TryHackMe Common Linux Privesc

Saikat Karmakar | Aug 16 : 2021

What does "privilege escalation" mean?

At it's core, Privilege Escalation usually involves going from a lower permission to a higher permission. More technically, it's the exploitation of a vulnerability, design flaw or configuration oversight in an operating system or application to gain unauthorized access to resources that are usually restricted from the users.

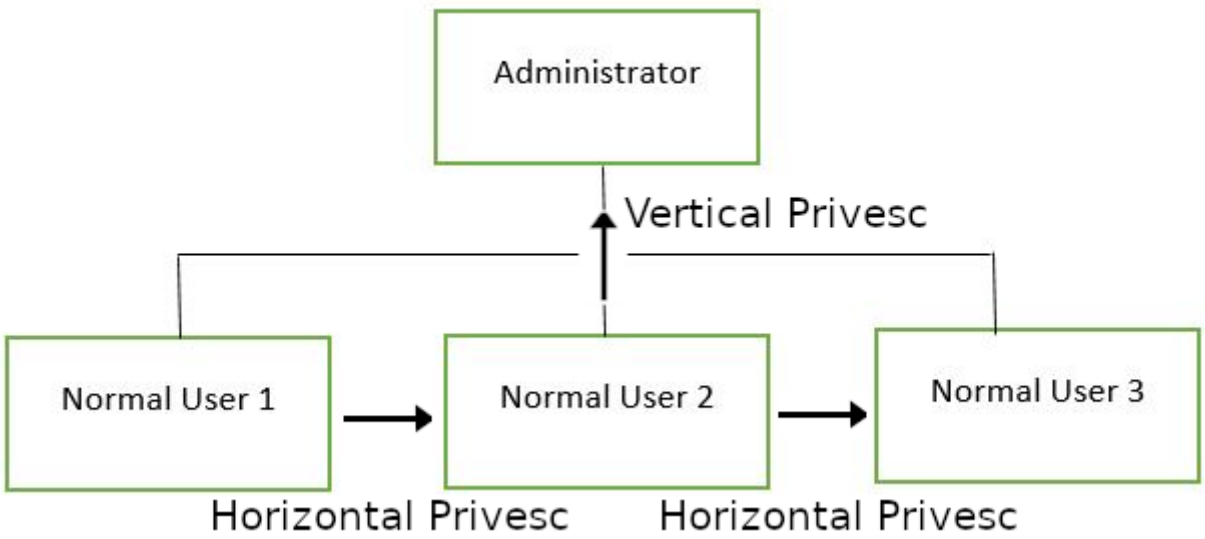
Why is it important?

Rarely when doing a CTF or real-world penetration test, will you be able to gain a foothold (initial access) that affords you administrator access. Privilege escalation is crucial, because it lets you gain system administrator levels of access. This allow you to do many things, including:

- Reset passwords
- Bypass access controls to compromise protected data
- Edit software configurations
- Enable persistence, so you can access the machine again later.
- Change privilege of users
- Get that cheeky root flag ;)

As well as any other administrator or super user commands that you desire.

Privilege Tree:



There are two main privilege escalation variants:

Horizontal privilege escalation: This is where you expand your reach over the compromised system by taking over a different user who is on the same privilege level as you. For instance, a normal user hijacking another normal user (rather than elevating to super user). This allows you to inherit whatever files and access that user has. This can be used, for example, to gain access to another normal privilege user, that happens to have an SUID file attached to their home directory (more on these later) which can then be used to get super user access. [Travel sideways on the tree]

***Vertical privilege escalation (privilege elevation):** This is where you attempt to gain higher privileges or access, with an existing account that you have already compromised. For local

privilege escalation attacks this might mean hijacking an account with administrator privileges or root privileges. [Travel up on the tree]

What is LinEnum?

LinEnum is a simple bash script that performs common commands related to privilege escalation, saving time and allowing more effort to be put toward getting root. It is important to understand what commands LinEnum executes, so that you are able to manually enumerate privesc vulnerabilities in a situation where you're unable to use LinEnum or other like scripts. In this room, we will explain what LinEnum is showing, and what commands can be used to replicate it.

