Travaux pratiques –TP
Analyse réseaux et injection de trafic

Partie 1 : 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()

Les niveaux (par rapport au modèle TCP/IP) sont représentés de la manière suivante :

- Niveau 2 : Ether()
- Niveau 3 : IP()
- Niveau 4 : TCP (), UDP ()
- Autre : ICMP (), ARP(), etc.

Pour un envoyer un paquet sur le réseau, il y a plusieurs possibilités :

- send(Ether()/IP(), count=10) : permet d'envoyer 10 paquets au niveau 3.
- Sendp(Ether()/IP(dst="192.0.2.1") : permet d'envoyer un paquet à destination de 192.0.2.1
- srloop(IP(dst=8.8.8.8)/ICMP()) : permet d'envoyer des paquets en permanence et afficher la réponse sur l'ecran.

Résumé :

- send() pour envoyer un paquet de la couche 3 ;
- sendp() pour envoyer un paquet de la couche 2 ;
- sr1() pour envoyer un paquet de la couche 3 et retourner la première réponse ;
- srp1() pour envoyer un paquet de la couche 2 et retourner la première réponse ;
- sr() pour envoyer un paquet de la couche 3 et retourner toutes les réponses ;
- srp() pour envoyer un paquet de la couche 2 et retourner toutes les réponses.

Exercice 3 : Construire un message ICMP, visualiser son contenu et l'envoyer sur le réseau :

On commence par afficher le contenu de Ether(), IP() et ICMP()
>> Ether().show()

>> IP().show()
>> ICMP().show()

On déclare les unités de données :

>> trame=Ether()
>> ip=IP(dst="192.168.1.1")
>> icmp=ICMP()

Empilement des couches : l'opérateur / peut être utilisé comme opérateur de composition entre deux couches. Pour envoyer notre message ICMP, nous utilisons la fonction sendp() :

sendp(trame/ip/icmp,count=100)

Il est aussi possible de déclarer un objet et l'envoyer :
>> MonMessage=trame/ip/icmp
>> sendp(MonMessage, count=100)

Pour visualiser le contenu : `ls(MonMessage)`

Il est aussi possible d'envoyer le message directement de la manière suivante:
>> sendp(Ether()/IP(dst="192.168.1.1"))/ICMP(), count=100)

Voir le paquet en Hex

```
hexdump()
hexdump(MonMessage)
```

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

Il est aussi possible d'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())

Exercice 4 : Envoyer une requête SYN et afficher la réponse

Afficher le contenu de TCP avec la commande : TCP().show()

Envoyer un message TCP avec le drapeau TCP SYN=1 sur le port 80 vers une machine locale :

sendp(Ether()/IP(dst="72.14.207.99")/TCP(dport=80,flags="S"),count=100)

Envoyer un message TCP avec le drapeau TCP SYN=1 sur les ports 80 à 443 vers une machine locale :

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

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

Exercice 5 : sniffer des paquets et afficher le résultat

Pour sniffer, on utilise la commande sniff.

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

pkts = sniff(filter="tcp and port 443", count = 10)
pkts.show()
```

Pour mettre le résultat dans un fichier pdf et afficher le contenu des pauets :
`pkts.pdfdump("resultats.pdf")`

Exercice 6 : Générer des paquets mal formés

Avec Scapy, il est possible d'envoyer des paquets avec n'importe quel contenu. Par exemple, la commande suivante, envoie sur le réseau un paquet IP avec la version 2 !!!

```
sendp(Ether()/IP(dst="10.1.1.5", ihl=10, version=2)/ICMP())
```

Vérifier sur la machine distante via wireshark le contenu.

Pour usurper une adresse IP source, taper la commande suivante :

```
sendp(Ether()/IP(src="8.8.8.8", dst="10.1.1.5")/ICMP(),count=100)
```

Vérifier sur la machine distante que le serveur 8.8.8.8 est la source de ces pings.

Pour usurper une adresse MAC source, taper la commande suivante :

```
sendp(Ether(src="12:34:56:78:9a:bc")/IP(src="8.8.8.8",
dst="10.1.1.5")/ICMP(),count=100
```

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

Exercice 7 : Corruption de cache ARP

L'empoisonnement du cache ARP (*ARP cache poisoning*) est une technique qui permet de rediriger tous les flux entre deux machines communiquant sur un réseau local, quel que soit le protocole encapsulé par Ethernet. Les machines qui implémentent la couche TCP/IP utilisent leurs adresses IP pour communiquer. Sur un réseau Ethernet, les machines communiquent en utilisant leurs adresses MAC.

Lorsqu'une machine tente d'établir une communication, elle doit d'abord récupérer l'adresse MAC du destinataire. Une requête ARP (arp-request) est alors envoyée en broadcast (diffusion) sur le réseau local. Toutes les machines reçoivent ce paquet, mais seul le destinataire répond, puisqu'il a identifié sa propre adresse IP dans l'entête ARP. Il forge alors une réponse ARP (arp-reply) adressée à l'émetteur qui, connaissant à présent l'adresse physique de son interlocuteur, peut entamer une discussion avec lui. Cette adresse MAC sera stockée dans une table dynamique appelée table *arp* pour des utilisations futures :

```
[192.168.1.1]$ arp -a
 Adresse Internet        Adresse physique      Type
 192.168.1.254           00-30-84-9d-e4-c5     dynamique
```

Pour empoisonner cette table au niveau de la machine victime, il suffit de lui envoyer une réponse *arp* avec une adresse MAC correspondant à celle de l'attaquant. Résultat : le trafic de la victime sera dirigé vers la machine de l'attaquant et non pas à la passerelle légitime.

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

```
sendp(Ether(src=MAC_Attaquant, dst=MAC_Victime)/ARP(op=2, psrc=IP_Gateway),
inter=RandNum(10,40), loop=1 )
```

Pour router le trafic vers la vraie passerelle, taper la commande suivante:

```
echo 1 > /proc/net/ipv4/ip_forwarding
```

Partie 2 : 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*}

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 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.*

Scan très rapide
nmap -T5 192.168.1.0/24

Afficher les paquets pendant le scan
nmap --packet-trace 192.168.1.1