

## Travaux pratiques –TP

### Analyse réseaux et attaques

#### Partie 1 : Analyse réseaux avec Wireshark

Wireshark (appelé Ethereal jusqu'à 2006) est l'analyseur réseau le plus populaire, il fournit des informations sur des protocoles réseaux et applicatifs. Wireshark utilise la librairie réseau pcap pour capturer les paquets réseaux. Il y a deux types de filtres dans Wireshark :

- Filtres de capture : pour sélectionner les données à enregistrer
- Filtres d'affichage : pour rechercher des informations à l'intérieur des données capturées

La syntaxe des deux types de filtres est complètement différente.

#### Exercice 1 : Filtres de capture

protocole	direction	hôte(s)	valeur	opérations logiques	autre expression
tcp	dst	10.1.1.1	80	and	tcp dst 10.2.2.2 3128

- Protocole : ether, fddi, ip, arp, rarp, decnet, lat, sca, mopr, mopdl, tcp and udp.
- Direction: src, dst, src and dst, src or dst
- Hôte(s) : net, port, host, portrange
- Operations logiques : not, and, or

Lancer Wireshark et filtrer le trafic selon les filtres suivants :

- tcp dst port 80 : ne capturer que les paquets avec un port destination TCP de 80
- ip src host 10.1.1.1 : ne capturer que les paquets du host 10.1.1.1
- host 10.1.2.3 : ne capturer que les paquets d'adresse source ou destination égal à 10.1.2.3
- src portrange 2000-2500 : ne capturer que les paquets qui ont un port entre 2000 et 2500
- not icmp : ne capturer pas d'icmp
- src host 10.7.2.12 and not dst net 10.200.0.0/16 : ne capturer que les paquets qui ont une adresse source 10.7.2.12 et qui n'ont pas destinés au réseau 10.200.0.0/16

#### Exercice 2 : Filtres d'affichage

Protocole	Champs 1	Champs 2	Opérateur de comparaison	Valeur	Operations logiques	Autre expression
-----------	----------	----------	--------------------------	--------	---------------------	------------------

#### Opérateurs de comparaison

==	Equal
!=	Non égal (Not Equal)
>	Plus grand que
<	Plus petit que
>=	Plus grand ou égale à <=Plus petit ou égal à

#### Expressions logiques

&&	Logical AND (et)
	Logical OR (ou)
^^	Logical XOR (ou)
!	Logical NOT (non)

Tester les filtres suivants et analyser les résultats :

- `tcp.flags.syn == 0x02`
- `ip.addr == 10.1.1.1`
- `tcp.dstport == 25`
- `tcp.port == 25`
- `ip.src != 10.1.2.3 and ip.dst != 10.4.5.6`

### **Partie 2 : Injection de trafic avec l'outil Scapy**

Scapy est un outil Open Source écrit en python, il permet de manipuler, forger, décoder, émettre, recevoir les paquets d'une multitude de protocoles (ARP, DHCP, DNS, ICMP, IP...). Scapy est principalement utilisé pour des tests de sécurité.

- Distributé sous GPLv2
- <http://www.secdev.org/projects/scapy/>
- Doc : <http://www.secdev.org/projects/scapy/doc/index.html>

Pour le télécharger : `apt-get install python-scapy`

Lancer scapy : `scapy`

Voir les différentes fonctions : `lsc()`

Pour sortir : `exit()`

### **Exercice 3 :**

Construire un paquet et visualiser son contenu.

```
>>> a=IP(ttl=10)
>>> a
< IP ttl=10 |>
>>> a.src
'127.0.0.1'
>>> a.dst="192.168.1.1"
>>> a
< IP ttl=10 dst=192.168.1.1 |>
>>> a.src
'192.168.8.14'
>>> del(a.ttl)
>>> a
< IP dst=192.168.1.1 |>
>>> a.ttl
64
```

Empilement des couches : l'opérateur `/` a été utilisé comme opérateur de composition entre deux couches. Pour construire un message ping :

```
a = IP(dst="192.168.1.1", src="10.10.10.1" )/ICMP()
```

Visualiser le contenu :

```
ls(a)
```

Envoyer un ping :

```
a = IP(dst="192.168.1.1")/ICMP(type=8, code=0)
a.summary()
res = sr(a) //envoyer le paquet a
```

Lancer Wireshark sur la machine distante et vérifier la réception du ping.

