Privilege Escalation Guide

This guide assumes that you have established a reverse shell and have upgraded it

Some terms to be aware of as we go:

setuid (set user ID): is a special permission in Unix-like operating systems (including Linux) that can be assigned to executable files. When a file has the setuid permission, it allows the user who runs the file to temporarily gain the permissions of the file's owner rather than the permissions of the user who executed the file.

* When an executable file has the setuid permission and is executed, the process runs with the permissions of the file's owner, not the user who executed it.
* This can be useful for allowing regular users to perform tasks that require elevated privileges without giving them permanent access to those privileges.
* This obviously is a security risk and is how we will gain our privilege escalation.

Step 1: Finding setuid Files:

We start with a command that searches for files in the main directory that have a special permission (setuid) for the owner.



Let’s break it down:

* **find /:** Start looking for files from the main directory (root).
* **-perm -u=s:** Search for files with a special permission called "setuid" for the owner. This special permission allows a file to run with the permissions of its owner.
* **-type f:** Look only for regular files, not directories or other types of files.
* **2>/dev/null:** Ignore any error messages that might come up during the search.
* **|:** Take the results of the first command and pass them as input to the next command.
* **xargs ls -l:** For each file found, show detailed information using the ls -l command.

**An output like this would come out:**

A screenshot of a computer

Description automatically generated

So, what do these files mean. Let’s take one and break it down the first one:   


* **-rwsr-xr-x:** This represents the file's permissions.
  1. The first character (-) indicates that it is a regular file.
  2. The next three characters (rws) indicate the file has read, write, and setuid permissions for the owner.
  3. The next three characters (r-x) indicate read and execute permissions for the group.
  4. The last three characters (r-x) indicate read and execute permissions for others (non-owner).
* **1 root root:** This part indicates the file's owner and group.
  1. The number 1 indicates that this file is linked to (or is) a single inode (a data structure on a filesystem).
  2. The first root is the owner of the file.
  3. The second root is the group associated with the file.
* **16712:** This is the file's size in bytes.
* **Jan 31 2002:** This indicates the date when the file was last modified or created.
* **/opt/admin:** This is the path to the file on the filesystem.

Putting it all together, the file /opt/admin is a regular file with setuid permissions. The setuid permission is indicated by the s in the owner's permission set (rws). This means that when this file is executed, it will run with the permission of the owner (root), potentially allowing the user who runs it to perform actions with elevated privileges.

Step 2: Exploring setuid directories and files:

Now that we have some files and directories to work with, let’s explore the one that we saw above. So we cd/ into the directory and “cat” the file and we get something like this:

A screen shot of a computer

Description automatically generated

So this looks like a program that has two libraries (<stdio.h> & <unistd.h> and then code to execute a program. The program itself if you look at the language, can be inferred that it is using “setuid” to make a shell that operates at a root level.

Step 3: Executing the root shell

So how do we execute this file? "ls -al” is a good way to examine even hidden files (ls = list, a = all, l = long format).

A screenshot of a computer screen

Description automatically generated

“admin” is the only real file that could be executable, but we can’t as it is a binary file (feel free to try) so we use the magic wand of “./” which allows even binary executables to run, in fact this operator is generally used for executable files. **USE ./admin and you should see no error and a the next command line ready to be input.**

Note: you would never actually see this in the real world, this is basically a backdoor that is in plain site.

After you have executed ./admin, you’ve now created a root shell. Type in “id” and you should see that you are now a root user.

A black background with white text

Description automatically generated

But unfortunately you’ve downgrade the shell in this process, bummer. But we can re-upgrade real quick

Step 4: Upgrade the Root Shell

We start with python3 -c ‘import pty; pty.spawn(“/bin/bash”)’   
Note: Your shell is basic, and backspace is a big problem…. Unless you use “Ctrl Backspace” life-saver in this case.



After this we just do what we did with our original upgrade:   
A black background with white text

Description automatically generated

And then make sure you match up your root shell size with your KALI Machine size:

A screen shot of a computer code

Description automatically generated

So we do this in our root:   
A black background with white text

Description automatically generated

You now have an upgraded root shell….. Nice.