Rapport : Mini Système de Détection d'Intrusion (IDS) avec Scapy

1. Objectif du projet

L'objectif de ce mini-projet est de mettre en place un **IDS simple** (Intrusion Detection System) capable d'observer et d'identifier :

- des paquets **ICMP** (ex : ping),
- des paquets **TCP SYN**, typiques des **scans de ports** réalisés par des attaquants (ex : Nmap).

Le script développé avec **Python + Scapy** doit détecter ces paquets en temps réel et afficher des logs permettant d'analyser le trafic.

2. Environnement technique

- **Ubuntu 22.04** (WSL)
- Python 3 + Scapy
- Interface écoutée : `lo` (loopback)

Commandes installées et utilisées :

- `pip install scapy`

- `ping` pour générer des ICMP
- `sudo nmap -sS 127.0.0.1` pour générer des SYN

3. Script IDS utilisé

Le script suivant a été exécuté :

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

# Compteurs globaux
icmp_count = 0
syn_count = 0

# Variables pour détecter ICMP flood
icmp_count_sec = 0
start_time = time.time()


def analyse_packet(pkt):
    global icmp_count, syn_count, icmp_count_sec, start_time

    if pkt.haslayer(ICMP):
        icmp_count += 1
        icmp_count_sec += 1
        print(f"[ICMP] {pkt[IP].src} -> {pkt[IP].dst}")

    # Détection flood : plus de 50 paquets ICMP en 1 seconde
    if time.time() - start_time >= 1:
        if icmp_count_sec > 50:
            print(f"[ALERTE] ICMP Flood détecté : {icmp_count_sec}
paquets/seconde")
        icmp_count_sec = 0
        start_time = time.time()


    if pkt.haslayer(TCP) and pkt[TCP].flags == "S":
        syn_count += 1
        dport = pkt[TCP].dport
        print(f"[SYN] {pkt[IP].src} -> {pkt[IP].dst} port {dport}")


    if pkt.haslayer(TCP):
        flags = pkt[TCP].flags
```

```
        # Null Scan : aucun flag
        if flags == 0:
            print("[NULL SCAN] Paquet TCP sans flags détecté")

        # FIN scan
        if flags == "F":
            print("[FIN SCAN] Paquet FIN détecté")

        # XMAS scan : FIN + PSH + URG
        if "F" in flags and "P" in flags and "U" in flags:
            print("[XMAS SCAN] Paquet XMAS détecté")


print("Sniffing en cours sur l'interface loopback (lo)... (CTRL+C
pour arrêter)")

sniff(
    iface="lo",
    filter="icmp or tcp",
    prn=analyse_packet,
    store=False
)
```

4. Tests réalisés

4.1. Test ICMP (Ping)

Commande utilisée :

```
narm@pcn:~/projets/mini-projets/projet_ids$ for i in {1..5}; do ping -c1 127.0.0.1 >
/dev/null; done
```

Résultat observé dans l'IDS :

```
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
```

Conclusion :
Le script détecte correctement les paquets ICMP envoyés en boucle.

4.2. Test ICMP Flood (attaque volumétrique)

Commande utilisée :

```
for i in {1..500}; do ping -c1 127.0.0.1 > /dev/null; done
```

Résultat observé dans l'IDS :

```
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ICMP] 127.0.0.1 -> 127.0.0.1
[ALERTE] ICMP Flood détecté : 898 paquets/seconde
```

Conclusion :
Le système identifie correctement une anomalie basée sur un volume excessif de paquets ICMP par seconde. Cela simule une attaque de type **ICMP Flood / DoS**.

4.3. Test d’un scan Nmap (TCP SYN)

Commande utilisée :

```
sudo nmap -sS 127.0.0.1
```

```
/projet_ids$ sudo nmap -sS 127.0.0.1
```

Résultat observé :

```
[SYN] 127.0.0.1 -> 127.0.0.1 port 23
[SYN] 127.0.0.1 -> 127.0.0.1 port 23
[SYN] 127.0.0.1 -> 127.0.0.1 port 111
[SYN] 127.0.0.1 -> 127.0.0.1 port 111
[SYN] 127.0.0.1 -> 127.0.0.1 port 587
[SYN] 127.0.0.1 -> 127.0.0.1 port 587
[SYN] 127.0.0.1 -> 127.0.0.1 port 22
[SYN] 127.0.0.1 -> 127.0.0.1 port 22
[SYN] 127.0.0.1 -> 127.0.0.1 port 443
[SYN] 127.0.0.1 -> 127.0.0.1 port 443
[SYN] 127.0.0.1 -> 127.0.0.1 port 1025
[SYN] 127.0.0.1 -> 127.0.0.1 port 1025
```

Les messages affichent un **grand nombre de ports testés** (153 ports testés), ce qui correspond parfaitement au comportement d’un scan SYN.

Conclusion :
L’IDS détecte les tentatives de découverte de ports, ce qui simule un comportement d’attaquant.

4.4. Test SYN Scan (scan de ports TCP)

Commande utilisée :

```
sudo nmap -sS 127.0.0.1
```

Résultat observé dans l'IDS :

```
[SYN] 127.0.0.1 -> 127.0.0.1 port 60020
[SYN] 127.0.0.1 -> 127.0.0.1 port 2809
[SYN] 127.0.0.1 -> 127.0.0.1 port 9502
[SYN] 127.0.0.1 -> 127.0.0.1 port 9502
[SYN] 127.0.0.1 -> 127.0.0.1 port 9917
[SYN] 127.0.0.1 -> 127.0.0.1 port 42
[SYN] 127.0.0.1 -> 127.0.0.1 port 6123
[SYN] 127.0.0.1 -> 127.0.0.1 port 8994
```

Conclusion :
Le script identifie les paquets TCP avec flag SYN, typiques d'un scan de ports furtif. Cela valide la capacité à repérer une reconnaissance réseau.

4.5. Test Null Scan

Commande utilisée :

