File Permissions

Eric Missimer

IDs Associated with a Process

- Real User/Group ID - who we really are
- Effective User/Group ID - used for file access permission checks
- Supplementary Group IDs - this is how we can belong to more than one group
- Saved Set-User-ID/Set-Group-ID - saved by exec functions

Real User/Group ID

- Taken from our entry in the password file when we log in
- Values normally do not change
- Most of the time the effective and Saved Set-User/Group-ID are the same as the real

Effective User/Group ID

- This is the access rights used when we open a file
- Compare the EUID/EGID to the files `st_uid` and `st_gid` (from the `stat` struct)
- Usually the same as as the RUID unless the set-user-id and/or set-group-id bits are set – will talk about this later

Effective User/Group ID

- This is the access rights used when we open a file
- Compare the EUID/EGID to the files `st_uid` and `st_gid` (from the `stat` struct)
- Usually the same as as the RUID unless the set-user-id and/or set-group-id bits are set – will talk about this later

Saved Set-User-ID/Set-Group-ID – and this too

File Access Permissions

`st_mode` encodes the access permissions of a file:

`st_mode` Mask	Meaning
`S_IRUSR`	user-read
`S_IWUSR`	user-write
`S_IXUSR`	user-execute
`S_IRGRP`	group-read
`S_IWGRP`	group-write
`S_IXGRP`	group-execute
`S_IROTH`	other-read
`S_IWOTH`	other-write
`S_IXOTH`	other-execute

File Access Permissions

- Remember read and execute permissions on directory files are different
- Read lets use view the files in it, execute privileges are necessary for each directory on a file
- Ex: to open `\usr\include\stdio.h` you need execute permissions on:
  - `\`
  - `\usr`
  - `\usr\include`

Access Test Performed by Kernel

Remember kernel controls access to files. This gives us security. When we try to open a file the kernel does the following:

- If the effective user ID of the process is 0 (the superuser), access is allowed.
- If the effective user ID of the process equals the owner ID of the file access is allowed if the appropriate user access permission bit is set. Otherwise, permission is denied. By appropriate access permission bit, we mean that if the process is opening the file for reading, the user-read bit must be on. Similarly for write and execute.
- If the effective group ID of the process or one of the supplementary group IDs of the process equals the group ID of the file, access is allowed if the appropriate group access permission bit is set. Otherwise, permission is denied.
- If the appropriate other access permission bit is set, access is allowed. Otherwise, permission is denied.

access()

- What if we want to test whether we can access a file
- We can use `access(const char *pathname, int mode)` to test whether a process has the ability to access a file
- `mode` can be one of the following:
  - `F_OK` – tests for the existence of a file.
  - bitwise OR of one or more of `R_OK` (read), `W_OK` (write), and `X_OK` (execute)
- The check is done using the calling process's real UID and GID, rather than the effective IDs
- This allows set-user-ID programs to easily determine the invoking user's authority (promise, close to talking about set-user-ID)

set-user-id & set-group-id

- Ever wonder how `sudo` lets you run things as root?

set-user-id & set-group-id

- Ever wonder how `sudo` lets you run things as root?
- It has the set-user-id bit set, note the `s` instead of `x`:

```
---s--x--x. 1 root root 121K Feb 28  2013 /usr/bin/sudo
```

- So what happens when we execute a program with the set-user-id bit set?

set-user-id & set-group-id

- Ever wonder how `sudo` lets you run things as root?
- It has the set-user-id bit set, note the `s` instead of `x`:

```
---s--x--x. 1 root root 121K Feb 28  2013 /usr/bin/sudo
```

- So what happens when we execute a program with the set-user-id bit set?
- The *effective* user ID is the same as the owner of the file

set-user-id & set-group-id

- Ever wonder how `sudo` lets you run things as root?
- It has the set-user-id bit set, note the `s` instead of `x`:

```
---s--x--x. 1 root root 121K Feb 28  2013 /usr/bin/sudo
```

- So what happens when we execute a program with the set-user-id bit set?
- The *effective* user ID is the same as the owner of the file
- Why is this useful?

setuid and seteuid

- `int setuid(uid_t uid)`
  - If the process has appropriate privileges, setuid() shall set the real user ID, effective user ID, and the saved set-user-ID of the calling process to uid.
  - If the process does not have appropriate privileges, but uid is equal to the real user ID or the saved set-user-ID, setuid() shall set the effective user ID to uid; the real user ID and saved set-user-ID shall remain unchanged.
- `int seteuid(uid_t euid)`
  - sets the effective user ID of the calling process. Unprivileged processes may only set the effective user ID to the real user ID, the effective user ID or the saved set-user-ID.

setuid and seteuid

- `int setuid(uid_t uid)`
  - If the process has appropriate privileges, setuid() shall set the real user ID, effective user ID, and the saved set-user-ID of the calling process to uid.
  - If the process does not have appropriate privileges, but uid is equal to the real user ID or the saved set-user-ID, setuid() shall set the effective user ID to uid; the real user ID and saved set-user-ID shall remain unchanged.
- `int seteuid(uid_t euid)`
  - sets the effective user ID of the calling process. Unprivileged processes may only set the effective user ID to the real user ID, the effective user ID or the saved set-user-ID.
- Code example