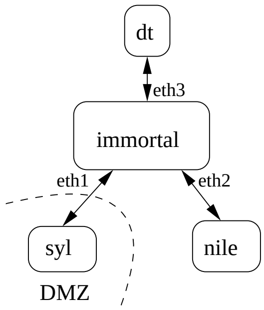
Gestion réseau

Le but de ce TP est de manipuler un peu la gestion de la sécurité réseau : mettre en place un firewall filtrant en utilisant iptables, activer le support SSL d'un site web et l'authentification d'utilisateurs. Bien sûr, ce n'est qu'un minuscule aperçu de tout ce que l'on doit mettre en place pour un réseau, mais cela devrait être représentatif des principes généraux.

Note : désolé, vous ne pouvez pas utiliser votre portable, les chemins sont hardcodés dans les images UML (qui sont d'ailleurs très grosses).

Pour démarrer le TP, lancez `~sthibault/uml/net`, cela met plusieurs minutes à se lancer, le temps de copier la base de travail, et que les machines virtuelles démarrent (au moins 3 minutes, que vous pouvez voir sous la forme de processus `linux-2.6.32` dans `top`). En attendant, commencez à lire ce sujet.

Finalement 4 terminaux s'ouvrent, qui correspondent à ces 4 machines :



1 Firewall

L'objectif de la première partie est de mettre en place un firewall filtrant au niveau de la passerelle `immortal` : `dt` sera considérée comme étant une machine quelconque sur Internet (l'« extérieur »), la machine `syl` sera en zone dite Démilitarisée (DMZ), c'est-à-dire qu'elle est visible depuis l'extérieur, tandis que `nile` ne fait partie que du réseau interne : elle n'est pas visible directement depuis l'extérieur.

La configuration des interfaces réseau et des tables de routages étant déjà faite, il ne vous est demandé que de tester le bon fonctionnement de la configuration fournie : logguez-vous en tant que root sur les machines, et regardez la configuration dans `ip addr ls` et `ip -0 route ls` pour repérer les adresses IPv4 et IPv6 et routes associées.

Les détails de `iptables` et `ip6tables` sont disponibles dans `man iptables` et `man ip6tables`.

Notez que l'on peut regarder le contenu d'une table avec `iptables -L -v`, vider le contenu d'une chaîne avec `iptables -F FORWARD`, supprimer une règle déjà ajoutée avec
`iptables -D FORWARD numéro_de_ligne`, ajouter une règle à la fin d'une chaîne avec `iptables -A FORWARD ...`, ajouter une règle au début d'une chaîne avec
`iptables -I FORWARD ...`,

1.1 Par défaut, tout est interdit !

A priori, on veut s'interdire toute retransmission par défaut, et n'autoriser que ce que l'on veut, on définit donc sur `immortal` la politique par défaut DROP :

```
iptables -P FORWARD DROP
ip6tables -P FORWARD DROP
```

Rien ne peut donc passer la machine firewall : tout est supprimé sans préavis. Essayez par exemple depuis `nile` de lancer `wget 82.225.10.2` pour accéder au serveur web `dt`, on n'obtient plus de réponse.

On veut tout de même permettre aux machines internes (donc ici `nile`) de faire ce qu'elles veulent, ainsi :

```
iptables -A FORWARD -i eth2 -j ACCEPT
ip6tables -A FORWARD -i eth2 -j ACCEPT
```

On accepte ainsi ce qui vient (*input*) d'`eth2`. Ce n'est cependant pas suffisant. Regardez dans `tcpdump -i eth2` et dans `tcpdump -i eth3`, on voit bien passer le début de la connexion (`[S]`), mais pas la réponse (`[S]` + `ack`).

On veut donc également laisser passer les réponses à des requêtes déjà acceptées :

```
iptables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
ip6tables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
```

Relancez le `wget`, cette fois-ci la réponse passe.

1.2 Un deuxième exemple

On veut permettre à n'importe qui d'accéder au serveur web qui tourne sur `syl` :

```
iptables -A FORWARD -p tcp -d 147.210.10.2 --dport 80 -j ACCEPT
ip6tables -A FORWARD -p tcp -d 2001:660:6101:10::2 --dport 80 -j ACCEPT
```

Ici le protocole est donc `tcp`, et l'on précise en plus l'IP de destination (`syl`) et le port de destination, 80. Utilisez `wget` sur `dt` pour vérifier que cela fonctionne. Note : en IPv6, il faut utiliser `wget 'http://[2001:660:6101:10::2]'`

De plus, on peut utiliser l'option `-s` pour filtrer aussi sur l'IP de source. Ajoutez une règle pour permettre seulement à la machine `dt` d'accéder au port `ssh` (22) de `syl`. Testez (avec le login `toto`). Utilisez `ifconfig eth0 82.225.10.3` pour changer l'adresse ip de `dt` pour vérifier qu'une autre adresse IP ne peut pas se connecter.

Il faut cependant noter que c'est loin d'être une protection absolue : l'`IP spoofing` est une attaque classique qui consiste à se faire passer pour une autre machine, en apparaissant sous son IP (on le voit bien ici, il a suffit d'utiliser `ifconfig`). Ce n'est donc au mieux qu'une gêne de plus pour un attaquant éventuel.

