université de BORDEAUX

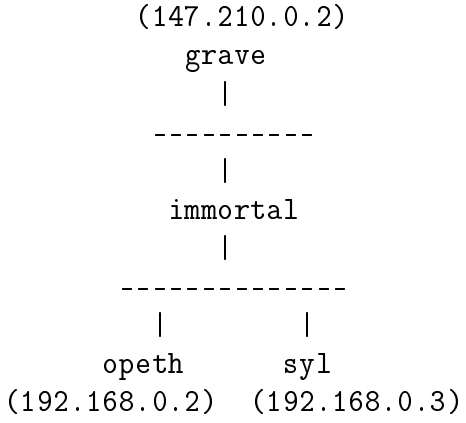
# TP4 - Scapy

## 1 Manipulation de paquets avec Scapy

Scapy est un logiciel libre de manipulation de paquets réseau écrit en Python. C'est un outil (interactif) très puissant qui permet en quelques lignes de Python (contre des centaines en langage C) d'implémenter des fonctions réseau de base : *ping*, *nmap* (syn scan), *traceroute*, ... En pratique, Scapy fait deux choses simples : envoyer des paquets et recevoir les réponses.

### 1.1 Démarrage d'un réseau virtuel

Lancez le réseau virtuel *QemuNet* avec la topologie suivante :
`/net/ens/qemunet/qemunet.sh -x -s /net/ens/qemunet/demo/gw.tgz`




Le réseau est déjà configuré correctement (adresses IP et routage). Lancez `tcpdump -i eth0` sur la passerelle `immortal` afin d'espionner le trafic échangé entre les autres machines. Lancez `netstat -tupl` pour voir quels services (et donc quels ports) sont ouverts sur `opeth` (ou `syl`).

### 1.2 Prise en main de Scapy

Lancez `scapy3` sur la machine `grave` en tant que *root* (ou *sudoer*). En effet, Scapy court-circuite l'interface de programmation traditionnelle des *Sockets* pour effectuer des manipulations de bas-niveau et il nécessite à ce titre des permissions supplémentaires.

L'invite >>> d'un interpréteur *Python3* apparaît. Vous pouvez maintenant programmer en Scapy. Par exemple :

```
$ scapy3
>>> x=IP()
>>> x.show()
>>> exit        # pour quitter scapy
```

Il est également possible d'écrire des programmes Scapy sous forme d'un script Python, qu'il faut enregistrer avec un éditeur de texte, comme nano ou emacs.

```
#!/usr/bin/env python3
import sys
from scapy.all import *

x = IP()
x.show()
```

Vous pouvez ensuite exécuter votre script comme ceci :

```
$ python3 test.py
```

Survolez rapidement la documentation pour découvrir les nombreuses possibiltés de Scapy : https://scapy.readthedocs.io/en/latest/introduction.html

## 1.3 Ping

Regardez dans le fichier ping.py un exemple d'utilisation de Scapy qui envoie un ping (ICMP) puis récupère la réponse. Essayez-le en recopiant le programme ligne par ligne, ou en faisant un copier/coller.

```
ping = IP(dst='192.168.0.2')/ICMP(type='echo-request')
ping.show()
pong = sr1(ping)
pong.show()
```

## 1.4 ARP

Rappelez le fonctionnement du protcole ARP. Notez que le protocole ARP ne dispose que de deux opérations : la requête (*Who Has*) et la réponse. On peut alors utiliser ce protocole pour effectuer un *ping* dans le réseau local Ethernet. Il s'agit d'envoyer une requête ARP. Si la machine répond, c'est bien qu'elle est en vie !

Commencez par construire une trame Ethernet avec `Ether()` vers l"adresse de broadcast `FF:FF:FF:FF:FF:FF` et encapsulez le datagramme `ARP()` à destination de l'adresse IP visée. Pour envoyer et recevoir une trame Ethernet, il faut utiliser la fonction `srp1()` (à la place de la fonction `sr1()` réservé aux paquets IP). On peut aussi utiliser dans cette fonction l'option `timeout=1` pour limiter le temps d'attente à 1 seconde, dans le cas où il n'y a pas de réponse.

### 1.5 Services UDP : Daytime et Echo

Suivez l'exemple du fichier daytime.py qui envoie un paquet UDP sur le port `daytime` (13) puis récupère et affiche la date envoyée en réponse. Essayez pas à pas.

Vous avez pu remarquer que le service `udp echo` est ouvert (port 7). Testez ce service en envoyant le message 'hello'. Quelle est la réponse ? Expliquez à quoi sert le *padding* dans la réponse ?

### 1.6 Syn Scan

Pour tester si un service TCP est disponible, il suffit d'essayer de s'y connecter, en envoyant un datagramme TCP à destination du port visé avec le flag SYN. Essayez avec les ports 80 et 81... Dans l'en-tête TCP, il faut mettre le champs `flags="S"` pour SYN. En déduire une méthode pour déterminer si un port donné est ouvert (ou fermé).

Écrivez une fonction qui prend en paramètre une adresse IP et qui imprime la liste des ports ouverts entre 1 et 100. Il s'agit donc de faire la même chose que ce que l'on a fait à la main ci-dessus, dans une boucle for. Pour tester si le flag S (SYN) est présent, il suffit d'utiliser `answer.payload.flags.S`. De même pour A (ACK) ou R (RST).

Pour ouvrir un port supplémentaire sur `opeth`, lancez-y `nc -l -p 42 &` par exemple.

## 2 Pour aller plus loin

### 2.1 Traceroute

A l'aide de Scapy, écrire un programme *traceroute*, qui cherche la route vers une destination IP. Le principe consiste à jouer avec le champs TTL (time to leave) de l'en-tête IP en le faisant croitre à partir de 1... pour provoquer une erreur sur les routeurs qui vont rejeter consécutivement ce paquet en renvoyant un message d'erreur ICMP (time to leave exceeded) !

### 2.2 Une connexion TCP

Essayez d'établir (à la main, donc) une connexion vers le service `echo` mais cette fois-ci en TCP. Il s'agit d'effectuer la poignée de main TCP en trois temps : SYN, SYN/ACK, ACK. Attention, il faudra donc bien sûr récupérer le numéro de séquence, et renvoyer les bons numéro d'acquittement, etc. Ignorez l'apparition de datagrammes RST dans le tcpdump, ils sont produits par le noyau parce que Scapy est fourbe, mais ça ne gênent pas !