Filtrage réseau avec netfilter/iptables

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 des autres supports 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

Cet article est écrit avec DocBook XML sur un système Debian GNU/Linux. Il est disponible en version imprimable au format PDF : `interco.netfilter.qa.pdf`.

Toutes les commandes utilisées dans ce document ne sont pas spécifiques à une version particulière des systèmes GNU/Linux. C'est la distribution Debian GNU/Linux qui est utilisée pour les tests présentés. Voici une liste des paquets contenant les commandes :

- procps - utilitaires pour le système de fichiers /proc
- iproute2 - outils de contrôle du trafic et du réseau
- ifupdown - outils de haut niveau pour configurer les interfaces réseau
- iputils-ping - outils pour tester l'accessibilité de noeuds réseaux

- iputils-tracepath - Tools to trace the network path to a remote host
- hping3 - Active Network Smashing Tool
- thc-ipv6 - The Hacker Choice's IPv6 Attack Toolkit
- iptables - outils d'administration pour le filtrage de paquets et le NAT
- iptstate - top-like interface to your netfilter connection-tracking table
- conntrack - programme pour modifier les tables conntrack

Conventions typographiques

Tous les exemples d'exécution des commandes sont précédés d'une invite utilisateur ou prompt spécifique au niveau des droits utilisateurs nécessaires sur le système.

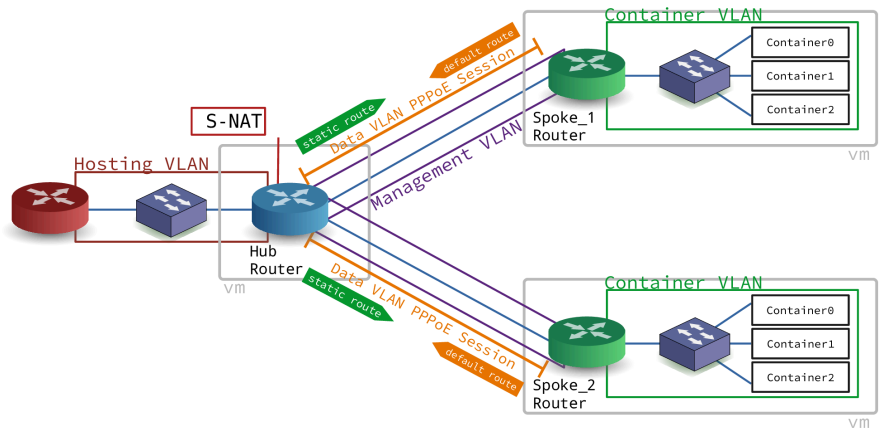
- Toute commande précédée de l'invite `$` ne nécessite aucun privilège particulier et peut être utilisée au niveau utilisateur simple.
- Toute commande précédée de l'invite `#` nécessite les privilèges du super utilisateur.

Le recours aux commandes grep et fmt sert à optimiser la quantité et l'espace occupés dans les copies d'écran données en réponse aux questions. Tous les “tubes“ utilisés pour les copies d'écran peuvent être supprimés de façon à afficher toutes les informations sur un espace qui occupe toute la largeur de la console.

2. Architecture réseau étudiée et filtrage

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.



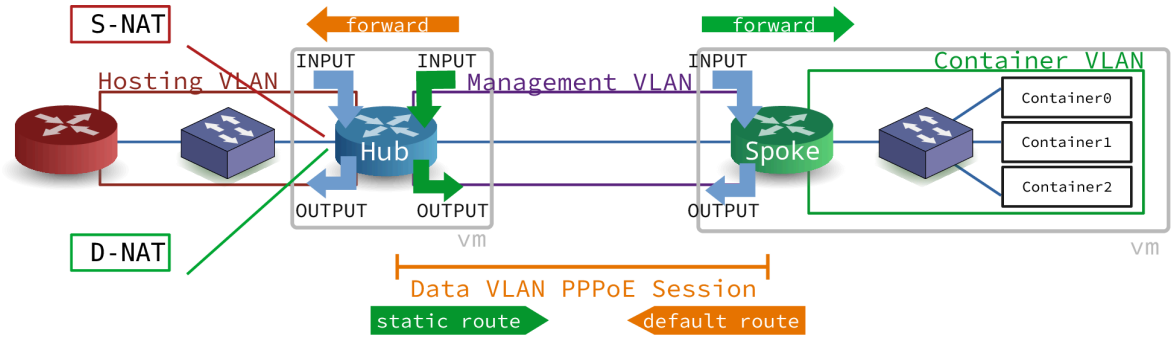
Topologie entre deux routeurs Hub et Spoke avec PPPoE

Routeur centralHubBroadband Remote Access ServerBRAS

Ce routeur réalise une interconnexion LAN/WAN. Il fournit un accès Internet aux routeurs de sites distants via ses interfaces WAN. Il dispose de son propre accès Internet via son interface LAN.

Routeur d'extrémitéSpokeCustomer Premises EquipmentCPE

Ce routeur réalise aussi une interconnexion LAN/WAN. À la différence du routeur Hub, il obtient l'accès Internet sur son interface WAN et il met cet accès à disposition d'un réseau local de site représenté par des conteneurs LXD.



Topologie Hub & Spoke et filtrage

Routage et traduction d'adresses (situation de départ)

Les manipulations qui suivent supposent que la topologie Hub & Spoke est en place et fonctionnelle. On s'appuie sur le support précédent de la série : Topologie Hub & Spoke avec le protocole PPPoE

- Le routeur Hub doit s'assurer que le trafic réseau qu'il route vers et depuis l'Internet correspond bien au plan d'adressage défini. Dans ce but, il attribue les adresses du lien point à point ainsi qu'une route statique à destination du réseau d'extrémité distant.

  Le routeur Hub assure la traduction des adresses sources du réseau distant vers l'Internet.

- Le routeur Spoke doit obtenir son adresse IPv4 de réseau étendu via PPP et assurer le routage de son réseau local. Il dispose d'une route par défaut qui désigne le lien point à point comme seul accès vers l'Internet.

Les questions ci-dessous ont pour objectif de valider le fonctionnement du routage et de la traduction des adresses sources en sortie du routeur Hub vers l'Internet.

Pour traiter les questions, on doit effectuer quelques opérations de vérification et de préparation.

1. On doit d'abord s'assurer que les réseaux des conteneurs de chaque branche de la topologie sont en place. On affiche la liste des conteneurs de chaque routeur Spoke.

```
etu@Spoke1Vert:~$ lxc ls
WARNING: cgroup v2 is not fully supported yet, proceeding with partial confinement
+----------+---------+------------------+----------------------+-----------+-----------+
|   NAME   |  STATE  |       IPV4       |         IPV6         |   TYPE    | SNAPSHOTS |
+----------+---------+------------------+----------------------+-----------+-----------+
| spoke1C0 | RUNNING | 10.0.1.10 (eth0) | fda0:7a62:1::a (eth0) | CONTAINER | 0         |
+----------+---------+------------------+----------------------+-----------+-----------+
| spoke1C1 | RUNNING | 10.0.1.11 (eth0) | fda0:7a62:1::b (eth0) | CONTAINER | 0         |
+----------+---------+------------------+----------------------+-----------+-----------+
| spoke1C2 | RUNNING | 10.0.1.12 (eth0) | fda0:7a62:1::c (eth0) | CONTAINER | 0         |
+----------+---------+------------------+----------------------+-----------+-----------+
```

```
etu@Spoke2Vert:~$ lxc ls
WARNING: cgroup v2 is not fully supported yet, proceeding with partial confinement
+----------+---------+------------------+----------------------+-----------+-----------+
|   NAME   |  STATE  |       IPV4       |         IPV6         |   TYPE    | SNAPSHOTS |
+----------+---------+------------------+----------------------+-----------+-----------+
| spoke2C0 | RUNNING | 10.0.2.10 (eth0) | fda0:7a62:2::a (eth0) | CONTAINER | 0         |
+----------+---------+------------------+----------------------+-----------+-----------+
| spoke2C1 | RUNNING | 10.0.2.11 (eth0) | fda0:7a62:2::b (eth0) | CONTAINER | 0         |
+----------+---------+------------------+----------------------+-----------+-----------+
| spoke2C2 | RUNNING | 10.0.2.12 (eth0) | fda0:7a62:2::c (eth0) | CONTAINER | 0         |
+----------+---------+------------------+----------------------+-----------+-----------+
```

2. On configure le shell par défaut dans les conteneurs des deux routeurs Spoke de façon à pouvoir exécuter des scripts dans ces conteneurs.

   - Création du script qui définit bash comme shell par défaut.

```
etu@Spoke1Vert:~$ cat << EOF > setBashasSh.sh
#!/bin/sh

# make /bin/sh symlink to bash instead of dash:
echo "dash dash/sh boolean false" | debconf-set-selections
DEBIAN_FRONTEND=noninteractive dpkg-reconfigure dash
EOF
```

- Copie du script dans chaque conteneur.

```
etu@Spoke1Vert:~$ for i in {0..2}
do
 lxc file push setBashasSh.sh spoke1C$i/root/
done
```

- Exécution du script dans chaque conteneur.

```
etu@Spoke1Vert:~$ for i in {0..2}
do
 lxc exec spoke1C$i -- sh /root/setBashasSh.sh
done
```

Q1. Comment tracer le chemin suivi par les paquets IPv4 et IPv6 d'un conteneur à un autre conteneur du site distant de l'autre branche de la topologie ?

