Mac OS X Filesystem ACLs

Josh Wisenbaker



AFP548
.COM

Table of Contents

Introduction 4

Background 4

The Problem *4*

The Solution *5*

ACL Basics 5

Atomic Theory *5*

Generated Unique Identifiers *5*

Access Control Entries *6*

Access Control Lists *6*

ACL Details 6

Permission Settings *6*

ACE Application Precedence *7*

Usage and Propagation *8*

Inheritance *9*

Deployment Example 10

The Players *10*

Enable Access Control Lists *10*

Directory Tree Setup *11*

ACL Planning and Creation *12*

ACL Tuning and Troubleshooting *14*

Wrapping up the ACE Definitions *18*

Test, Test, Test *18*

Conclusion 19

Appendix A **20**

ACL accounting ***20***

Version History **22**

Introduction

There has been a long-standing issue with Mac OS X in the fact that it lacked Access Control Lists, or ACLs for short. This weakness in the operating system has often led to frustration among sysadmins as they struggled to provide flexible file and service access to their user communities. With Mac OS X 10.4 Apple has added pervasive ACL, DAC, and SACL support to the OS. This step forward is one of the most significant in the history of the Mac OS.

You would think that the addition of ACLs would lead to widespread acceptance and usage among Mac sysadmins, but this has not been the case. In some situations people had found a workaround with POSIX file modes, or had reached some other solution they could settle with. These sort of solutions are often not necessary with the new file-system ACLs and it would behoove sysadmins to not deploy them in the future. More often than not though, the ability to now apply thirteen different access rights has proved bewildering to many. This whitepaper is an effort to reduce the confusion that the file-system ACLs can bring and help make sense of the deployment options that exist.

Background

The Problem

Traditionally, Mac OS X has been limited to the standard POSIX permission model of read, write, and execute. This system has served UNIX based OSes well for many years now, but is starting to show its age. The main concern is that it isn't very flexible and a host of special modes, such as the "sticky bit", have been added over time. Furthermore, the growth and acceptance of Windows as a server platform has helped to drive the utilization and understanding of ACLs. It's not too hard to find a situation where you get stuck with the POSIX system. Let's take a look at a simple example.

Let's say that you have a directory that:

- The "accounting" group needs r/w access to,
- You need to deny the "everyone" mode any access at all,
- You need to allow a second group, "auditing" r-o access.

That's it. You are stuck.

Mac OS X Server can not assign a group as a owner. This is a surprise to many longtime ASIP admins.

With the ability to hit the limits of POSIX file modes so rapidly, you can just imagine what happens when you want to allow more complex access to your files. ACLs allow for a amazing amount of flexibility that the POSIX system simply doesn't have.

The Solution

The solution that Apple came up with was to bring ACLs to Mac OS X. The Apple model isn't quite POSIX ACLs, but rather is closer to the Windows model. This brings with it some very nice benefits.

- Mac OS X is now compatible with Windows filesystem ACLs.
- The usage model, to a degree, apes Windows; a system that is well understood.

The adoption of ACLs brings Mac OS X into the era of modern UNIX systems. The deployment of filesystem ACLs does take some planning and understanding though.

ACL Basics

Atomic Theory

It's easiest to understand the ACLs if we break down how they work and are applied before we try to understand the thirteen different modes. There are three different key system properties that allow ACLs to function:

- The GUID (Generated Unique Identifier) is a value assigned to just about everything in Mac OS X. Your user account has both a traditional UID and a new GUID.
- An ACE (Access Control Entry) is the individual rule that determines what access is given to any particular user or group.
- An ACL (Access Control List) is a list of ACEs that can be attached to a directory or file.

What filesystems can use ACLs?

Only HFS+.

Let's take a look at each of these in more detail...

Generated Unique Identifiers

The GUID is a unique identifier for just about everything in Mac OS X. Your user account, group accounts, disk volumes, just about anything really in the OS that needs to be identified as an absolute singularity.

