Sudo and other ways to avoid root!

It's time to admit my guilt. :−(For someone who can write an article on why you shouldn't be root, I'm still use it too much myself. Bit by bit, I'm finding ways to avoid it.

When I first had my own Linux system, I learned early that most of the system setup tasks I need to do had to be done as root. Isn't it easier to log in as root and do all my work that way? It certainly was, and for a few months, I did exactly that. The problem was that I didn't know how to efficiently perform tasks as root without logging out and logging back in − ugh! Well, I've been using Linux for seven years and I've got a few techniques that I'd like to share.

X as a non−root user

Here's the first mistake most people make; we start out running X Windows as the root user. Suddenly everything becomes simple; one has immediate access to all files, system administration tasks become easy, one can install new apps, etc.

Here's the problem; a lot of programs shouldn't be run as root. If you start up a file manager as yourself and choose to delete /home, well, your own files will disappear and you'll be cursing yourself all the way to the drawer with the backup tapes. Everyone else's files will still be there. If you did the same thing with the file manager running as root, you'll be staring at a lynch mob very quickly. See my "Undeleting files with MC" article if you don't believe it can happen.

File sharing programs, too, shouldn't be run as root either. If someone exploits a hole in an Irc or Gnutella client, the worst they'll be able to do is work with the files you own. Run them as root suddenly *everyone's* files are completely open to their view, including the all−important /etc/shadow password file and your coworkers personal email files. Not good.

Do yourself a favor. Run X under your own account, and use the following tips as ways to handle all the tasks that need to be run as root.

su and su −

Let's say someone's typing up the outgoing dialup connection and I want to know who the culprit is. tcpdump would show the packets flowing over the wire. Let's try it:

```
[wstearns@sparrow wstearns]$ tcpdump −i ppp0 −qtn
tcpdump: socket: Operation not permitted
```

Oh shoot, that's right. This shell, like all the other apps running under X, is running as wstearns. You'd think I'd know that from the prompt, wouldn't you? :−)

Well, I don't want to completely log out of X just for one command. **su** allows one to **S**ubstitute your current **U**ser for another one. If you don't specify who you wish to become, by default you'll become root. Before you start to drool at the hacking possibilities, you still need the password for the target user unless you're currently root. :−)

```
[wstearns@sparrow wstearns]$ tcpdump −i ppp0 −qtn
tcpdump: socket: Operation not permitted
[wstearns@sparrow wstearns]$ su −
Password: <enter root's password>
```

```
[root@sparrow /root]# tcpdump −i ppp0 −qt
User level filter, protocol ALL, datagram packet socket
tcpdump: listening on ppp0
<reflectix.xs4all.nl.29788 > ME.33085: tcp 1448 (DF) [tos 0x10]
> ME.33085 > reflectix.xs4all.nl.29788: tcp 0 (DF)
 <reflectix.xs4all.nl.29788 > ME.33085: tcp 1448 (DF) [tos 0x10]
> ME.33085 > reflectix.xs4all.nl.29788: tcp 0 (DF)
 <reflectix.xs4all.nl.29788 > ME.33085: tcp 1448 (DF) [tos 0x10]
> ME.33085 > reflectix.xs4all.nl.29788: tcp 0 (DF)
 <reflectix.xs4all.nl.29788 > ME.33085: tcp 1448 (DF) [tos 0x10]

7 packets received by filter
[root@sparrow /root]# exit
[wstearns@sparrow wstearns]$
```

Oh yes, that's my download that's tying up the line. Oops!

You may have noticed that I used **su −** instead of **su** in the above example. The "−" gives you the target user's environment instead of your own. This is most evident when you forget the dash and the shell can't find commands in /sbin and /usr/sbin:

```
[wstearns@sparrow wstearns]$ su
Password: <enter root's password>
[root@sparrow wstearns]# tcpdump −i ppp0 −qt
bash: tcpdump: command not found
[root@sparrow wstearns]# exit
[wstearns@sparrow wstearns]$
```

I was left in the current *wstearns* directory instead of being placed in root's home directory in the above example. Also, because /usr/sbin isn't in my path, I'd have to specify the whole path to tcpdump. Generally you'll want to use **su −**.

su − −c <command>

There's an even quicker version, if you literally want to run a single command. **−c** tells su to run a command and immediately return you to being the original user:

```
[wstearns@sparrow wstearns]$ whoami
wstearns
[wstearns@sparrow wstearns]$ su − −c whoami
Password: <enter root's password>
root
[wstearns@sparrow wstearns]$
```

The man and info pages for su talk more about this command and the command line options available.

sudo

Even the aformentioned **su** has it's drawbacks. You need to enter a password every time you want to run a command. Even worse, it's the *root* password!