Rechercher le paquet contenant la commande tracepath qui permet d'afficher le chemin suivi par le trafic réseau.

Par exemple, on se place sur le routeur Spoke2Vert et on installe le paquet `iputils-tracepath` dans les conteneurs avant de lancer les relevés du chemin de bout en bout.

```
etu@Spoke2Vert:~$ for i in {0..2}
do
 lxc exec spoke1C$i -- apt install iputils-tracepath
done
```

une fois le paquet installé, on doit pouvoir contacter les adresses IPv4 et IPv6 des conteneurs situés à l'autre extrémité de la topologie Hub & Spoke.

Par exemple, on relève le chemin entre les conteneurs `spoke2C2` et `spoke1C0`.

```
etu@Spoke2Vert:~$ lxc exec spoke2C2 -- tracepath 10.0.1.10
WARNING: cgroup v2 is not fully supported yet, proceeding with partial confinement
 1?: [LOCALHOST]                      pmtu 1500
 1:  10.0.2.1                                              0.798ms
 1:  10.0.2.1                                              0.100ms
 2:  10.0.2.1                                              0.156ms pmtu 1492
 2:  10.47.3.1                                             1.086ms
 3:  10.47.1.2                                             1.489ms
 4:  10.0.1.10                                             2.518ms reached
     Resume: pmtu 1492 hops 4 back 4
```

```
etu@Spoke2Vert:~$ lxc exec spoke2C2 -- tracepath fda0:7a62:1::a
WARNING: cgroup v2 is not fully supported yet, proceeding with partial confinement
 1?: [LOCALHOST]                       0.043ms pmtu 1492
 1:  fda0:7a62:2::1                                        1.183ms
 1:  fda0:7a62:2::1                                        0.205ms
 2:  2001:678:3fc:12c::2                                   1.059ms
 3:  fda0:7a62:1::1                                        2.003ms
 4:  fda0:7a62:1::a                                        3.453ms reached
     Resume: pmtu 1492 hops 4 back 4
```

Les résultats obtenus avec l'exécution de la commande tracepath montrent que le routage des paquets IPv4 et IPv6 est fonctionnel.

Q2. Comment caractériser la traduction d'adresses source en sortie du routeur Hub ?

La fonction de traduction d'adresse entre dans cadre du filtrage réseau et fait appel aux mêmes outils : netfilter/iptables.

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

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

```
etu@HubBleu:~$ sudo apt -y install conntrack
```

Dans le même temps, on ouvre une autre console sur le routeur Spoke1Vert. C'est à partir de cette console que l'on lance des téléchargements depuis le serveur inetdoc.net à l'aide de la commande wget.

```
etu@Spoke1Vert:~$ for i in {0..2}
do
 lxc exec spoke1C$i -- apt -y install wget
done
```

Sur le routeur Hub, on affiche la liste des enregistrements de la table NAT.

- Requête IPv4 depuis le conteneur :

  ```
  root@container0:~# while true
  do
   wget -4 -O /dev/null https://inetdoc.net/pdf/iproute-cheatsheet.pdf
   sleep 3
  done
  ```

  Liste des enregistrements :

  ```
  etu@HubBleu:~$ sudo conntrack -f ipv4 -L
  udp      17 17 src=10.0.1.10 dst=9.9.9.9 sport=49165 dport=53
      src=9.9.9.9 dst=10.141.0.162 sport=53 dport=49165 mark=0 use=1
  tcp      6 432000 ESTABLISHED src=172.16.0.230 dst=10.141.0.162 sport=40278 dport=22
      src=10.141.0.162 dst=172.16.0.230 sport=22 dport=40278 [ASSURED] mark=0 use=1
  udp      17 20 src=10.0.1.10 dst=9.9.9.9 sport=36074 dport=53
      src=9.9.9.9 dst=10.141.0.162 sport=53 dport=36074 mark=0 use=1
  tcp      6 1 CLOSE src=10.0.1.10 dst=89.234.156.195 sport=44860 dport=443
      src=89.234.156.195 dst=10.141.0.162 sport=443 dport=44860 [ASSURED] mark=0 use=1
  tcp      6 7 CLOSE src=10.0.1.10 dst=89.234.156.195 sport=44864 dport=443
      src=89.234.156.195 dst=10.141.0.162 sport=443 dport=44864 [ASSURED] mark=0 use=1
  udp      17 27 src=10.0.1.10 dst=9.9.9.9 sport=45443 dport=53
      src=9.9.9.9 dst=10.141.0.162 sport=53 dport=45443 mark=0 use=1
  udp      17 24 src=10.0.1.10 dst=9.9.9.9 sport=33499 dport=53
      src=9.9.9.9 dst=10.141.0.162 sport=53 dport=33499 mark=0 use=1
  tcp      6 4 CLOSE src=10.0.1.10 dst=89.234.156.195 sport=44862 dport=443
      src=89.234.156.195 dst=10.141.0.162 sport=443 dport=44862 [ASSURED] mark=0 use=1
  conntrack v1.4.6 (conntrack-tools): 9 flow entries have been shown.
  ```

- Requête IPv6 depuis le conteneur :

  ```
  root@container0:~# while true
  do
   wget -6 -O /dev/null https://inetdoc.net/pdf/iproute-cheatsheet.pdf
   sleep 3
  done
  ```

  Liste des enregistrements :

  ```
  etu@HubBleu:~$ sudo conntrack -f ipv6 -L
  tcp      6 5 CLOSE src=fda0:7a62:1:0:216:3eff:feda:e1a dst=2a03:7220:8083:c300::1 sport=47472 d
      src=2a03:7220:8083:c300::1 dst=2001:678:3fc:12c::2 sport=443 dport=47472 [ASSURED] mark=0 u
  tcp      6 8 CLOSE src=fda0:7a62:1:0:216:3eff:feda:e1a dst=2a03:7220:8083:c300::1 sport=47474 d
      src=2a03:7220:8083:c300::1 dst=2001:678:3fc:12c::2 sport=443 dport=47474 [ASSURED] mark=0 u
  tcp      6 431998 ESTABLISHED src=fe80:1d6::1 dst=fe80:1d6::2 sport=39218 dport=2222
      src=fe80:1d6::2 dst=fe80:1d6::1 sport=2222 dport=39218 [ASSURED] mark=0 use=1
  conntrack v1.4.6 (conntrack-tools): 3 flow entries have been shown.
  ```

3. Les outils de filtrage réseau

Sur un système GNU/Linux, les fonctions de filtrage réseau sont réparties entre les espaces mémoire noyau (kernelspace) et utilisateur (userspace). Les fonctions de filtrage réseau sont disponibles sous forme de modules qui sont chargés dynamiquement dans la mémoire du système en cours d'exécution en fonction de la syntaxe des règles de filtrage ajoutées.

Q3. Quel est le paquet le plus important pour les manipulations sur les fonctions de filtrage réseau ?

Rechercher dans la liste des paquets les mots clés tels que `iptables` ou `firewall`.

La partie userspace des fonctions de filtrage réseau s'appelle iptables. On lance donc une recherche avec ce mot clé dans la base de données des paquets Debian.

```
$ aptitude search ~iiptables
i   iptables              - administration tools for packet filtering and NAT
i   iptables-persistent   - boot-time loader for netfilter rules, iptables plugin
```

Q4. Comment visualiser les modules chargés dynamiquement en fonction de l'utilisation des règles de filtrage réseau ?

Utiliser la commande qui sert à lister les modules chargés en mémoire avant et après avoir consulté les tables de filtrage réseau pour la première fois.

La commande lsmod sert à lister les modules chargés en mémoire. Voici un exemple de liste de modules relatifs au filtrage.

```
$ $ lsmod | egrep '(ip_|nf_)' | fmt -t -w80
nf_conntrack_netlink    57344  0
nf_nat                  49152  2 nft_chain_nat,xt_MASQUERADE
nf_conntrack           176128  3 nf_nat,nf_conntrack_netlink,xt_MASQUERADE
nf_defrag_ipv6          24576  1 nf_conntrack
nf_defrag_ipv4          16384  1 nf_conntrack
libcrc32c               16384  2 nf_conntrack,nf_nat
nf_tables              241664  7 nft_compat,nft_counter,nft_chain_nat
nfnetlink               16384  3 nft_compat,nf_conntrack_netlink,nf_tables
ip_tables               32768  0
x_tables                53248  4 nft_compat,ip_tables,xt_limit,xt_MASQUERADE
```

Q5. Quels sont les outils de sauvegarde et de restauration des jeux de règles de filtrage réseau fournis avec le paquet iptables-persistent ?

Consulter la liste des fichiers du paquet.

La liste des fichiers du paquet fait apparaître les outils iptables-save et iptables-restore qui permettent respectivement de sauvegarder et de restaurer l'ensemble des règles de toutes les tables utilisées.

Ces programmes sont indispensables pour éditer, insérer ou retirer des règles sans avoir à se préoccuper de l'ordre de saisie. De plus, le programme de restauration se charge de l'effacement des règles précédentes.

Q6. Comment visualiser les enregistrements d'états de suivi des communications réseau ?

Rechercher la chaîne `conntrack` dans la liste des paquets.

La section «7.2 Les entrées de conntrack» du Tutoriel iptables décrit précisément les différents champs du suivi de communication.

