Projet No 1 :
Client-serveur Python pour des tests de connectivité

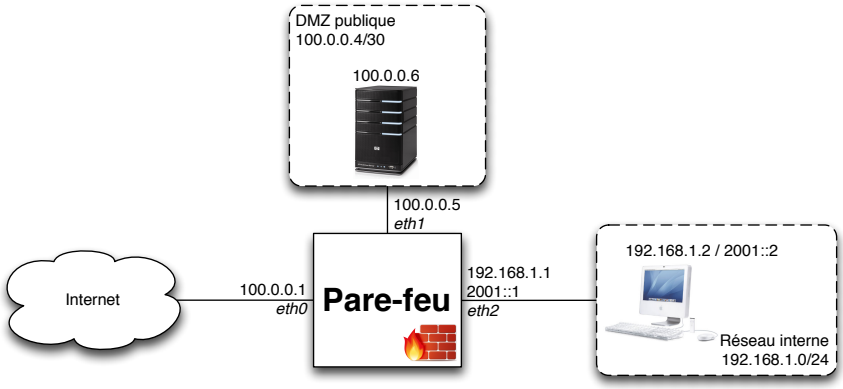
En bref

En vue du projet pare-feu du module ASSR-SECU1, l'objectif du projet est de programmer une application client-serveur en Python pour effectuer des tests de connectivité :

- A différentes adresses (IPv4 et IPv6)
- Pour différents ports
- Pour les protocoles TCP et UDP ainsi que ICMP (limité au ping)

L'application cliente effectuera les tests dont les résultats seront présentés sous la forme d'un tableau récapitulatif. Ces tests doivent permettre de vérifier que les règles de filtrage sont correctes.

Contexte : projet pare-feu



Le schéma ci-dessus illustre un exemple d'architecture réseau ayant pour élément central un pare-feu filtrant des flux entre différents réseaux (Internet, DMZ publique, réseau interne). Ce projet vise à fournir un outil permettant de tester les règles de filtrage à l'aide de simple test de connectivité.

Pour cela :

- La partie serveur doit être active sur chaque réseau.
- La partie cliente doit être également exécuté sur chacun des réseaux vers les autres afin de vérifier le comportement du pare-feu

En pratique, ce réseau sera simulé à l'aide de quatre machines virtuelles (VM) :

- Une VM simulant une machine sur Internet, d'adresse IPv4 100.0.0.2
- Une VM simulant un serveur en DMZ, d'adresse IPv4 100.0.0.6
- Une VM simulant le réseau interne, d'adresse IPv4 192.168.1.2 et IPv6 2001::2
- Une VM simulant le pare-feu et qui possède trois interfaces réseau :
    - eth0, reliée à Internet, d'adresse IPv4 100.0.0.2
    - eth1, reliée à la zone DMZ, d'adresse IPv4 100.0.0.5
    - eth2, reliée au réseau interne, d'adresse IPv4 192.168.1.1 et IPv6 2001::1

Programmation de la partie serveur

La partie serveur doit être programmée de telle sorte que :

- L'application puisse écouter en parallèle (usage de Thread) sur plusieurs ports TCP et UDP ;
- La liste des ports TCP et UDP à écouter soit paramétrable (par exemple, à l'aide de listes et dictionnaires du langage Python) ;
- A chaque réception de données, un serveur retourne la chaine 'ok', valeur qui sera attendue par le client pour valider le test de connectivité ;
- L'application affiche dans la console l'adresse IP et le port destination à chaque paquet reçu, quelque soit le protocole et le port. Cet affichage permettra de détecter si une règle de filtrage ne fonctionne que dans un seul sens (par exemple, un paquet est autorisé dans un sens mais la réponse associée est bloquée dans le sens retour).

Programmation de la partie client

La partie client doit être programmée de telle sorte que :

- L'application contacte successivement tous les serveurs :
  - En fonction d'une liste d'adresse IP : le client doit détecter son IP pour éliminer son IP de la liste des serveurs à contacter. Cette liste doit inclure toutes les adresses du pare-feu.
  - Pour des tests selon les protocoles UDP/TCP et une série de ports. Pour cela, envoyer des données et vérifier que le serveur retourne bien la chaine 'ok'
  - Pour tester le ping (utiliser la commande ping via le module subprocess).
- Le résultat des tests est présenté sous la forme d'un tableau synthétique, affiché dans la console (voir exemple ci-dessous).

```
+-----------+------+--------+--------+--------+
| @IP cible | ping | tcp/22 | tcp/25 | udp/53 |
+-----------+------+--------+--------+--------+
+ DMZ       | ok   | X      | ok     | ok     |
+-----------+------+--------+--------+--------+
+ Intranet  | X    | X      | X      | X      |
+-----------+------+--------+--------+--------+
+ FW eth0   | ok   | X      | X      | ok     |
+-----------+------+--------+--------+--------+
+ FW eth1   | X    | X      | X      | X      |
+-----------+------+--------+--------+--------+
+ FW eth1   | X    | X      | X      | X      |
+-----------+------+--------+--------+--------+
```