

Introduction à Scapy

■ ■ ■ Contrôle d'erreur

1 – Différence « checksum » et « CRC » :

- Créez à l'aide des fonctions et objets disponibles dans Scapy deux trames Ethernet contenant :
 - un paquet UDP contenant les données : '\x00\x00' ;
 - un paquet UDP contenant les données : '\xFF\xFF' ;
- Comparez les valeurs *calculées* des checksums présents dans la couche UDP : sont-ils identiques ? Pourquoi ?
Les valeurs sont identiques, car avec le checksum, l'addition de 65535 est équivalent à ajouter 0 !
- Calculez le CRC-32 de chacune de ces trames : sont-ils différents ? Pourquoi ?
Les valeurs sont différentes car le CRC est capable de détecter une modification de la trame d'une longueur de 16 bits.

■ ■ ■ Détection de trafic d'audit

2 – Écrire un programme utilisant Scapy permettant de détecter si on est la cible d'un scan de port :

```

1 #!/usr/bin/python3
2
3 from scapy.all import *
4
5 traces = {}
6
7 # traces est un dico de dico {@IPsrc:{port:temps, ...port:temps}, ...}
8 def verifier_attaque(a,t):
9     if len(t)<10:
10         return
11     temps_min = min(t.values())
12     temps_max = max(t.values())
13     if (temps_max-temps_min) < 10:
14         print ("Attaque possible depuis %s"%a)
15
16 def traiter_trame(t):
17     if (not TCP in t):
18         return
19     if (t.sprintf("%TCP.flags%")!='S'):
20         return
21     adresse_source = t[IP].src
22     port_destination = t[TCP].dport
23     temps = t.time
24     if (adresse_source in traces):
25         table_ports = traces[adresse_source]
26         table_ports[port_destination] = temps
27         print (traces)
28         verifier_attaque(adresse_source,table_ports)
29     else :
30         traces[adresse_source]={port_destination:temps}
31
32 sniff(count=0,filter="tcp",prn=traiter_trame)

```

3 – Réalisation d'un serveur de « Port Knocking ».

Vous écrirez le programme Python utilisant Scapy et réalisant :

- ★ la détection de cette séquence de tentative de connexion correspondant à du « port knocking » ;
- ★ l'autorisation à l'aide de NetFilter de la connexion vers l'application (le travail du « PK daemon »):
 - la séquence de tentative de connexion TCP est la suivante : [2027, 3230, 2001, 17377] ;
 - l'application attend en TCP sur le port 16400 ;
 - le serveur hébergeant l'application et votre programme possède l'@IP 192.168.1.137.

```
1#!/usr/bin/python3
2
3from scapy.all import *
4import commands
5
6sequence = [2027, 3230, 2001, 17377]
7adresse_serveur = '192.168.1.137'
8adresse_client = ''
9port_application = 16400
10rang_sequence = 0
11
12def traiter_paquet(p):
13    global rang_sequence
14    if (p.sprintf('%TCP.flags%') == 'S'):
15        if ((rang_sequence == 0) and (p[TCP].dport == sequence[0])):
16            adresse_client = p[IP].src
17            if ((p[TCP].dport == sequence[rang_sequence]) and (p[IP].src == adresse_client)):
18                rang_sequence += 1
19                print ("Knock...",rang_sequence)
20                if (rang_sequence == len(sequence)):
21                    rang_sequence = 0
22                    commands.getoutput('sudo iptables -A INPUT -s %s -p tcp --dport %d -m state
23--state NEW, ESTABLISHED -j ACCEPT'%(adresse_client,port_application))
24                sniff(filter="(dst host %s) and tcp"%adresse_serveur, prn=traiter_paquet)
```