Voici un échantillon capturé sur le routeur HubBleu après avoir lancé une mise à jour du catalogue des paquets sur les conteneurs du routeur Spoke2Vert.

```
$ sudo conntrack -L | fmt -t -w80
conntrack v1.4.6 (conntrack-tools): 7 flow entries have been shown.
tcp      6 431999 ESTABLISHED src=172.16.0.230 dst=10.141.0.162 sport=40626
   dport=22 src=10.141.0.162 dst=172.16.0.230 sport=22 dport=40626 [ASSURED]
   mark=0 use=1
udp      17 23 src=10.0.2.10 dst=9.9.9.9 sport=53336 dport=53 src=9.9.9.9
   dst=10.141.0.162 sport=53 dport=53336 [ASSURED] mark=0 use=1
tcp      6 113 TIME_WAIT src=10.0.2.10 dst=151.101.12.204 sport=48494
   dport=80 src=151.101.12.204 dst=10.141.0.162 sport=80 dport=48494 [ASSURED]
   mark=0 use=1
udp      17 27 src=10.0.2.11 dst=9.9.9.9 sport=39790 dport=53 src=9.9.9.9
   dst=10.141.0.162 sport=53 dport=39790 mark=0 use=1
udp      17 23 src=10.0.2.10 dst=9.9.9.9 sport=59674 dport=53 src=9.9.9.9
   dst=10.141.0.162 sport=53 dport=59674 mark=0 use=1
tcp      6 117 TIME_WAIT src=10.0.2.11 dst=151.101.12.204 sport=50252
   dport=80 src=151.101.12.204 dst=10.141.0.162 sport=80 dport=50252 [ASSURED]
   mark=0 use=1
udp      17 27 src=10.0.2.11 dst=9.9.9.9 sport=43617 dport=53 src=9.9.9.9
   dst=10.141.0.162 sport=53 dport=43617 [ASSURED] mark=0 use=1
```

4. Protection de base des routeurs ***Hub*** et ***Spoke***

Le but de cette section est de mettre en place le routage avant de passer aux fonctions de filtrage réseau proprement dites. Elle correspond à la vue Topologie PPP et routage.

Voici une liste de fonctions de protection à mettre en œuvre sur tous les types de routeurs.

Protection contre l'usurpation des adresses sourcesrpfilterBCP38

Ces fonctions de protection comprennent une partie noyau ainsi qu'une partie filtrage avec le module `rpfilter` à implanter dans la table `raw` qui assure un filtrage sans état. Voir Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing.

Les tests de validation de ces mécanismes peuvent se faire à l'aide de la commande hping3. Les résultats doivent être visibles aussi bien dans les journaux systèmes que sur les compteurs des règles de la table `raw`. En avant pour la chasse aux martiens !

Protection contre les dénis de services ICMPmodule `netfilter_limit`

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 SSHfail2ban

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

Q7. 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 iptables les options relatives aux listes et aux compteurs.

La visualisation des compteurs de correspondance des règles de filtrage est indispensable pour qualifier le fonctionnement du filtrage

C'est l'option `-L` qui permet l'affichage des listes.

C'est l'option `-v` qui permet d'obtenir les valeurs des compteurs de correspondance avec chaque règle.

Voici un exemple dans le contexte de la maquette sur le routeur Spoke1Vert. Une règle a déjà été insérée dans la table `raw`. Elle permet de visualiser les compteurs de correspondance qui montrent que la règle a bien été utilisée.

```
etu@Spoke1Vert:~$ sudo iptables -vL -t raw
# Warning: iptables-legacy tables present, use iptables-legacy to see them
Chain PREROUTING (policy ACCEPT 112K packets, 113M bytes)
 pkts bytes target     prot opt in     out     source       destination
   60  4996 DROP       all  --  any    any     anywhere     anywhere  rpfilter invert /* BCP38 */

Chain OUTPUT (policy ACCEPT 20480 packets, 1365K bytes)
 pkts bytes target     prot opt in     out     source       destination
```

Q8. Comment activer la protection contre l'usurpation des adresses sources au niveau du noyau ?

Rechercher les informations relatives à la fonction Reverse Path Forwarding du noyau Linux. Identifier les rôles des 3 valeurs possibles de cette fonction.

La documentation est à cette adresse : Kernel IP sysctl.

Le fichier de configuration principal `/etc/sysctl.conf` dispose de plusieurs entrées relatives à cette fonction. Voici un extrait dans le contexte de la maquette.

```
$ grep rp_filter /etc/sysctl.conf
net.ipv4.conf.default.rp_filter=1
net.ipv4.conf.all.rp_filter=1
```

Voici la liste des valeurs actives au moment de l'exécution de la commande.

```
etu@Spoke1Vert:~$ sudo sysctl -ar '\.rp_filter'
net.ipv4.conf.all.rp_filter = 1
net.ipv4.conf.asw-host.rp_filter = 1
net.ipv4.conf.default.rp_filter = 1
net.ipv4.conf.enp0s2.rp_filter = 1
net.ipv4.conf.enp0s2/470.rp_filter = 1
net.ipv4.conf.enp0s2/471.rp_filter = 1
net.ipv4.conf.lo.rp_filter = 0
net.ipv4.conf.ovs-system.rp_filter = 1
net.ipv4.conf.ppp0.rp_filter = 1
net.ipv4.conf.sw-vlan1.rp_filter = 1
net.ipv4.conf.veth52dfe1cc.rp_filter = 1
net.ipv4.conf.veth73d0058f.rp_filter = 1
net.ipv4.conf.vethc394c229.rp_filter = 1
```

Voici l'extrait de la documentation officielle qui donne les explications sur les 3 valeurs possibles du paramètre `rp_filter`.

```
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.
```

Q9. Comment enregistrer les tentatives d'usurpation d'adresses dans les journaux système ?

Rechercher les entrées de l'arborescence `/proc` relatives aux paquets “martiens“.

Rechercher aussi le paramètre relatifs aux “martiens“ dans le fichier `/etc/sysctl.conf`.

On a activé la “journalisation des martiens“ en éditant le fichier `/etc/sysctl.conf`.

```
$ grep martians /etc/sysctl.conf
net.ipv4.conf.all.log_martians = 1
```

On vérifie que le paramètre est bien actif sur le système.

```
$ sudo sysctl -ar 'all.*martians'
net.ipv4.conf.all.log_martians = 1
```

Si ce n'est pas le cas, il ne faut pas oublier de parcourir à nouveau les fichiers de paramètres à l'aide de la commande sysctl.

```
$ sudo sysctl --system
```

Q10. Comment valider la fonction de blocage des tentatives d'usurpation d'adresses entre le routeur Hub et les routeurs Spoke ?

Installer le paquet `hping3` sur le routeur Hub.

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.

Voici un premier exemple de test effectué sur le routeur Hub dans le contexte de la maquette.

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.

```
etu@HubBleu:~$ sudo hping3 -1 -a 10.0.2.12 --fast -c 10 10.0.2.11
HPING 10.0.2.11 (ppp0 10.0.2.11): icmp mode set, 28 headers + 0 data bytes

--- 10.0.2.11 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 Spoke, on peut consulter les traces des tentatives d'usurpation d'adresses à l'aide de la commande suivante.

```
etu@Spoke2Vert:~$ grep martian /var/log/kern.log
```

Il est aussi possible de lancer un test avec une série d'adresses IP source aléatoires. Voici un exemple de commande qui provoquera un nombre de blocages aléatoire en fonction des correspondances.

```
etu@HubBleu:~$ sudo hping3 -1 --rand-source --fast -c 100 10.0.1.11
```

Q11. Comment filtrer les tentatives d'usurpation d'adresses source au plus tôt de façon à limiter le coût de traitement de ces paquets falsifiés sur le système ?

Identifier le nom de la table de filtrage sans état et rechercher la fonction associée au filtrage des adresses sources usurpées. Rechercher dans les pages de manuels `iptables-extensions` les informations relatives au module `rpfilter`.

Ajouter une règle spécifique dans la table de traitement sans état pour les protocoles IPv4 et IPv6.

La table de filtrage sans état est appelée : `raw`. Après consultation des exemples donnés dans les pages de manuels, on aboutit aux deux règles suivantes que l'on applique sur les routeurs Spoke.

```
$ sudo iptables -t raw -A PREROUTING -m rpfilter --invert -m comment --comment "BCP38" -j DROP
$ sudo ip6tables -t raw -A PREROUTING -m rpfilter --invert -m comment --comment "BCP38" -j DROP
```

On peut ensuite sauvegarder ces règles dans les fichiers systèmes utilisés par le service `iptables-persistent`.

```
$ sudo sh -c "iptables-save >/etc/iptables/rules.v4"
$ sudo sh -c "ip6tables-save >/etc/iptables/rules.v6"
```

Q12. Comment caractériser les nouvelles règles de filtrage entre le routeur Hub et les routeurs Spoke ?

Pour les tests IPv4, il suffit de reprendre les mêmes tests que ceux effectués plus haut avec la commande hping3.

Installer le paquet `thc-ipv6` sur le routeur Hub pour disposer des outils de tests spécifiques au protocole IPv6.

Rechercher dans les pages de manuels de la commande atk6-thcping6 les options qui permettent de générer du trafic ICMP avec une adresse source falsifiée à destination d'un conteneur hébergé sur un routeur Spoke.

Dans le contexte de la maquette, les requêtes falsifiées sont émises depuis le routeur HubBleu à destination du routeur Spoke1Vert sur lequel les règles de filtrage IPv4 et IPv6 ont été implantées.

- Pour le protocole IPv4, on reprend la commande hping3 avec les mêmes paramètres de dans la question sur la protection au niveau du noyau Linux et on relève le compteur des paquets “jetés“ sur le routeur Spoke.

