Scan réseau couche 2 (ARP)

Pour et contre :

Pour :

- Rapide
- Non-protégé

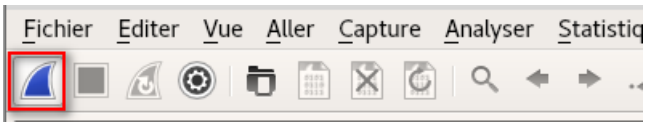
Il peut-être important de noter les informations récupérées ou d'inclure dans un fichier les informations récupérées grâce à l'option > *fichier*

Wireshark

Lancez Wireshark

Choisissez votre carte réseau eth0

Lancez la capture



Utilisation de Scapy

Lancez Scapy

```
INFO: Can't import python gnuplot wrapper . Won't be able to plot.
WARNING: No route found for IPv6 destination :: (no default route?)
Welcome to Scapy (2.2.0)
>>>
```

Un prompt apparaît.

Scapy dispose déjà de la structure pour les requêtes ARP, pour l'afficher utiliser la fonction display.

```
>>> ARP().display()
```

```
>>> ARP().display()
###[ ARP ]###
  hwtype= 0x1
  ptype= 0x800
  hwlen= 6
  plen= 4
  op= who-has
  hwsrc= 00:0c:29:49:2e:99
  psrc= 192.168.99.142
  hwdst= 00:00:00:00:00:00
  pdst= 0.0.0.0
```

Les différents champs de la requête ARP apparaissent :

hwtype=
ptype=
hwlen=
plen=
op=
hwsrc=
psrc=
hwdst=
pdst=

Par défaut tous les champs sources sont remplis avec les informations de la machine.

Il ne faut jamais travailler sur la structure ARP fournie par scapy.

Il faut créer une variable avec le contenu de la structure ARP.

```
>>> arp_request = ARP()
```

Il faut ensuite changer l'adresse IP de destination.

```
>>> arp_request.pdst="192.168.99.2"
```

Affichez la structure de arp_request

```
>>> arp_request.display()
###[ ARP ]###
  hwtype= 0x1
  ptype= 0x800
  hwlen= 6
  plen= 4
  op= who-has
  hwsrc= 00:0c:29:49:2e:99
  psrc= 192.168.99.142
  hwdst= 00:00:00:00:00:00
  pdst= 192.168.99.2
```

L'adresse de destination a changé (pdst=192.168.99.2)
Les champs sources sont remplis automatiquement avec la configuration de la machine.

Il est donc possible de changer l'adresse MAC, IP source, etc.

Envoyez votre requête ARP grâce à la fonction sr1

```
>>> sr1 (arp_request)
```

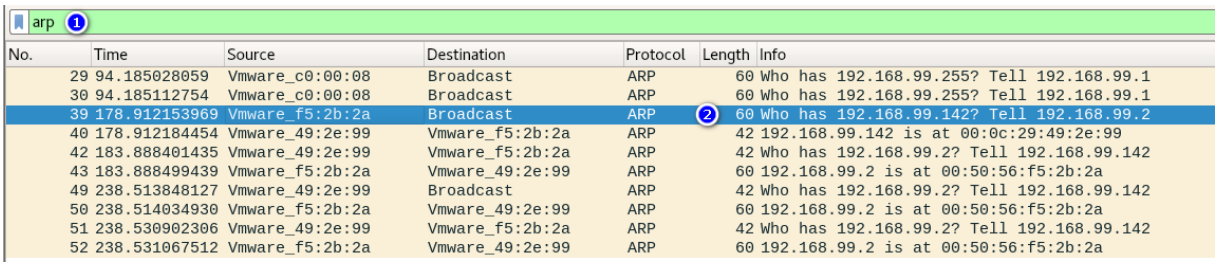
Le paquet est envoyé et scapy nous informe d'une éventuelle réponse.

```
Begin emission:
*Finished to send 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
<ARP  hwtype=0x1 ptype=0x800 hwlen=6 plen=4 op=is-at hwsrc=00:50:56:f5:2b:2a psrc=192.168.99.2 hw
dst=00:0c:29:49:2e:99 pdst=192.168.99.142 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00' |>>
```

