inet filter
```

```
table inet filter {
        chain forward {
                type filter hook forward priority filter; policy drop;
                oifname "ppp0" ct state new counter packets 41 bytes 3078 accept
                ct state established,related accept
                iifname "ppp0" meta l4proto { icmp, ipv6-icmp } ct state new counter packets 2 bytes 188 accept
                iifname "ppp0" tcp dport 2222 ct state new counter packets 0 bytes 0 accept
                iifname "ppp0" meta l4proto { tcp, udp } th dport { 80, 443 } ct state new counter packets 1 bytes 60 accep
                counter packets 0 bytes 0 comment "count dropped packets"
        }
}
```

Q35. Comment tester les règles relatives aux flux entrants ?

Il suffit de reprendre les séquences de tests présentées à la fin du support de travaux pratiques précédent : *Topologie Hub & Spoke avec le protocole PPPoE* à la question 21.

On reprend l'accès aux pages Web depuis le routeur ***Hub***.

```
for addr in {10..12}
do
    wget -O /dev/null http://10.0.10.$addr 2>&1 | grep "HTTP "
done
```

```
for addr in {10..12}
do
    wget -O /dev/null http://[fda0:7a62:a::$(printf "%x" $addr)] 2>&1 | grep "HTTP "
done
```

Ensuite, on vérifie la nouvelle valeur de comptage sur la règle de filtrage concernée.

```
sudo nft list table inet filter
```

```
table inet filter {
        chain forward {
                type filter hook forward priority filter; policy drop;
                oifname "ppp0" ct state new counter packets 0 bytes 0 accept
                ct state established,related counter packets 81 bytes 13581 accept
                iifname "ppp0" meta l4proto { icmp, ipv6-icmp } ct state new counter packets 0 bytes 0 accept
                iifname "ppp0" tcp dport 2222 ct state new counter packets 0 bytes 0 accept
                iifname "ppp0" meta l4proto { tcp, udp } th dport { 80, 443 } ct state new counter packets 9 bytes 600 acce
                counter packets 0 bytes 0 comment "count dropped packets"
        }
}
```

Une fois ces règles basiques en place, on peut aborder les filtrages réseau spécifiques à la topologie de travaux pratiques.

6. Traduction d'adresses destination sur le routeur Hub

Dans cette partie, on doit compléter les règles de filtrage sur le routeur ***Hub*** en appliquant la même politique que sur les routeurs ***Spoke*** pour les flux traversants : tout ce qui n'est pas autorisé est interdit. En complément à cette politique par défaut, on doit aussi répondre à deux objectifs :

- Autoriser le trafic provenant des routeurs ***Spoke*** vers l'Internet.
- Autoriser l'accès aux services Web hébergés sur les conteneurs des réseaux d'hébergement des sites distants à l'aide de la traduction des adresses destination.

  Voici un exemple de correspondances de numéros de ports pour l'accès aux différents services web.

Tableau 1. Correspondance entre numéro de port et service Web

numéros de port Hub : http,https	conteneur
8010,8453	`10.0.10.10` `fda0:7a62:a::a`
8011,8454	`10.0.10.11` `fda0:7a62:a::b`
8012,8455	`10.0.10.12` `fda0:7a62:a::c`
8020,8463	`10.0.20.10` `fda0:7a62:14::a`
8021,8464	`10.0.20.11` `fda0:7a62:14::b`
8022,8465	`10.0.20.12` `fda0:7a62:14::c`

Avant d'aborder les questions, on commence par afficher l'état courant du jeu de règles de filtrage sur le routeur ***Hub***.

```
#!/usr/sbin/nft -f

flush ruleset

# Outbound interface
define RED_VLAN = enp0s1.360

table inet nat {
    chain postrouting {
        type nat hook postrouting priority 100;
        oifname $RED_VLAN counter masquerade
    }
}

table inet raw {
    chain rpfilter {
        type filter hook prerouting priority raw; policy accept;
        iifname $RED_VLAN fib saddr . iif oif 0 counter packets 0 bytes 0 drop
    }

    # ICMP Rate Limiting Rules
    chain icmpfilter {
        type filter hook prerouting priority raw; policy accept;
        icmp type echo-request limit rate 10/second burst 5 packets counter accept
        icmp type echo-request counter drop
    }
}

table inet filter {

    chain forward {
        type filter hook forward priority 0; policy drop;

