TP 1: Automatisation du paramétrage du filtrage par des iptables sur un serveur

04/02/2025

I)

On commence par exécuter la commande :
« sudo apt install python3-paramiko »."

Puis, on vérifie s'il est bien installé la commande suivante :
« python3 -c "import paramiko; print(paramiko.__version__)" »

```
root@SRV2:~# python3 -c "import paramiko; print(paramiko.__version__)"
2.12.0
```

II)

Ensuite, je crée un fichier nommé « iptables_rules.txt », contenant toutes les règles iptables. :

root@mehdi:/home/mehdi/tp# cat iptables_rules.txt

```
# Autorise localhost
iptables -A INPUT -i lo -j ACCEPT
iptables -A OUTPUT -o lo -j ACCEPT

# Autorise SSH
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT

# On bloque tout par défaut
iptables -P INPUT DROP
iptables -P OUTPUT DROP
iptables -P FORWARD DROP

# Autorise HTTP port 80
iptables -A INPUT -p tcp --dport 80 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 80 -j ACCEPT

# Autorise HTTPS port 443
iptables -A INPUT -p tcp --dport 443 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 443 -j ACCEPT

# Autorise ping
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT
```

III)

Puis, je crée le script qui permet l'automatisation et que je nomme « ssh.py »:

```
import paramiko

# Configuration du serveur
SERVER_IP = "127.0.0.1"
USERNAME = "root"
PASSWORD = "12345"
RULES_FILE = "iptables_rules.txt"
LOG_FILE = "iptables_log.txt"

# Connexion SSH au serveur
def connect_ssh():
    client = paramiko.SSHClient()
    client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    client.connect(SERVER_IP, username=USERNAME, password=PASSWORD)
    return client

# Lecture des règles iptables
def read_rules(file_path):
    with open(file_path, "r") as file:
        return [line.strip() for line in file if line.strip() and not line.startswith("#")]

# Exécution des règles sur le serveur
def apply_iptables_rules(client, rules):
    with open(LOG_FILE, "w") as log:
        for rule in rules:
            stdin, stdout, stderr = client.exec_command(f"sudo {rule}")
            exit_status = stdout.channel.recv_exit_status()
            log.write(f"Commande: {rule}\nCode retour: {exit_status}\n\n")
            print(f"{rule} -> Code retour: {exit_status}")

def main():
    client = connect_ssh()
    rules = read_rules(RULES_FILE)
    apply_iptables_rules(client, rules)
    client.close()

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

IV)

Ensuite, je peux modifier le fichier « /etc/ssh/sshd_config » et configurer l'accès à SSH pour l'utilisateur root grâce à l'option « PermitRootLogin yes» :

```
Port 22
ListenAddress 0.0.0.0
PermitRootLogin yes
PasswordAuthentication yes
PubkeyAuthentication yes
```

V)

Ensuite je vérifie que le sevice ssh est bien active :

```
root@mehdi:/home/mehdi/tp# systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
     Loaded: loaded (/lib/systemd/system/ssh.service; enabled; preset: enabled)
     Active: active (running) since Tue 2025-02-04 15:42:23 CET; 1h 9min ago
       Docs: man:sshd(8)
             man:sshd_config(5)
    Process: 537 ExecStartPre=/usr/sbin/sshd -t (code=exited, status=0/SUCCESS)
   Main PID: 575 (sshd)
      Tasks: 1 (limit: 2283)
     Memory: 1.7M
        CPU: 100ms
     CGroup: /system.slice/ssh.service
             └─575 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

févr. 04 15:42:23 mehdi systemd[1]: Starting ssh.service - OpenBSD Secure Shell
févr. 04 15:42:23 mehdi sshd[575]: Server listening on 0.0.0.0 port 22.
févr. 04 15:42:23 mehdi systemd[1]: Started ssh.service - OpenBSD Secure Shell
févr. 04 15:44:01 mehdi sshd[2749]: Accepted publickey for root from 127.0.0.1
févr. 04 15:44:01 mehdi sshd[2749]: pam_unix(sshd:session): session opened for
févr. 04 15:44:01 mehdi sshd[2749]: pam_env(sshd:session): deprecated reading o
lines 1-19/19 (END)
```

VI)

Je peux désormais exécuter le script et voir son bon fonctionnement avec le code retour qui est 0 (opérationnel).

