Université de Limoges | FACULTÉ DES SCIENCES ET TECHNIQUES

# Master 1ère année

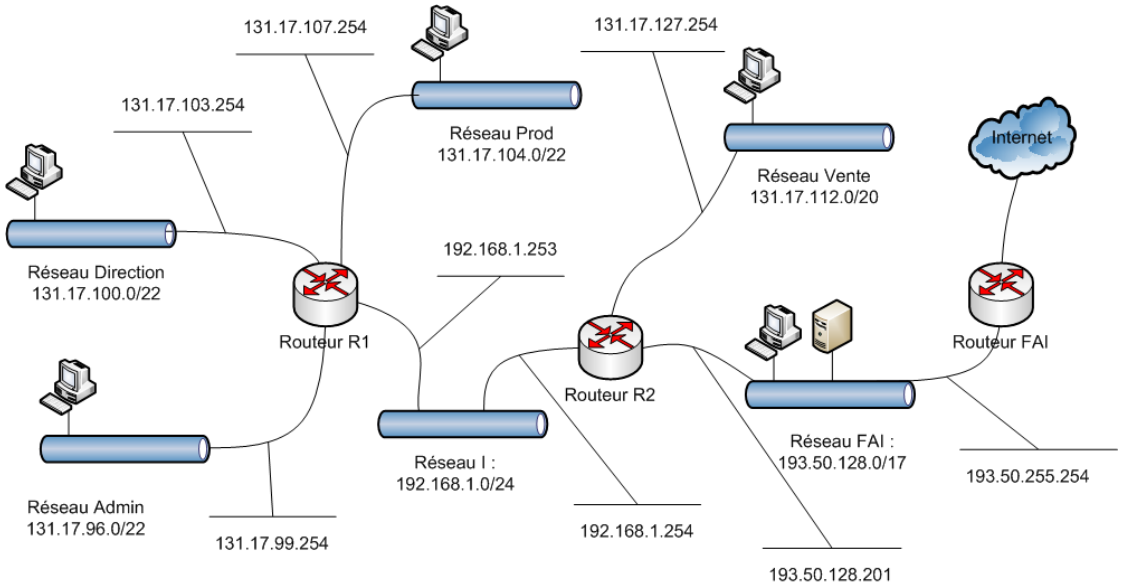
## Réseaux Avancés II

*Examen — mai 2012*

*Durée : 2h — Documents autorisés (sauf sur support électronique)*

**IPv6 – (9 points)**

**1 – 4pts** Soit le réseau d'une entreprise `131.17.96.0/19` décrit par le schéma suivant :




Cette entreprise vient d'obtenir auprès du RIPE l'adresse IPv6 suivante : `2001:660:6201::/48`.
Le responsable du système d'information vous demande de réaliser une migration du réseau vers IPv6.
Il vous fournit la description suivante :

- ⋆ il existe deux bâtiments distincts dans l'entreprise : $\text{Bâtiment}_A$ et $\text{Bâtiment}_B$ ;
- ⋆ les différents réseaux se répartissent :

« géographiquement » de la façon suivante :

| Network | Location |
|---|---|
| Direction | $\text{Bâtiment}_A$ |
| Vente | $\text{Bâtiment}_B$ |
| Prod | $\text{Bâtiment}_A$ |
| Admin | $\text{Bâtiment}_A$ |
| Réseau I | $\text{Bâtiment}_A$ |

« par usage » de la façon suivante :

| Type |
|---|
| Visiteur |
| Chargé de clientèle |
| Ouvrier |
| Direction |
| Admin |
| Infrastructure |

a. Décomposez en nombre de bits, l'@IPv6 fournie, en tenant compte des « Types » & « Location ».

b. De combien de sous-réseaux pourra-t-on bénéficier après cette décomposition et pour chaque combinaison « type/location » ?

c. Donnez un plan d'adressage en IPv6 pour ce réseau, en favorisant la localisation.
*Vous donnerez pour chaque réseau utilisé l'adresse IPv6 qui lui correspond.*

d. Comment les machines connectées dans les différents réseaux vont-elles apprendre le préfixe réseau auquel elles appartiennent ?

e. Est-ce que pendant la période de configuration matérielle de ce plan d'adressage (où il n'y aura plus de routage), les machines du réseau « Prod » pourront-elles continuer à communiquer entre elles et pourquoi ?

**2 –** Soit l'@MAC « `da:ec:6d:59:96:2e` » d'une interface réseau :

**2pts**

a. Donnez son @IPv6 de lien obtenue par auto-configuration ;

b. Par quelle @IPv6 multicast de type « *solicitation de voisin* », cette machine pourra-t-elle être jointe (protocole « Neighbor Discovery ») ?

c. Si le préfixe global du réseau auquel appartient cette machine est le `2001:660:6201::/48` quelle sera son adresse globale ?

d. Lors de l'envoi d'une trame d'ethertype `0x86DD` en unicast vers cette interface, qu'elle sera l'@MAC destination de cette trame ?

**3 –** Analysez la trame suivante :

**3pts**

```
0000   00 10 75 1A D4 8A 00 26  BB 15 8E 87 86 DD 60 00   ..u....&......‘.
0010   00 00 00 1C 2C 40 FE 80  00 00 00 00 00 00 02 26   ....,@.........&
0020   BB FF FE 15 8E 87 FE 80  00 00 00 00 00 00 02 10   ................
0030   75 FF FE 1A D4 8A 06 00  00 00 00 00 00 00 00 50   u..............P
0040   13 89 00 00 04 E8 00 01  7F 29 50 12 20 00 65 6D   .........)P. .em
0050   00 00                                             ..
```

## ▬▬▬ Tunnels – (2 points)

**4 –** Questions :

**2pts**

a. Quelles différences entre un tunnel SIT, « Simple Internet Transition », et un tunnel GRE, « Generic Routing Encapsulation » ?

b. Quelles différences entre un tunnel GRE et la technologie MPLS ?