Where to get LinEnum

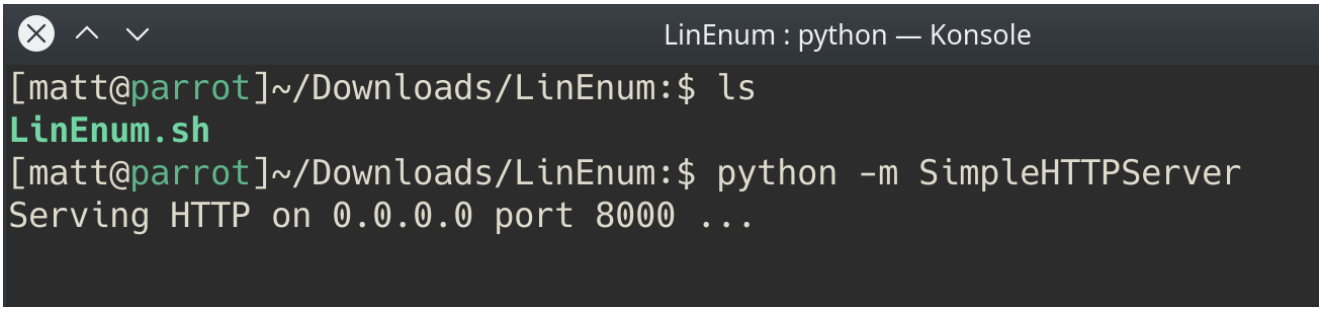
You can download a local copy of LinEnum from:

https://github.com/rebootuser/LinEnum/blob/master/LinEnum.sh

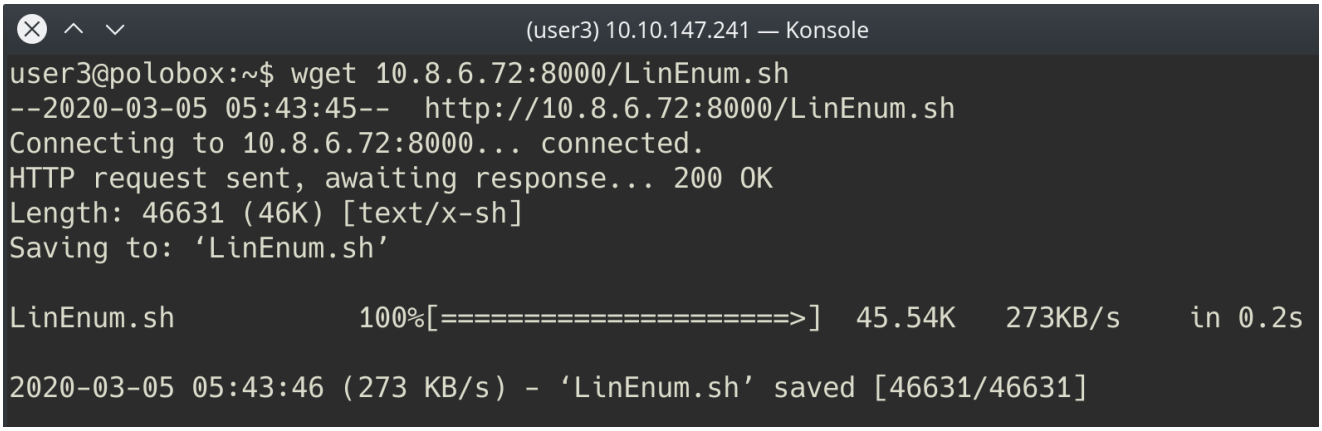
It's worth keeping this somewhere you'll remember, because LinEnum is an invaluable tool.

How do I get LinEnum on the target machine?

There are two ways to get LinEnum on the target machine. The first way, is to go to the directory that you have your local copy of LinEnum stored in, and start a Python web server using **"python3 -m http.server 8000"** [1]. Then using **"wget"** on the target machine, and your local IP, you can grab the file from your local machine [2]. Then make the file executable using the command **"chmod +x FILENAME.sh"**.