1.3 Protection zone privée en IPv4

On s'intéresse maintenant à la machine `nile` : son adresse, IPv4, `172.16.0.2`, fait parties des adresses privées (de même que `192.168.x.y` et `10.x.y.z`). C'est la situation typique d'un réseau domestique : le Fournisseur d'Accès Internet ne fournit pas d'adresse IPv4 publique pour chacun des équipements. Actuellement tout fonctionne car `dt` passe par `immortal` pour atteindre `172.16.0.2`, mais sur un vrai réseau, le Fournisseur d'Accès Internet filtrerait tout paquet à destination d'un réseau privé, c'est-à-dire que l'on a en fait :

```
iptables -A INPUT   -i eth3 -d 172.16.0.0/12 -j DROP
iptables -A FORWARD -i eth3 -d 172.16.0.0/12 -j DROP
```

Il faut donc masquer ces adresse privées :

```
iptables -t nat -A POSTROUTING ! -o eth2 -s 172.16.0.0/12 -j MASQUERADE
```

Lancez `tcpdump -i eth2` sur `immortal` et `tcpdump -i eth0` sur `dt`, et lancez `wget 82.225.10.2` depuis `nile`, voyez que `immortal` remplace l'adresse de `nile` par la sienne, à la volée, avec un nouveau numéro de port. Il s'agit du NAT (Network address Translation).

1.4 Protection zone privée en IPv6

Notez que nous n'avons effectué de masquerading qu'en IPv4. En effet, l'adresse IPv6 que `nile` possède est publique, c'est la situation courante : comme il n'y a pas de pénure d'IPv6, les Fournisseurs d'Accès Internet fournissent tout un sous-réseaux d'adresses publiques.

Il n'y a donc plus besoin d'effectuer de NAT, ce qui coûte bien moins cher. Il faut par contre tout de même protéger notre réseau privé, nous l'avons en fait déjà fait en utilisant la politique `INPUT` par défaut `DROP`.

1.5 Pour aller plus loin...

`iptables` est capable de bien d'autres choses bien sûr, son `man` regorge d'options intéressantes. Par exemple,

```
iptables -A FORWARD -d 147.210.10.2 -p tcp --dport 22 -m state --state NEW -m limit --limit 5/m -j ACCEPT
```

limite le nombre de connexions ssh vers `syl` à 5 par minute. Testez. Pourquoi la machine `dt` n'est pas affectée par la limitation ?

Pour vérifier l'ensemble des ports visibles d'une machine donnée depuis une autre machine, vous pouvez utiliser la commande `nmap`, vérifiez ainsi votre configuration de firewall.

2 Protection d'un serveur web

2.1 Des mots de passe

On se propose de protéger l'accès au serveur web de `syl` par un mot de passe, pour que tout en étant visible depuis n'importe où à l'extérieur, seuls des utilisateurs autorisés aient le droit d'accéder aux pages. Il suffit pour cela sur `syl` de mettre dans `/var/www` un fichier `.htaccess` contenant (pour ceux qui ne connaissent pas `vi`, `nano` est disponible)

```
AuthName "mon site"
AuthType Basic
Require valid-user
AuthUserFile /home/toto/htpasswd
```

On indique ici qu'on va utiliser un fournisseur d'authentification basique (*i.e.* par le fichier `htpasswd`), et que l'accès est restreint aux utilisateurs valides. `AuthName` est utilisé pour distinguer éventuellement différentes parties du site. Il faut par ailleurs au niveau du serveur apache la permission de changer ce genre de paramètres : dans `/etc/apache2/sites-available/default`, ajoutez à la fin de la partie `Directory /var/www/` :

```
AllowOverride AuthConfig
```

et rechargez la configuration à l'aide de `/etc/init.d/apache2 reload`

Il ne nous reste plus qu'à créer le fichier `/home/toto/htpasswd` contenant les logins et mot de passe, sous la forme :

```
login:mot_de_passe_crypté
```

Le mot de passe crypté peut être fabriqué à l'aide de la commande `mkpasswd` Testez depuis `dt` à l'aide de `wget --user=login --password=mot_de_passe`

Problème : le mot de passe est transmis en clair sur le réseau, on peut le voir sur immortal avec `tcpdump -A -i eth1`, en passant le champ `Authorization: Basic` de la requête http à un coup de `base64 -d`

Ce n'est donc pas vraiment satisfaisant ! On va donc ajouter une couche de chiffrement.

2.2 HTTPS

Au lieu de `http://147.210.10.2/`, on va donc faire ouvrir par wget `https://147.210.10.2/`

Commencez déjà par ouvrir le port `443` (https) dans le firewall.

Il faut aussi activer le port https dans la configuration apache ainsi que charger le module ssl, à l'aide des commandes suivantes :

```
a2ensite default-ssl
a2enmod ssl
/etc/init.d/apache2 restart
```

Réessayez, aucun mot de passe n'est demandé ! Il faut effectivement dans `/etc/apache2/sites-available/default-ssl` faire la même modification que dans `default`

Vous noterez que wget n'est pas très content des certificats fournis par le serveur web. Il faudrait effectivement obtenir un certificat fourni par une authorité standard et le fournir aux options `SSLCertificateFile` et `SSLCertificateKeyFile` On peut utiliser l'option `--no-check-certificate` pour s'en dispenser.

Par ailleurs, on préfèrera rediriger automatiquement les client de la version http vers la version https, en ajoutant simplement

`Redirect / https://147.210.10.2/`

au début de la section `VirtualHost` de `/etc/apache2/sites-available/default`

On peut mettre `Redirect permanent` pour signifier au logiciel que c'est une redirection définitive.

Testez en ouvrant la version `http` dans wget, il est redirigé immédiatement sur la version `https`.