Rapport de Mini-Projet : Honeypot & Analyse d'Événements

1. Introduction

Dans le cadre d'un exercice pratique d'observation de comportements réseau suspects, un mini-honeypot a été déployé sur une machine virtuelle Ubuntu.
L'objectif est simple : **exposer un faux service** et analyser les connexions qui tentent d'interagir avec lui.

Ce projet vise à comprendre :

- comment un service accessible peut attirer des connexions,
- comment les logs se construisent,
- quelles actions typiques reproduisent un attaquant ou un bot,
- comment interpréter des tentatives d'accès.

Environnement utilisé :

- VM Ubuntu 20.04 / 22.04
- Python 3
- Netcat et Nmap pour générer des événements
- Script Python simule un faux service SSH (port 2222)

2. Mise en place du Honeypot

2.1. Description du Honeypot

Un honeypot minimaliste a été développé en Python.
Il écoute sur le port **2222** et enregistre toutes les connexions entrantes ainsi que les messages envoyés par les clients.

Il simule un service SSH fictif, ce qui permet d'observer :

- les scans de ports,
- les tentatives de connexion,
- les commandes envoyées par un potentiel attaquant.

2.2. Architecture

```
Attaquant (scan / connexion)
        ↓
  Honeypot TCP (port 2222)
        ↓
       Logs → analyse
```

2.3. Étapes d'installation

1. Création du script :

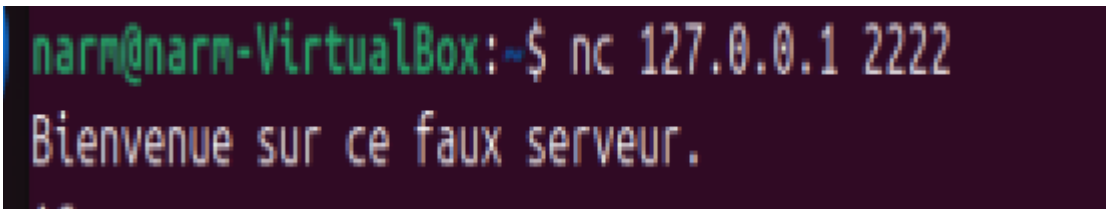
```
nano mini_honeypot.py
```

2. Script Python lancé avec :

```
python3 mini_honeypot.py
```

3. Connexions générées :

```
nc 127.0.0.1 2222
```



4. Enregistrement automatique dans `honeypot.log`

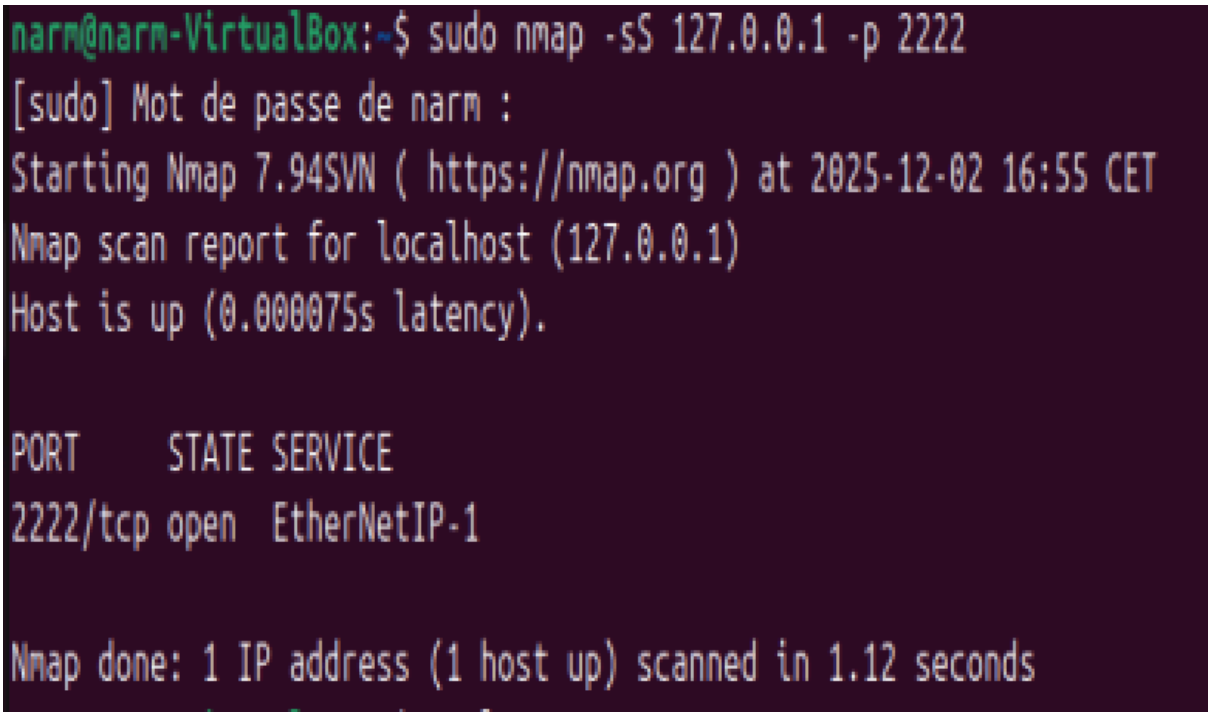
3. Génération des événements

Durant l'expérience, plusieurs scénarios de test ont été simulés afin d'obtenir des logs réalistes.