You can view the GUID of a user or group with a variety of tools. Most users will probably use Workgroup Manager's Inspector or `dscl`. Figures 1 and 2 demonstrate how to view a user's GUID.

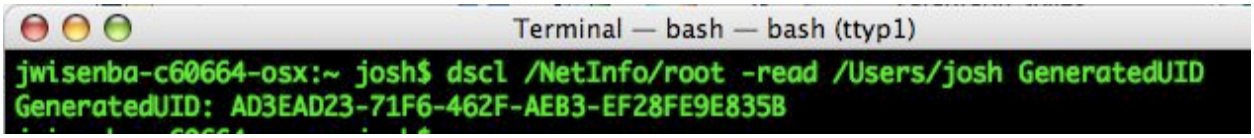


Figure 1. Viewing a user's GUID with dscl.

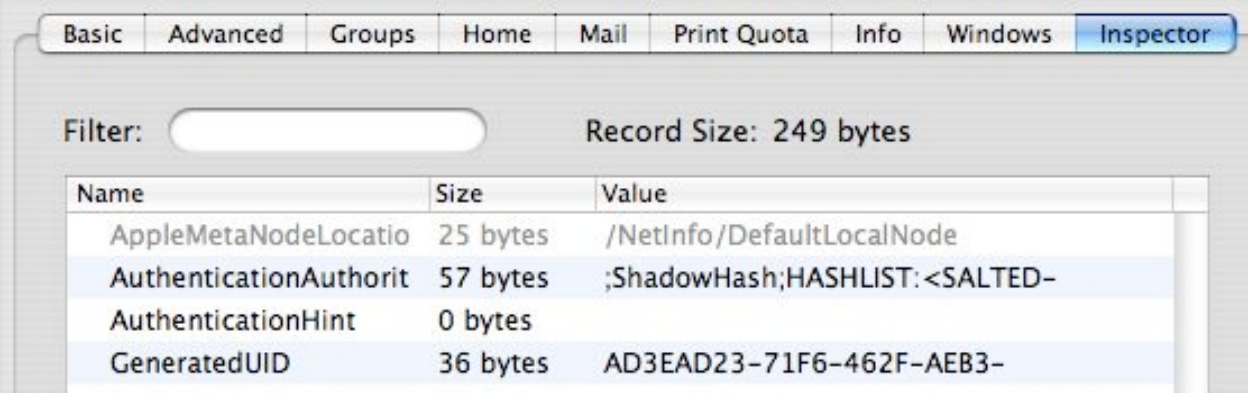


Figure 2. Viewing a user's GUID with Workgroup Manager's Inspector tab.

GUIDs are assigned to lots of of other things in Mac OS X as well. For example you can use the GUID of a filesystem to mount it via `fstab`.

Access Control Entries
ACEs are the building blocks of an ACL and they specify what controls are to be applied to a file or folder. Each ACE contains four bits of info:

- The user or group the ACE applies to. This is stored using the GUID of the record to prevent any confusion.
- The permission type, indicating if the permission to be applied is an Allow or a Deny.
- The actual permission settings that are to be allowed or denied.
- The inherited field specifies if this ACE was inherited from the parent folder.

At its core that's all there is to an ACE. It's simply a record of the four mentioned fields.

Access Control Lists
After all of the discussion about how complex ACLs can be to deploy, it might surprise you that the ACLs are not the hard part. An ACL is simply the list of ACEs that are to be applied to a file or folder. Nothing more; nothing complicated.

As we are about to see though, the devil truly is in the details...

ACL Details

Permission Settings
As I've already said, there are thirteen possible permissions that can be applied to files and folders with the new ACL model. This is a huge jump over the read-write-execute model we are used to on Mac OS X. To keep you from going insane, the permissions are broken down into three basic categories.

Administration

There are two Administration privileges:

- *Change Permissions*: User can change standard permissions.
- *Take Ownership*: User can take ownership of a file or folder.

