



Finding Potential SUID/SGID Vulnerabilities on Linux & Unix Systems

Welcome back, my nascent hackers!

We have spent a lot of time in previous tutorials focused on hacking the ubiquitous Windows systems, but the vast majority of "heavy iron" around the world are Linux or Unix systems. Linux and Unix dominate the world of Internet web servers with over 60% of the market. In addition, Linux and Unix servers are the operating system of choice for major international corporations (including almost all the major banks) throughout the world.

A common flaw in Linux and Unix operating systems are the SUID binaries. We learned in [this tutorial](#) how Linux handles permissions. In addition to the read, write and execute privileges, Linux/Unix has what is often referred to as the set user ID (SUID) and the set group ID (SGID) bit. These bits are set when we need an application/binary to run with the permission of the owner (SUID) or group (SGID), rather than the user.

For instance, when a regular user needs to change their password which exists in the */etc/shadow* file, they will need the permissions of "root" to change their password as */etc/shadow* is owned by "root" and no one else has permissions to write to it. This is resolved by setting the SUID bit, which gives the user the permissions of root while using */etc/shadow*.

If we look at the */etc/passwd*, we see its permissions as below.

- **-r-s--x 1 root root 15104 Mar 13 2012 /usr/bin/passwd**

Note that where we would expect an **x** in the first stanza of permissions (the owner's) there is an **s**. This **s** indicates that the sticky, or special, bit is set. This means that when this file is accessed by someone other than root, they have root's permissions while they are running or using it.

Exploiting the SUID Bit

Imagine, if you will, a scenario where I run a program that has the SUID bit set and has root privileges momentarily, I can re-engineer it to do something other than what it was designed to do. For instance, accessing the */etc/shadow* file. If I can get it to do my bidding—even for just a moment—while my permissions are root's, I can own the system.

Elevating Privileges

This SUID bit can, at times, be exploited to elevate privileges once we have hacked a system, have only a terminal, and have only regular user permissions. Also, if we are a local user and want to elevate our privileges, we can look to exploit applications that have the SUID or SGID bit set. If we can find an app that uses root permissions to run, we may be able to manipulate it to give us its elevated status—even if for a moment.

A good example of such a hack was the old eFax program that installed by default on nearly every KDE Linux system. The program was set up to use root privileges in order to send faxes from your computer. It was possible to use these privileges of the eFax program to access the */etc/shadow* file and print out the entire contents. Of course, once it grabbed the */etc/shadow* file, it was just a matter of cracking the hashes (most importantly the root user hash) to take total control of the system.

What I want to do here is show you how you can find these binaries, programs, and apps on your Linux, Mac or Unix system that have root privileges, if even for only a moment, and may be able to be manipulated to elevate your privileges when you only have limited user privileges.

Step 1 Finding Files with the SUID Bit Set

The "find" command in Linux is powerful utility that enables us to find files that meet some specified criteria. In this case, we want the find command to search the entire file system to locate files that have permissions with the SUID bit set, that are owned by root. We can accomplish by typing the following command:

- **debian > find / -perm +4000 -user root -type f -print**

Let's break this command down to its individual parts.

- **/** says start at the top (root) of the file system and search every directory
- **-perm** says look for the permissions that follow
- **+4000** is the numerical representation of the SUID bit permission
- **-user** says look for files that are owned by the following user
- **root** is the user whose files we are looking for
- **-type** defines the type of file we are looking for
- **f** represents a regular file (not directories or special files)
- **-print** tells the command to print to standard out the path to the file

When we run this command against a Debian 7 GNOME system, we get the following results.



```
File Edit View Search Terminal Help
/usr/lib/openssh/ssh-keysign
/usr/sbin/exim4
/usr/sbin/pppd
/usr/bin/lppasswd
/usr/bin/sudoedit
/usr/bin/procmail
/usr/bin/passwd
/usr/bin/sudo
/usr/bin/chsh
/usr/bin/chfn
/usr/bin/newgrp
/usr/bin/gpasswd
/usr/bin/pkexec
/usr/bin/traceroute6.iputils
/usr/bin/X
/bin/ping6
/bin/fusermount
/bin/su
/bin/ping
/bin/umount
/bin/mount
find: `/.pulse': Permission denied
find: `/root': Permission denied
```

As you can see, most of what was returned are BASH commands and other tried-and-true applications. Since these commands and applications have been tested over years, you are unlikely to find the vulnerabilities you need. What we are looking for are new untried and untested applications that might have the sticky bit set and may have been carelessly coded.

