Rapport Réseaux

I. Exploration des outils de reconnaissance réseau

❖ Nmap (Network Mapper) :

Cette commande lance des scripts pour identifier les vulnérabilités connues sur l'hôte

➢ **Permet de :**

- Découvrir les hôtes actifs sur un réseau.
- Identifier les ports ouverts, les services exécutés et leur version.
- Évaluer les vulnérabilités potentielles en utilisant des scripts spécifiques.

Elle utilise différentes options pour scanner des réseaux parmi elles on a :

1. **Balayage d'hôtes actifs :**

   nmap -sn 192.168.1.0/24

   - Cette commande effectue un balayage sans port (-sn) pour détecter uniquement les hôtes actifs sur le réseau.
   - Idéal pour identifier rapidement les machines allumées sans interagir directement avec leurs services.

2. **Balayage des ports :**

   nmap -p 1-65535 192.168.1.100

   - Explore tous les ports (de 1 à 65535) sur une machine spécifique pour déterminer lesquels sont ouverts.
   - Utile pour identifier les services exposés par un hôte.

3. **Identification des services et versions :**

   nmap -sV 192.168.1.100

   - Permet de détecter les versions des services exécutés sur les ports ouverts.
   - Exemple : un port 22 ouvert pourrait indiquer un service SSH fonctionnant sous OpenSSH.

4. **Balayage avec détection des OS :**

   nmap -O 192.168.1.100

   - Utilise des empreintes pour deviner le système d'exploitation de l'hôte distant.

5. **Utilisation de scripts NSE (Nmap Scripting Engine) :**

   nmap --script vuln 192.168.1.100

   - Cette commande lance des scripts pour identifier les vulnérabilités connues sur l'hôte.

❖ Netcat (nc) :

C’est un outil polyvalent pour tester la connectivité réseau et interagir directement avec des hôtes via des ports.

➢ **Elle Permet :**

- La Connexion à des hôtes distants via des ports spécifiques.
- La Configuration de Netcat pour écouter un port et attendre des connexions entrantes.
- L’Usage de Netcat pour tester la connectivité réseau ou comme outil de diagnostic.

Elle utilise différentes options parmi elles on a :

1. **Connexion à un service distant :**

   nc 192.168.1.100 80

- Permet de se connecter au port 80 (HTTP) d’un hôte distant.
- Utile pour envoyer des requêtes manuelles à des services réseau.

2. **Écoute d'un port :**

   nc -lvp 4444

   - Configure Netcat pour écouter sur le port 4444.
   - Fréquemment utilisé pour tester des connexions entrantes ou lors de transferts de fichiers.

3. **Transfert de fichiers avec Netcat :**

   - **Côté émetteur :**

     nc -lvp 4444 < fichier.txt

   - **Côté récepteur :**

     nc 192.168.1.100 4444 > fichier.txt

   - Pratique pour échanger des fichiers sans configuration complexe.

4. **Test de connectivité :**

   nc -zv 192.168.1.100 20-25

   - Vérifie si les ports 20 à 25 sont ouverts sur un hôte.
   - Le paramètre -z effectue un balayage sans ouvrir de session complète, tandis que -v (verbose) affiche les résultats en détail.

   ➢ **Resumé :**

- **Nmap** est indispensable pour découvrir des informations sur le réseau et les machines qui y sont connectées.
- **Netcat** est un couteau suisse permettant de tester rapidement la connectivité, d'envoyer des données ou d'écouter des connexions entrantes.

- En combinant les deux outils, vous pouvez cartographier un réseau, identifier des services vulnérables et interagir avec eux directement pour des tests ou des attaques simulées.

II. Analyse des paquets et des protocoles

❖ Tcpdump :

C'est un outil qui permet de capturer et analyser des paquets réseau directement depuis le terminal. Il diagnostique aussi les problèmes réseau ou observer les échanges entre machines.

➢ **Commandes explorées :**

1. **Capture basique de paquets :**

```
tcpdump
```

- Par défaut, Tcpdump capture tout le trafic réseau sur l'interface principale.
- Fournit une vue brute des paquets échangés.

2. **Filtrage par protocole :**

```
tcpdump -i eth0 icmp
```

- Capture uniquement les paquets ICMP (comme les requêtes de ping) sur l'interface eth0.
- Utile pour isoler des protocoles spécifiques.