There are five Read privileges:

- *Read Attributes*: User can view things like name, modified date, and size.
- *Read Extended Attributes*: This one is interesting. It allows the user to read 3rd party added file attributes.
- *List Folder Contents (Read Data)*: User can read the contents of a folder (And read files).
- *Traverse Folder (Execute File)*: User can open subfolders (And execute programs).
- *Read Permissions*: User can read standard permissions.

There are six Write privileges:

- *Write Attributes*: User can modify attributes like name or date.
- *Write Extended Attributes*: User can modify other added attributes.
- *Create Files (Write Data)*: User can create new files (And modify files).
- *Create Folder (Append Data)*: User can create new folders (And add data to files).
- *Delete*: User can delete the file or folder.
- *Delete Subfolder and Files*: User can delete subfolders and files.

Many of these permissions could be reached by application of special POSIX modes before, but you were severely limited in the ability to deploy them due to the User Owner, Group Owner, Others model. With the traditional permissions, for example, you can not assign different permissions for two different groups. With ACLs this is no problem at all, you simply create an ACE for each group.

ACE Application Precedence

When Mac OS X evaluates what access a user has to a folder or file it follows a strict ordering of precedence. It's actually very simple, but understanding the rules of precedence is very important. Here is how it works:

- ACLs are evaluated first.

- Mac OS X starts at the first ACE and evaluates them down the list, top to bottom.
- Mac OS X combines the ACL rights with any POSIX rights it finds for the user or group.
- If no ACEs are found to apply to a particular user or group, then only POSIX permissions are applied for that user.
- If a deny permission is found it overrides all others. Mac OS X automatically orders ACLs to put deny rules at the top of the list.

Evaluation for any particular permission stops when a match is found!

So there are a few different things we should take from this list. The first is that permissions are cumulative. If you set the standard POSIX others to `rw` then that is what anyone not specifically listed in the ACL is going to get. The second is the atomic option of the deny bomb.

Any deny will override any allow!

I would reserve the deny permission for cases when you really need it. In many cases it is sufficient to simply not grant access to others. However, use of deny is very useful when you want to restrict the permissions of a single user, but not any particular groups they are a member of.

As you might expect, trying to determine what rights a user or group is going to receive becomes quite the puzzle. Luckily, Apple has provided the very nice Effective Permissions Inspector in Workgroup Manager. This tool becomes invaluable when setting up your ACL structures.

Confused yet? So far this has been pretty simple. In most cases it isn't the actual permission model or the evaluation order that trips people up. It is the usage model that is used when deploying.

Usage and Propagation

So now we come to the real crux of most people's issues with ACLs, usage and propagation. The first part, usage, is rather simple. On Mac OS X you apply ACLs to folders, not files. Application of ACEs to files is only accomplished through inheritance. At first this seems as if it is a limitation, but it's a model that Apple has chosen in order to keep the sysadmin, i.e. you, from drowning in a sea of permissions. If you ever do find a situation where you need to set an ACL on an individual file, then you can do so with `chmod`. You do have the ability to re-apply an inherited ACL to a file in Workgroup Manager if you had removed the inherited entries.

Inheritance
When deciding how to propagate your ACLs into your directory structure there are four choices of inheritance. It's in understanding this inheritance model that most sysadmins get tripped up.

- Apply to this folder: *Applies permissions to this specific folder*
- Apply to child folders: *Apply permissions to subfolders of this specific folder*
- Apply to child files: *Apply permissions to the files in this specific folder*
- Apply to all descendants: *Apply permissions to all descendants.*

From these four methods we can divine twelve different inheritance scenarios. For example, if you wanted to set an ACL on a folder and have all new files and folders inherit the permissions you need to select all four types of inheritance.

1. Apply to this folder will set the ACL on the folder itself.
2. Apply to child folders will tell new, first level, subfolders to inherit the ACL.
3. Apply to child files will tell new, first level, files to inherit the ACL.
4. Apply to all descendants tells the OS to recurse through all subfolders and files.

Technically, the all descendants option controls the `limit_inherit` property that prevents ACEs from being further inherited.

There are two important things to see here.