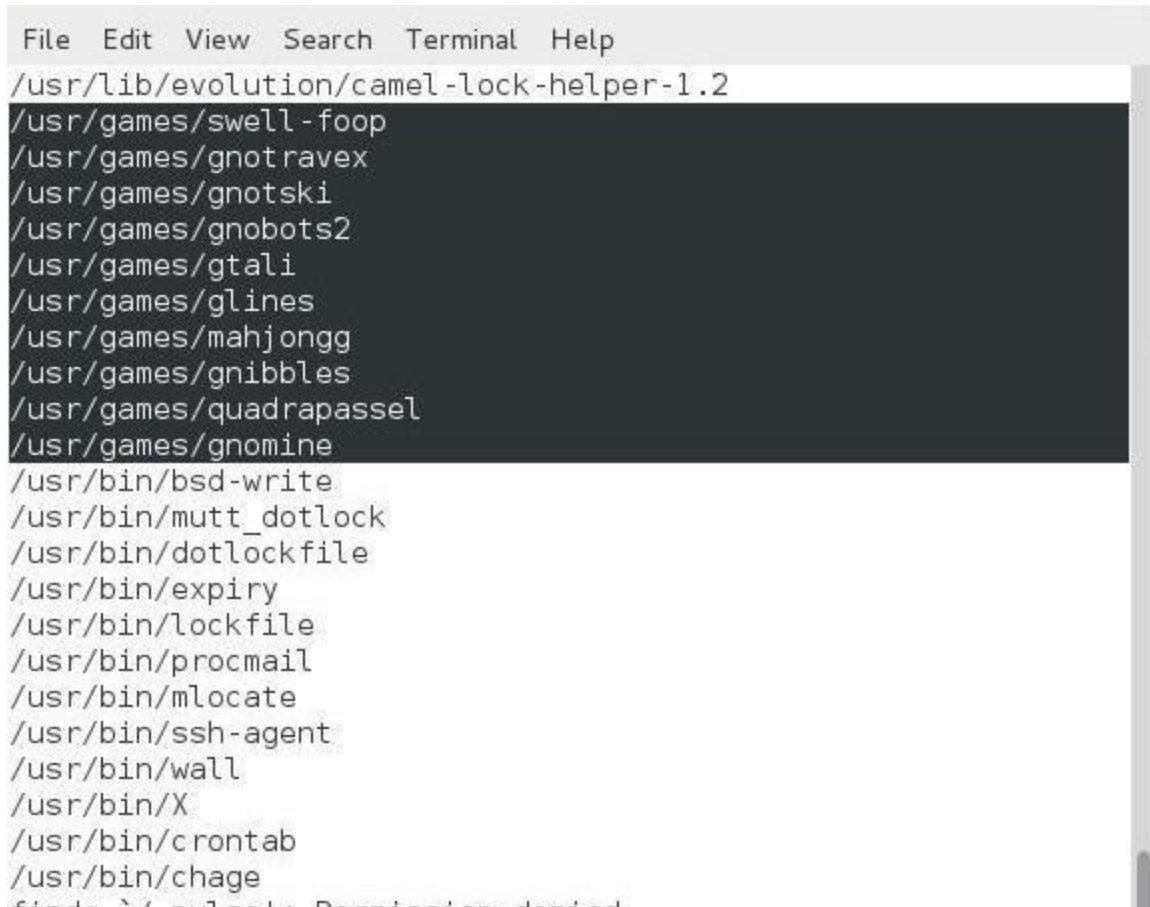
Step 2 Looking for SGID Bit Set

Now, let's look for programs that have the SGID bit set. We can find them by using the following command.

- **debian > find / -perm +2000 -user root -type f -print**

The only difference here is that instead of looking for +4000 permissions as in Step #1, now we are looking for +2000 permissions (SGID).

When we run this command on the Debian 7 system, we get the following output.



```
File Edit View Search Terminal Help
/usr/lib/evolution/camel-lock-helper-1.2
/usr/games/swell-foop
/usr/games/gnotravex
/usr/games/gnotski
/usr/games/gnobots2
/usr/games/gtali
/usr/games/glines
/usr/games/mahjongg
/usr/games/gnibbles
/usr/games/quadrapassel
/usr/games/gnomine
/usr/bin/bsd-write
/usr/bin/mutt_dotlock
/usr/bin/dotlockfile
/usr/bin/expiry
/usr/bin/lockfile
/usr/bin/procmail
/usr/bin/mlocate
/usr/bin/ssh-agent
/usr/bin/wall
/usr/bin/X
/usr/bin/crontab
/usr/bin/chage
```

I have highlighted numerous games that were returned from this search. This means that these games run with the permissions of the root group permissions. These might be a fertile group to seek to manipulate for privilege escalation as many games are poorly coded and might be good candidates to manipulate to gain root privileges.

I will continue to provide you the tools and techniques to hack with the best hackers in the world, so keep coming back, my nascent hackers.