        # Allow outbound new connections
        iifname "ppp*" ct state new counter accept

        # Allow established and related connections
        ct state established,related counter accept

        # Count dropped packets
        counter comment "count dropped packets"
    }
}
```

Q36. Comment appliquer la politique par défaut, autoriser et enregistrer dans le mécanisme de suivi des états les flux traversants depuis les routeurs *Spoke* ?

Reprendre le jeu de règles de la Section 5, « Filtrage des flux réseaux traversant les routeurs Spoke » et adapter ce jeu au contexte du routeur *Hub*.

Il faut notamment choisir sur quelle(s) interface(s) la règle doit s'appliquer pour ne pas bloquer les flux entre routeurs *Spoke*.

Voici un exemple de règles basées sur l'utilisation des interfaces de raccordement des routeurs *Spoke*.

```
table inet filter {
    chain forward {
        type filter hook forward priority filter; policy drop;
        iifname "ppp*" ct state new counter packets 6 bytes 504 accept
        ct state established,related counter packets 18 bytes 1512 accept
        counter packets 0 bytes 0 comment "count dropped packets"
    }
}
```

Dans ce cas de figure le trafic entrant par les interfaces PPP est autorisé à traverser le routeur *Hub*.

Q37. Comment valider l'utilisation de ces nouvelles règles à partir d'un routeur *Spoke* ?

Il suffit de lancer une mise à jour de catalogue de paquets dans les conteneurs hébergés sur un routeur *Spoke*.

On peut aussi utiliser la commande conntrack pour afficher la liste des enregistrements en cours sur le routeur *Hub* pendant les transactions des conteneurs.

Voici un exemple de trace qui montre que les flux initiés par les conteneurs sont correctement acheminés par le routeur *Hub*.

- Côté routeur *Spoke*, on initie le trafic.

```
for i in {0..2}
do
    echo ">>>>>>>>>>>>>>>> c$i"
    incus exec c$i -- apt update
done
```

- Côté routeur ***Hub***, on relève “rapidement” la liste des enregistrements relatif à une adresse de conteneur.

```
sudo conntrack -L -j 10.0.10.1
```

```
tcp      6 36 TIME_WAIT src=10.0.10.10 dst=151.101.2.132 sport=39664 dport=80
    src=151.101.2.132 dst=192.168.104.130 sport=80 dport=39664 [ASSURED] mark=0 use=1
```

Q38. Comment implanter les règles de traduction d'adresses IPv4 et IPv6 destination pour ouvrir l'accès aux services Web placés dans les conteneurs des réseaux d'hébergement des routeurs ***Spoke*** ?

Rechercher la chaîne présente par défaut à utiliser pour les traitements de traduction d'adresses et de ports destination dans la représentation graphique *Packet Flow in Netfilter*.

Une fois la chaîne identifiée, rechercher des exemples de règles de filtrage de type `dnat` et une adaptation au critères de redirection de ports définis dans le tableau ci-dessus.

Bien sûr, les adresses IPv4 et IPv6 des conteneurs hébergés sur les routeur ***Spoke*** doivent être modifiées en fonction du plan d'adressage du document *Topologie Hub & Spoke avec le protocole PPPoE*.

Le nom de chaîne recherchée est `prerouting`. En effet, la traduction d'adresse destination doit se faire avant la décision de routage des paquets IPv4 ou IPv6.

Voici un extrait du fichier `/etc/nftables.conf` qui ne comprend que les règles définies dans la table `nat`.

```
table inet nat {
    chain postrouting {
        type nat hook postrouting priority 100;
        oifname $RED_VLAN counter masquerade
    }

