Permissions

Joseph Hallett

January 12, 2023

In the beginning there was `root`...

And it was good.
The `root` user is all powerful.

- The *super user*
- The *system administor*
- UID 0

And so root beget `init` which beget `login`...

And so other users came to pass

- Each less powerful than the original root

For inside the password file...

In the bowels of the computer's configuration directory `/etc/`:

```
$ grep -Ev '^_' /etc/passwd | column -ts :
```

Username	Password	UID	GID	GECOS	Home Directory	Shell
root	*	0	0	Charlie &	/root	/bin/ksh
daemon	*	1	1	The devil himself	/root	/sbin/nologin
operator	*	2	5	System &	/operator	/sbin/nologin
bin	*	3	7	Binaries Commands and Source	/	/sbin/nologin
build	*	21	21	base and xenocara build	/var/empty	/bin/ksh
sshd	*	27	27	sshd privsep	/var/empty	/sbin/nologin
www	*	67	67	HTTP Server	/var/www	/sbin/nologin
nobody	*	32767	32767	Unprivileged user	/nonexistent	/sbin/nologin
joseph	*	1000	1000	Joseph Hallett,,,	/home/joseph	/usr/local/bin/bash

See `man 5 passwd` or your OS's manual pages.

(Can anyone spot what OS I use?)

And inside the group file....

```
$ grep -Ev '^_' /etc/group | column -ts :
```

Groupname	Password	GID	Members
wheel	*	0	root,joseph
daemon	*	1	daemon
kmem	*	2	root
sys	*	3	root
tty	*	4	root
operator	*	5	root
bin	*	7	
wsrc	*	9	joseph
users	*	10	
auth	*	11	
games	*	13	
staff	*	20	root,joseph
wobj	*	21	joseph
sshd	*	27	
guest	*	31	root
utmp	*	45	
crontab	*	66	
www	*	67	
network	*	69	
authpf	*	72	
dialer	*	117	
nogroup	*	32766	
nobody	*	32767	
joseph	*	1000	

Something very similar

- Each group can have *multiple* members
- No passwords ever actually listed
  - (They're in `/etc/shadow`)

For all files were owned by a user and a group...

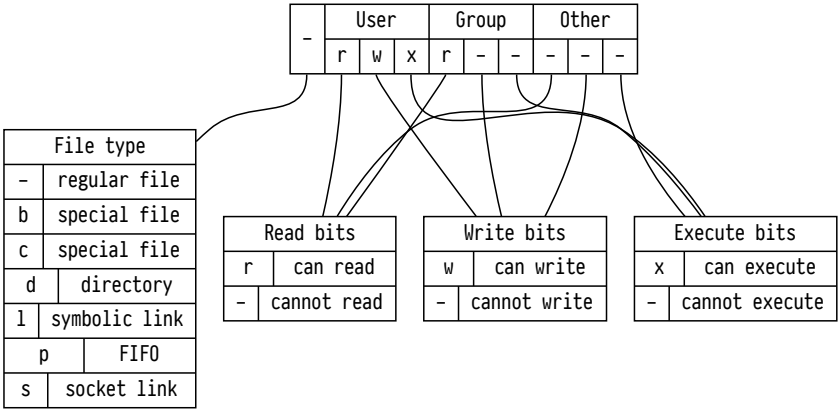
`ls -loh /etc/`

Permissions		UID	GID	File flags	Size	Filename	
drwxr-xr-x	5	root	wheel	-	512B	May 20 2022	ConsoleKit
drwxr-xr-x	2	root	wheel	-	512B	Nov 25 13:25	ImageMagick
drwxr-xr-x	7	root	wheel	-	512B	Nov 16 20:19	X11
-rw-r--r--	1	root	wheel	-	20.5K	Nov 6 12:41	abcde.conf
drwx------	2	root	wheel	-	512B	Nov 16 19:39	acme
-rw-r--r--	1	root	wheel	-	1.7K	Sep 22 19:03	adduser.conf
drwxr-xr-x	2	root	wheel	-	512B	Nov 16 19:39	amd
-rw-r--r--	1	root	wheel	-	271B	Oct 30 19:14	anthy-conf
drwxr-xr-x	3	root	wheel	-	512B	Nov 25 13:27	apache2
-rw-r--r--	1	root	wheel	-	1.8K	Nov 14 10:34	authentication$_{\texttt{milter.json}}$