What if you wanted to give some junior administrators the ability to, say, add new user accounts to the system? You could tell them to use su, but if you give them that root password, they'll suddenly become senior administrators. :−(

sudo fixes these problems and more. I'd like to install a new rpm on my system:

```
[wstearns@sparrow updates-sparrow]$ rpm -Uvh cyrus-sasl-1.5.24-11.i386.rpm
cannot open Packages index using db3 - Permission denied (13)

--> The rpm database cannot be opened in db3 format.
    If you have just upgraded the rpm package you need to convert
    your database to db3 format by running "rpm --rebuilddb" as root.

error: cannot open Packages database in /var/lib/rpm
[wstearns@sparrow updates-sparrow]$
```

It's rpm's way of saying I can't write to the rpm database, so I can't install new software.

```
[wstearns@sparrow updates-sparrow]$ sudo rpm -Uvh cyrus-sasl-1.5.24-11.i386.rpm
Password: <enter wstearns' password>
cyrus-sasl                  ##################################################
[wstearns@sparrow updates-sparrow]$
```

It looks a little like **su - -c**, doesn't it? I give it a command to run and it gets run as root. I only had to enter my own password, not root's. That, by itself, makes it a more useful tool than su.

If I run another command in 5 minutes, I don't need to enter my password again:

```
[wstearns@sparrow updates-sparrow]$ sudo rpm --erase cyrus-sasl
[wstearns@sparrow updates-sparrow]$
```

Sudo also logs all commands to /var/log/secure:

```
Oct 29 22:26:49 sparrow sudo: wstearns : TTY=pts/21 ; PWD=/mnt/redhat ; USER=root ; COMMAND=/bin/
Oct 29 22:30:03 sparrow sudo: wstearns : TTY=pts/21 ; PWD=/mnt/redhat ; USER=root ; COMMAND=/bin/
```

Before a normal user can use sudo, root needs to declare which users can run which commands as root. In this example, lets say that root on my box decided to let me install new software and unmount drives. She'll run the **visudo** command to edit /etc/sudoers; after I make my changes and exit, visudo will check the file to make sure she hasn't made any syntax errors. Here's what the /etc/sudoers file might look like:

```
# User privilege specification
root    ALL=(ALL) ALL

wstearns        ALL=(root) /bin/rpm,/bin/umount
```

If I try to execute something else, sudo blocks me and logs my attempt to /var/log/secure:

```
[wstearns@sparrow wstearns]$ sudo mcedit test
Sorry, user wstearns is not allowed to execute '/usr/bin/mcedit test' as root on sparrow.
[wstearns@sparrow wstearns]$ su - -c 'tail --lines=1 /var/log/secure'
Password: <enter root's password>
Oct 29 22:46:53 sparrow sudo: wstearns : command not allowed ; TTY=pts/21 ; PWD=/home/wstearns ;
[wstearns@sparrow wstearns]$
```

Be careful about handing out sudo privileges. In the above example, root granted me the right to install and remove rpms on the system. What if I decided to be malicious and replaced the normal util-linux package on the system with a new util-linux package that had a trojan horse /bin/login? The next time root logged in, the /bin/login might mail me the password root used. Even giving someone the ability to run **less** under sudo allows that user to spawn a shell as root from less with the **!** command.

Before you give anyone sudo privileges to a command, think through all of the ways that someone might misuse them. Could they somehow replace /etc/passwd or /etc/shadow with their own version? **dd**, **tar**, **ln** or any editor are just a few of the commands that can do this if run as root.

As with su, see the man pages for sudo, visudo, and sudoers for more details.

vlock −a

If I'm logged in on a few terminals and need to leave for a few minutes, I don't want to have to log out of all of them, only to log right back in after I get some more coffee, especially if I have some programs running on them. Instead, I find a free terminal and run **vlock −a**:

```
[wstearns@sparrow wstearns]$ vlock −a
The entire console display is now completely locked.
You will not be able to switch to another virtual console.
Please enter the password to unlock.
wstearns's Password: <enter wstearns' password>
[wstearns@sparrow wstearns]$
```

vlock doesn't log me out or stop any programs I might have running, it just doesn't let me type any more commands on that terminal. Also, by using the **−a**, I can't even switch to another terminal until I type my password; this protects all of my other terminals and X sessions that may be running.

Make sure you run this on a console; the **−a** doesn't work in an xterm.

You guessed it; **man vlock** talks more about the command.

William is an Open−Source developer, enthusiast, writer, and advocate from New Hampshire, USA. His day job at SANS pays him to work on network security and Linux projects.

Last updated 12/18/2003.