    chain prerouting {
        type nat hook prerouting priority dstnat; policy accept;
        # Spoke1
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8010, 8453 } \
            dnat ip to 10.0.10.10
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8010, 8453 } \
            dnat ip6 to fda0:7a62:a::a
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8011, 8454 } \
            dnat ip to 10.0.10.11
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8011, 8454 } \
            dnat ip6 to fda0:7a62:a::b
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8012, 8455 } \
            dnat ip to 10.0.10.12
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8012, 8455 } \
            dnat ip6 to fda0:7a62:a::c
        # Spoke2
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8020, 8463 } \
            dnat ip to 10.0.20.10
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8020, 8463 } \
            dnat ip6 to fda0:7a62:14::a
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8021, 8464 } \
            dnat ip to 10.0.20.11
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8021, 8464 } \
            dnat ip6 to fda0:7a62:14::b
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8022, 8465 } \
            dnat ip to 10.0.20.12
        iifname $RED_VLAN meta l4proto { tcp, udp } th dport { 8022, 8465 } \
            dnat ip6 to fda0:7a62:14::c
    }
}
```

Relativement à l'unique règle de la chaîne `postrouting`, il est nécessaire d'ajouter deux règles par destination en distinguant les protocoles IPv4 et IPv6. Dans le contexte de la maquette, il y a trois conteneurs hébergés sur chaque routeur ***Spoke***. On doit donc ajouter 6 règles pour chacun d'eux.

Les règles de traduction d'adresses de la chaîne `prerouting` sont structurées de la façon suivante :

- Une même règle traite les protocoles de couche transport TCP et UDP.
- Les numéros de ports sont aussi regroupés. Par exemple : { 8010, 8453 }.
- Relativement aux règles des autres parties du document, les protocoles IPv4 et IPv6 sont séparés et les adresses destination sont explicitement définies pour chaque routeur ***Spoke***.

Q39. Comment tester le ces nouvelles règles de traduction d'adresse destination sur le routeur ***Hub*** ?

Comme il s'agit de flux réseau entrants sur le routeur ***Hub***, il faut nécessairement lancer les tests depuis le réseau d'infrastructure, c'est-à-dire le VLAN rouge.

La connexion au Shell de l'hyperviseur nous permet de lancer nos tests à destination de l'adresse du routeur ***Hub*** dans le réseau d'infrastructure, c'est-à-dire le VLAN rouge. Voici un exemple avec l'adresse définie dans le plan d'adressage de la maquette utilisée pour ce document.

```
wget http://192.168.104.130:8010
--2024-10-13 15:47:25--  http://192.168.104.130:8010/
Connexion à 192.168.104.130:8010…
```

Si l'adresse IPv4 et le numéro de port utilisés pour ce test sont corrects, cela ne suffit toutefois pas.

Q40. Comment autoriser les flux de la traduction d'adresse destination à traverser le routeur ***Hub*** ?

Rechercher à nouveau le nom de la chaîne présente par défaut qui traite les flux réseau qui traversent le routeur ***Hub***.

Une fois la chaîner identifiée, rechercher la syntaxe des règles qui admettent les premiers paquets de traitement de la traduction d'adresse destination dans le mécanisme de suivi des communications du système de filtrage.

C'est la chaîne `forward` qui nous intéresse ici. On a déjà créé des règles dans cette chaîne qui traitent les flux sortants du routeur ***Hub*** provenant des routeurs ***Spoke***.

Voici un exemple de jeu de règles pour la chaîne `forward` adapté au plan d'adressage de la maquette.

```
table inet filter {
    chain forward {
        type filter hook forward priority 0; policy drop;

        # Allow outbound new connections
        iifname "ppp*" ct state new counter accept

        # Allow inbound new connections to the web services hosted on spoke routers
        # Spoke1
        iifname $RED_VLAN ip daddr 10.0.10.10 meta l4proto { tcp, udp } \
            th dport { 8010, 8453 } ct state new counter accept
        iifname $RED_VLAN ip6 daddr fda0:7a62:a::a meta l4proto { tcp, udp } \
            th dport { 8010, 8453 } ct state new counter accept
        iifname $RED_VLAN ip daddr 10.0.10.11 meta l4proto { tcp, udp } \
            th dport { 8011, 8454 } ct state new counter accept
        iifname $RED_VLAN ip6 daddr fda0:7a62:a::b meta l4proto { tcp, udp } \
            th dport { 8011, 8454 } ct state new counter accept
        iifname $RED_VLAN ip daddr 10.0.10.12 meta l4proto { tcp, udp } \
            th dport { 8012, 8455 } ct state new counter accept
        iifname $RED_VLAN ip6 daddr fda0:7a62:a::c meta l4proto { tcp, udp } \
            th dport { 8012, 8455 } ct state new counter accept
        # Spoke2
        iifname $RED_VLAN ip daddr 10.0.20.10 meta l4proto { tcp, udp } \
            th dport { 8020, 8463 } ct state new counter accept
        iifname $RED_VLAN ip6 daddr fda0:7a62:14::a meta l4proto { tcp, udp } \
            th dport { 8020, 8463 } ct state new counter accept
        iifname $RED_VLAN ip daddr 10.0.20.11 meta l4proto { tcp, udp } \
            th dport { 8021, 8464 } ct state new counter accept
        iifname $RED_VLAN ip6 daddr fda0:7a62:14::b meta l4proto { tcp, udp } \
            th dport { 8021, 8464 } ct state new counter accept
        iifname $RED_VLAN ip daddr 10.0.20.12 meta l4proto { tcp, udp } \
            th dport { 8022, 8465 } ct state new counter accept
        iifname $RED_VLAN ip6 daddr fda0:7a62:14::c meta l4proto { tcp, udp } \
            th dport { 8022, 8465 } ct state new counter accept