```
root@mehdi:/home/mehdi/tp# python3 ssh.py
iptables -A INPUT -i lo -j ACCEPT -> Code retour: 0
iptables -A OUTPUT -o lo -j ACCEPT -> Code retour: 0
iptables -A INPUT -p tcp --dport 22 -j ACCEPT -> Code retour: 0
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT -> Code retour: 0
iptables -P INPUT DROP -> Code retour: 0
iptables -P OUTPUT DROP -> Code retour: 0
iptables -P FORWARD DROP -> Code retour: 0
iptables -A INPUT -p tcp --dport 80 -j ACCEPT -> Code retour: 0
iptables -A OUTPUT -p tcp --sport 80 -j ACCEPT -> Code retour: 0
iptables -A INPUT -p tcp --dport 443 -j ACCEPT -> Code retour: 0
iptables -A OUTPUT -p tcp --sport 443 -j ACCEPT -> Code retour: 0
iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT -> Code retour: 0
iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT -> Code retour: 0
```

TP 2: Gestion utilisateurs et groupes

I)

Je crée un script nommé « gestion_user_group.py », qui me permet d'automatiser l'administration de mon serveur :

```
import subprocess
import os

# Fonction pour exécuter une commande shell
def run_command(command):
    try:
        subprocess.run(command, shell=True, check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        print(f"[OK] Commande exécutée : {command}")
    except subprocess.CalledProcessError as e:
        print(f"[ERREUR] Commande : {command}\nErreur : {e.stderr.decode()}")

# Fonction pour vérifier si un groupe existe
def group_exists(group_name):
    try:
        result = subprocess.run(f"getent group {group_name}", shell=True, stdout=subprocess.PIPE)
        return result.stdout.decode().strip() != ""
    except Exception:
        return False

# Fonction pour vérifier si un utilisateur existe
def user_exists(username):
    try:
        result = subprocess.run(f"getent passwd {username}", shell=True, stdout=subprocess.PIPE)
        return result.stdout.decode().strip() != ""
    except Exception:
        return False

# Fonction principale pour ajouter un utilisateur
def add_user():
    username = input("Entrez le nom de l'utilisateur : ").strip()
    password = input("Entrez le mot de passe : ").strip()
    group = input("Entrez le groupe : ").strip()

    # Vérifier si le groupe existe, sinon le créer
    if not group_exists(group):
        print(f"Le groupe '{group}' n'existe pas. Création...")
        run_command(f"sudo groupadd {group}")

    # Vérifier si l'utilisateur existe déjà
    if user_exists(username):
        print(f"L'utilisateur '{username}' existe déjà !")
        return

    # Ajouter l'utilisateur
    home_dir = f"/home/{username}"
    shell = "/bin/bash"

    print(f"Création de l'utilisateur '{username}' avec le groupe '{group}'...")
    run_command(f"sudo useradd -m -d {home_dir} -s {shell} -g {group} {username}")

    # Définir le mot de passe
    run_command(f"echo '{username}:{password}' | sudo chpasswd")

    # Enregistrer les informations dans un fichier
    with open("users_created.txt", "a") as file:
        file.write(f"{username}:{group}\n")

    print(f"✅ Utilisateur '{username}' créé avec succès et enregistré dans 'users_created.txt'.")

    # Ajouter des droits supplémentaires à l'utilisateur
    add_permissions(username)

# Fonction pour ajouter des droits à l'utilisateur
def add_permissions(username):
    print("\n🔧 Configuration des permissions supplémentaires")
    print("1. Ajouter l'utilisateur au groupe sudo (permissions administrateur)")
    print("2. Donner un accès en lecture/écriture à un dossier spécifique")
    print("3. Ne pas attribuer de permissions supplémentaires")
    choice = input("Choisissez une option (1, 2 ou 3) : ").strip()

    if choice == "1":
        print(f"Ajout de '{username}' au groupe sudo...")
        run_command(f"sudo usermod -aG sudo {username}")
        print(f"✅ L'utilisateur '{username}' a maintenant les privilèges administrateur.")

    elif choice == "2":
        directory = "/home/partage/"
        if os.path.exists(directory):
            print(f"Ajout des droits d'accès en lecture/écriture sur {directory}...")
            run_command(f"sudo chown {username}:{username} {directory}")
            run_command(f"sudo chmod -R 770 {directory}")
            print(f"✅ L'utilisateur '{username}' peut maintenant accéder et modifier les fichiers dans '{directory}'.")
        else:
            print(f"❌ Le dossier '{directory}' n'existe pas. Permission non appliquée.")

    elif choice == "3":
```

```
        print("⏭️ Aucune permission supplémentaire ajoutée.")

    else:
        print("❌ Option invalide. Aucune permission ajoutée.")

