Access Control and the Filesystem

CSC 510

Unix Access Control Rules

- Access control decisions are based on which user is attempting to perform an action
- Objects have owners who have broad control over the objects. (Here objects are files, processes, etc.
- The user owns the objects that user creates
- There is a special user called “root” that can act as the owner of any object and perform administrative operations

Filesystem Access Control

- In the classic model, all files have an owner and a group.
- The “ls -l” command provides a long listing that includes information about owners, groups, and permissions
- The permissions are listed as a 10 character string
    - The first character represents the file’s type and mode
    - Characters 2-4 represent the owner’s read, write, and execute permissions
    - Characters 5-7 represent the group’s read, write, and execute permissions
    - Characters 8-10 represent other (global) read, write, and execute permissions

File Types

File Type	Symbol	Created By
Regular file	`-`	editors, `cp`, etc.
Directory	`d`	`mkdir`
Character device	`c`	`mknod`
Block device	`b`	`mknod`
Local domain socker	`s`	`socket` system call
Named pipe	`p`	`mkfifo`
Symbolic link	`l`	`ln`

The chmod command

- The `chmod` command can change the mode (permissions) of a file, the first argument is the permissions (in octal or mnemonic syntax) and the arguments that follow are files that should be changed
- Octal permission bits

Octal	Binary	Perm	Octal	Binary	Perm
0	000	`---`	4	100	`r--`
1	001	`--x`	5	101	`r-x`
2	010	`-w-`	6	110	`rw-`
3	011	`-wx`	7	111	`rwx`

- Example: `chmod 711 myfile` sets all permissions for the user (owner) and execute only permission to everyone else

The `chmod` command (continued)

- `chmod` mnemonic syntax: combine a set of targets with an operator and a set of permissions (see the man page for details)
- Examples

`u+w`	adds `w` to the owner
`ug=rw,o=r`	gives `r/w` to owner and group, and `r` for others
`a-x`	removes `x` for all categories
`g=u`	sets the group to be the same as the owner

Special Permissions

- user + s (SUID) – executes a file as the owner of the file regardless of the user running the program
- group + s (SGID) – if set on a file, allows the file to be executed as the group that ownd the file. If set on a directory, any files created in the directory will have the group set to the directory owner.
- other + t (sticky) – when set on a directory, restricts deletion of files; only the owner of the file can delete the file.
- Note: when listed with “ls” these settings take place in the execute bit location

Setting Special Permissions

- The octal method adds a digit to the beginning of the setting: 0 no change, 1 sticky, 2 SGID, and 4 SUID.
  - Example: `chmod 2770 dir` sets the SGID bit on dir
- The mnemonic method uses `s` for SUID and SGID, and `t` for sticky

Process Ownership

- The owner of a process can send signals to the process (see the `kill` man page)
- A process has multiple identities associated with it:
  - Real UID/GID: the real user/group that started the process
  - Effective UID/GID: the user/group that the process is running as for access control purposes (normally the same as the real UID/GID)
  - Saved UID/GID: IDs that are available for the process to invoke. Typically used when a process running with elevated privileges needs to perform some under-privileged work so the process can switch back and forth

Management of Root Privileges

- root login: `root` is user on the system with a password (usually a bad idea and some systems default to disabling the root password)
- `su` command: login as a regular user, then switch to the root user. An advantage of this approach is that this gets logged (but root still has a password)
- `sudo` command: run a single command as root
  - consults the `/etc/sudoers` file to determine if the command can be performed by the user (see the sudoers man page for the file format)
  - keeps a log of executed commands