        # Allow established and related connections
        ct state established,related counter accept

        # Count dropped packets
        counter comment "count dropped packets"
    }
}
```

Comme dans le cas de la chaîne `prerouting`, il est nécessaire de décrire le plus finement possible le premier flux autorisé à entrer dans le mécanisme de suivi des communication. Cela engendre un nombre important de règles même si on parvient à regrouper les deux protocoles de la couches transport et les numéros de ports.

Une fois ce jeu de règles implanté, ce n'est toujours pas suffisant pour accéder aux site web de chaque conteneur du réseau d'hébergement. On peut tout de même qualifier les règles à l'aide des compteurs et de l'affichage des enregistrements du suivi des communications. Voici un exemple.

On commence par lancer une requête qui ne parvient pas à charger la page web du service nginx.

```
wget http://192.168.104.130:8010
--2024-10-13 18:34:00--  http://192.168.104.130:8010/
Connexion à 192.168.104.130:8010… échec : Connexion terminée par expiration du délai d'attente.
```

Côté routeur ***Hub***, on peut examiner les compteurs relatifs à la règle qui autorise l'accès au service Web du conteneur à l'adresse IPv4 `10.0.10.10`.

```
sudo nft list table inet filter | grep '10.0.10.10'
    iifname "enp0s1.360" ip daddr 10.0.10.10 meta l4proto { tcp, udp } th dport { 8010, 8453 }
        ct state new counter packets 22 bytes 1320 accept
```

L'affichage de la liste des enregistrements de suivi des communication au niveau du routeur ***Hub*** montre qu'aucune réponse à la requête n'a été vue avec la clé `UNREPLIED`.

```
 sudo conntrack -f ipv4 -L | grep "10.0.10.10"
conntrack v1.4.8 (conntrack-tools): 2 flow entries have been shown.
tcp      6 87 SYN_SENT src=172.16.0.6 dst=192.168.104.130 sport=48430 dport=8010
    [UNREPLIED] src=10.0.10.10 dst=172.16.0.6 sport=8010 dport=48430 mark=0 use=1
```

Q41. Comment reconfigurer les services Web des conteneurs pour qu'ils soient en écoute sur les numéros de ports définis dans les règles de filtrage ci-dessus ?

On sait depuis l'installation des services nginx qu'ils sont tous en écoute sur le port `80`. On doit donc reconfigurer ces mêmes services avec les numéros de ports définis dans le tableau donné au début de cette partie.

⚠ Attention

Pour les besoins de nos tests, on se contente du protocole HTTP et on ne traite pas les ports HTTPS.

On doit réaliser deux tâches sur chaque routeur ***Spoke***.

- Définir le numéro de port en écoute pour chaque service nginx sur chaque conteneur.
- Compléter la liste des ports autorisés dans les règles de filtrage Web.

Voici un exemple des traitements à réaliser sur le routeur ***Spoke1***.

Pour reconfigurer les services Web, on lance un script avec comme celui-ci.

```
for i in {0..2}
do
    echo ">>>>>>>>>>>>>>>> c$i"
    incus exec c$i -- \
        sh -c "sed -i 's/listen 80/listen $((8010 + i))/' /etc/nginx/sites-enabled/default"
    incus exec c$i -- \
        sh -c "sed -i 's/listen \[::\]:80/listen \[::\]:$(( 8010 + i))/' /etc/nginx/sites-enabled/default"
    incus exec c$i -- systemctl restart nginx
done
```

Pour la liste des ports autorisés, on complète la ligne des services Web de la chaîne `forward` de la table `filter`. Voici une copie de la table.

