Faculté des Sciences & Techniques — Université de Limoges

Master 1ère année

Audit&SécuRés

TP n°1

Introduction à Scapy

Contrôle d'erreur

1 – Différence « checksum » et « CRC » :

a. Créez à l'aide des fonctions et objets disponibles dans Scapy deux trames au format Ethernet contenant :
   1. un paquet UDP contenant les données : `b'\x00\x00'` ;
   2. un paquet UDP contenant les données : `b'\xFF\xFF'` ;

   Les deux trames posséderont les mêmes informations suivantes :
   - adresse IP source : `164.81.50.10` ;
   - adresse IP destination : `164.81.50.20` ;
   - port source : `6578` ;
   - port destination : `7869` ;

b. Comparez les valeurs *calculées* des checksums présents dans la couche UDP : sont ils identiques ? Pourquoi ?

c. Calculez le CRC-32 de chacune de ces trames : sont-ils différents ? Pourquoi ?

Dans Scapy :

- il est possible de calculer la valeur du CRC-32 sur une séquence d'octets donnée sous forme d'une chaîne de caractère :
```
1   crc32(b'\x01\x02\x03')
```
- vous pouvez créer une trame de la manière suivante :
```
1   trame=Ether()/IP()/UDP()/b'contenu'
```

Indication : le calcul du CRC-32 dans une trame Ethernet s'applique sur les octets de la trame associés aux champs : adresse MAC destination, adresse MAC source, type et contenu.

Détection de trafic d'audit

2 – Il est possible de récupérer l'heure de capture d'un paquet dans Scapy donnée au format EPOCH (décalage en seconde par rapport au 1/1/1970) :

xterm
```
pef@ncc1701:~$ sudo scapy3
INFO: Please, report issues to https://github.com/phaethon/scapy
WARNING: IPython not available. Using standard Python shell instead.
Welcome to Scapy (3.0.0)
>>> l=sniff(count=5)
>>> l[0].time
1646949441.439771
>>>
now exiting InteractiveConsole...
pef@ncc1701:~/Sync$ date -d @1646949441.439771
Thu 10 Mar 2022 10:57:21 PM CET
```

Écrire un programme utilisant Scapy permettant de **détecter si on est la cible d'un scannage de port**, c-à-d une succession de tentative de connexion vers des ports TCP différents dans un temps très court et pour une même adresse IP source.

Vous prendrez les valeurs suivantes pour configurer le seuil de détection :

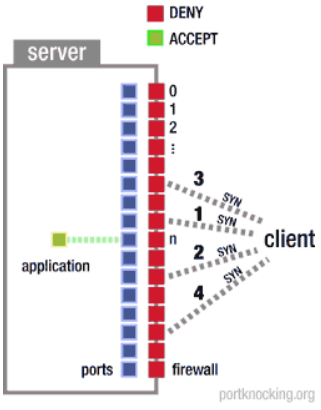
- ▷ *un intervalle de temps de 10s ;*
- ▷ *un nombre de ports d'au moins 5.*

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

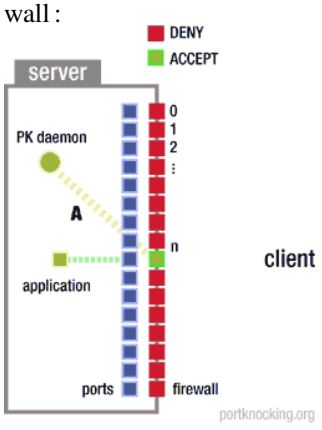
Qu'est-ce que le « Port Knocking *» ?*

Il consiste à autoriser la connexion d'un client à une application **uniquement après** que ce client ait effectué une **séquence précise de tentative de connexion** vers des **ports fermés** du serveur :

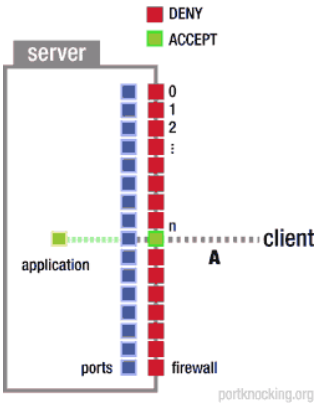
Le client effectue sa séquence de tentative de connexion :



Le serveur détecte cette séquence avec le « PK » daemon qui autorise la connexion avec le firewall :



Le client peut alors se connecter à l'application :



Vous écrirez le programme Python utilisant Scapy et réalisant :

- **la détection** de cette séquence de tentative de connexion correspondant à du « *port knocking* » :
  - la séquence de tentative de connexion TCP est la suivante : `[2027, 3230, 2001, 17377]` ;
- **l'autorisation** à l'aide de NetFilter de la connexion vers l'application (le travail du « PK daemon ») :
  - l'application attend en TCP sur le port `16400` ;
    Vous utiliserez la commande `socat` pour définir votre serveur :

```
xterm
$ socat stdio tcp-listen:16400,fork
```

  - par défaut la connexion vers l'application sera rejetée à l'aide d'un `tcp-reset` ;
  - si la séquence de « *port knocking* » est correctement reconnue la règle sera désactivée ou supprimée.

Pour réaliser le « *port knocking* », vous utiliserez la commande socat sur votre hôte en utilisant l'adresse IP de votre VM comme destination :

```
xterm
$ socat stdio tcp:192.168.x.y:2027
$ socat stdio tcp:192.168.x.y:3230
...
$  socat stdio tcp:192.168.x.y:17377
```

Vous vérifierez alors que l'accès à votre serveur attendant sur le port 16400 :

```
xterm
$ socat stdio tcp:192.168.x.y:16400
hello ?
```