3. **Capture par adresse IP :**

```
tcpdump host 192.168.1.100
```

- Montre uniquement les paquets échangés avec l'hôte spécifié.
- Très utile pour surveiller une machine particulière sur le réseau.

4. **Filtrage par port :**

tcpdump port 80

- Affiche uniquement le trafic échangé sur le port HTTP (80).
- Peut être combiné avec d'autres filtres comme src, dst, etc.

5. **Enregistrement dans un fichier :**

tcpdump -w capture.pcap

- Sauvegarde les paquets capturés dans un fichier .pcap pour une analyse ultérieure.
- Idéal pour travailler ensuite avec un outil graphique comme Wireshark.

6. **Lecture d'un fichier PCAP :**

tcpdump -r capture.pcap

- Permet d'afficher les paquets d'une capture précédemment sauvegardée.

❖ Wireshark :

C'est un Analyseur graphique des paquets capturés pour une exploration approfondie des protocoles et des échanges réseau. Il permet de détecter des anomalies, décoder des échanges et extraire des données spécifiques.

➢ Fonctionnalités explorées :

1. **Filtrage des protocoles :**
   - Exemple : http, tcp, icmp, ou udp dans la barre de filtres.
   - Permet de se concentrer sur un protocole spécifique pour une analyse ciblée.
2. **Analyse de flux (Stream Analyses) :**
   - Clic droit sur un paquet > **"Follow → TCP Stream"** ou **"Follow → HTTP Stream"**.
   - Affiche les échanges complets d'une session, pratique pour comprendre le contexte d'une requête ou d'une réponse.

3. **Recherche dans les paquets :**

    - Recherche de chaînes spécifiques comme par exemple : flag, password, dans le contenu des paquets.
    - Utile pour extraire des informations sensibles ou cachées.

4. **Exportation des objets :**

    - Menu : **File > Export Objects > HTTP**.
    - Permet d'extraire des fichiers ou des données échangées sur HTTP.
    - Utilisé dans un défi pour découvrir un fichier contenant un flag.

➢ **Resumé :**

- Tcpdump est un outil puissant pour capturer rapidement des données réseau en ligne de commande.

Il est léger et permet de capturer tout ou partie du trafic en fonction de critères précis.

- Wireshark complète Tcpdump avec une interface graphique conviviale pour analyser les captures en profondeur.

Ses outils d'exportation et de suivi de flux facilitent la recherche de données spécifiques dans des captures complexes.

III. Manipulation des paquets

❖ Scapy :

Scapy est un logiciel et un module pour Python permettant de forger, envoyer, réceptionner et manipuler des paquets réseau.

- **Permet :**

    - Génération et envoi de paquets Ethernet, IP, TCP/IP pour des tests et des simulations.
    - Construction de paquets ARP pour rediriger du trafic ou répondre à des requêtes spécifiques.
    - Utilisation avancée de champs comme seq, ack, et les flags TCP (SYN, ACK).

❖ Composition et manipulation des paquets avec Scapy

Comprendre les Paquets Réseau :

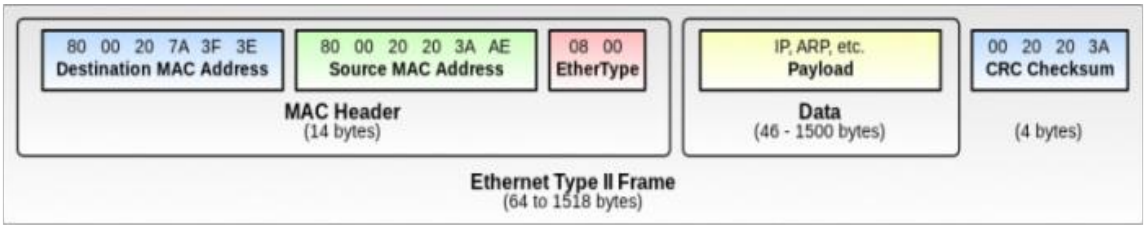
1. **Les paquets réseau** sont des unités de données formées de plusieurs couches, chacune correspondant à une étape du processus de communication :
    - **Couche lien (Ethernet)** : Permet la transmission des données sur le réseau local.
    - **Couche réseau (IP)** : Dirige les données vers leur destination en utilisant des adresses IP.
    - **Couche transport (TCP/IP)** : Assure une communication fiable et ordonnée entre les hôtes.
2. **Structure d'un paquet :** Chaque couche ajoute un en-tête contenant des informations essentielles, comme :
    - La source et la destination (MAC pour Ethernet, IP pour la couche réseau).
    - Les ports et flags (TCP pour la couche transport).