3.1. Scan réseau

Commande utilisée :

```
sudo nmap -sS 127.0.0.1 -p 2222
```



Résultat observé :

- Le port **2222** apparaît comme **ouvert**, ce qui reproduit la phase de reconnaissance d'un cyberattaquant.

3.2. Tentatives de connexion

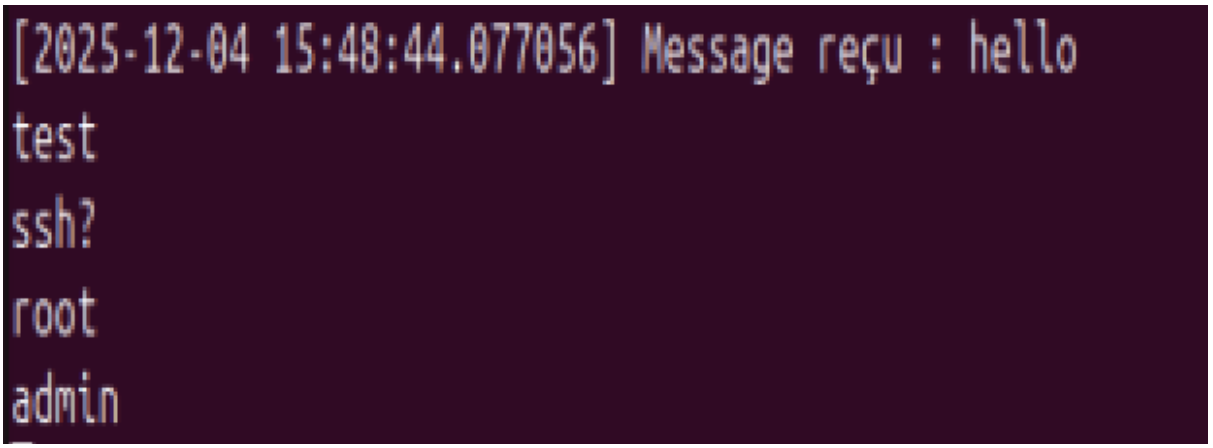
Connexion manuelle :

```
nc 127.0.0.1 2222
```

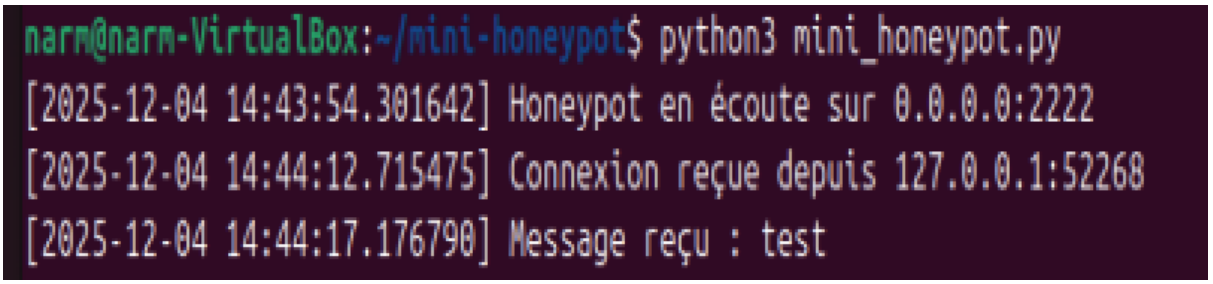
Messages envoyés :

- **“test”**
- **hello**
- **test**
- **ssh?**
- **root**
- **admin**

Terminal émetteur



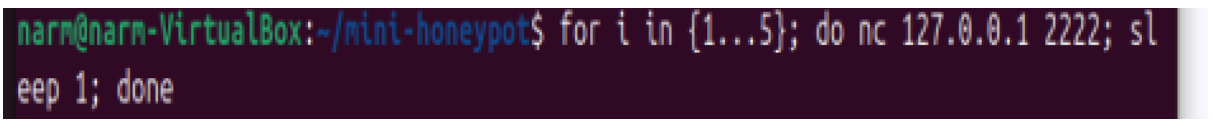
Terminal recepteur



3.3. Connexions automatisé

Script pour générer plusieurs tentatives successives :

```
for i in {1..10}; do nc 127.0.0.1 2222; sleep 1; done
```



Comportement simulé :

- attaques automatisées,
- scans agressifs,
- bots essayant plusieurs connexions.

