Chapter 4

Authorization

This section deals with users, groups, and permissions for the files in the UNIX system. Dealing with setting these things up is the job of the system administrator. In fact, with the exception of setting permissions and ACLs (both discussed later), all of setup in these sections (e.g. adding users, groups, sudoers entries) must be done by the system administrator.

4.1 Users (/etc/passwd)

Every user on the UNIX system is associated with a specific user account that keeps track of certain settings specific to the user. These are things like the location of your home directory, your default shell. The data for these accounts is stored in the **/etc/passwd** file. This is a plain text file consisting of one-line entries, each entry being a colon delimited list of values:

```
Name:Password:UserID:PrincipalGroupID:Gecos:HomeDirectory:Shell
```

Lets examine each entry:

Name The username of the account.

Password The user's encrypted password. This field is covered more in the password section.

UserID The user's identification number. Every user on the system is assigned a unique number. There are certain UIDs that are reserved for specific user accounts. For example, UID 0 is always associated with the super-user account.

PrincipalGroupID The user's principal group identification number. All files created by the user are associated with the user's principal group. This field is covered more in the groups section.

Gecos This field has no defined syntax and is used for general information on the user. This includes things like the user's real name, phone number, room number, etc.

HomeDirectory The path to the user's home directory.

Shell The path to the user's shell.

Lets take a look at some passwd entries:

```
root:x:0:0:root:/root:/bin/bash
www-data:x:33:33:www-data:/var/www:/bin/sh
gdm:x:105:107:Gnome Display Manager:/var/lib/gdm:/bin/false
thomson:x:1008:1008:Thomson Nguyen,,,:/home/thomson:/bin/bash
aoaks:x:1010:1010:Aaron Oaks,,,:/home/aoaks:/bin/bash
abhi:x:1011:1011:Abhi Yerra,,,:/home/abhi:/bin/bash
```

You may notice that some of these users, like `gdm` and `www-data` seem like odd choices for a person's user name. These are referred to as **system user** accounts. These accounts are used by certain system users for access control. For example, the `www-data` user account is used by the webserver. By having the webserver run as the `www-data` user, if for some reason the apache server is compromised, only the files accessible by the `www-data` user or accessible.

4.2 Groups (/etc/group)

Every user in the UNIX system is associated with a collection of groups. These groups allow permissions to be set that effective for anybody that is a member of that group. For example, a set of users could collaborate on a file by allowing read and write access on that file to anybody who is a member of a certain group. This will be discussed later in the permissions section. Groups are specified in the **/etc/group** file, in a format similar to that of the passwd file:

```
Name:Password:GroupID:Members
```

Lets examine each entry:

Name The name of the group.

Password The encrypted password for the group. This is rarely used.

GroupID The group's identification number. Each group is assigned a unique number. There are certain GIDs that are reserved for specific groups. For example, GID 0 is always associated with the `root` group. This is the `GroupID` that is specified in the `/etc/passwd` file. Every user is a member of at least one group. The group specified in the `passwd` file is called their **principal group**.

Members These are users who are secondary members of the group. As discussed earlier, ever member specifies a principal group in the `passwd` file. This field specifies users who are members of the group but do not declare the group as their principal group.

Lets look at some example group entries:

```
video:x:44:sle,jchu,webcam,aoaks,thomson
plugdev:x:46:griffin
audio:x:29:griffin,sle,mpd,aoaks
wheel:x:108:sle,dima,angel,thomson,aoaks,sluo,yury
aoaks:x:1010:
```

On this system, the convention is that when a new user is created, a new group is created for that user. Lets consider the `aoaks` user. If you look back at the example `passwd` entries, you will see that the user `aoaks` declares his principal group to be `1010`. Looking at these group entries, this means that the his principal group is `aoaks`. Any files created by `aoaks` will be owned by `aoaks` and in the group `aoaks`. However, notice that the user `aoaks` appears as a member of several other groups. This means that he will be given whatever permission is associated with these groups.

For example, the `wheel` group is the group that system administrators are members of. Since `aoaks` is a member of this group, he has whatever abilities are granted to the `wheel` group. This typically means he can do things like run high level commands. This will be talked about more in the permissions section.

4.3 Passwords (/etc/shadow)

The `/etc/passwd` file must be readable by everybody on the system in order for programs to make use of it. Originally, this presented a problem because it forced the encrypted password hashes to be readable by everybody. While they were encrypted, anybody could use a password cracking program and with time, reverse the password hashes. This was solved by moving the password hash into a secure file called **/etc/shadow**. This file is accessible only by secure system accounts. You may also have noticed that all of the password fields in the `passwd` file contain only an `x` and not a password hash. This is by convention and is used to mean that the password has been stored securely in `/etc/shadow`.

Note that the passwords stored here are created using the **crypt** encryption function. If the value in the password field is not a valid output of —crypt—, (usually a `*` or a `!`), the user cannot log in using password authentication.

A typical `shadow` entry looks like this:

```
aoaks:$1$djckutDKdkdcdmsdkuR23dj5DKjcd2l:13439:0:99999:7:::
```

The most important fields are the first two, which specify the user name and the encrypted password hash. The other fields contain password aging data. This includes things like password age, until the password expires, etc. The

specifications of these values varies between platforms, so we won't go into this here.