Manipulation des Paquets Ethernet

1. **Rôle et Fonctionnalité :**
    - Le protocole **Ethernet** constitue la base de toutes les communications réseau dans un LAN (Réseau local).

- Il identifie les appareils via des adresses MAC uniques.
- Permet des transmissions ciblées (unicast) ou à large diffusion (broadcast).

  Voici la structure qu'une trame Ethernet doit présenter :



  Le **CRC** permet le contrôle d'intégrité de la trame : si on le modifiait, la trame deviendrait invalide et inutile. Il ne nous reste donc que 3 champs modifiables :

  - **dst** : représente l'adresse mac du destinataire
  - **src :** représente l'adresse mac de l'émetteur
  - **Type** : représente le type de protocole (dépend du contenu de la partie "data" dans notre cas vide)

2. **Créer un paquet Ethernet :**

   - **Utilisation de Scapy** pour générer un paquet Ethernet avec une adresse de diffusion

     Code: Python

     from scapy.all import Ether

     packet = Ether(dst="ff:ff:ff:ff:ff:ff")

     sendp(packet)

     .

     Sent 1 packets.

NB : Dans le jargon Scapy, un point "." représente un envoi.

Ajout de la Couche Réseau (IP)

1. **Concept de la Couche IP :**

- Fournit un moyen de router les paquets d'un réseau à un autre via des adresses IP.
- Champs clés observés :
    - **Adresse source (src)** : IP de l'expéditeur.
    - **Adresse destination (dst)** : IP du destinataire.

2. **Créer un paquet IP avec Scapy :**
    - Ajout d'une destination spécifique au paquet :

Code: Python

```
from scapy.all import IP

ip_packet = IP(dst="192.168.1.1")

send(ip_packet.show())
```

Explication

« from scapy.all import IP »

- Cette ligne importe la classe IP du module Scapy.
- La classe IP permet de créer et de manipuler des paquets IP (Internet Protocol)
- **dst="192.168.1.1"** : Définit l'adresse IP de destination du paquet

`send(ip_packet)` :

- Enverra un paquet IP vide (sans charge utile) à l'adresse 192.168.1.1

3. **Combinaison Ethernet + IP :**
    - Superposition de couches :

Code: Python

```
from scapy.all import Ether, IP

packet = Ether(dst="ff:ff:ff:ff:ff:ff") / IP(dst="192.168.1.1")
```

sendp(packbet, iface="eth0")

Explication :

- ✓ **packet = Ether(dst="ff:ff:ff:ff:ff:ff") / IP(dst="192.168.1.1")**

Cette ligne crée une trame Ethernet contenant un paquet IP.

- ✓ **Ether(dst="ff:ff:ff:ff:ff:ff")**:

  Définit une trame Ethernet avec l'adresse MAC de destination :

  - **ff:ff:ff:ff:ff:ff** : Adresse de diffusion (broadcast).
  - La trame sera envoyée à **tous les appareils** sur le même réseau local.

- ✓ **IP(dst="192.168.1.1")** :

  - Définit un paquet IP avec , **dst="192.168.1.1"** : L'adresse IP de destination.
  - Par défaut, l'adresse IP source sera remplie automatiquement par Scapy en fonction de votre configuration réseau.

- ✓ / : slash

- En Scapy, l'opérateur / est utilisé pour encapsuler ou empiler des couches de protocole.
- Ici, il empile un paquet IP dans une trame Ethernet.

- ✓ sendp(packet, iface="eth0")