```
etu@HubBleu:~$ sudo hping3 -1 -a 10.0.2.12 --fast -c 100 10.0.2.11
HPING 10.0.2.11 (ppp1 10.0.2.11): icmp mode set, 28 headers + 0 data bytes

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

```
etu@Spoke1Vert:~$ sudo iptables -vL -t raw
# Warning: iptables-legacy tables present, use iptables-legacy to see them
Chain PREROUTING (policy ACCEPT 393K packets, 457M bytes)
 pkts bytes target     prot opt in     out     source    destination
   71  5304 DROP       all  --  any    any     anywhere  anywhere   rpfilter invert /* BCP38 */

Chain OUTPUT (policy ACCEPT 94964 packets, 5522K bytes)
 pkts bytes target     prot opt in     out     source    destination
```

- Pour le protocole IPv6, on utilise la commande atk6-thcping6 avec l'adresse d'un conteneur comme source et l'adresse du routeur Spoke dans le VLAN supervision (violet) comme destination.

```
etu@HubBleu:~$ sudo atk6-thcping6 -n 10 enp0s2.470 fda0:7a62:1:0:216:3eff:feda:e1a fe80:1d6::2
0000.000  ping packet sent to fe80:1d6::2
0000.000  ping packet sent to fe80:1d6::2
0000.000  ping packet sent to fe80:1d6::2
0000.000  ping packet sent to fe80:1d6::2
0000.000  ping packet sent to fe80:1d6::2
0000.000  ping packet sent to fe80:1d6::2
0000.000  ping packet sent to fe80:1d6::2
*** buffer overflow detected ***: terminated
Abandon
```

```
etu@Spoke1Vert:~$ sudo ip6tables -vL -t raw
# Warning: ip6tables-legacy tables present, use ip6tables-legacy to see them
Chain PREROUTING (policy ACCEPT 37580 packets, 6219K bytes)
 pkts bytes target     prot opt in     out     source     destination
   14   784 DROP       all      any    any     anywhere   anywhere     rpfilter invert /* BCP38

Chain OUTPUT (policy ACCEPT 7599 packets, 1505K bytes)
 pkts bytes target     prot opt in     out     source     destination
```

4.2. Protection contre les dénis de service ICMP

Q13. Comment peut-on se protéger contre un nombre de sollicitations ICMP trop important ?

Rechercher dans le guide Tutoriel iptables la correspondance Limit qui permet de définir un seuil au delà duquel les nouveaux flux réseau ne sont plus acceptés.

Il faut ajouter une règle spécifique au protocole ICMP après celle qui assure le traitement des flux déjà enregistrés dans les tables de suivi d'état (Stateful).

Dans le contexte de la maquette, les nouvelles règles de filtrage sont appliquées sur le routeur Spoke2Vert et le trafic “malveillant“ est généré sur le routeur HubBleu à destination des conteneurs du réseau local du site distant.

On commence par afficher la liste des règles de la table par défaut appelée netfilter de façon à vérifier si la règle générale de suivi des enregistrements est présente ou non dans les chaînes `INPUT` et `FORWARD`.

Dans la copie d'écran ci-dessous, on constate qu'aucune règle de filtrage n'a été appliquée au moment du test.

```
etu@Spoke2Vert:~$ sudo iptables -vL
# Warning: iptables-legacy tables present, use iptables-legacy to see them
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source     destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source     destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source     destination
```

Le résultat de la commande `sudo ip6tables -vL` doit être identique à la copie d'écran ci-dessus.

On ajoute les deux règles générales de suivi des conversations en premier dans les chaînes `INPUT` et `FORWARD` pour le protocole IPv4.

```
etu@Spoke2Vert:~$ sudo iptables -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
etu@Spoke2Vert:~$ sudo iptables -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
etu@Spoke2Vert:~$ sudo iptables -vL
# Warning: iptables-legacy tables present, use iptables-legacy to see them
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source      destination
    0     0 ACCEPT     all  --  any    any     anywhere    anywhere     ctstate RELATED,ESTABLISH

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source      destination
    0     0 ACCEPT     all  --  any    any     anywhere    anywhere     ctstate RELATED,ESTABLISH

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source      destination
```

On ajoute aussi deux règles identiques pour le protocole IPv6.

```
etu@Spoke2Vert:~$ sudo ip6tables -A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
etu@Spoke2Vert:~$ sudo ip6tables -A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
```

Maintenant que le trafic relatif ou appartenant à un flux enregistré dans la table de suivi d'état est accepté, nous pouvons définir les conditions dans lesquelles un nouveau flux entre de le système de suivi d'état. Ici, on s'intéresse au protocole ICMP et au module Limit. Voici un exemple de règle qui restreint le trafic ICMP à 2 nouvelles entrées par seconde sur les chaînes `INPUT` et `FORWARD`.

- Pour le protocole IPv4.

```
etu@Spoke2Vert:~$ sudo iptables -A INPUT -p icmp -m limit --limit 2/sec -m conntrack --ctstate
etu@Spoke2Vert:~$ sudo iptables -A FORWARD -p icmp -m limit --limit 2/sec -m conntrack --ctstat
```

- Pour le protocole IPv6.

```
etu@Spoke2Vert:~$ sudo ip6tables -A INPUT -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
etu@Spoke2Vert:~$ sudo ip6tables -A FORWARD -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
```

Q14. Comment qualifier le fonctionnement des règles de limitation du nombre de nouvelles requêtes ICMP ?

Rechercher les options de la commande hping3 qui permettent de générer des flux ICMP en utilisant des adresses IPv4 source aléatoires.

Attention ! Il faut positionner la politique par défaut en mode "tout ce qui n'est pas autorisé est interdit“ sur le routeur cible le temps du test de qualification.

On rappelle que dans le contexte de la maquette, les règles de filtrage sont appliquées sur le routeur Spoke2Vert et le trafic “malveillant“ est généré sur le routeur HubBleu à destination des conteneurs du réseau local du site distant.

- On commence par modifier la politique par défaut dans la chaîne `FORWARD` sur le routeur Spoke2Vert.

```
$ sudo iptables -P FORWARD DROP
$ sudo ip6tables -P FORWARD DROP
```

- On lance la génération de trafic ICMP à partir du routeur HubBleu. Dans l'exemple ci-dessous, ce sont 100 paquets ICMP `echo-request` qui sont envoyés avec une adresse IPv4 source aléatoire à destination du conteneur `10.0.2.11`.

```
etu@HubBleu:~$ sudo hping3 -1 --rand-source --fast -c 100 10.0.2.11
```

  À la fin de l'émission, les résultats montrent que 76% des 100 requêtes ont été rejetées.

```
--- 10.0.2.11 hping statistic ---
100 packets transmitted, 24 packets received, 76% packet loss
round-trip min/avg/max = 1.1/5.0/9.3 ms
```

- On se place sur le routeur Spoke2Vert et on affiche la liste des règles de la chaîne `FORWARD` avec les compteurs de paquets.

```
etu@Spoke2Vert:~$ sudo iptables -vL FORWARD
# Warning: iptables-legacy tables present, use iptables-legacy to see them
Chain FORWARD (policy DROP 127 packets, 3556 bytes)
 pkts bytes target     prot opt in     out     source     destination
  143  4004 ACCEPT     all  --  any    any     anywhere   anywhere     ctstate RELATED,ESTABLIS
   72  2016 ACCEPT     icmp --  any    any     anywhere   anywhere     limit: avg 2/sec burst 5
```

  Cet échantillon montre que 127 paquets ont été mis à la poubelle et non routés jusqu'au conteneur.

- Comme le jeu de règles sur le routeur Spoke2Vert est trop restreint pour être acceptable par les conteneurs, on replace la politique par défaut à `ACCEPT` sur la chaîne `FORWARD`.

```
$ sudo iptables -P FORWARD ACCEPT
$ sudo ip6tables -P FORWARD ACCEPT
```

4.3. Protection contre les robots de connexion au service SSH

Q15. Quel est la fonction du paquet fail2ban ?

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

```
apt install fail2ban
```

```
aptitude show fail2ban | grep -A2 Desc | fmt -t -w80
```

```
Description : ban hosts that cause multiple authentication errors Fail2ban
 monitors log files (e.g. /var/log/auth.log, /var/log/apache/access.log)
 and temporarily or persistently bans failure-prone addresses by updating
 existing firewall rules.  Fail2ban allows easy specification of different
 actions to be taken such as to ban an IP using iptables or hostsdeny rules,
 or simply to send a notification email.
```

Le rôle du service fail2ban est de repérer les erreurs d'authentification dans les journaux des différents services actifs et de créer une chaîne `iptables` qui bloque les tentatives de connexion suivantes.

Q16. 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    43975 root    3u  IPv4 7613072      0t0  TCP *:2222 (LISTEN)
sshd    43975 root    4u  IPv6 7613074      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.

Q17. Quel est le fichier de configuration du service SSH qui permet de définir le numéro de port en écoute avec le protocole TCP ?

Repérer le répertoire qui contient les éléments de configuration du service SSH.