Il y a donc un retour de la machine 192.168.99.2 avec son adresse MAC. (hwdst)

Analysez la requête ARP dans wireshark



Il est possible d'envoyer une requête via la structure ARP de base (sans création d'une variable) et de modifier le contenu seulement pour un envoi.

```
>>> sr1(ARP(pdst="192.168.99.2"))
Begin emission:
*Finished to send 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
<ARP  hwtype=0x1 ptype=0x800 hwlen=6 plen=4 op=is-at hwsrc=00:50:56:f5:2b:2a psrc=192.168.99.2 hw
dst=00:0c:29:49:2e:99 pdst=192.168.99.142 |<Padding  load='\x00\x00\x00\x00\x00\x00\x00\x00\x00\x
00\x00\x00\x00\x00\x00\x00\x00\x00' |>>
>>>
```

Il est possible de mieux structurer la réponse en la mettant dans une variable pour l'afficher avec la fonction display.

```
>>> reponse = sr1(ARP(pdst="192.168.99.2"))
Begin emission:
*Finished to send 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
>>>
```

Affichez la réponse avec la fonction display.

```
>>> reponse.display()
###[ ARP ]###
  hwtype= 0x1
  ptype= 0x800
  hwlen= 6
  plen= 4
  op= is-at
  hwsrc= 00:50:56:f5:2b:2a
  psrc= 192.168.99.2
  hwdst= 00:0c:29:49:2e:99
  pdst= 192.168.99.142
###[ Padding ]###
     load= '\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00'
```

On récupère donc la syntaxe de la réponse. La variable reponse peut donc

être modifiée et réutilisée.

Effectuez différentes requêtes avec scapy en modifiant l'adresse source, MAC etc... et analysez dans wireshark vos différentes requêtes.

Pour quitter Scapy :

```
>>> quit()
root@kali:~#
```

ARPing

Arping envoie une requête ARP sur une ou plusieurs machines, afin de recevoir une réponse potentielle, ce qui permet de savoir si l'adresse machine est active ainsi que son adresse MAC.

```
root@kali:~# arping 192.168.99.146
ARPING 192.168.99.146
60 bytes from 00:0c:29:ce:94:f9 (192.168.99.146): index=0 time=42.794 usec
60 bytes from 00:0c:29:ce:94:f9 (192.168.99.146): index=1 time=172.914 usec
60 bytes from 00:0c:29:ce:94:f9 (192.168.99.146): index=2 time=217.292 usec
60 bytes from 00:0c:29:ce:94:f9 (192.168.99.146): index=3 time=128.814 usec
60 bytes from 00:0c:29:ce:94:f9 (192.168.99.146): index=4 time=160.639 usec
^C
--- 192.168.99.146 statistics ---
5 packets transmitted, 5 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.043/0.144/0.217/0.058 ms
```

Par défaut arping ne cesse d'envoyer des requêtes jusqu’à l'arrêt manuel de l'attaquant (ctrl+C)

Avec l'option -c il est possible de préciser le nombre de requêtes.

```
root@kali:~# arping 192.168.99.146 -c 1
ARPING 192.168.99.146
60 bytes from 00:0c:29:ce:94:f9 (192.168.99.146): index=0 time=125.420 usec

--- 192.168.99.146 statistics ---
1 packets transmitted, 1 packets received,   0% unanswered (0 extra)
rtt min/avg/max/std-dev = 0.125/0.125/0.125/0.000 ms
```

```
1 0.000000000   Vmware_49:2e:99   Broadcast   ARP   58 Who has 192.168.99.146? Tell 192.168.99.142
2 0.000251069   Vmware_ce:94:f9   Vmware_49:2e:99   ARP   60 192.168.99.146 is at 00:0c:29:ce:94:f9
```

Lancez une seconde requête qui ne peut aboutir.

```
root@kali:~# arping 192.168.99.147
ARPING 192.168.99.147
Timeout
Timeout
Timeout
^C
--- 192.168.99.147 statistics ---
4 packets transmitted, 0 packets received, 100% unanswered (0 extra)
```

Comparez les deux et trouvez une chaîne de caractères présente dans la réponse valide et non présente dans la réponse non aboutit.

```
60 bytes from
```

(réponses : bytes from)

1ere optimisation :

Il est possible avec quelques fonctions d'optimiser quelques outils.

