1 Introduction

AFS allows a higher degree of security than standard Unix. While standard Unix provides read, write, and execute permissions for three different groupings of security (owner, group, and other), AFS allows setting (among others) read, list, insert, delete, write, for individual or groups of AFS users. This higher level of flexibility greatly enhances the security possibilities available for your files.

2 Setting AFS Permissions Attributes

Most of you are probably familiar with the fs command. With fs, you can set and reset AFS permissions flags. The fs command can also perform many other AFS functions which are beyond the scope of this tutorial; we will only discuss the permission-setting functions here. For your own personal edification, type fs help to see the full list of fs’s options. To examine the permissions list for a given directory, type the following command:

faulkner% fs la directory

This will display a list of the AFS users and groups who are explicitly mentioned in the permissions list for the AFS directory directory, as well as their associated permission rights. To set and reset AFS security flags on a given directory, the format of the fs command is:

faulkner% fs sa directory AFS_username permissions

Where directory is the target AFS directory, AFS_username is the target AFS user that you with to explicitly specify in directory’s permissions list, and permissions are the actual flags that you want to set. Note that you reset flags by not setting them.

AFS_username can not only take the value of any AFS user ID, it can also take the value of the special AFS group system:anyuser. This group represents all AFS users. That is, whatever permissions that you assign to system:anyuser for a given directory will be used for all users not otherwise explicitly stated in the permissions list. For example:

faulkner% fs sa ~ system:anyuser rl

will give read and list privilidges to any user not otherwise listed in the AFS permissions for your home directory. You can also take away someone’s access with the special permission none:

faulkner% fs sa ~ system:anyuser none
This will remove the AFS group `system:anynone` from the permissions lists of your home directory. That is, unless an AFS user is explicitly stated in the permissions list of your home directory, they will have no access rights. **BE CAREFUL!** AFS does not protect against stupidity! It is legal to give yourself the `none` permissions setting on your own directory!

The settable flags that the `fs` command will accept are:

<table>
<thead>
<tr>
<th>Flag</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>r</td>
<td>Read</td>
</tr>
<tr>
<td>l</td>
<td>List</td>
</tr>
<tr>
<td>i</td>
<td>Insert</td>
</tr>
<tr>
<td>d</td>
<td>Delete</td>
</tr>
<tr>
<td>w</td>
<td>Write</td>
</tr>
<tr>
<td>k</td>
<td>Lock</td>
</tr>
<tr>
<td>a</td>
<td>Administrate</td>
</tr>
</tbody>
</table>

The `Administrate` flag enables a user or group to set or reset the AFS permissions flags on a given directory. It should also be noted that AFS permissions, like most things in Unix, are inheritable. That is, if you make a subdirectory, it will inherit the same permissions as its parent (which of course, are changeable).

### 3 AFS Groups

AFS provides for the ability to specify multiple AFS users under one logical name. This ability is called grouping, and is slightly different than the standard Unix implementation of grouping. These groups are most useful when working on projects that require several AFS users to have access to the same files.

AFS groups are accessed with the `pts` command. As a normal AFS user, you can create, edit, and destroy groups that begin with your AFS id. For example, the AFS user `rplant` could create a group for his Mentor Graphics (Mentor is fun, yeah!) project with the following command:

```
faulkner% pts creategroup rplant:mentor_proj
```

```
Group rplant:mentor_proj has id -612
```

AFS will return some non-important numeric identifier for this group. Once the group has been created, several operations are possible. These include: adding AFS users to the group, listing the AFS users in the group, and deleting AFS users from the group.

```
faulkner% pts adduser jpage rplant:mentor_proj
faulkner% pts membership rplant:mentor_proj
```

```
Members of rplant:mentor_proj (id: -612) are:
  jpage
```

```
faulkner% pts removeuser jpage rplant:mentor_proj
```

The main purpose for defining AFS groups is that they can be used as the AFS_username argument in the fs command. So if rplant executed the following commands:

```
faulkner% pts adduser jpage rplant:mentor_proj1
faulkner% fs sa ~ rplant:mentor_proj1 rl
faulkner% fs sa ~/mentor rplant:mentor_proj1 write
```

then he and jpage would be able to work on the same Mentor files with full writing privileges. Note how it was not necessary to set the administrate flag to the group’s access; this flag should only be set for the owner and the special AFS group system:administrators (select OUC personnel). As more users are placed into the rplant:mentor_proj1 group, they will have all the access rights that have been defined for rplant:mentor.proj1.

4 Application

With AFS groups, you can dynamically define, associate, and change a group of users with a set of AFS permissions. The most practical application (as related to Dr. Henry’s class) is to have the team leader apply for more disk space in AFS (you’ll need it for the large projects that Dr. Henry will assign) using the form provided. Turn it in to any consultant in the public Fitzpatrick lab. The team leader should create an AFS group with the other members of the team, as well as Dr. Henry (ehenry) and the teaching assistants (msheliga and kbalemar). Then, using the fs command, set permissions for that AFS group to read and list (rl) in the directory hierarchy all the way down to the destination top-level Mentor directory. In the top-level Mentor directory, it is necessary to give write permissions to the rplant:mentor_proj1 group so that the members of this group and read and write to the schematic datafiles located in this directory.

As described in the preceding paragraph, it is only necessary for rplant to give read and list privileges to rplant:mentor_proj1 in his home directory. This group does not need full writing privileges in rplant’s home directory; it only needs read and list in order to reach the ~mentor subdirectory. This can also be taken a step further; rplant only has to grant write privileges in the top-level Mentor directory and the schematic directories of those projects for which rplant:mentor_proj1 need to work on. If rplant has other Mentor schematics for other projects, he does not need to give rplant:mentor_proj1 access to them at all. For example, rplant could define rplant:mentor_proj2 and give that group access to selected schematics.

5 If you ignored the above advice...

Since Mentor takes the liberty of creating copious numbers of directories under your top-level Mentor directory, if you have already created some schematics and then decided to apply the AFS groups to them, follow the same procedure listed above to set the permissions up to and including your top-level Mentor directory. Then execute the following command:

```
faulkner% find top_level_mentor_directory/schematic_name -type d
-exec fs sa {} AFS_group_name write \; -print
```
This will find all directories under the `top_level_mentor_directory/schematic_name` and execute the `fs sa` command to set the `write` flags for the AFS group `AFS.group.name`. Look up the man page for `find` if you want more information.