## ▬▬▬ Routage dynamique – (2 points)

**5 –** Questions :

**2pts**

a. En étudiant la table de routage d'un routeur, peux-t-on savoir si elle a été construite à l'aide du protocole RIP ou OSPF ? Pourquoi ?

b. Est-il possible de reconstruire la topologie complète du réseau grâce à **un seul arbre** obtenu par l'algorithme de Dijkstra à partir d'un des routeurs du réseau (dans un protocole de routage par « états de lien ») ? Pourquoi ?

## ▬▬▬ Programmation Python avec Scapy – (7 points)

**6 –** Le « ping » étant interdit par une entreprise, celle-ci vous demande de simuler son fonctionnement à

**3pts** l'aide du protocole UDP (qui lui est *bizarrement* autorisé) et qui, à chaque réception d'un paquet UDP sur le port `55555` :

- ⋆ vérifie qu'il contient le texte « `PING UDP :` » + un « numéro d'ordre » ;
- ⋆ renvoie à l'émetteur un datagramme IP :
  - ◇ de même « id », c-à-d identifiant de datagramme ;
  - ◇ contenant un datagramme UDP :
    - ⋆ contenant le texte « `PING UDP :` » + le même « numéro d'ordre » que celui reçu.

a. Est-il nécessaire d'utiliser Scapy et pourquoi ?

b. Écrivez le programme en Python mettant en œuvre ce serveur (pour information, l'@IP du serveur où tournera votre programme est `192.168.10.10`).

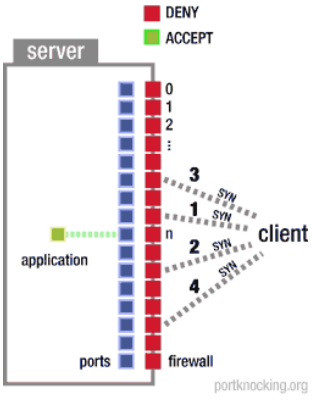
## 7 – Réalisation d'un serveur de « *Port Knocking* ».

**4pts**

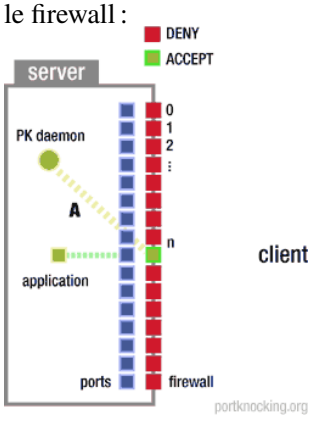
*Qu'est-ce que le « Port Knocking » ?*

Il consiste à autoriser la connexion d'un client à une application **uniquement après** que ce client ait effectué une **séquence précise de tentative de connexion** vers des **ports fermés** du serveur :

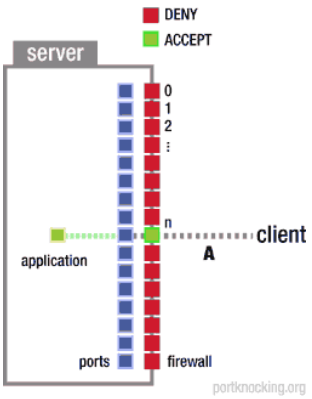
Le client effectue sa séquence de tentative de connexion :




Le serveur détecte cette séquence avec le « PK » daemon qui autorise la connexion avec le firewall :




Le client peut alors se connecter à l'application :




Vous écrirez le programme Python utilisant Scapy et réalisant :

- ⋆ la détection de cette séquence de tentative de connexion correspondant à du « port knocking » ;
- ⋆ l'autorisation de la connexion vers l'application (le travail du « PK daemon ») :
  - la séquence de tentative de connexion TCP est la suivante : `[2027, 3230, 2001, 17377]` ;
  - l'application attend en TCP sur le port `16400` ;
  - le serveur hébergeant l'application et votre programme possède l'@IP `192.168.1.137`.

### Quelques rappels

```
>>> ls(IP)
version    : BitField             = (4)
ihl        : BitField             = (None)
tos        : XByteField           = (0)
len        : ShortField           = (None)
id         : ShortField           = (1)
flags      : FlagsField           = (0)
frag       : BitField             = (0)
ttl        : ByteField            = (64)
proto      : ByteEnumField        = (0)
chksum     : XShortField          = (None)
src        : Emph                 = (None)
dst        : Emph                 = ('127.0.0.1')
options    : PacketListField      = ([])

>>> import commands
>>> commands.getoutput('echo toto')
```

```
>>> ls(TCP)
sport      : ShortEnumField       = (20)
dport      : ShortEnumField       = (80)
seq        : IntField             = (0)
ack        : IntField             = (0)
dataofs    : BitField             = (None)
reserved   : BitField             = (0)
flags      : FlagsField           = (2)
window     : ShortField           = (8192)
chksum     : XShortField          = (None)
urgptr     : ShortField           = (0)
options    : TCPOptionsField      = ({})

>>> ls(UDP)
sport      : ShortEnumField       = (53)
dport      : ShortEnumField       = (53)
len        : ShortField           = (None)
chksum     : XShortField          = (None)
```

```
sniff(count=0, prn=None, filter=None, timeout=None)
    Sniff packets
    sniff([count=0,] [prn=None,] [filter=None,]) -> list of packets

      count: number of packets to capture. 0 means infinity
        prn: function to apply to each packet. If something is returned,
             it is displayed. Ex:
             ex: prn = lambda x: x.summary()
     filter: filter packets according to tcpdump syntax filter
    timeout: stop sniffing after a given time (default: None)
```