

# Privileges: who can control what

## Introduction to Unix

May 24, 2008  
Rabat, Morocco

Hervey Allen



# Goal

Understand the following:

- The Unix security model
- How a program is allowed to run
- Where user and group information is stored
- Details of file permissions

# Some philosophy

- It's available!
- Wait, what was that? Oh yeah, it's available!
- It's has some very powerful features.
- It's ubiquitous in Unix (`visudo`, `vipw`, `vigr`, etc.)
- Not that hard to learn after initial learning curve.
- Impress your friends and family with your arcane knowledge of computers.

# Users and Groups

- Unix understands Users and Groups
- A user can belong to several groups
- A file can belong to only one user and one group at a time
- A particular user, the superuser “*root*” has extra privileges (uid = “0” in /etc/passwd)
- Only root can change the ownership of a file

# Users and Groups cont.

- User information in `/etc/passwd`
- User info in db-format in `/etc/pwd.db`
- User password hashes in db-format in `/etc/spwd.db`
- Group information is in `/etc/group`
- `/etc/passwd` and `/etc/group` divide data fields using “:”

# A program runs...

1. A program may be run by a user, when the system starts or by another process.
2. Before the program can execute the kernel inspects several things:
  - a) Looks up the numeric ID values for uid and gid of the user in the file `/etc/passwd`.
  - b) Is the execute bit set on the program file?
  - c) Does whoever ran the program, or the program itself have the required privileges to do what is requested?
  - d) In most cases, while executing, a program inherits the privileges of the user/process who started it.

# A program in detail

When we type:

```
ls -l /usr/bin/top
```

We'll see:

```
-r-xr-xr-x 1 root wheel 46112 Apr 28 10:52 /usr/bin/top
```

What does all this mean?





# Access rights

- Files are owned by a *user* and a *group* (ownership)
- Files have permissions for the user, the group, and *other*
- “*other*” permission is often referred to as “world”
- The permissions are *Read*, *Write* and *Execute* (R, W, X)
- The same applies to all files

# Some special cases

When looking at the output from “`ls -l`” in the first column you might see:

`d` = directory

`-` = regular file

`l` = symbolic link

`s` = Unix domain socket

`p` = named pipe

`c` = character device file

`b` = block device file

# Some special cases cont

In the Owner, Group and other columns you might see:

<code>s</code>	<code>=</code>	<code>setuid</code>	[when in Owner column]
<code>s</code>	<code>=</code>	<code>setgid</code>	[when in Group column]
<code>t</code>	<code>=</code>	<code>sticky bit</code>	[when at end]

## Some References

<http://www.tuxfiles.org/linuxhelp/filepermissions.html>

<http://www.cs.uregina.ca/Links/class-info/330/Linux/linux.html>

[http://www.onlamp.com/pub/a/bsd/2000/09/06/FreeBSD\\_Basics.html](http://www.onlamp.com/pub/a/bsd/2000/09/06/FreeBSD_Basics.html)

# File permissions

There are two ways to set permissions when using the `chmod` command:

Symbolic mode:

*testfile* has permissions of `-r--r--r--`

U G O\*

`$ chmod g+x testfile ==> -r--r-xr--`

`$ chmod u+wx testfile ==> -rwxr-xr--`

`$ chmod ug-x testfile ==> -rw--r--r--`

U=user, G=group, O=other (world)

# File permissions cont.

## Absolute mode:

We use octal (base eight) values represented like this:

<u>Letter</u>	<u>Permission</u>	<u>Value</u>
R	read	4
W	write	2
X	execute	1
-	none	0

For each column, User, Group or Other you can set values from 0 to 7. Here is what each means:

0= ---	1= --x	2= -w-	3= -wx
4= r--	5= r-x	6= rw-	7= rwx

# File permissions cont.

## Numeric mode cont:

Example index.html file with typical permission values:

```
$ chmod 755 index.html
```

```
$ ls -l index.html
```

```
-rwxr-xr-x 1 root wheel 0 May 24 06:20 index.html
```

```
$ chmod 644 index.html
```

```
$ ls -l index.html
```

```
-rw-r--r-- 1 root wheel 0 May 24 06:20 index.html
```

# Inherited permissions

Two critical points:

1. The permissions of the directory in which a file resides determines what a user can do to the file.
2. The permissions of the file determine what a user can do to the data in the file.

Example:

If a directory is owned by another user, then you cannot delete a file in the directory, even if you have write (w) access to the file, but you can update the data in the file.

# Conclusion

To reinforce these concepts let's do some exercises.

In addition, a very nice reference on using the `chmod` command is:

*An Introduction to Unix Permissions -- Part Two*

By Dru Lavigne

[http://www.onlamp.com/pub/a/bsd/2000/09/13/FreeBSD\\_Basics.html](http://www.onlamp.com/pub/a/bsd/2000/09/13/FreeBSD_Basics.html)