Visualiser la réponse *echo reply* sur la machine source :

```
res[0].summary() // voir le premier résultat
```

Utiliser les champs d'une trame au niveau 2 :

```
sendp(Ether(dst="08:11:96:f6:42:12")/IP(dst="192.168.1.1")/ICMP())
```

Encapsulation

```
a=Ether()/IP()/ICMP()
sendp(a)
```

Envoyer une requête SYN

```
sendp(IP(dst="72.14.207.99")/TCP(dport=80, flags="S"))
sendp(IP(dst="192.168.1.1")/TCP(sport=666, dport=(440, 443), flags="S"))
```

Lancer Wireshark sur la machine distante et vérifier la réception du SYN.

Voir le paquet en Hex

```
hexdump()
hexdump(pkt)
```

### Exercice 4 :

Fragmentation de paquets

Envoyer deux messages de tailles différentes et regarder le résultat sur la machine distante.

```
send( fragment(IP(dst="10.0.0.5")/ICMP()/("X"*1472)) )
send( fragment(IP(dst="10.0.0.5")/ICMP()/("X"*1473)) )
```

Dans la première commande, combien de paquets la machine distante a-t-elle reçu ? Dans la deuxième ? Pourquoi ?

Sniffing

Pour sniffer, on utilise la commande sniff.

```
pkts = sniff(count=10)
pkts.show()
```

```
pkts = sniff(filter="icmp and host 192.168.1.1", count = 2)
pkts.show()
```

### Exercice 5:

Générer un paquet mal formé

```
sendp(IP(dst="10.1.1.5", ihl=2, version=3)/ICMP())
```

Générer d'autres paquets mal formés et visualiser le résultat avec Wireshark

Land attack

```
sendp(IP(src=target, dst=target) / ICMP() / ("X"*1472)), loop=1)
```

Envoyer un TCP SYN sur chaque port et attendre un SYN-ACK, RST ou ICMP error

```
res,unans = sr( IP(dst="target")/TCP(flags="S", dport=(1,1024)))  
Visualiser le résultat sur Wireshark (réception de 1024 paquets)
```

```
unans[1].summary()
```

### Exercice 6:

Faire une traceroute

```
res,unans=traceroute(["www.voila.com"],maxttl=20)  
res.show()  
res.graph() # pour voir le résultat de traceroute en image  
res.graph(type="ps",target="| lp") # choisir le type du fichier  
res.graph(target="> /tmp/graph.svg") # sauvegarder le fichier  
res.trace3D(), pour voir le résultat en 3D # nécessite une librairie graphique,  
à installer
```

Ecrivez une commande qui vous permet d'afficher le même résultat que traceroute.

```
rep,non_rep=sr( IP(dst='209.85.143.100', ttl=(1,25)) / ICMP(), timeout=1 )  
rep.summary()
```

Voir toutes les adresse MAC /IP dans un réseau : `arping(" 192.168.1.* ")`

Statistiques à vérifier avec : `netstat -s`

Corruption de cache ARP

Cette attaque empêche un client de joindre directement la passerelle par une corruption de cache ARP

```
sendp( Ether(dst=clientMAC)/ARP(op="who-has", psrc=gateway, pdst=client),  
loop=1 )
```

### Exercice 7:

Ecrivez un script pour capturer 10 paquets et les afficher

```
#!/usr/bin/env python  
from scapy.all import *  
a=sniff(count=10)  
a.summary()
```

```
chmod +x scapysniff.py
```

```
./scapysniff.py
```