Envoie la trame via l'interface eth0

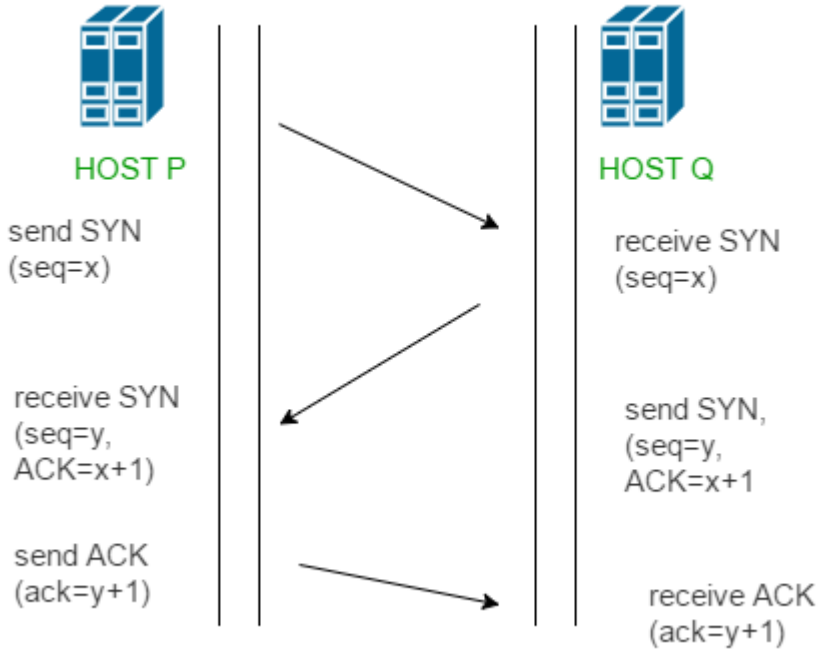
Ajout de la Couche Transport (TCP)

1. **Comprendre TCP :**

Le protocole TCP est un protocole qui garantit une transmission fiable grâce à :

- Des numéros de séquence pour suivre les données.
- Un mécanisme de vérification appelé handshake.

- Le handshake TCP suit trois étapes :
    - **SYN** : Début de la connexion.
    - **SYN-ACK** : Confirmation par le serveur.
    - **ACK** : Finalisation par le client.



2. **<u>Étape 1 (SYN)</u>** : Le client veut se connecter au serveur. Il envoie un message avec le flag **SYN** pour dire qu'il souhaite commencer une communication et indique un numéro de séquence de départ.
3. **<u>Étape 2 (SYN + ACK)</u>** : Le serveur répond avec un message **SYN-ACK**. Cela signifie qu'il a reçu la demande du client (**ACK**) et qu'il est prêt à démarrer une communication avec son propre numéro de séquence (**SYN**).
4. **<u>Étape 3 (ACK)</u>** : Le client envoie un dernier message **ACK** pour confirmer qu'il a ru la réponse du serveur. La connexion est alors établie, et ils peuvent commencer à échanger des données.

Créer un paquet TCP avec Scapy :

Voici les **flags TCP les plus utilisés** et leur rôle :

- **S (SYN)** : Utilisé pour initier une connexion TCP.
- **A (ACK)** : Confirme la réception de données ou d'autres messages.
- **F (FIN)** : Sert à terminer une connexion TCP proprement.
- **R (RST)** : Réinitialise une connexion en cas de problème ou de connexion incorrecte.
- **P (PSH)** : Demande au destinataire de traiter immédiatement les données.
- Construction d'un paquet TCP avec un drapeau(flag) SYN :

Code: Python

```
from scapy.all import TCP

tcp_packet = TCP (dport=80, flags="S")

send(tcp_packet)
```

Explication :

- Le code actuel **tente d'envoyer** un paquet TCP avec un drapeau SYN au port 80. Cependant, sans la couche IP, il est incomplet et ne pourra pas atteindre la destination.

tcp_packet = TCP(dport=80, flags="S")

- **TCP** : Crée un paquet TCP.
- **dport=80** : Spécifie le port de destination. Ici, le port 80 est utilisé, qui est le port standard pour les communications HTTP.
- **flags="S"** : Définit le drapeau TCP à SYN, qui est utilisé pour initier une connexion TCP dans le processus de handshake.

5. **Combinaison Ethernet + IP + TCP :**

- Superposition complète des couches :

Code: Python

from scapy.all import Ether, IP, TCP

full_packet = Ether(dst="ff:ff:ff:ff:ff:ff") / IP(dst="192.168.1.1") / TCP(dport=80, flags="S")

sendp(full_packet)

Explication :

Ce paquet complet encapsule :

1. Une couche Ethernet pour l'adresse MAC.
2. Une couche IP pour l'adresse réseau.
3. Une couche TCP pour établir une connexion.

Exploration Avancée (ARP et Spoofing)

1. **Protocole ARP :**

ARP associe les adresses IP aux adresses MAC dans un réseau local.