C'est le fichier `/etc/ssh/sshd_config` qui contient les paramètres du serveur. Dans le cas de ces manipulations, on a décommenté la ligne avec le mot clé `Port`.

```
$ grep ^Port /etc/ssh/sshd_config
Port 2222
```

Attention ! Si on édite ce fichier de configuration, les modifications ne sont prises en compte qu'au redémarrage du service via la commande `sudo systemctl restart ssh`.

Q18. Quels sont les deux fichiers de configuration principaux fournis à l'installation du paquet `fail2ban` ?

Rechercher dans l'arborescence des fichiers de configuration, les informations relatives aux traitements assurés en cas de détection d'erreurs de connexion à n'importe quel service, puis les informations spécifiques au service SSH.

Dans le répertoire `/etc/fail2ban`, on identifie les deux fichiers demandés.

- Le fichier `/etc/fail2ban/jail.conf` donne la liste des paramètres par défaut en cas de détection de tentatives de connexions en erreur. Voici une extraction des premiers paramètres généraux. Ici, le temps de maintien d'une adresse source en prison (`bantime`) est de 10 minutes mais ce temps est multiplié par deux en cas de récidive.

```
etu@Spoke1Vert:~$ sed -n '/^\[DEF/,/fail2ban_agent/p' /etc/fail2ban/jail.conf | egrep -v '(^#|^
[DEFAULT]
bantime.increment = true
bantime.factor = 2
ignoreself = true
ignorecommand =
bantime  = 10m
findtime  = 10m
maxretry = 5
maxmatches = %(maxretry)s
backend = systemd
usedns = warn
logencoding = auto
enabled = false
mode = normal
filter = %(name)s[mode=%(mode)s]
destemail = root@localhost
sender = root@<fq-hostname>
mta = sendmail
protocol = tcp
chain = <known/chain>
port = 0:65535
fail2ban_agent = Fail2Ban/%(fail2ban_version)s
```

- Le fichier `/etc/fail2ban/jail.d/defaults-debian.conf` contient les paramètres par défaut pour la distribution avec une section spécifique au service SSH. C'est ce fichier que l'on édite pour l'adapter ua contexte des routeurs de la topologie. On spécifie le numéro de port identifié dans les questions précédentes ainsi qu'un traitement particulier. Voici un exemple de configuration appliqué à la maquette.

```
[sshd]
enabled = true
filter  = sshd
action = %(action_)s
maxretry = 3
banaction = iptables-new
```

Là encore, les modifications effectuées sur la configuration ne sont prises en compte qu'au redémarrage du service : `sudo systemctl restart fail2ban`.

Q19. 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 iptables et consulter l'état de la prison fail2ban.

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

```
etu@HubBleu:~$ ssh -p 2222 etu@10.47.1.2
etu@10.47.1.2's password:
Permission denied, please try again.
etu@10.47.1.2's password:
Permission denied, please try again.
etu@10.47.1.2's password:
etu@10.47.1.2: Permission denied (publickey,password).
etu@HubBleu:~$ ssh -p 2222 etu@10.47.1.2
ssh: connect to host 10.47.1.2 port 2222: Connection refused
```

On relève ensuite les résultats côté routeur Spoke.

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.

```
etu@Spoke1Vert:~$ sudo iptables -vL
Chain INPUT (policy ACCEPT 335K packets, 401M bytes)
 pkts bytes target     prot opt in     out     source      destination
    2   120 f2b-sshd   tcp  --  any    any     anywhere    anywhere   state NEW tcp dpt:2222

Chain FORWARD (policy ACCEPT 60353 packets, 58M bytes)
 pkts bytes target     prot opt in     out     source      destination

Chain OUTPUT (policy ACCEPT 95465 packets, 5602K bytes)
 pkts bytes target     prot opt in     out     source      destination

Chain f2b-sshd (1 references)
 pkts bytes target     prot opt in     out     source      destination
    2   120 REJECT     all  --  any    any     10.47.1.1   anywhere   reject-with icmp-port-unrea
    0     0 RETURN     all  --  any    any     anywhere    anywhere
```

Toujours sur le routeur Spoke, on relève l'état du service fail2ban et plus particulièrement celui de la "prison" spécifique au protocole SSH.

```
etu@Spoke1Vert:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 0
|  |- Total failed: 3
|  `- Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
   |- Currently banned: 1
   |- Total banned: 1
   `- Banned IP list: 10.47.1.1
```

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

```
etu@HubBleu:~$ ssh -p 2222 fe80::5c93:b536:53e7:f976%ppp0
etu@fe80::5c93:b536:53e7:f976%ppp0's password:
Permission denied, please try again.
etu@fe80::5c93:b536:53e7:f976%ppp0's password:
Permission denied, please try again.
etu@fe80::5c93:b536:53e7:f976%ppp0's password:
etu@fe80::5c93:b536:53e7:f976%ppp0: Permission denied (publickey,password).
etu@HubBleu:~$ ssh -p 2222 fe80::5c93:b536:53e7:f976%ppp0
ssh: connect to host fe80::5c93:b536:53e7:f976%ppp0 port 2222: Connection refused
```

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

```
etu@Spoke1Vert:~$ sudo fail2ban-client status sshd
Status for the jail: sshd
|- Filter
|  |- Currently failed: 1
|  |- Total failed: 7
|  `- Journal matches: _SYSTEMD_UNIT=sshd.service + _COMM=sshd
`- Actions
   |- Currently banned: 2
   |- Total banned: 2
   `- Banned IP list: 10.47.1.1 fe80::490a:39d4:1a05:f33d
```

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

```
etu@Spoke1Vert:~$ sudo ip6tables -vL
# Warning: ip6tables-legacy tables present, use ip6tables-legacy to see them
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source    destination
    3   240 f2b-sshd   tcp      any    any     anywhere  anywhere     state NEW tcp dpt:2222

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source    destination

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source    destination

Chain f2b-sshd (1 references)
 pkts bytes target     prot opt in     out     source    destination
    2   160 REJECT     all      any    any     fe80::490a:39d4:1a05:f33d  anywhere   reject-with
    1    80 RETURN     all      any    any     anywhere  anywhere
```

5. Règles de filtrage communes à toutes les configurations

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 : ***tout trafic réseau non autorisé est interdit***.

- On fait le choix d'un filtrage 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 celle qui laisse passer le trafic qui correspond ou qui est relatif à une communication déjà enregistrée dans les tables.

- Dans le but de simplifier l'étude du filtrage, on fait le choix d'autoriser tous les flux sortants émis par les routeurs Hub et Spoke. On laisse donc la politique par défaut à `ACCEPT` pour les chaînes `OUTPUT` des routeurs.

On commence par afficher les règles actives sur les différents routeurs 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.

- Régles de filtrage IPv4 côté Hub : fichier `/etc/iptables/rules.v4`.

```
#~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~ N A T
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A POSTROUTING -o enp0s2.300 -j MASQUERADE
COMMIT
#~~~~~~~~~~~~ F I L T E R
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
COMMIT
```

- Régles de filtrage IPv6 côté Hub : fichier `/etc/iptables/rules.v6`.

```
#~~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~~ N A T
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A POSTROUTING -o enp0s2.300 -j MASQUERADE
COMMIT
#~~~~~~~~~~~~~ F I L T E R
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -s fe80::/10 -j ACCEPT
-A INPUT -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
COMMIT
```

- Régles de filtrage IPv4 côté Spoke : fichier `/etc/iptables/rules.v4`.

```
#~~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~~ F I L T E R
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
COMMIT
```

- Régles de filtrage IPv6 côté Spoke : fichier `/etc/iptables/rules.v6`.

```
#~~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~~ F I L T E R
*filter
:INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -s fe80::/10 -j ACCEPT
-A INPUT -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
COMMIT
```

Q20. Dans les jeux de règles déjà en place, comment identifier les règles qui traitent les flux réseau dèjà enregistrés dans le suivi de communication ?

La section «7.3. États de l'espace utilisateur» du Tutoriel iptables décrit les correspondances entre les états et les flux réseau.

Le tableau de la section «7.3. États de l'espace utilisateur» permet de sélectionner les états `ESTABLISHED` et `RELATED` que l'on retrouve en première position dans les chaînes `INPUT` et `FORWARD`.

Voici un exemple qui illustre l'utilisation de ces règles dans le contexte de la maquette. L'évolution des compteurs, montre qu'une règle est effectivement utilisée dans le traitement du trafic réseau.

```
etu@Spoke2Vert:~$ sudo ip6tables -vL
# Warning: ip6tables-legacy tables present, use ip6tables-legacy to see them
Chain INPUT (policy ACCEPT 26 packets, 2192 bytes)
 pkts bytes target     prot opt in     out     source     destination
  591 54216 ACCEPT     all      any    any     anywhere   anywhere     ctstate RELATED,ESTABLISHE
    0     0 ACCEPT     icmp     any    any     anywhere   anywhere     limit: avg 2/sec burst 5

Chain FORWARD (policy ACCEPT 6 packets, 480 bytes)
 pkts bytes target     prot opt in     out     source     destination
    6   360 ACCEPT     all      any    any     anywhere   anywhere     ctstate RELATED,ESTABLISHE
    0     0 ACCEPT     icmp     any    any     anywhere   anywhere     limit: avg 2/sec burst 5

Chain OUTPUT (policy ACCEPT 505 packets, 97048 bytes)
 pkts bytes target     prot opt in     out     source     destination
```

Q21. Quelles règles faut-il ajouter pour autoriser les nouveaux flux réseau depuis et vers l'interface de boucle locale (chaîne `INPUT`) ?