Exemple :

```
root@kali:~# for addr in $(seq 1 254); do arping 192.168.99.$addr -c 1 |grep "bytes from" >> test.txt & done
[2] 3604
[3] 3606
```

Affichez le contenu de test.txt

```
root@kali:~# cat test.txt
60 bytes from 00:50:56:c0:00:08 (192.168.99.1): index=0 time=13.891 msec
60 bytes from 00:50:56:f5:2b:2a (192.168.99.2): index=0 time=262.161 usec
60 bytes from 00:0c:29:ce:94:f9 (192.168.99.146): index=0 time=9.078 msec
60 bytes from 00:50:56:e0:15:8a (192.168.99.254): index=0 time=2.458 msec
60 bytes from 00:0c:29:94:2b:bf (192.168.99.253): index=0 time=15.429 msec
root@kali:~#
```

Arping a scanné tout le réseau et listé seulement les réponses valides.

2eme Optimisation

La seconde optimisation consiste à retirer le test pour récupérer seulement l'adresse IP.

```
root@kali:~# for addr in $(seq 1 254); do arping 192.168.99.$addr -c 1 |grep "bytes from" |
cut -d " " -f 5 | cut -d "(" -f 2 | cut -d ")" -f 1 >> iplist.txt & done
```

Affichez le contenu de ipliste.txt

```
root@kali:~# cat iplist.txt
192.168.99.1
192.168.99.2
192.168.99.146
192.168.99.253
192.168.99.254
root@kali:~#
```

Essayez de comprendre la syntaxe de la commande.

Nmap

Lancez un scan ARP sur une adresse via nmap

```
root@kali:~# nmap 192.168.99.146 -sn

Starting Nmap 7.60 ( https://nmap.org ) at 2017-12-14 16:55 CET
Nmap scan report for 192.168.99.146
Host is up (0.00027s latency).
MAC Address: 00:0C:29:CE:94:F9 (VMware)
Nmap done: 1 IP address (1 host up) scanned in 13.26 seconds
root@kali:~#
```

Nmap nous indique que l'hôte est actif.

Il est aussi possible de scanner une plage réseau.

```
root@kali:~# nmap 192.168.99.1-254 -sn
```

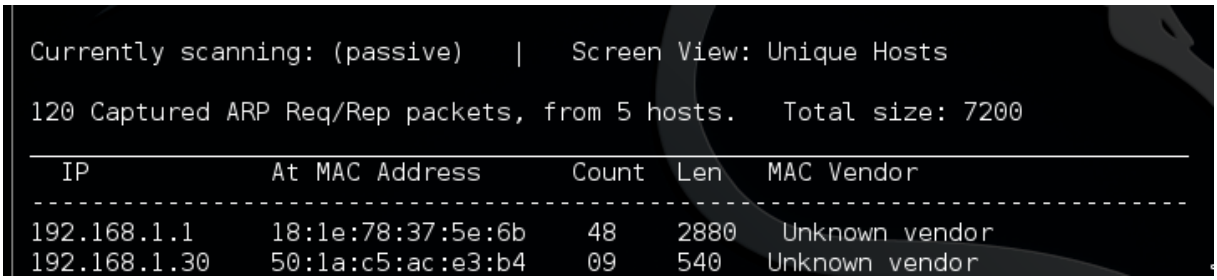
NetDiscover

Scannez une plage réseau via netdiscover.

```
root@kali:~# netdiscover -r 192.168.99.0/24
```