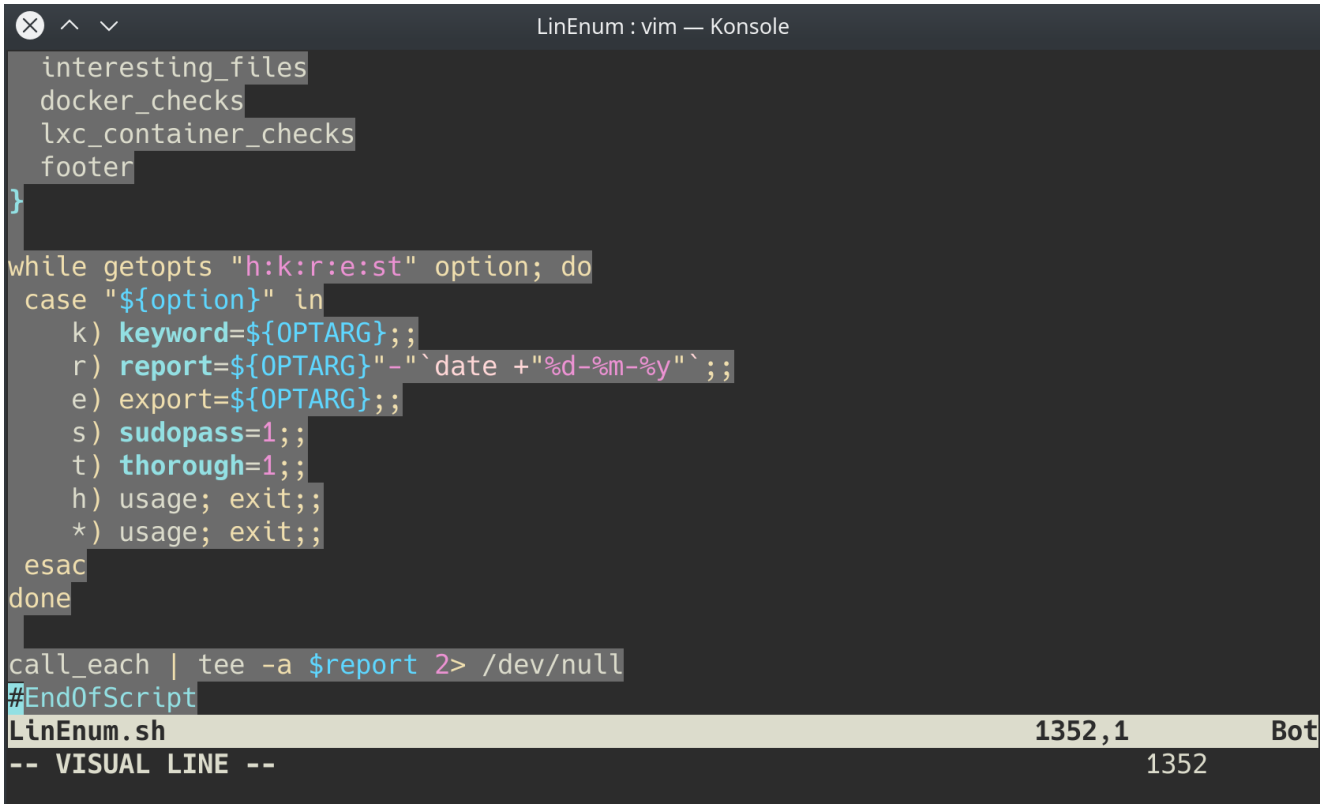
[1]