Pour que les processus locaux au système puissent communiquer entre eux, il est ***essentiel*** d'autoriser le trafic sur l'interface de boucle locale `lo`.

On insère une nouvelle règle sur la chaîne `INPUT` qui admet tous les nouveaux paquets entrant sur l'interface de `lo` sans tenir compte de la table de suivi des communications.

La même règle est insérée pour les protocoles IPv4 et IPv6. On utilise les numéros de lignes pour insérer les nouvelles règles en postion 2.

```
etu@Spoke2Vert:~$ sudo iptables -I INPUT 2 -i lo -j ACCEPT
etu@Spoke2Vert:~$ sudo ip6tables -I INPUT 2 -i lo -j ACCEPT
```

On relève un exemple de résultat en affichant la liste des règles actives avec leurs numéros.

```
etu@Spoke2Vert:~$ sudo iptables -vL --line-numbers
# Warning: iptables-legacy tables present, use iptables-legacy to see them
Chain INPUT (policy ACCEPT 939 packets, 364K bytes)
num   pkts bytes target     prot opt in     out     source     destination
1      955 1336K ACCEPT     all  --  any    any     anywhere   anywhere     ctstate RELATED,ESTAB
2        0     0 ACCEPT     all  --  lo     any     anywhere   anywhere
3        0     0 ACCEPT     icmp --  any    any     anywhere   anywhere     limit: avg 2/sec burs

Chain FORWARD (policy ACCEPT 12 packets, 825 bytes)
num   pkts bytes target     prot opt in     out     source     destination
1      864  653K ACCEPT     all  --  any    any     anywhere   anywhere     ctstate RELATED,ESTAB
2        0     0 ACCEPT     icmp --  any    any     anywhere   anywhere     limit: avg 2/sec burs

Chain OUTPUT (policy ACCEPT 573 packets, 34052 bytes)
num   pkts bytes target     prot opt in     out     source     destination
```

À partir de ce jeu de règles, on peut lancer un test ICMP et relever les compteurs d'utilisation de la nouvelle règle.

```
etu@Spoke2Vert:~$ ping -q -c 4 ::1
PING ::1(::1) 56 data bytes

--- ::1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3074ms
rtt min/avg/max/mdev = 0.116/0.139/0.162/0.022 ms
```

```
etu@Spoke2Vert:~$ sudo ip6tables -vL INPUT --line-numbers
# Warning: ip6tables-legacy tables present, use ip6tables-legacy to see them
Chain INPUT (policy ACCEPT 33 packets, 2968 bytes)
num   pkts bytes target     prot opt in     out     source    destination
1     1445  131K ACCEPT     all      any    any     anywhere  anywhere     ctstate RELATED,ESTABL
2        2   208 ACCEPT     all      lo     any     anywhere  anywhere
3        0     0 ACCEPT     ipv6-icmp  any    any     anywhere  anywhere     limit: avg 2/sec bur
```

Q22. Quelles règles faut-il ajouter pour autoriser les nouvelles connexions SSH et les intégrer dans la table de suivi des communications ?

Le protocole de couche transport utilisé est TCP et le numéro de port utilisé par le service SSH est `2222`.

La section «7.3. États de l'espace utilisateur» du Tutoriel iptables décrit les correspondances entre les états et les flux réseau. Rechercher la clé relative aux nouveaux flux entrants.

Le tableau de la section «7.3. États de l'espace utilisateur» permet de sélectionner l'état `NEW`.

Voici un exemple d'ajout de règles dans le contexte de la maquette.

```
etu@Spoke2Vert:~$ sudo iptables -A INPUT -p tcp --syn --dport 2222 \
  -m conntrack --ctstate NEW -m comment --comment SSH -j ACCEPT
etu@Spoke2Vert:~$ sudo ip6tables -A INPUT -p tcp --syn --dport 2222 \
  -m conntrack --ctstate NEW -m comment --comment SSH -j ACCEPT
```

Comme précédemment, on peut relever les compteurs suite à une nouvelle connexion SSH.

```
etu@Spoke2Vert:~$ sudo ip6tables -vL INPUT
# Warning: ip6tables-legacy tables present, use ip6tables-legacy to see them
Chain INPUT (policy ACCEPT 41 packets, 3608 bytes)
 pkts bytes target     prot opt in     out     source    destination
 2227  206K ACCEPT     all      any    any     anywhere  anywhere     ctstate RELATED,ESTABLISHED
    2   208 ACCEPT     all      lo     any     anywhere  anywhere
    0     0 ACCEPT     ipv6-icmp  any    any     anywhere  anywhere     limit: avg 2/sec burst 5
    1    80 ACCEPT     tcp      any    any     anywhere  anywhere     tcp dpt:2222 flags:FIN,SYN,
```

Q23. Quelle est l'instruction qui définit la politique par défaut à appliquer sur les chaînes de la table `netfilter` ?

Il s'agit d'appliquer le principe de filtrage énoncé en début de section qui veut que tout trafic non autorisé soit interdit.

La section «9.3. Commandes» du Tutoriel iptables donne la syntaxe de configuration de ***cible par défaut*** pour les chaînes : `INPUT`, `FORWARD` et `OUTPUT`.

On consulte la documentation et on relève la commande -P. Ensuite, on sélectionne la politique par défaut adaptée au contexte : `DROP`.

Voici un exemple sur un routeur Spoke.

```
etu@Spoke2Vert:~$ sudo iptables -P INPUT DROP
etu@Spoke2Vert:~$ sudo ip6tables -P INPUT DROP
etu@Spoke2Vert:~$ sudo iptables -P FORWARD DROP
etu@Spoke2Vert:~$ sudo ip6tables -P FORWARD DROP
etu@Spoke2Vert:~$ sudo iptables -P OUTPUT ACCEPT
etu@Spoke2Vert:~$ sudo ip6tables -P OUTPUT ACCEPT
```

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

6. Règles de filtrage sur le routeur *Hub*

Dans cette section, on doit compléter les règles de filtrage pour répondre à deux objectifs :

- Le routeur Hub doit autoriser le trafic issu des routeurs Spoke vers l'Internet.
- Les demandes de connexion aux services Web hébergés sur les conteneurs desservis par les routeurs Spoke doivent être redirigées via 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	`10.0.1.10` `fda0:7a62:1:0:216:3eff:feda:e1a`
8011,8454	`10.0.1.11` `fda0:7a62:1:0:216:3eff:fec4:d325`
8012,8455	`10.0.1.12` `fda0:7a62:1:0:216:3eff:fe66:86fb`
8020,8463	`10.0.2.10` `fda0:7a62:2:0:216:3eff:feda:e1a`
8021,8464	`10.0.2.11` `fda0:7a62:2:0:216:3eff:fec4:d325`
8022,8465	`10.0.2.12` `fda0:7a62:2:0:216:3eff:fe66:86fb`

Avant d'aborder les questions, on commence par afficher le contenu des deux fichiers `/etc/iptables/rules.v4` et `/etc/iptables/rules.v6` qui correspondent à la situation initiale avant de répondre aux objectifs de cette section.

- Jeu de règles pour le protocole IPv4.

```
#~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~ N A T
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A POSTROUTING -o enp0s2.300 -j MASQUERADE
COMMIT
#~~~~~~~~~~~~ F I L T E R
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
COMMIT
```

- Jeu de règles pour le protocole IPv6.

```
#~~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~~ N A T
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A POSTROUTING -o enp0s2.300 -j MASQUERADE
COMMIT
#~~~~~~~~~~~~~ F I L T E R
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -s fe80::/10 -j ACCEPT
-A INPUT -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
COMMIT
```

Q24. Comment autoriser et enregistrer dans le mécanisme de suivi des états les flux entrants par les interfaces WAN du routeur Hub ?

Rechercher dans les pages de manuels de la commande iptables le moyen de désigner plusieurs interfaces en une seule règle.

C'est le symbole `+` qui permet de regrouper les interfaces `ppp0` et `ppp1` dans une même règle de filtrage.

On ajoute donc les deux règles suivantes sur le routeur Hub.

```
etu@HubBleu:~$ sudo iptables -A FORWARD -i ppp+ -m conntrack --ctstate NEW -j ACCEPT
etu@HubBleu:~$ sudo ip6tables -A FORWARD -i ppp+ -m conntrack --ctstate NEW -j ACCEPT
```

Q25. Comment valider l'utilisation de ces deux nouvelles règles à partir d'un routeur Spoke ?

Il suffit de lancer un téléchargement depuis un routeur Spoke en utilisant successivement les protocoles IPv4 et IPv6. Ensuite, on relève les enregistrements sur le routeur Hub à l'aide de la commande conntrack.