Voir les machines qui sont online

```
#!/usr/bin/python from
scapy.all import *
for ip in range(0, 256):
    packet = IP(dst="192.168.0." + str(ip), ttl=20)/ICMP()
    reply = sr1(packet)
    if "192.168." in reply.src: print reply.src, "is online"
```

### **Partie 3** : Scan de réseaux avec NMAP

NMAP est abréviation de mappeur de réseau. Cet outil est utilisé pour scanner les ports sur une machine (locale ou distante) et peut être installé sur Windows, Sun Solaris, Linux pour scanner même les grands réseaux pour obtenir des détails :

- du système d'exploitation
- des ports ouverts
- des logiciels utilisés pour un service et leurs versions,
- du fournisseur de la carte réseau

NMAP peut être utilisé par les pirates pour analyser les vulnérabilités des systèmes

Syntaxe de la commande : **nmap** [Scan Type...] [Options] {target specification}

#### **Exercice 8:**

Vérifiez un port particulier sur une machine

```
nmap -p portnumber hostname
nmap -p 53 192.168.0.1
```

Pour scanner les ports ouverts sur une machine

```
nmap hostname
nmap 192.168.0.1
```

Pour balayer les ports ouverts sur un réseau

```
nmap network ID/subnet-mask
nmap 192.168.1.0/24
```

Analyser uniquement les ports avec l'option -F

```
Scan de ports 70% plus rapide que le scan normal
nmap -F hostname
-F pour la rapidité, cette option n'ajoute pas d'autre scan.
nmap -F 192.168.1.1
```

Scanner une machine avec l'option -v pour le mode verbose

Scanner la machine et donner le maximum de détails

```
nmap -v hostname  
nmap -v 192.168.1.1
```

Scanner les ports ouverts du protocole TCP sur une machine

```
nmap -sT hostname  
s pour le balayage et T pour balayer seulement les ports TCP  
nmap -sT 192.168.1.1
```

Scanner les ports ouverts du protocole UDP sur une machine

```
nmap -sU hostname  
U pour balayer seulement les ports UDP, permission root
```

Scanner sur une machine les services et leurs versions

```
nmap -sV hostname  
s pour le balayage et V indique la version des services sur la machine  
nmap -sV 192.168.1.1
```

Vérifier les protocoles tels que TCP, UDP, ICMP, IGMP supportés sur une machine.

```
nmap -sO hostname  
nmap -sO localhost
```

Vérifier le système d'exploitation en cours d'exécution

```
nmap -O hostname  
-O pour vérifier le système d'exploitation, avec un scan par default  
nmap -O google.com
```

Scanner les ports TCP sur target.com; verbose mode

```
nmap -v target.com
```

Pinger le réseau pour trouver les machines actives et ensuite les scanner

```
nmap -sS -O target.com/24
```

Trouver les serveurs qui ont des adresses qui se terminent par .2.3, .2.4, or .2.5

```
nmap -v -p 80 *.*.2.3-5
```

Scanner plusieurs @IP

```
nmap 192.168.1.1 192.168.1.2 192.168.1.3  
nmap 192.168.1.1,2,3
```

Scanner des *ranges* d'@IP

`nmap 192.168.1.1-20`

Scanner un ensemble de machines

`nmap 192.168.1.*`

Exclusion de certaines adresses pendant le scan

`nmap 192.168.1.0/24 --exclude 192.168.1.5`

`nmap 192.168.1.0/24 --exclude 192.168.1.5,192.168.1.254`

Afficher les états des ports

`nmap --reason 192.168.1.1`

Faire un scan rapide

`nmap -F 192.168.1.1`

Scan très rapide

`nmap -T5 192.168.1.0/24`

Afficher les paquets pendant le scan

`nmap --packet-trace 192.168.1.1`

Mettre le résultat dans un fichier

`nmap 192.168.1.1 > output.txt`