4. Analyse des logs

Les logs ont été générés dans le fichier `honeypot.log`.

4.1. Volume total

- Nombre total de connexions : **9**
- Période observée : **3h40**

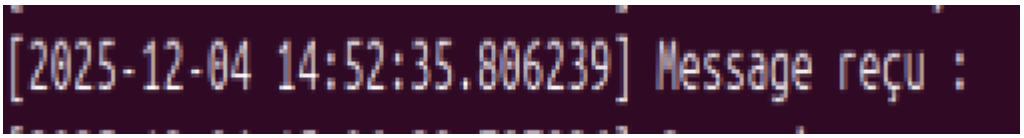
4.2. IP sources

- Dans ce test local, les connexions proviennent de `127.0.0.1`
- Dans un vrai déploiement, tu aurais plusieurs IP globales (botnets, scanners…).

4.3. Contenu observé

Exemples à compléter :

- Tentatives de connexion sans message



- Répétitions rapides → comportement typique d'outils automatiques

```
[2025-12-04 22:45:41.737115] Connexion reçue depuis 127.0.0.1:53998
[2025-12-04 22:45:41.755205] Message reçu : ping1
[2025-12-04 22:45:41.980038] Connexion reçue depuis 127.0.0.1:54004
[2025-12-04 22:45:41.980796] Message reçu : ping2
[2025-12-04 22:45:42.242360] Connexion reçue depuis 127.0.0.1:54014
[2025-12-04 22:45:42.242716] Message reçu : ping3
[2025-12-04 22:45:42.456459] Connexion reçue depuis 127.0.0.1:54024
[2025-12-04 22:45:42.456697] Message reçu : ping4
[2025-12-04 22:45:42.671028] Connexion reçue depuis 127.0.0.1:54032
[2025-12-04 22:45:42.671498] Message reçu : ping5
```

- Détections de scans SYN (Nmap)

```
[2025-12-04 15:48:38.280669] Connexion reçue depuis 127.0.0.1:54336
```

4.4. Interprétation

Les comportements observés correspondent à :

- une **phase de reconnaissance** (scan de port)
- des **connexions répétées**, typiques d'un script ou d'un bot
- des **tentatives d'interaction** avec un port ressemblant à un SSH

Ce sont des patterns classiques dans la chaîne d'attaque MITRE ATT&CK ("Discovery").

5. Risques et enseignements

Ce qu'un attaquant réel pourrait faire :

- Scanner la machine à la recherche de services vulnérables

- Tester des mots de passe ou des commandes
- Exploiter une vulnérabilité SSH
- Installer un malware ou établir une porte dérobée
- Effectuer un mouvement latéral (pivot) dans le réseau

Ce que montre ce mini-projet :

- Tout service exposé attire du trafic suspect
- Même un faux service reçoit des connexions automatiquement
- L'importance des logs pour comprendre les comportements réseau
- La facilité avec laquelle un attaquant peut détecter un port ouvert

7. Conclusion

Ce mini-projet a permis de :

- comprendre le fonctionnement d'un honeypot
- observer des connexions suspectes
- analyser un fichier de logs
- se familiariser avec la détection et la collecte d'événements réseau

Il démontre qu'un simple port ouvert suffit pour attirer des interactions, et illustre l'importance de la surveillance réseau.

Annexe A : Code du honeypot

```
import socket
from datetime import datetime

HOST = "0.0.0.0"
PORT = 2222
LOGFILE = "honeypot.log"
```

```
def log(message: str):
    """Écrit un message dans le fichier de log et sur l'écran."""
    line = f"[{datetime.now()}] {message}"
    print(line)
    with open(LOGFILE, "a") as f:
        f.write(line + "\n")

def main():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind((HOST, PORT))
        s.listen()
        log(f"Honeypot en écoute sur {HOST}:{PORT}")

        while True:
            conn, addr = s.accept()
            ip, remote_port = addr
            log(f"Connexion reçue depuis {ip}:{remote_port}")

            try:
                data = conn.recv(1024)
                if data:
                    try:
                        text = data.decode(errors="ignore").strip()
                    except:
                        text = str(data)
                    log(f"Message reçu : {text}")
                # on répond quelque chose au client
                conn.sendall(b"Bienvenue sur ce faux serveur.\n")
            except Exception as e:
                log(f"Erreur avec {ip} : {e}")
            finally:
                conn.close()

if __name__ == "__main__":
    main()
```