NetDiscover permet de mettre son interface en écoute et d'attendre différentes requêtes ARP du réseau, de les lister, ce qui permet de ne pas émettre de requête ARP (détectable via IDS)

```
root@kali:~# netdiscover -p
```

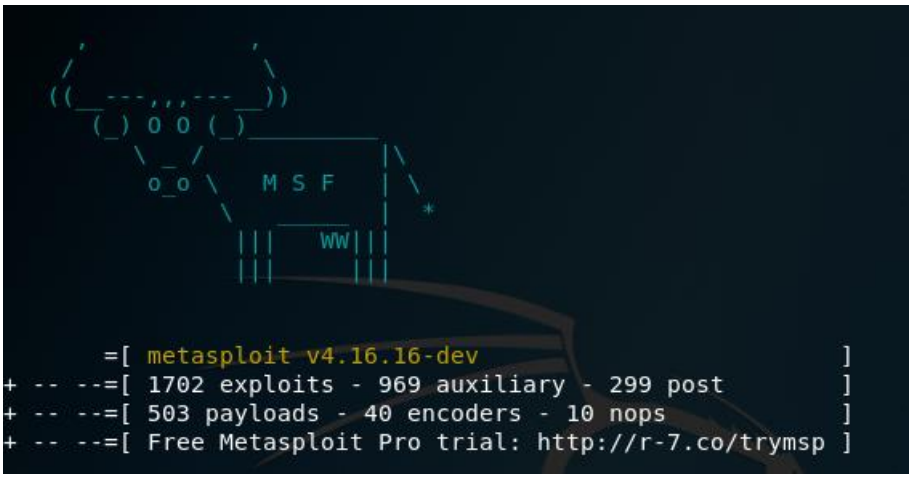


On récupère donc au fur et à mesure les adresses IP et MAC du réseau.

Metasploit

Lancez Metasploit.

```
root@kali:~# msfconsole
```



Choisir le module arp_sweep qui permet le scan ARP.

```
msf > use auxiliary/scanner/discovery/arp_sweep
msf auxiliary(arp_sweep) >
```

Listez les options disponibles du module

```
msf auxiliary(arp_sweep) > show options

Module options (auxiliary/scanner/discovery/arp_sweep):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   INTERFACE                   no        The name of the interface
   RHOSTS                      yes       The target address range or CIDR identifier
   SHOST                       no        Source IP Address
   SMAC                        no        Source MAC Address
   THREADS    1                yes       The number of concurrent threads
   TIMEOUT    5                yes       The number of seconds to wait for new data
```

Il est possible de lancer des commandes shell sous le prompt msf.

```
msf auxiliary(arp_sweep) > ifconfig
[*] exec: ifconfig

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.99.142  netmask 255.255.255.0  broadcast 192.168.99.255
        inet6 fe80::77e2:9bab:aefd:c926  prefixlen 64  scopeid 0x20<link>
        ether 00:0c:29:49:2e:99  txqueuelen 1000  (Ethernet)
        RX packets 545426  bytes 803803465 (766.5 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 206262  bytes 13493072 (12.8 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Il faut donc remplir tous les champs requis (au minimum), listés dans show

options

```
msf auxiliary(arp_sweep) > set interface eth0
```

Vérifiez grâce à show options que le champ interface a bien été modifié

```
msf auxiliary(arp_sweep) > show options

Module options (auxiliary/scanner/discovery/arp_sweep):

   Name       Current Setting  Required  Description
   ----       ---------------  --------  -----------
   INTERFACE  eth0             no        The name of the interface
```

```
msf auxiliary(arp_sweep) > set rhosts 192.168.99.0/24
rhosts => 192.168.99.0/24
```

Il est possible de modifier l'adresse MAC et IP source, cependant la réponse à la requête ne va pas revenir à la machine attaquante.

Vérifiez vos changements à l'aide de show options

```
msf auxiliary(arp_sweep) > show options
```

Lancez le scan

```
msf auxiliary(arp_sweep) > run
```