Voici un exemple de relevé avec un téléchargement suffisamment volumineux pour collecter la liste des entrées de suivi d'état sur le routeur Hub.

```
etu@Spoke2Vert:~$ wget -4 -O /dev/null https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.9.1.t
```

```
etu@HubBleu:~$ sudo conntrack -f ipv4 -L
tcp      6 300 ESTABLISHED src=10.47.3.2 dst=151.101.121.176 sport=60962 dport=443 \
   src=151.101.121.176 dst=10.141.0.162 sport=443 dport=60962 [ASSURED] mark=0 use=2
```

```
etu@Spoke2Vert:~$ wget -6 -O /dev/null https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.9.1.t
```

```
etu@HubBleu:~$ sudo conntrack -f ipv6 -L
tcp      6 300 ESTABLISHED src=fda0:7a62:2::1 dst=2a04:4e42:1d::432 sport=49156 dport=443 \
   src=2a04:4e42:1d::432 dst=2001:678:3fc:12c::2 sport=443 dport=49156 [ASSURED] mark=0 use=2
```

Q26. Comment implanter les règles de traduction d'adresses IPv4 et IPv6 destination de façon à rendre accessibles les services Web configurés dans les conteneurs situés dans les réseaux desservis par les routeurs Spoke ?

Il faut rechercher la syntaxe des règles de la cible DNAT à appliquer dans la table des règles de traduction d'adresses (nat) ainsi que la syntaxe des règles à ajouter dans la chaîne `FORWARD` de la table `netfilter`.

Ces nouvelles règles doivent être conformes au tableau de correspondance donné en début de section. Bien sûr, les adresses doivent être modifiées en fonction du plan d'adressage du document Topologie Hub & Spoke avec le protocole PPPoE.

Comme indiqué dans l'énoncé de la question, l'ajout des règles comprend deux parties : les règles de la table `nat` et les règles de la table `netfilter`.

Dans le contexte de la maquette, on a édité les fichiers `/etc/iptables/rules.v4` et `/etc/iptables/rules.v6`.

- Pour le protocole IPv4.

```
#~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~ N A T
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8010 -m conntrack --ctstate NEW \
  -m comment --comment Spoke1C0 -j DNAT --to 10.0.1.10:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8453 -m conntrack --ctstate NEW \
  -m comment --comment Spoke1C0 -j DNAT --to 10.0.1.10:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8011 -m conntrack --ctstate NEW \
  -m comment --comment Spoke1C1 -j DNAT --to 10.0.1.11:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8454 -m conntrack --ctstate NEW \
  -m comment --comment Spoke1C1 -j DNAT --to 10.0.1.11:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8012 -m conntrack --ctstate NEW \
  -m comment --comment Spoke1C2 -j DNAT --to 10.0.1.12:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8455 -m conntrack --ctstate NEW \
  -m comment --comment Spoke1C2 -j DNAT --to 10.0.1.12:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8020 -m conntrack --ctstate NEW \
  -m comment --comment Spoke2C0 -j DNAT --to 10.0.2.10:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8463 -m conntrack --ctstate NEW \
  -m comment --comment Spoke2C0 -j DNAT --to 10.0.2.10:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8021 -m conntrack --ctstate NEW \
  -m comment --comment Spoke2C1 -j DNAT --to 10.0.2.11:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8464 -m conntrack --ctstate NEW \
  -m comment --comment Spoke2C1 -j DNAT --to 10.0.2.11:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8022 -m conntrack --ctstate NEW \
  -m comment --comment Spoke2C2 -j DNAT --to 10.0.2.12:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8465 -m conntrack --ctstate NEW \
  -m comment --comment Spoke2C2 -j DNAT --to 10.0.2.12:443
-A POSTROUTING -o enp0s2.300 -j MASQUERADE
COMMIT
#~~~~~~~~~~~~ F I L T E R
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
-A INPUT -p tcp --syn --dport 2222 -m conntrack --ctstate NEW \
  -m comment --comment SSH -j ACCEPT
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
-A FORWARD -p tcp --syn --dport 2222 -m conntrack --ctstate NEW \
  -m comment --comment SSH -j ACCEPT
-A FORWARD -i ppp+ -m conntrack --ctstate NEW -j ACCEPT
-A FORWARD -d 10.0.1.10/32 -p tcp --syn -m multiport --dports 80,443 \
  -m comment --comment Spoke1C0 -j ACCEPT
-A FORWARD -d 10.0.1.11/32 -p tcp --syn -m multiport --dports 80,443 \
  -m comment --comment Spoke1C1 -j ACCEPT
-A FORWARD -d 10.0.1.12/32 -p tcp --syn -m multiport --dports 80,443 \
  -m comment --comment Spoke1C2 -j ACCEPT
-A FORWARD -d 10.0.2.10/32 -p tcp --syn -m multiport --dports 80,443 \
  -m comment --comment Spoke2C0 -j ACCEPT
-A FORWARD -d 10.0.2.11/32 -p tcp --syn -m multiport --dports 80,443 \
  -m comment --comment Spoke2C1 -j ACCEPT
-A FORWARD -d 10.0.2.12/32 -p tcp --syn -m multiport --dports 80,443 \
  -m comment --comment Spoke2C2 -j ACCEPT
COMMIT
```

- Pour le protocole IPv6.

```
#~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~ N A T
*nat
:PREROUTING ACCEPT [0:0]
:INPUT ACCEPT [0:0]
:POSTROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8010 \
  -m comment --comment Spoke1C0 -j DNAT --to [fda0:7a62:1:0:216:3eff:feda:e1a]:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8453 \
  -m comment --comment Spoke1C0 -j DNAT --to [fda0:7a62:1:0:216:3eff:feda:e1a]:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8011 \
  -m comment --comment Spoke1C1 -j DNAT --to [fda0:7a62:1:0:216:3eff:fec4:d325]:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8454 \
  -m comment --comment Spoke1C1 -j DNAT --to [fda0:7a62:1:0:216:3eff:fec4:d325]:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8012 \
  -m comment --comment Spoke1C2 -j DNAT --to [fda0:7a62:1:0:216:3eff:fe66:86fb]:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8455 \
  -m comment --comment Spoke1C2 -j DNAT --to [fda0:7a62:1:0:216:3eff:fe66:86fb]:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8020 \
  -m comment --comment Spoke2C0 -j DNAT --to [fda0:7a62:2:0:216:3eff:feda:e1a]:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8463 \
  -m comment --comment Spoke2C0 -j DNAT --to [fda0:7a62:2:0:216:3eff:feda:e1a]:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8021 \
  -m comment --comment Spoke2C1 -j DNAT --to [fda0:7a62:2:0:216:3eff:fec4:d325]:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8464 \
  -m comment --comment Spoke2C1 -j DNAT --to [fda0:7a62:2:0:216:3eff:fec4:d325]:443
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8022 \
  -m comment --comment Spoke2C2 -j DNAT --to [fda0:7a62:2:0:216:3eff:fe66:86fb]:80
-A PREROUTING -i enp0s2.300 -p tcp --syn --dport 8465 \
  -m comment --comment Spoke2C2 -j DNAT --to [fda0:7a62:2:0:216:3eff:fe66:86fb]:443
-A POSTROUTING -o enp0s2.300 -j MASQUERADE
COMMIT
#~~~~~~~~~~~~ F I L T E R
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -s fe80::/10 -j ACCEPT
-A INPUT -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
-A INPUT -p tcp --syn --dport 2222 -m conntrack --ctstate NEW -m comment --comment SSH -j ACCEP
-A INPUT -m limit --limit 1/sec -m conntrack --ctstate INVALID -j DROP
-A INPUT -m limit --limit 1/sec -j NFLOG
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
-A FORWARD -p tcp --syn --dport 2222 -m conntrack --ctstate NEW -m comment --comment SSH -j ACC
-A FORWARD -i ppp+ -m conntrack --ctstate NEW -j ACCEPT
-A FORWARD -d fda0:7a62:1:0:216:3eff:feda:e1a/128 -p tcp --syn -m multiport --dports 80,443 \
  -m conntrack --ctstate NEW -m comment --comment Spoke1C0 -j ACCEPT
-A FORWARD -d fda0:7a62:1:0:216:3eff:fec4:d325/128 -p tcp --syn -m multiport --dports 80,443 \
  -m conntrack --ctstate NEW -m comment --comment Spoke1C1 -j ACCEPT
-A FORWARD -d fda0:7a62:1:0:216:3eff:fe66:86fb/128 -p tcp --syn -m multiport --dports 80,443 \
  -m conntrack --ctstate NEW -m comment --comment Spoke1C2 -j ACCEPT
-A FORWARD -d fda0:7a62:2:0:216:3eff:feda:e1a/128 -p tcp --syn -m multiport --dports 80,443 \
  -m conntrack --ctstate NEW -m comment --comment Spoke2C0 -j ACCEPT
-A FORWARD -d fda0:7a62:2:0:216:3eff:fec4:d325/128 -p tcp --syn -m multiport --dports 80,443 \
  -m conntrack --ctstate NEW -m comment --comment Spoke2C1 -j ACCEPT
-A FORWARD -d fda0:7a62:2:0:216:3eff:fe66:86fb/128 -p tcp --syn -m multiport --dports 80,443 \
  -m conntrack --ctstate NEW -m comment --comment Spoke2C2 -j ACCEPT
-A FORWARD -m limit --limit 1/sec -m conntrack --ctstate INVALID -j DROP
-A FORWARD -m limit --limit 1/sec -j NFLOG
COMMIT
```

Pour rétablir les lignes des copies d'écran ci-dessus, il est possible d'utiliser la commande ci-dessous avec laquelle le fichier `rules.txt` contient les lignes coupées avec le caractère `\`.

```
$ sudo sed '/^[ \-].*\\$/N;s/\\\n *//' rules.txt
```

Comme pour toutes les autres sections, on n'oublie pas de sauvegarder le jeu des règles qui ont été validées.

```
$ sudo sh -c "iptables-save >/etc/iptables/rules.v4"
$ sudo sh -c "ip6tables-save >/etc/iptables/rules.v6"
```

7. Règles de filtrage sur le routeur *Spoke*

Comme pour la section précédente sur le routeur Hub, on doit compléter le jeu de règles de filtrage pour répondre à deux objectifs :

- Le routeur Spoke doit autoriser et enregistrer dans la table de suivi d'état les flux réseaux sortants issus du réseau des conteneurs.
- Ce même routeur Spoke doit autoriser et enregistrer dans la table de suivi d'état les flux réseaux entrants à destination des services Web hébergés par les conteneurs.