The first is that the child folders and files settings only apply to the first level of any given directory; if you want to make inheritance recursive you need to use the apply to all descendants setting.

The second, and probably more commonly misunderstood part, is <u>when</u> the inheritance happens. Mac OS X Server only propagates ACLs at two, very specific, times.

- The kernel will propagate the ACLs at file or directory creation time.
- You can use Workgroup Manager, or similar administrative tool, to force a propagation.

If one of these events does not happen then there will be no propagation of ACLs. A very common problem that I see comes from a misunderstanding of the propagation rules.

Setting the inheritance will not automatically propagate ACLs to existing folders!

If you make a change to your ACLs they will not, and will never, be applied to existing folders unless you explicitly apply them yourself.

You can use the way inheritance and propagation work as a powerful tool when setting up your ACLs. Apple has a very nice little illustration of the all the possible applications of propagation on page 23 of the *File Services Administration* guide. (I'll defer to the Apple Legal Ninjas and let you look it up for yourself rather than reproduce it here.)

Deployment Example

So let's take a look at setting up a fairly simple filesystem structure for a design shop. Admittedly this example is a bit contrived as it only uses one sharepoint, but it works to illustrate the power of ACLs. For this example we will be using Workgroup Manager against a headless server.

The Players
In this scenario we are working with a graphics design shop. In this shop we have three different groups of users.

- *Managers*: This group needs full access to everything.
- *Designers*: This group needs read and write access to job folders, but not to administrative data.
- *Freelancers*: This group needs read only access to job folders, but read and write to a freelancer area and a drop box to turn finished work in for review.

In addition to these three groups we have one additional group, named *global*, that only has the other groups as members. In general I would now make a habit of this as it gives you quick access to a group that encompasses all of your users. In Workgroup Manager you can press the "Show Inherited Groups" button to see all groups a user is a member of. Figures 3 and 4 show nested and inherited groups in action.

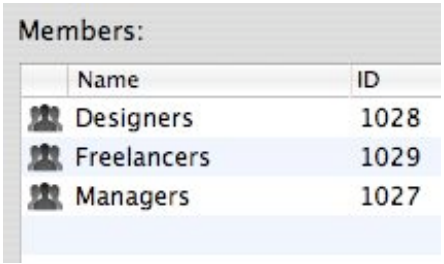


Figure 3. Nested groups inside Global.

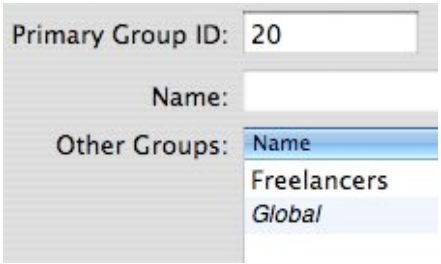


Figure 4. Inherited group in italics.

Enable Access Control Lists
The first thing that you need to do is make sure that ACLs are enabled on the volume. They should be on by default, but it never hurts to check and the setting is a bit hidden.

To find it look in the Sharing section of Workgroup Manager, click on the “All” tab, and then select the volume to enable ACLs on. Figure 5 shows the ACL checkbox location.

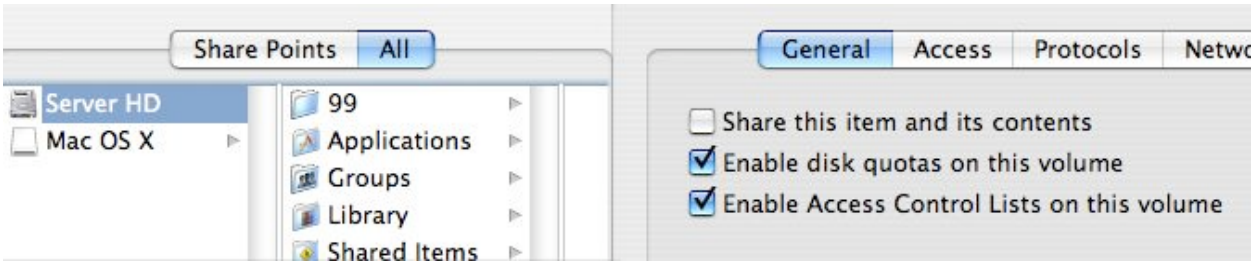


Figure 5. Enabling ACLs in Workgroup Manager.