```
table inet filter {
    chain forward {
        type filter hook forward priority 0; policy drop;

        # Allow outbound new connections
        oifname "ppp0" ct state new counter accept

        # Allow established and related connections
        ct state established,related counter accept

        # Allow specific inbound traffic
        # ICMP IPv4 + IPv6
        iifname "ppp0" meta l4proto {icmp, ipv6-icmp} ct state new counter accept
        # SSH
        iifname "ppp0" tcp dport 2222 ct state new counter accept
        # HTTP(S)
        iifname "ppp0" meta l4proto {tcp, udp} th dport \
            {80, 443, 8010, 8011, 8012, 8453, 8454, 8455} ct state new counter accept

        # Count dropped packets
        counter comment "count dropped packets"
    }
}
```

Pour terminer, les tests d'accès depuis l'hyperviseur sont maintenant concluants pour les protocoles IPv4 et IPv6.

```
for p in {0..2}
do
    wget -O /dev/null http://192.168.104.130:$((8010 + $p))
done
```

```
--2024-10-13 19:24:24--  http://192.168.104.130:8010/
Connexion à 192.168.104.130:8010… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 615 [text/html]
Sauvegarde en : « /dev/null »

/dev/null  100%[=============================================>]     615  --.-KB/s    ds 0s

2024-10-13 19:24:24 (33,6 MB/s) — « /dev/null » sauvegardé [615/615]

--2024-10-13 19:24:24--  http://192.168.104.130:8011/
Connexion à 192.168.104.130:8011… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 615 [text/html]
Sauvegarde en : « /dev/null »

/dev/null  100%[=============================================>]     615  --.-KB/s    ds 0s

2024-10-13 19:24:24 (34,4 MB/s) — « /dev/null » sauvegardé [615/615]

--2024-10-13 19:24:24--  http://192.168.104.130:8012/
Connexion à 192.168.104.130:8012… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 615 [text/html]
Sauvegarde en : « /dev/null »

/dev/null  100%[=============================================>]     615  --.-KB/s    ds 0s

2024-10-13 19:24:24 (36,3 MB/s) — « /dev/null » sauvegardé [615/615]
```

```
for p in {0..2}
do
    wget -O /dev/null http://[2001:678:3fc:168::82]:$((8010 + $p))
done
```

```
--2024-10-13 19:50:30--  http://[2001:678:3fc:168::82]:8010/
Connexion à [2001:678:3fc:168::82]:8010… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 615 [text/html]
Sauvegarde en : « /dev/null »

/dev/null  100%[=============================================>]     615  --.-KB/s    ds 0s

2024-10-13 19:50:30 (33,5 MB/s) — « /dev/null » sauvegardé [615/615]

--2024-10-13 19:50:30--  http://[2001:678:3fc:168::82]:8011/
Connexion à [2001:678:3fc:168::82]:8011… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 615 [text/html]
Sauvegarde en : « /dev/null »

/dev/null  100%[=============================================>]     615  --.-KB/s    ds 0s

2024-10-13 19:50:30 (39,0 MB/s) — « /dev/null » sauvegardé [615/615]

--2024-10-13 19:50:30--  http://[2001:678:3fc:168::82]:8012/
Connexion à [2001:678:3fc:168::82]:8012… connecté.
requête HTTP transmise, en attente de la réponse… 200 OK
Taille : 615 [text/html]
Sauvegarde en : « /dev/null »

/dev/null 100%[=============================================>]     615  --.-KB/s    ds 0s

2024-10-13 19:50:30 (42,4 MB/s) — « /dev/null » sauvegardé [615/615]
```

C'est gagné ! La traduction d'adresse destination au niveau du routeur ***Hub*** est validée.

7. Pour conclure...

Ce document a présenté les concepts fondamentaux du filtrage réseau avec `netfilter/nftables` sur Linux. Il a couvert la configuration de base des règles de filtrage, la protection contre les attaques courantes, et la mise en place de traductions d'adresses sur une topologie *Hub & Spoke*.

Les travaux pratiques ont permis d'explorer concrètement l'implémentation des règles de filtrage ***stateful*** et ***stateless***. Les étudiants ont pu configurer et tester des règles pour sécuriser les routeurs, contrôler le trafic traversant, et mettre en place de la traduction d'adresse source puis destination.

Ce support fournit une base solide pour comprendre et mettre en œuvre le filtrage réseau avec les outils de filtrage des systèmes Linux. Les compétences acquises permettront aux étudiants de concevoir et déployer des politiques de sécurité réseau efficaces dans des environnements réels.