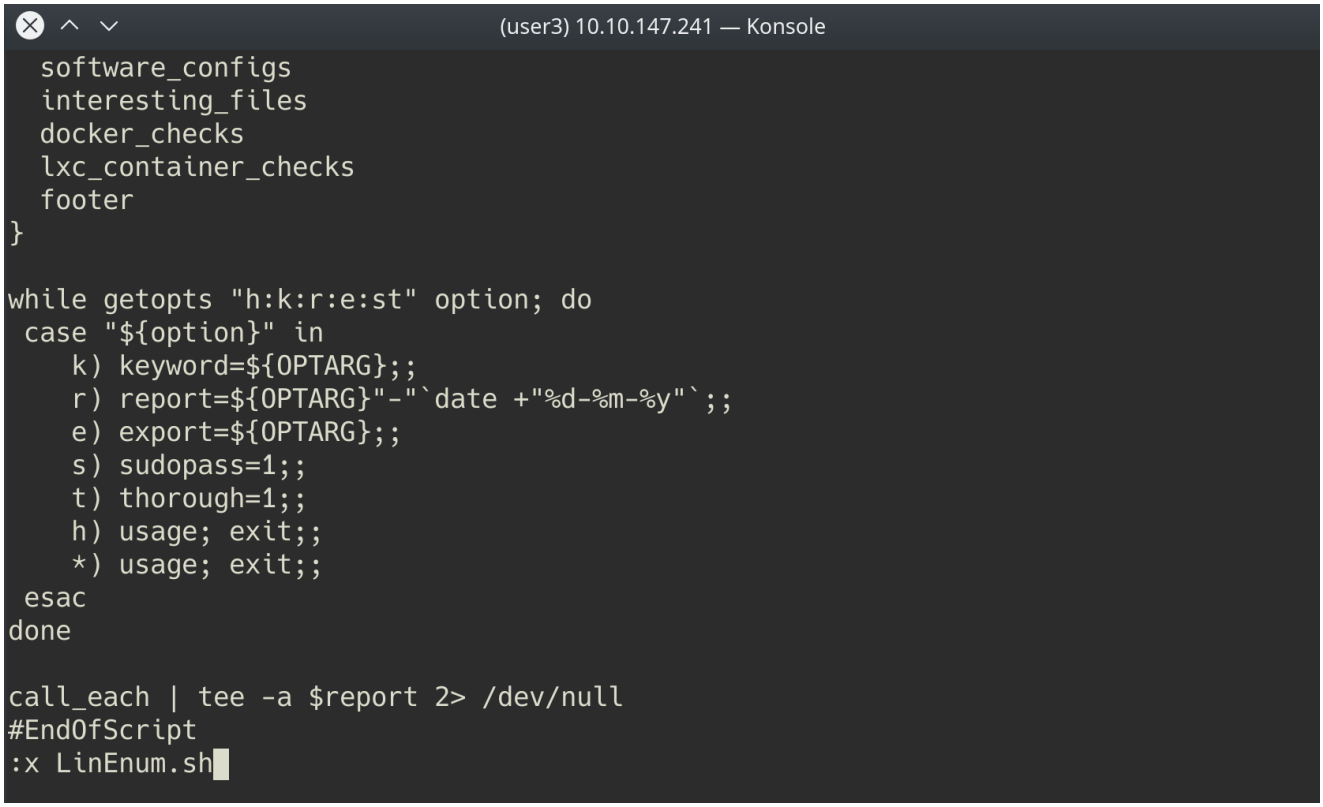
[2]

Other Methods

In case you're unable to transport the file, you can also, if you have sufficient permissions, copy the raw LinEnum code from your local machine [1] and paste it into a new file on the target, using Vi or Nano [2]. Once you've done this, you can save the file with the **".sh"** extension. Then make the file executable using the command **"chmod +x FILENAME.sh"**. You now have now made your own executable copy of the LinEnum script on the target machine!



[1]



[2]

Running LinEnum

LinEnum can be run the same way you run any bash script, go to the directory where LinEnum is and run the command "./LinEnum.sh".

Understanding LinEnum Output

The LinEnum output is broken down into different sections, these are the main sections that we will focus on:

Kernel Kernel information is shown here. There is most likely a kernel exploit available for this machine.

