SUID and SGID Holes and Weaknesses

---

by The Bronc Buster
bbuster@succeed.net
http://www2.succeed.net/~bbuster
Made for The Infected
www.infected.com

How do most hacks take place? 99% of them all are accomplished by someone getting something to do something it was not designed to do. How do they get whatever they use to preform something it's not suposed to do you ask? Well they defeat it, or they find a weakness in it. Some come with built it holes (like the tons of sendmail bugs) and some are simpley not configured correctly, the latter is what I am planning on "enlightening" you on. How SUID and SGID work, what to do to find files that use them, and how they can be used to make Unix do things you are not suposed to be doing. This paper IN NOT a step by step instruction manual on how to hack a certin hole, but how to recongnise a hole in the first place, as I figure I haven't seen any papers on this subject yet. Lets get started!

What you will need

---

1) A basic understanding of the Unix Operating System, and File Structure
2)A Brain (manditory)
3) Basic knowledge of shell scripts

Section 1

Before we start, I want to go over CHMOD, CHOWN, CHGRP, File Permissions, and how SUID and SGID work as you must understand these to be able to find these weaknesses.

CHMOD - uses the Chmod() system call to modify a files permissions

Useage: chmod [-Rfh] [agou] [+-=] [rwxXstugol] Filename

CHOWN - uses the Chown() system call to change the owner of a file. Only root can change this under most modern versions of Unix

Useage: chown [-fRh] owner filename

CHGRP - uses they Chgrp() system call to change the group of a file. Behaves like CHOWN above, must be root to change a files group.

Useage: chgrp [-fRh] group filename

File Permissions - When you do an "ls" command and the files come up you get the permissions to each file, or whats right you and your group, the owner and his group, and the world have on this file or directory. Exapmles are:

```
# ls -al
total 98
dwrxr--r--   evildude      bad              100   Feb 30   12:01    passwdcracked
-wrxrw-r--    evildude     bad            12020   Feb 29   13:02    hacks.txt
-wrsr-s--t    evildude     bad          56800   Feb   1   05:30    lard*
```

Ok, lets tear this down from here. Possible permissions on standard Unix systems are:

r READ gives you permission to use the open() system call to read the contents of that file

w WRITE gives you permission to overwrite files with new ones or to modify existing ones. You are allowed to use the write(), truncate() or ftruncate() system calls

x EXECUTE Allows you to use the exec() system call to execute a program. Depending on how the execute bits are set determines how it will run

s SUID or SGID : if the "s" is set in the ---s------ then the program is SUID. SUID programs have the effective UID as the owner of the file.
if the "s" is set in the ------s--- then the program is SGID. SGID programs have the effective GID as the files owner.

t STICKY Sticky is mostly obsolete with files but some older systems still use it, don't worry if you run into a "t"

Section 2

So now what? As you can guess, if you can get a file that is SUID as "root", then you execute whatever the program is AS "root". Some exapmles of an SUID program are ones you use quite often, and are quite commen. the "passwd" program that allows you to change your password in the passwd file is SUID. It has special features that match you PID and UID to make sure that you can not change anyones password but yours, but it's still a good exapmle. The majority or all SUID programs are SUID to root, so they really become the Superuser, in theory.

What can you do with them? Well first you have to find one. A simple command to find all SUID files on your system is :

```
# find / \(-perm -004000 -o -perm -002000 \) -type f -print
```

Now that you have found all the SUID files you have to dertimine what, if any, files are vurnable to some type of attack. A few examples I have found are the "write" bug. This program as it come, has a SGID hole built into it. If it is CHOWNed to root, then it will also be SUID. What the problem is : it sends simple message to another users terminal. It is set SGID tty, because it need to have access to all the tty to be able to print a message on someone elses terminal because Unix does not allow users to have access to someone elses tty; if you did, it would be an easy task to capture someone keystrokes. When you run this command, it's shell escape, that lets it SGID has a huge hole, if you begin a line with an "!" (exclamation point) the person running the "write" and envoking a new shell could make this new shell do almost anything. This is the part of the code that makes it weak:

```
setgid(getgid());
    execl(getenv("SHELL"),"sh", "-c",arg, 0);
```

Thats it. Someone careless programming and wa la, a hole because of a misconfigured program. Most newer versions of Unix, from 1995 and up, have fixed this hole with job controls, but a clever hacker might figure out a way to change back the premissions and re-misconfigure this program, and have it run normal still.

Another famous hole using SUID had to do with "sendmail" and "vi". There is a little used program called "vi" (a real hacker will have mastered its use, hehehe), it's a text editor of sorts, with a ton of cryptic commands. Well has you ever been telneted into you shell and then get cut off and lose all your work you had going? Well someone at AT&T deceided to be nice and add a feature to "vi" that not only saved your work, but e-mailed you to tell you your work was safe and sound. Can you see the hole? "/usr/lib/preserve" is the hole. Perserve is the program that does this nifty stuff, i.e. saves a temp file in your directory and uses the /bin/mail to let you know your stuff is saved. To do what it does, like write to your directory and access your mail, it must be SUID root. The problem is a little-known shell varable called IFS, the internal field seperator, which "sh" uses to figure out where there are breaks in words and on eack line as it parses the information. IFS can be set buy anyone, usualy it is set to be a standard blank, tab, and linefeed charactor, but resettinf it to an "/" the running "vi" and then issueing the perserve command, it was possible to get /usr/lib/preserve to esecute a program in it's homw directory, /bin. Insted of running /bin/mail you could ask it to run /bin/hackscript . As perserve ran it got parsed as bin with mail as it's argument, using the "/" you could make a script to replace mail and as it's arguement and do some damage, i.e.:

```
#
#Shell script to make a SUID-root shell
#
```

```
cd /home/evildude/bin
cp /bin/sh ./hacked
chown root hacked
chmod 4755 hacked
```

There you have it, a "sh" of your own so you may Superuser anytime you like with no history to mark your tracks. Well after a few years, CERT let out an advisory (ya they are so slow as hell) about this hole and the SUID weekness and versions of Unix from 1992 and newer have a new version on perserve that has only the rights and privlidges needed to do it's job, and not SUID root (sorry).

As you can see, SUID and SGID files can be easy to find, the hard part is determining if one of the files could have a potential hole. Due to the attention to detail, and how ISPs and Unix systems are popping up all over the world, and some lack of experience on the part of some system admins, I don't think it would be very hard for someone with an understanding of Unix file structures and basic shell scripting couldn't find a file or two that might be exploited with minimal effort. Good Luck!

Bronc Buster
bbuster@succeed.net

Any suggestions of comments are welcomed.