Exercice 1 : Scanning de Réseau avec Nmap et Python

Objectif : Utiliser Nmap avec Python pour scanner les ports ouverts sur une machine cible.

Étape 1 : Installation des Dépendances Assurez-vous que Nmap est installé sur votre système et installez la bibliothèque `python-nmap` :

```
pip install python-nmap
```

Étape 2 : Écriture du Script Créez un fichier Python, nommé `nmap_scan.py`, et ajoutez le code suivant :

```
import nmap

def simple_port_scan(target, ports):
    nm = nmap.PortScanner()
    nm.scan(target, ports)
    for host in nm.all_hosts():
        print('Host : %s (%s)' % (host, nm[host].hostname()))
        print('State : %s' % nm[host].state())
        for proto in nm[host].all_protocols():
            print('----------')
            print('Protocol : %s' % proto)
            lport = nm[host][proto].keys()
            for port in sorted(lport):
                print('port : %s\tstate : %s' % (port, nm[host][proto][port]['state']))

# Remplacez '192.168.1.1' par l'adresse IP de votre cible et spécifiez les ports à scanner
simple_port_scan('192.168.1.1', '21-443')
```

Exercice : Modifiez le script pour scanner différents types de ports, ou différentes plages de ports, et observez les résultats.

Étape 3 : Exécution du Script Exécutez le script depuis votre terminal :

```
python nmap_scan.py
```

Exercice 2 : Analyse de Paquets avec Scapy

Objectif : Utiliser Scapy pour capturer et analyser les paquets réseau.

Étape 1 : Installation de Scapy Installez Scapy via pip :

```
pip install scapy
```

Étape 2 : Écriture du Script d'Analyse de Paquets Créez un fichier Python, nommé `packet_sniffer.py`, et ajoutez le code suivant :

```
from scapy.all import sniff, IP, TCP

def packet_callback(packet):
    if IP in packet:
        ip_src = packet[IP].src
        ip_dst = packet[IP].dst
        print(f"IP Source: {ip_src} --> IP Destination: {ip_dst}")

    if TCP in packet:
        tcp_sport = packet[TCP].sport
        tcp_dport = packet[TCP].dport
        print(f"TCP Source Port: {tcp_sport} --> TCP Destination Port: {tcp_dport}")

# Sniff continuously for 10 packets.
sniff(prn=packet_callback, count=10)
```

Exercice : Modifiez le script pour filtrer différents types de trafic, comme ICMP ou UDP. Essayez d'ajouter des conditions pour capturer des paquets spécifiques basés sur des critères comme des adresses IP ou des numéros de port.

Étape 3 : Exécution du Script Lancez le script depuis votre terminal (peut nécessiter des privilèges administrateur) :

```
sudo python packet_sniffer.py
```

Exercice 3 : Scanner les Ports avec Nmap pour Détecter les Versions des Services

Objectif : Utiliser Nmap avec Python pour identifier non seulement les ports ouverts, mais aussi les versions des services qui s'exécutent sur ces ports.

Étape 1 : Préparation du Script Assurez-vous que la bibliothèque `python-nmap` est installée. Si ce n'est pas le cas, installez-la avec pip :

```
pip install python-nmap
```

Étape 2 : Écriture du Script Créez un fichier Python, `nmap_service_detection.py`, et ajoutez le code suivant :

```
import nmap

def service_version_scan(target):
    nm = nmap.PortScanner()
    nm.scan(target, arguments='-sV')  # Utilise `-sV` pour la détection de version
    for host in nm.all_hosts():
```

```
        print(f'Scanning Host: {host}')
        for proto in nm[host].all_protocols():
            lport = nm[host][proto].keys()
            for port in sorted(lport):
                service = nm[host][proto][port]
                print(f'Port : {port}/tcp')
                print(f'Service : {service["name"]}')
                print(f'Product : {service["product"]}')
                print(f'Version : {service["version"]}')
                print(f'Extra Info: {service["extrainfo"]}')

# Exemple d'utilisation
service_version_scan('192.168.1.1')
```

Exercice : Modifiez le script pour scanner différents types d'appareils ou plages d'IP dans votre réseau.

Étape 3 : Exécution du Script Lancez votre script depuis le terminal pour voir quels services et quelles versions sont exposés sur la machine cible.