On commence par afficher le contenu des deux fichiers `/etc/iptables/rules.v4` et `/etc/iptables/rules.v6` d'un routeur Spoke qui correspondent à la situation initiale avant de traiter les questions de cette section.

- Jeu de règles pour le protocole IPv4.

```
#~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~ F I L T E R
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
-A INPUT -p tcp --syn --dport 2222 -m conntrack --ctstate NEW -m comment --comment SSH -j ACCEPT
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p icmp -m limit --limit 2/sec -m conntrack --ctstate NEW -j ACCEPT
-A FORWARD -p tcp --syn --dport 2222 -m conntrack --ctstate NEW -m comment --comment SSH -j ACCEPT
COMMIT
```

- Jeu de règles pour le protocole IPv6.

```
#~~~~~~~~~~~~ R A W
*raw
:PREROUTING ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A PREROUTING -m rpfilter --invert -m comment --comment BCP38 -j DROP
COMMIT
#~~~~~~~~~~~~ F I L T E R
*filter
:INPUT DROP [0:0]
:FORWARD DROP [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A INPUT -i lo -j ACCEPT
-A INPUT -s fe80::/10 -j ACCEPT
-A INPUT -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
-A INPUT -p tcp --syn --dport 2222 -m conntrack --ctstate NEW -m comment --comment SSH -j ACCEPT
-A FORWARD -m conntrack --ctstate RELATED,ESTABLISHED -j ACCEPT
-A FORWARD -p ipv6-icmp -m limit --limit 2/sec -j ACCEPT
-A FORWARD -p tcp --syn --dport 2222 -m conntrack --ctstate NEW -m comment --comment SSH -j ACCEPT
COMMIT
```

Q27. Comment autoriser et enregistrer dans le mécanisme de suivi des états les flux sortants par l'interface WAN du routeur Spoke ?

Rechercher dans les pages de manuels de la commande iptables le moyen de désigner une interface ainsi que le sens des flux qui transitent par cette interface.

C'est la directive `-o` qui permet de désigner les flux sortants par l'intreface `ppp0`.

On ajoute donc les deux règles suivantes sur les routeurs Spoke.

```
etu@Spoke2Vert:~$ sudo iptables -A FORWARD -o ppp0 -m conntrack --ctstate NEW -j ACCEPT
etu@Spoke2Vert:~$ sudo ip6tables -A FORWARD -o ppp0 -m conntrack --ctstate NEW -j ACCEPT
```

Q28. Comment valider l'utilisation de ces deux nouvelles règles à partir d'un routeur Spoke ?

Il suffit de lancer un téléchargement depuis un conteneur desservi par le routeur Spoke en utilisant successivement les protocoles IPv4 et IPv6. Ensuite, on relève les enregistrements sur le même routeur Spoke à l'aide de la commande conntrack.

Voici un exemple de relevé avec un téléchargement suffisamment volumineux pour collecter la liste des entrées de suivi d'état sur le routeur Spoke.

On commence par s'assurer que le paquet wget est bien installé sur le conteneur depuis lequel on effectue le test.

```
etu@Spoke2Vert:~$ lxc exec container0 -- apt install wget
```

On passe ensuite au téléchargement et au relevé de la table de suivi d'état.

```
etu@Spoke2Vert:~$ lxc exec container0 -- \
  wget -4 -O /dev/null https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.9.1.tar.xz
```

```
etu@Spoke2Vert:~$ sudo conntrack -f ipv4 -L
tcp      6 2 CLOSE src=10.0.2.10 dst=151.101.113.176 sport=59888 dport=443 \
    src=151.101.113.176 dst=10.0.2.10 sport=443 dport=59888 [ASSURED] mark=0 use=1
```

```
etu@Spoke2Vert:~$ lxc exec container0 -- \
  wget -6 -O /dev/null https://cdn.kernel.org/pub/linux/kernel/v5.x/linux-5.9.1.tar.xz
```

```
etu@Spoke2Vert:~$ sudo conntrack -f ipv6 -L
tcp      6 300 ESTABLISHED src=fda0:7a62:2:0:216:3eff:feda:e1a dst=2a04:4e42:3::432 sport=38384 d
    src=2a04:4e42:3::432 dst=fda0:7a62:2:0:216:3eff:feda:e1a sport=443 dport=38384 [ASSURED] mark
```

Q29. Comment autoriser les flux Web entrants par l'interface WAN vers les conteneurs ?

Rechercher dans les options de la commande iptables celles qui permettent de désigner les interfaces d'entrée et de sortie ainsi que les numéros de ports associés au service Web.

Les options utiles pour les interfaces sont `-i` pour l'entrée et `-o` pour la sortie. Les numéros de ports `80` et `443` sont regroupés avec le module `multiport`.

Voici un exemple des deux règles à ajouter.

```
etu@Spoke2Vert:~$ sudo iptables -A FORWARD -i ppp0 -o sw-vlan2 \
  -p tcp --syn -m multiport --dports 80,443 -m conntrack --ctstate NEW -j ACCEPT
etu@Spoke2Vert:~$ sudo ip6tables -A FORWARD -i ppp0 -o sw-vlan2 \
  -p tcp --syn -m multiport --dports 80,443 -m conntrack --ctstate NEW -j ACCEPT
```

Q30. Comment valider l'utilisation des deux règles ajoutées dans la question précédente ?

Reprendre, depuis le routeur Hub, l'utilsation de la commande wget telle qu'elle a été présentée dans la section Routeurs Spoke du support Topologie Hub & Spoke avec le protocole PPPoE.

Voici un exemple des résultats obtenus sur le routeur Hub de la maquette. Le code HTTP `200` montre que la requête a bien été traitée par le serveur Web de chaque conteneur.

```
etu@HubBleu:~$ for addr in 10.0.2.10 10.0.2.11 10.0.2.12;\
  do sh -c "wget -O /dev/null http://$addr 2>&1 | grep \"HTTP\" "; done
requête HTTP transmise, en attente de la réponse… 200 OK
requête HTTP transmise, en attente de la réponse… 200 OK
requête HTTP transmise, en attente de la réponse… 200 OK
```

```
etu@HubBleu:~$ for addr in fda0:7a62:2:0:216:3eff:feda:e1a \
  fda0:7a62:2:0:216:3eff:fec4:d325 \
  fda0:7a62:2:0:216:3eff:fe66:86fb; \
  do sh -c "wget -O /dev/null http://[$addr] 2>&1 | grep \"HTTP\" "; done
requête HTTP transmise, en attente de la réponse… 200 OK
requête HTTP transmise, en attente de la réponse… 200 OK
requête HTTP transmise, en attente de la réponse… 200 OK
```

On se place ensuite sur le routeur Spoke pour relever les compteurs des règles de filtrage.

```
etu@Spoke2Vert:~$ sudo iptables -vL FORWARD | grep http
    6   360 ACCEPT     tcp  --  ppp0   sw-vlan2  anywhere  anywhere \
    tcp flags:FIN,SYN,RST,ACK/SYN multiport dports http,https ctstate NEW
```

```
etu@Spoke2Vert:~$ sudo ip6tables -vL FORWARD | grep http
    3   240 ACCEPT     tcp      ppp0   sw-vlan2  anywhere  anywhere \
    tcp flags:FIN,SYN,RST,ACK/SYN multiport dports http,https ctstate NEW
```

8. Documents de référence

IETF & IANA

Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address SpoofingBCP 38`rp_filter`

Le document standard RFC2827 Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing est un guide de bonne pratiques pour se protéger contre l'usurpation des adresses sources. Dans le monde GNU/Linux, la fonction clé est appélée `rp_filter` pour Reverse Path Filtering.

Distribution Debian GNU/Linux

Manuel de référence Debian

Manuel de référence Debian : configuration du réseau : chapitre du manuel de référence Debian consacré à la configuration réseau.

Site inetdoc.net

Configuration d'une interface de réseau local

Configuration d'une interface de réseau local : identification du type d'interface, de ses caractéristiques et manipulations des paramètres. Ce support fournit une méthodologie de dépannage simple d'une connexion réseau.

Fonctions réseau du noyau Linux

Configuration des fonctions réseau & compilation du noyau Linux : présentation et configuration des fonctions réseau du noyau LINUX

Didacticiel sur Iptables

Tutoriel iptables : guide très complet sur le fonctionnement du filtrage réseau avec les noyaux Linux.

Guide Pratique du NAT

Guide Pratique du NAT : Ce document décrit comment réaliser du camouflage d'adresse IP, un serveur mandataire transparent, de la redirection de ports ou d'autres formes de Traduction d'adresse réseau (Network Address Translation ou NAT) avec le noyau Linux 2.4.