- Exemple d'un paquet ARP :

Code: Python

from scapy.all import ARP

arp_packet = ARP(op="who-has", pdst="192.168.1.1")

send(arp_packet)

Explication :

arp_packet = ARP(op="who-has", pdst="192.168.1.1")

- ARP() : Cette fonction crée un paquet ARP. Par défaut, Scapy remplit automatiquement certains champs si vous ne les spécifiez pas.
- Paramètres utilisés :
    1. op="who-has" : Spécifie l'opération ARP.
        - "who-has" est une requête pour demander l'adresse MAC associée à une adresse IP spécifique.
        - "is-at" serait une réponse ARP pour indiquer que l'adresse MAC associée est connue.
    2. pdst="192.168.1.1" : Définit l'adresse IP cible pour laquelle vous demandez l'adresse MAC.

En résumé, ce paquet demande "Qui possède l'adresse IP 192.168.1.1 ?" sur le réseau local

2. **Spoofing avec ARP :**
    - Simulation d'une fausse réponse ARP pour rediriger le trafic :

      Code: Python

      from scapy.all import ARP, Ether

      spoofed_packet=Ether(dst="ff:ff:ff:ff:ff:ff")/ ARP(op="is-at", psrc="192.168.1.1", hwsrc="00:11:22:33:44:55")

      sendp(spoofed_packet)

Explication :

Ce code met en œuvre une **attaque d'usurpation ARP (ARP spoofing)** en créant un paquet ARP falsifié. Voici une explication détaillée de chaque ligne :

“from scapy.all import ARP, Ether”

- **ARP** : Permet de créer des paquets ARP.
- **Ether** : Permet de construire une trame Ethernet, qui encapsulera le paquet ARP.
- **Création du paquet ARP falsifié**

spoofed_packet = Ether(dst="ff:ff:ff:ff:ff:ff") / ARP(op="is-at", psrc="192.168.1.1", hwsrc="00:11:22:33:44:55")

- **Ether(dst="ff:ff:ff:ff:ff:ff")** :
    - Spécifie l'adresse MAC de destination.
    - "ff:ff:ff:ff:ff:ff" est l'adresse de broadcast Ethernet, donc ce paquet sera diffusé à tous les appareils du réseau local.
- **ARP()** : Définit le contenu du paquet ARP :

1. **op="is-at"** : Indique que c'est une réponse ARP. Dans une réponse normale, un appareil dit : "Je suis à cette adresse MAC pour cette IP."

2. **psrc="192.168.1.1"** : Spécifie l'adresse IP source usurpée. Cela signifie que l'attaquant prétend être l'appareil avec l'IP **192.168.1.1**.

3. **hwsrc="00:11:22:33:44:55"** : Définit l'adresse MAC falsifiée associée à **192.168.1.1**. Ici, l'attaquant prétend que cette adresse MAC correspond à l'IP usurpée.

En résumé, ce paquet dit aux appareils du réseau local :

- "L'IP **192.168.1.1** est associée à l'adresse MAC **00:11:22:33:44:55**."
- Le paquet est diffusé sur le réseau local, et tous les appareils qui reçoivent ce paquet mettront à jour leur table ARP (cache ARP) avec cette information falsifiée.

Conséquences de l'usurpation ARP

Une fois ce paquet envoyé :

1. Les appareils du réseau croiront que l'IP **192.168.1.1** correspond à l'adresse MAC **00:11:22:33:44:55** (falsifiée).

2. Cela permet à l'attaquant de :
    - Intercepter le trafic destiné à l'IP usurpée.
    - Mener une attaque de type **Man-in-the-Middle (MITM)**.
    - Perturber les communications normales du réseau.

NB :

- **Sendp ()** : Permet d'envoyer des paquets au niveau 2 (liaison de données), comme Ethernet.
- **Send ()** : Envoie des paquets au niveau du réseau (couche 3), tels que des paquets IP, TCP, ou UDP.

Résumé des outils utilisés

- **Nmap, Netcat (nc) : Reconnaissance** réseau et diagnostic
- **Tcpdump**, **Wireshark** : Analyse et extraction de données dans des fichiers pcap.
- **Scapy** : Manipulation et génération de paquets.

Conclusion :

En explorant ces aspects, nous avons couvert des domaines clés en sécurité réseau, reconnaissance, analyse de paquets, et manipulation.

Fait par : Seydina Oumar