…

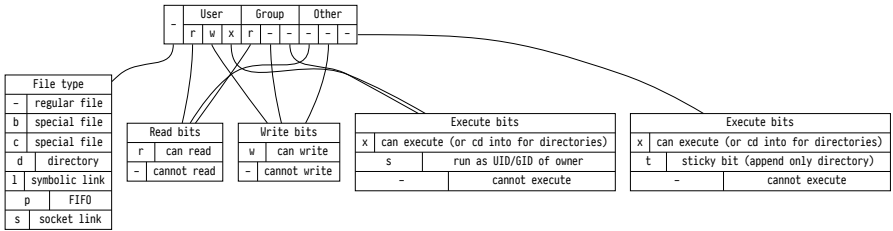
UNIX Discretionary Access Controls

And the owner of each file could set the *permissions* for each file



Actually its a *bit* more complex

And the owner of each file could set the *permissions* for each file



And, honestly, on some systems/filesystems it gets *even more* complex

- But this is 99.99% of everything you'll ever see or use

So what are those weird extra bits for

The *sticky* bit `t` is mostly for log directories and temporary directories

- You should be able to append to log files, but not delete them

The `setuid/setgid` bits are used for privilege separation.

For example how do you update your password?

Passwords are normally stored *securely* bin the shadow file `/etc/shadow`, or equivalent

- But I use OpenBSD...

```
ls -l /etc/spwd.db
```

-rw-r—– 1 root _shadow 40960 Dec 22 15:03 /etc/spwd.db

Changing passwords

The `passwd` program changes your password:

```
ls -l $(command -v passwd)
```

```
-r-sr-xr-x 1 root bin 21208 Jan 12 03:08 /usr/bin/passwd
```

Other useful `setuid` programs

`su` switch to user (by default root) with *their* password

`sudo` switch to user *if the sysadmin says you're allowed to* with *your* password

`doas` modern rewrite of `sudo` with less bugs and Spiderman references

See `man su` or `man sudo` or `man doas`...

- Or *Michael W. Lucas*'s excellent *Sudo Mastery*
- (You can do a lot with `sudo...`)

Generally `setuid` programs are dangerous and you want to use them extremely carefully!

How do you change who owns a file?

```
ls -l exam

-rw-r--r-- 1 joseph joseph 0 Jan 12 11:49 exam

chown joseph:staff exam
# Alternatively...
chown :staff exam
ls -l exam

-rw-r--r-- 1 joseph staff 0 Jan 12 11:49 exam
```

(See `man 1 chown`)

How do you change a file's permissions

```
chmod go-wx exam
ls -l exam

-rw-r--r-- 1 joseph staff 0 Jan 12 11:49 exam
```

Footnote

Some people like to use octal (base 8) to express permissions, where r=4, w=2, x=1...
Instead of saying `go-wx` to remove `w` and `x` bits from the group and other permissions they'll say:
`chmod 744 exam`
I suggest you give these people a wide berth.

- (but you should know how to do it)

Recap

Systems have users!

- The UNIX DAC lets you set file permissions!
- `setuid` and `setgid` programs exist!
- Root's firstname is *Charlie*!

`chmod` to change permissions

`chown` to change file owners

One more thing...

Traditionally the `root` user can do anything...
In *most* modern operating systems this has been split up a bit more

- For example Linux uses *capabilities* to set what things any user can do
- ...and *namespaces* to allow multiple root users with different capabilities

`man 7 capabilities` if you want to know more

- ...but *most of the time* you won't need to know about them...
- Unless you use Docker...

This is a lie, you really *should* know about them... but unless you're routinely in the habit of writing sysadmin tools or privileged programs you *won't normally* need to touch them. Hey, I'm a security researcher I think this stuff is fascinating but other people don't. Don't get it meself: it isn't *that* complex but hey ho. I tried.