Can we read/write sensitive files: The world-writable files are shown below. These are the files that any authenticated user can read and write to. By looking at the permissions of these sensitive files, we can see where there is misconfiguration that allows users who shouldn't usually be able to, to be able to write to sensitive files.

SUID Files: The output for SUID files is shown here. There are a few interesting items that we will definitely look into as a way to escalate privileges. SUID (Set owner User ID up on execution) is a special type of file permissions given to a file. It allows the file to run with permissions of whoever the owner is. If this is root, it runs with root permissions. It can allow us to escalate privileges.

Crontab Contents: The scheduled cron jobs are shown below. Cron is used to schedule commands at a specific time. These scheduled commands or tasks are known as “cron jobs”. Related to this is the crontab command which creates a crontab file containing commands and instructions for the cron daemon to execute. There is certainly enough information to warrant attempting to exploit Cronjobs here.

There's also a lot of other useful information contained in this scan. Lets have a read!

SUID/GUID

Finding and Exploiting SUID Files

The first step in Linux privilege escalation exploitation is to check for files with the SUID/GUID bit set. This means that the file or files can be run with the permissions of the file(s) owner/group. In this case, as the super-user. We can leverage this to get a shell with these privileges!

What is an SUID binary?

As we all know in Linux everything is a file, including directories and devices which have permissions to allow or restrict three operations i.e. read/write/execute. So when you set permission for any file, you should be aware of the Linux users to whom you allow or restrict all three permissions. Take a look at the following demonstration of how maximum privileges (rwx-rwx-rwx) look:

r = read

w = write

x = execute

user	group	others
rwx	rwx	rwx
421	*421*	*421*

The maximum number of bit that can be used to set permission for each user is 7, which is a combination of read (4) write (2) and execute (1) operation. For example, if you set permissions using **"chmod"** as **755**, then it will be: rwxr-xr-x.

But when special permission is given to each user it becomes SUID or SGID. When extra bit **“4”** is set to user(Owner) it becomes **SUID** (Set user ID) and when bit **“2”** is set to group it becomes **SGID** (Set Group ID).

Therefore, the permissions to look for when looking for SUID is:

SUID:

rws-rwx-rwx

GUID:

rwx-rws-rwx

Finding SUID Binaries

We already know that there is SUID capable files on the system, thanks to our LinEnum scan. However, if we want to do this manually we can use the command: **"find / -perm -u=s -type f 2>/dev/null"** to search the file system for SUID/GUID files. Let's break down this command.

find - Initiates the "find" command

/ - Searches the whole file system

-perm - searches for files with specific permissions

-u=s - Any of the permission bits *mode* are set for the file. Symbolic modes are accepted in this form

-type f - Only search for files

2>/dev/null - Suppresses errors

Exploiting Writeable /etc/passwd

Exploiting a writable /etc/passwd

Continuing with the enumeration of users, we found that **user7** is a member of the **root** group with **gid 0.** And we already know from the **LinEnum** scan that **/etc/passwd** file is writable for the user. So from this observation, we concluded that **user7** can edit the /etc/passwd file.

Understanding /etc/passwd

The /etc/passwd file stores essential information, which is required during login. In other words, it stores user account information. The /etc/passwd is a **plain text file**. It contains a list of the system's accounts, giving for each account some useful information like user ID, group ID, home directory, shell, and more.

The /etc/passwd file should have general read permission as many command utilities use it to map user IDs to user names. However, write access to the /etc/passwd must only limit for the superuser/root account. When it doesn't, or a user has erroneously been added to a write-allowed group. We have a vulnerability that can allow the creation of a root user that we can access.