# Exécution du script
if __name__ == "__main__":
    print("\n🚀 Ajout d'un utilisateur Linux\n")
    add_user()
```

II)

Je peux exécuter le script et observer son exécution :

```
root@mehdi:/home/mehdi/tp# python3 gestion_user_group.py

🚀 Ajout d'un utilisateur Linux

Entrez le nom de l'utilisateur : testuser
Entrez le mot de passe : testuser
Entrez le groupe : testuser
Le groupe 'testuser' n'existe pas. Création...
[OK] Commande exécutée : sudo groupadd testuser
Création de l'utilisateur 'testuser' avec le groupe 'testuser'...
[OK] Commande exécutée : sudo useradd -m -d /home/testuser -s /bin/bash -g testuser testuser
[OK] Commande exécutée : echo 'testuser:testuser' | sudo chpasswd
✅ Utilisateur 'testuser' créé avec succès et enregistré dans 'users_created.txt'.

🔒 Configuration des permissions supplémentaires
1. Ajouter l'utilisateur au groupe sudo (permissions administrateur)
2. Donner un accès en lecture/écriture à un dossier spécifique
3. Ne pas attribuer de permissions supplémentaires
Choisissez une option (1, 2 ou 3) : 3
⏭️Aucune permission supplémentaire ajoutée.
root@mehdi:/home/mehdi/tp#
```

Je peux aussi consulter les utilisateurs que mon script a créés dans le fichier nommé : « users_created.txt » :

```
root@mehdi:/home/mehdi/tp# cat users_created.txt
coucou:coucou
tes:test
robert:robert
john:john
testuser:testuser
root@mehdi:/home/mehdi/tp#
```

TP 3: Conception d'un script menu permettant l'exécution de plusieurs scripts automatisés

I)

Dans le répertoire où j'ai créé tous mes fichiers, je vais créer un script menu.py qui me permettra de choisir quel script je veux exécuter.

```
root@mehdi:/home/mehdi/tp# ls
gestion_user_group.py  iptables_rules.txt  ssh.py
iptables_log.txt       menu.py             users_created.txt
```

II)

Voici le code du script « menu.py »

root@mehdi:/home/mehdi/tp# cat menu.py

```
import subprocess
import os
import time

# Fonction pour exécuter une commande shell
def run_command(command):
    try:
        subprocess.run(command, shell=True, check=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)
        print(f"[OK] Commande exécutée : {command}")
    except subprocess.CalledProcessError as e:
        print(f"[ERREUR] Commande : {command}\nErreur : {e.stderr.decode()}")

# Fonction pour exécuter le script de gestion des utilisateurs et des groupes
def gestion_utilisateurs():
    print("\n🚀 Lancement du script de gestion des utilisateurs et des groupes...\n")
    os.system("python3 gestion_user_group.py")  # Remplace par le bon chemin si nécessaire

# Fonction pour exécuter le script de gestion des iptables
def gestion_iptables():
    print("\n🔥 Lancement du script de configuration des iptables...\n")
    os.system("python3 ssh.py")  # Remplace par le bon chemin si nécessaire

# Fonction du menu principal
def menu():
    while True:
        print("\n📌 Menu Principal - Sélectionnez une option")
        print("1. 🔧 Gérer les utilisateurs et groupes")
        print("2. 🔥 Configurer iptables")
        print("3. ❌ Quitter")

        choix = input("\nEntrez votre choix (1, 2 ou 3) : ").strip()

        if choix == "1":
            gestion_utilisateurs()
        elif choix == "2":
            gestion_iptables()
        elif choix == "3":
            print("\n👋 Fermeture du programme... À bientôt !")
            break
        else:
            print("\n❌ Choix invalide. Veuillez entrer 1, 2 ou 3.")
        time.sleep(2)  # Petite pause avant de revenir au menu

# Lancer le menu
if __name__ == "__main__":
    menu()
```

III)

Désormais on peut exécuter le script « menu.py » :