Permissions :

Root The source of power in a computer system is the root user. This is the system administrator/superuser.
User ID (UID) = 0

Users Below root are the users. These represent the individuals or entities who interact with the system - these users have permissions dictating what they can & cannot read, write & execute.

You can see the users by looking in /etc/passwd or running the command

```
grep -Ev '^_' /etc/passwd
```

this filters out everything starting with _

You can also create users by using

```
sudo adduser NAME
```
• Choose a password

Once you've created a new user you can check its there by running

```
tail etc/passwd
```
or `ls -l /home`

To change to that user you run

```
SU NAME
```
at which point you'll have to enter that users' password, you'll notice this triggers a change in command line.

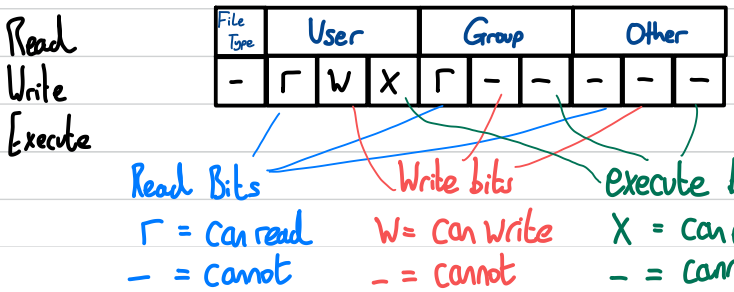
Groups Next we look at groups. These are a collection of users who share common access permissions or belong to a specific category or roll. Organising users this way simplifies access control & permission management.

You can view groups by running :

```
grep -Ev '^_' /etc/group
```

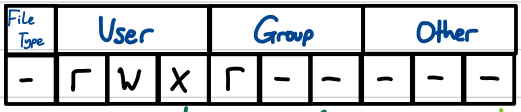
This shows all groups, their group ID (GID) & their members.

All files are owned by a user & a group. These relationships can be seen using `ls -l`
The owner can set the permissions for each file.
One last time, let's look at the UNIX permission bit structure (UNIX Discretionary Access Controls)



File type
- = regular file
b = special file
c = special file
d = directory
l = symbolic link
p = FIFO
s = Socket link

Slight Extension.



execute bits (User ~ Group)
X = can execute / cd in for directories
S = run as UID/GID of owner
- = Cannot execute

Execute bit (Other)
X = can execute / cd in for directories
t = sticky bit / append only directory
- = cannot

Set UID bit allows the file to be executed with the privileges of the file owner, a good example of this is the change password program, where it will temporarily grant users root access to change their password.

this bit only applies to directories

Other Useful Setuid programs

We've seen

SU NAME

This switches user (to root if username unspecified) using their password.

SUDO NAME

This switches to a user if the system admin says you're allowed to using your password

doas

Modern rewrite of sudo.

These setuid commands are dangerous ~ Should be used with extreme caution

- Changing who owns a file

To change who owns a file, we use the change owner command - chown, like so:

Usage

Chown USER:GROUP FILE

If you only want to change one (ie the user owner or the group owner), simply omit it but keep the colon -

Chown USER: FILE

Chown :GROUP FILE

Changing Permissions of a file

To do this, we use chmod. When using chmod, we need to tell it 3 things:

1. Who we are setting the permissions for — WHO
2. What change are we making (adding/removing) — WHAT
3. Which of the permissions are we setting. — WHICH

Usage:

(no spaces)

chmod WHO WHAT WHICH FILE

WHO -
- u = user
- g = group
- o = other
- a = all

WHAT -
- '-' = remove permission,
- '+' = set permission,
- '=' = set to specific permission config, (set some & remove others)

WHICH -
- 'r' = read
- 'w' = write
- 'x' = execute

Examples

chmod u+x file

This is one we've seen before, we now know it means we're giving the user who owns this file permission to execute it

chmod go-wx file

This removes write & execute permissions from everyone except the user who owns file

chmod u=rwx, go=r file

This wipes out all existing permissions giving the owning user full permissions & everyone else read-only access