Directory Tree Setup

Now we need to do is create a root for the sharepoint. In this case we are going to use the new, create folder button in Workgroup Manager to make a new folder named “AFP_Design”. Since the purpose of this exercise is to demonstrate ACL usage we will set the POSIX group and everyone mode to “none”. Figure 6 illustrates our basic sharepoint setup.

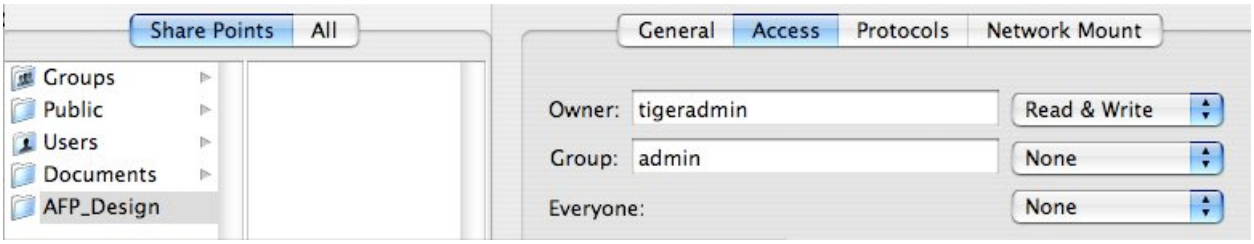


Figure 6. Root of the AFP_Design directory tree.

Inside the root of our sharepoint we are going to build our directory structure, and here is where we need to do some planning.

- The *Managers* group needs full control of everything from the root down.
- The *Designers* group needs read and write access to the Jobs, Freelance, and Freelance drop box areas. We don’t want them to have any access to the managers folder.
- The *Freelancers* group only needs read access to the Jobs folder, read and write access to the Freelance folder, and write only to the Freelance drop box.

When planning out fairly complex ACL controls it’s often easier to lay out the directories first. Inside the AFP_Design folder create three top level folders named Managers, Jobs, and Freelance. Inside Freelance make a folder named DropBox. Again, the new folder creation button in Workgroup Manager makes all of this very quick and easy. In the end it should look something like figure 7.

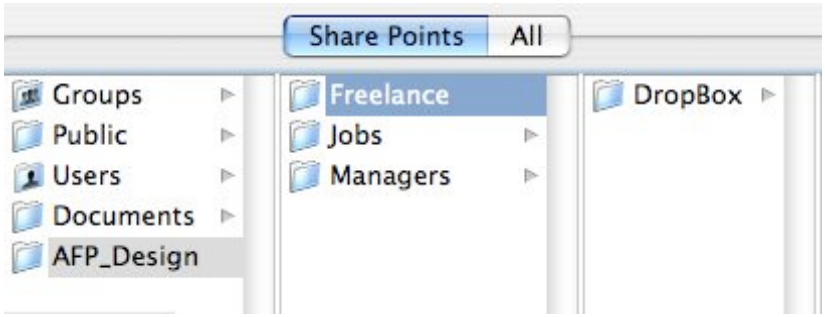


Figure 7. The AFP_Design sharepoint structure.

For this example we have a very simple directory setup. For more complicated deployments you will find it beneficial to grab a pencil and paper and plan that way.

ACL Planning and Creation

Now that we have our folders we can start building ACLs. It's best to start with the broad strokes and then focus in on the exceptions. If we think about things logically the most common right we will need is for the *Managers* to have full control of everything. The easiest way to do this is to create an ACE for them that grants full control and propagates fully. So select the AFP_Design sharepoint, click the Users & Group button, drag in the *Managers* group to the ACL pane, and select full control as the permissions setting as seen in Figure 8.