4.4 Permissions

Every file on the system is associated with a specific user (the **owner**) and a specific group. Lets look at some sample files:

```
aoaks@flood:~$ ls -l | head
total 16328
-rw-------   1 aoaks ocf  235830 2007-02-14 12:35 110bs07-rel.pdf
-rw-------   1 aoaks ocf 8515935 2007-02-14 20:56 aoaks
drwx------+  4 aoaks ocf       4 2007-01-27 19:40 class/
-r--r--r--   1 aoaks ocf   20480 2006-11-03 22:22 Coursework.xls
-rw-r--r--   1 aoaks ocf     135 2007-02-13 12:48 data
drwxr-xr-x+  2 aoaks ocf      47 2007-02-20 01:47 decal/
drwxr-xr-x   2 aoaks ocf      27 2007-01-28 17:06 decalsle/
-rw-------   1 aoaks ocf   81227 2007-01-21 17:48 decal-sle.zip
```

Among other output, first, third, and fourth column are most relevant to permissions. The third and fourth columns specify the files owner and group respectively. The first column shows the permissions that are set on the file.

Permissions are divided into three groups, then into three types. The permission groups are **owner**, **group**, and **other**, which specify permissions for the file's owner, members of the file's group, and everybody else on the system. For each group, there are three types of permissions: read, write, and execute. The meaning of these permissions differs for regular files and for directories.

4.4.1 Regular Files

Permissions on regular files mean what they say:

Read You have permission to read the data in the file.

Write You have permission to write data to the file.

Execute You have permission to execute the file as a program.

4.4.2 Directories

Permissions on directories have a slightly different meaning. Since a directory file contain information about the files contained within it, the permissions on a directory relate to the contents of the directory.

Read You have permission to view what files are in the directory.

Write You have permission to create or delete files in the directory.

Execute You have permission to `cd` into the directory and `stat` the files within it.

The execute permission on a directory is perhaps the most confusing of all the permissions. This requires some knowledge about the system calls involved with accessing data. In order to access a file, the system must first look up the permissions on the file. In order to get the permissions, the system runs the `stat` system call on the file's **inode** (the file's unique identifier in the filesystem). In order to look up the inode number, you have to be able search the directory file, which requires execute permission on the directory.

This has some interesting implications. If you don't set execute permission on a directory, the contents of the directory are inaccessible. Thus, even if you specify full permissions on a file, if you don't give execute permission to the directory that contains it, you won't be able to access the file. This works from the root directory up to the file in question. Therefore, if you have a file `/foo/bar/baz/myfile.txt`, you will need to have execute permission on `\`, `foo`, `bar`, and `baz` before you can even look up permissions on `myfile.txt`.

4.4.3 Special Permissions

In addition to the general permissions discussed above, there are also special permissions that can be used by the experienced administrator to accomplish specific goals. It is important that you know what you are doing before using these permissions. BE CAREFUL!!! THESE PERMISSIONS CAN INTRODUCE MAJOR SECURITY HOLES IF YOU USE THEM WITHOUT UNDERSTANDING THE IMPLICATIONS OF THEIR USE!

File setuid and setgid bits Normally when a file is executed, it runs with the permissions of the running user's UID and GID. If the `setuid` or `setgid` permissions are set, the file executes with the UID or GID of the file.

Directory setgid bit When the setgid bit is set on a directory, all new files and directories created within that directory take on the GID of the directory, rather than the principal GID of the user.

Sticky bit The sticky bit is usually applied now only to directories. When applied on a directory, files within the directory can only be renamed or deleted by the file's owner, the directory's owner, or the super-user. This is most commonly used on the `/tmp` directory. This directory typically gives full permissions to all users on the system so that all uses can use the temp space. Normally, since every user has write permission on the directory, they would have permission to remove other user's data. Applying the sticky bit fixes this problem.

4.5 ACLs

You may have noticed that two of the files in the sample file listing have `+` signs at the end of the permissions. This indicated that an ACL has been set on that file. Access Control Lists (**ACLs**) provide a way to modify permissions on a file in order to allow permissions that would be difficult to set up otherwise. For example, say I wanted to give `jameson` permission to modify the `Group_Info` file so that he could update his information when it changes. If I gave ownership of the file to him, I would no longer be in control of it. If I added him to my group, he would have permission to access anything in my group. The solution is to set an ACL on the file, specifying full permission for `jameson`. By doing this, I can give him the permissions he needs on a specific file without compromising my other files. ACLs can only be specified by the owner of the file. You can read up on ACLs in the man pages (`getfacl` and `setfacl`).

4.6 Sudo

`Sudo` is similar to ACLs in that it allows you to extend permissions to other users, but unlike ACLs which allow you to give permissions on files to other users, sudo allows you to execute commands as another user. It consults a central files called the `sudoers` file to see if you have permission to run the requested command as the requested user. If you do, sudo changes your UID and GID to the desired user and group and executes the command. This method can be very useful for delegating privileges. For example, the `adduser` command is used to add users to the system. In order to work, it must be run as `root`. If you as a system administrator want to give certain trustworthy users the ability to add new users to the system, you could add them all to a group, say `creators`, then specify in the sudoers file that all members of the group `creators` have permission to run `/usr/sbin/adduser` as root. You can learn more about the `sudo` system, read up on it in the man pages (`sudo` and `sudoers`).