Exercice 4 : Écoute et Filtrage de Paquets avec Scapy

Objectif : Utiliser Scapy pour capturer et filtrer les paquets ICMP, qui sont souvent utilisés pour le ping.

Étape 1 : Installation de Scapy Si Scapy n'est pas déjà installé :

```
pip install scapy
```

Étape 2 : Écriture du Script Créez un fichier Python, `icmp_filter_sniffer.py`, et ajoutez le code suivant :

```
from scapy.all import sniff, ICMP

def icmp_packet_callback(packet):
    if ICMP in packet:
        print(f"ICMP Packet: {packet.summary()}")

# Filtrer pour écouter uniquement les paquets ICMP
sniff(filter='icmp', prn=icmp_packet_callback, count=10)
```

Exercice : Adaptez le script pour écouter d'autres types de paquets, comme ceux utilisés pour les protocoles TCP ou UDP spécifiques.

Étape 3 : Exécution du Script Exécutez le script dans un terminal, idéalement avec des droits administrateur pour permettre à Scapy d'intercepter les paquets :

```
sudo python icmp_filter_sniffer.py
```

Pour continuer à développer vos compétences en Python appliquées à la sécurité réseau avec des exercices plus avancés, je vous propose de réaliser un exercice qui combine l'analyse de paquets avec des actions conditionnelles et une interaction plus dynamique avec le réseau. Cet exercice intégrera Scapy pour créer et analyser des paquets personnalisés, ainsi que pour répondre automatiquement à certaines conditions réseau.

Exercice 5 : Réponse Automatisée à des Événements Réseau avec Scapy

Objectif : Utiliser Scapy pour surveiller le trafic réseau spécifique et répondre automatiquement à des conditions pré-définies, comme répondre à des pings ou à des requêtes spécifiques.

Étape 1 : Installation de Scapy Si Scapy n'est pas déjà installé, installez-le avec pip :

```
pip install scapy
```

Étape 2 : Écriture du Script Créez un fichier Python, `auto_response_sniffer.py`, pour surveiller et répondre automatiquement au trafic ICMP (ping requests). Le script interceptera les paquets ICMP et enverra une réponse personnalisée.

```
from scapy.all import *
import time

def auto_respond(packet):
    if packet.haslayer(ICMP):
        if packet[ICMP].type == 8:  # Echo request
            print(f"Received ICMP request from {packet[IP].src}")
            # Préparer une réponse ICMP
            icmp_reply = IP(dst=packet[IP].src) / ICMP(type=0, id=packet[ICMP].id, seq=packe
            send(icmp_reply, verbose=0)
            print(f"Sent ICMP reply to {packet[IP].src}")

# Sniffer qui filtre les paquets ICMP et utilise la fonction auto_respond pour chaque paque
sniff(filter='icmp', prn=auto_respond)
```

Étape 3 : Fonctionnement du Script

1. **Interception de paquets ICMP** : Le script écoute tous les paquets ICMP (utilisés notamment pour les commandes `ping`).
2. **Réponse aux requêtes ICMP** : Lorsqu'un paquet ICMP de type "echo request" (ping request) est détecté, le script génère une réponse "echo reply" (ping reply) et l'envoie au demandeur.

Exercice 6 : Détection et Réponse à des Paquets SYN pour Simuler un Half-open Scan

Un autre scénario intéressant serait de simuler une réponse à un scan SYN (utilisé parfois pour identifier des ports ouverts sans établir de connexion complète).

Étape 1 : Script Python pour Simuler une Réponse SYN

```
from scapy.all import *

def handle_syn_packet(packet):
    if packet.haslayer(TCP) and packet[TCP].flags == 'S':  # SYN flag
        print(f"Received SYN on port {packet[TCP].dport} from {packet[IP].src}")
        # Créer un paquet SYN-ACK
        syn_ack_pkt = IP(dst=packet[IP].src) / TCP(dport=packet[TCP].sport, sport=packet[TCP
        send(syn_ack_pkt, verbose=0)
        print(f"Sent SYN-ACK to {packet[IP].src}")

# Filtrer les paquets TCP entrants et traiter les paquets SYN
sniff(filter="tcp", prn=handle_syn_packet)
```