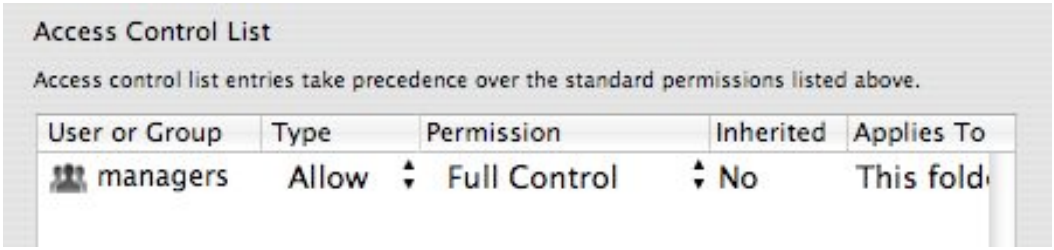


Figure 8. Managers have full control on the root of the sharepoint.

Since we pre-created our directory structure we will need to propagate the ACLs as we build them. Click the action button in the lower right corner of the Workgroup Manager window and select "Propagate permissions...". Since this is the first time we've opened this tool we should take a look at it with Figure 9.

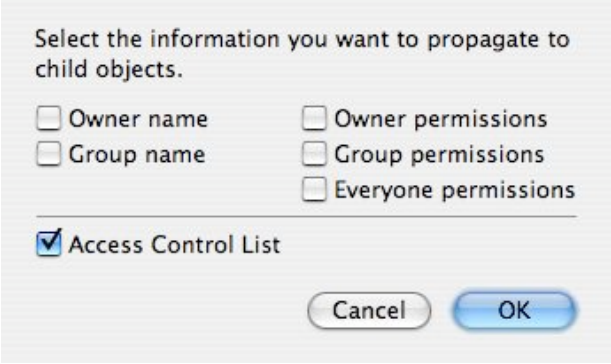


Figure 9. The permissions propagation tool.

Notice that there are several options presented. Not only can you propagate ACLs, but you can use the tool to propagate ownership and POSIX permissions as well. In our case we only want to propagate the ACL so leave it selected and click OK. A progress bar will appear to let you know that things are grinding away and once it is done we can examine the ACLs on any of the subfolders.

You must be logged into Workgroup Manager as the folder owner or root to propagate permissions!

Inherited ACEs are easy to pick out as they will not only be grayed out, but also clearly state that they are inherited as seen in Figure 10.



Figure 10. Inherited ACEs are grayed and noted.

It's one thing to see that an ACE is in an ACL, it's another to figure out what that means. It's time to take a look at the Effective Permissions Inspector (EPI), a tool that will become your best friend as you define access control policies. Here we can use the EPI to examine what rights various users will have inside our sharepoint structure. To use the EPI simply select the directory you wish to examine, click on the action button in Workgroup Manager, and select "Show Effective Permissions Inspector". Now drag any user you wish to the EPI and Mac OS X will compute, using the rules of precedence, the permissions that the user will receive.

Note that you can not use the EPI to evaluate what permissions a group will receive, only the permissions of individual users. If you were to use the Get Info window of the Finder to try to resolve permissions it will attempt to composite everything into a POSIX view of the world. It isn't always that clear in its results and you should always use the EPI when you can. Figures 11 and 12 show the results of examining the rights that a member of the *Managers* group has compared to a member of the *Freelancers* group on the DropBox directory respectively.