Understanding /etc/passwd format

The /etc/passwd file contains one entry per line for each user (user account) of the system. All fields are separated by a colon : symbol. Total of seven fields as follows. Generally, /etc/passwd file entry looks as follows:

```
test:x:0:0:root:/root:/bin/bash
```

[as divided by colon (:)]

1. **Username**: It is used when user logs in. It should be between 1 and 32 characters in length.
2. **Password**: An x character indicates that encrypted password is stored in /etc/shadow file. Please note that you need to use the passwd command to compute the hash of a password typed at the CLI or to store/update the hash of the password in /etc/shadow file, in this case, the password hash is stored as an "x".
3. **User ID (UID)**: Each user must be assigned a user ID (UID). UID 0 (zero) is reserved for root and UIDs 1-99 are reserved for other predefined accounts. Further UID 100-999 are reserved by system for administrative and system accounts/groups.
4. **Group ID (GID)**: The primary group ID (stored in /etc/group file)
5. **User ID Info**: The comment field. It allow you to add extra information about the users such as user's full name, phone number etc. This field use by finger command.
6. **Home directory**: The absolute path to the directory the user will be in when they log in. If this directory does not exists then users directory becomes /
7. **Command/shell**: The absolute path of a command or shell (/bin/bash). Typically, this is a shell. Please note that it does not have to be a shell.

How to exploit a writable /etc/passwd

It's simple really, if we have a writable /etc/passwd file, we can write a new line entry according to the above formula and create a new user! We add the password hash of our choice, and set the UID, GID and shell to root. Allowing us to log in as our own root user!

Exploiting Crontab

What is Cron?

The Cron daemon is a long-running process that executes commands at specific dates and times. You can use this to schedule activities, either as one-time events or as recurring tasks. You can create a crontab file containing commands and instructions for the Cron daemon to execute.

How to view what Cronjobs are active.

We can use the command **"cat /etc/crontab"** to view what cron jobs are scheduled. This is something you should always check manually whenever you get a chance, especially if LinEnum, or a similar script, doesn't find anything.

Format of a Cronjob

Cronjobs exist in a certain format, being able to read that format is important if you want to exploit a cron job.

`# = ID`

`m = Minute`

`h = Hour`

`dom = Day of the month`

`mon = Month`

`dow = Day of the week`

`user = What user the command will run as`

`command = What command should be run`

For Example,

 # m   h dom mon dow user command

```
17 * 1 * * root cd / && run-parts --report /etc/cron.hourly
```

How can we exploit this?

We know from our LinEnum scan, that the file autoscript.sh, on user4's Desktop is scheduled to run every five minutes. It is owned by root, meaning that it will run with root privileges, despite the fact that we can write to this file. The task then is to create a command that will return a shell and paste it in this file. When the file runs again in five minutes the shell will be running as root.

What is PATH?

PATH is an environmental variable in Linux and Unix-like operating systems which specifies directories that hold executable programs. When the user runs any command in the terminal, it searches for executable files with the help of the PATH Variable in response to commands executed by a user.

It is very simple to view the Path of the relevant user with help of the command **"echo $PATH"**.

How does this let us escalate privileges?

Let's say we have an SUID binary. Running it, we can see that it’s calling the system shell to do a basic process like list processes with "ps". Unlike in our previous SUID example, in this situation we can't exploit it by supplying an argument for command injection, so what can we do to try and exploit this?

We can re-write the PATH variable to a location of our choosing! So when the SUID binary calls the system shell to run an executable, it runs one that we've written instead!

As with any SUID file, it will run this command with the same privileges as the owner of the SUID file! If this is root, using this method we can run whatever commands we like as root!

Further Learning

There is never a "magic" answer in the huge area that is Linux Privilege Escalation. This is simply a few examples of basic things to watch out for when trying to escalate privileges_._The only way to get better at it, is to practice and build up experience. Checklists are a good way to make sure

you haven't missed anything during your enumeration stage, and also to provide you with a resource to check how to do things if you forget exactly what commands to use.

Below is a list of good checklists to apply to CTF or penetration test use cases.Although I encourage you to make your own using CherryTree or whatever notes application you prefer.

- https://github.com/netbiosX/Checklists/blob/master/Linux-Privilege-Escalation.md
- https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Linux%20-%20Privilege%20Escalation.md
- https://sushant747.gitbooks.io/total-oscp-guide/privilege*escalation*-_linux.html
- https://payatu.com/guide-linux-privilege-escalation