```
sudo nmap -sN 127.0.0.1
```

Résultat observé :

```
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
```

Conclusion :
Le script détecte les paquets TCP ne contenant aucun flag, une technique utilisée pour contourner certains pare-feux.

4.6. Test FIN Scan

Commande utilisée :

```
sudo nmap -sF 127.0.0.1
```

Résultat observé :

```
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
[NULL SCAN] Paquet TCP sans flags détecté
```

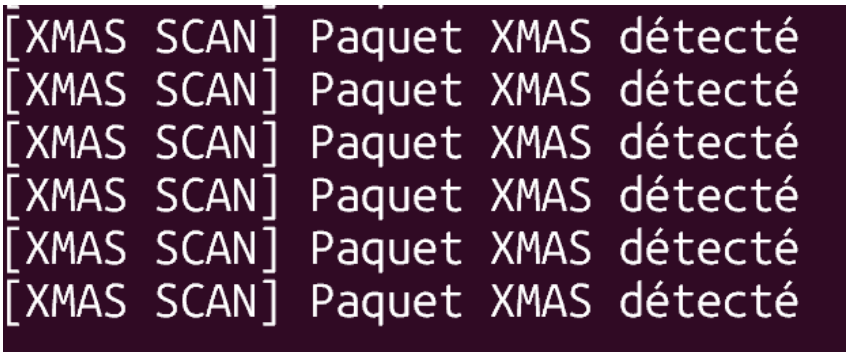
Conclusion :
La détection du flag F fonctionne, révélant la présence d'un scan furtif alternatif utilisé par les attaquants.

4.7. Test XMAS Scan

Commande utilisée :

```
sudo nmap -sX 127.0.0.1
```

Résultat observé :



Conclusion :
Le script identifie correctement les paquets dont les flags **FIN + PSH + URG** sont activés simultanément.
C'est une signature claire de l'attaque **XMAS Scan**, utilisée pour contourner certaines règles de filtrage.

5. Analyse générale des résultats

5.1. Détection ICMP (Ping et Flood)

Le script a bien identifié les paquets ICMP classiques générés via la commande ping. Lors du test d'**ICMP Flood**, une alerte a été déclenchée dès que le nombre de paquets ICMP dépassait 50 par seconde.
Ce comportement montre que le système est capable de détecter des attaques par **déni de service de type ICMP Flood** sur la base d'un seuil simple.

5.2. Détection des scans TCP (SYN, NULL, FIN, XMAS)

Les différents types de scans de ports exécutés avec **Nmap** (SYN, NULL, FIN, XMAS) ont tous été correctement détectés par le script :

- Les **paquets TCP avec flag SYN** ont été repérés, indiquant un **SYN Scan**.
- Les **paquets sans aucun flag** ont été classifiés comme **NULL Scan**.
- Les paquets avec uniquement le flag **FIN** ont été identifiés comme **FIN Scan**.
- Les paquets comportant simultanément les flags **FIN + PSH + URG** ont été reconnus comme **XMAS Scan**.

Chaque type de scan a généré une sortie spécifique dans le terminal, démontrant la capacité du script à différencier plusieurs techniques furtives d'exploration réseau.

5.3. Interprétation globale

L'IDS minimal développé avec **Scapy** a rempli sa mission :
il détecte en **temps réel** les paquets associés à des comportements d'analyse réseau ou de flood, et les affiche sous forme de log en console.

Malgré sa simplicité, ce script illustre bien les **fondamentaux d'un IDS** :

- **Écoute** sur une interface réseau (ici `lo`)
- **Filtrage** de certains types de paquets (ICMP, TCP)
- **Identification** d'anomalies courantes (flood, reconnaissance)
- **Classification** basique des types d'attaque

6. Limites du projet

Malgré son efficacité dans un cadre pédagogique, ce mini-IDS présente plusieurs limites :

- **Pas de journalisation** : les alertes sont seulement affichées dans le terminal, sans être enregistrées dans un fichier pour analyse ultérieure.
- **Pas de classification avancée** : les détections sont basées sur des conditions simples (flags ou seuils bruts), sans corrélation de comportement ou historique.
- **Analyse réseau limitée** : seuls les paquets **ICMP** et **TCP SYN** sont pris en compte, sans prise en charge d'autres protocoles ou flux applicatifs.
- **Environnement restreint** : le test a été réalisé uniquement sur **l'interface loopback (`lo`)**, ce qui ne reflète pas un réseau réel d'entreprise.

7. Améliorations possibles

Plusieurs pistes peuvent être explorées pour rendre cet IDS plus robuste :

- **Ajouter un fichier de log** (`ids.log`) pour historiser les détections.
- **Étendre la détection à d'autres scans TCP** : FIN, XMAS, NULL…
- **Créer une interface de visualisation** (avec `matplotlib`, `streamlit` ou `tkinter`) pour suivre en temps réel l'activité réseau.
- **Déclencher des alertes actives** (email, son, ou message dans une interface) en cas d'attaque détectée.
- **Écouter une interface réseau réelle** (`eth0`, `wlan0`, etc.) pour une surveillance sur réseau local.

8. Conclusion

Ce mini-projet a permis de :

- Développer un script IDS fonctionnel en Python avec Scapy.

- Simuler plusieurs types de comportements malveillants courants (ping flood, scan SYN, NULL, FIN, XMAS).

- Comprendre les signatures basiques d'attaque et comment les détecter à bas niveau.

Même s'il s'agit d'un projet simple, cette approche fournit une base solide pour s'initier à la surveillance réseau, à la cybersécurité défensive, et à la création d'outils de détection personnalisés.