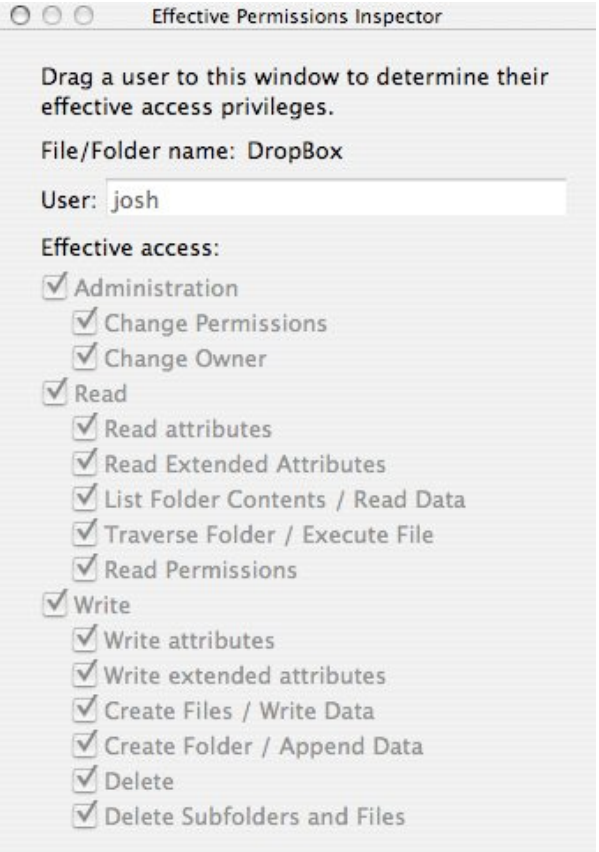


Figure 11. Josh has full control.

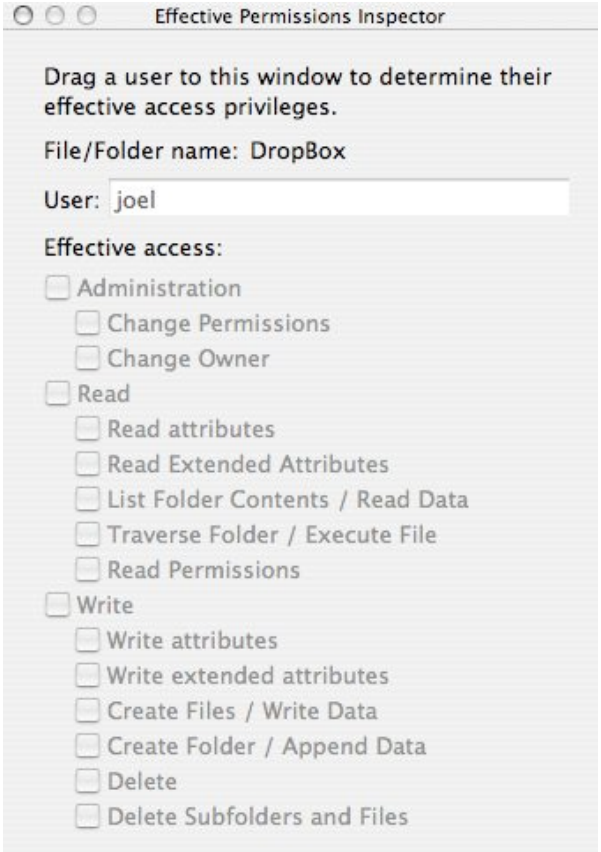


Figure 12. Joel has no access.

If we were to try and access the server via AFP as Joel he would not even see the AFP_Design sharepoint as an option. The AFP server is nice in that it does not present a share if a user has no access to it at all. This however can lead to confusion.

ACL Tuning and Troubleshooting

We know that we want the *Freelancers* users to have read and write access to the Freelance folder and write only access to the DropBox folder. We have already propagated the *Managers* group's Full Control to all of the directories, but there is nothing stopping us from defining additional explicit ACEs on any sub-folder. For that matter we could also remove any inherited ACEs or make the inherited entries explicit so that they could be modified. Both of these actions are found under the same action button as the EPI and the permissions propagation tool. For right now though we just want to add a Read and Write control for the *Freelancers* group to the Freelance folder and a Write control to the DropBox folder. In this case we don't want to propagate the permissions any further into the Freelance folder. In order to accomplish this we need to edit the ACE in greater detail. Simply select the desired ACE and then click the pencil button to edit it. As seen in Figure 13, we want to limit the propagation to just the explicit folder.

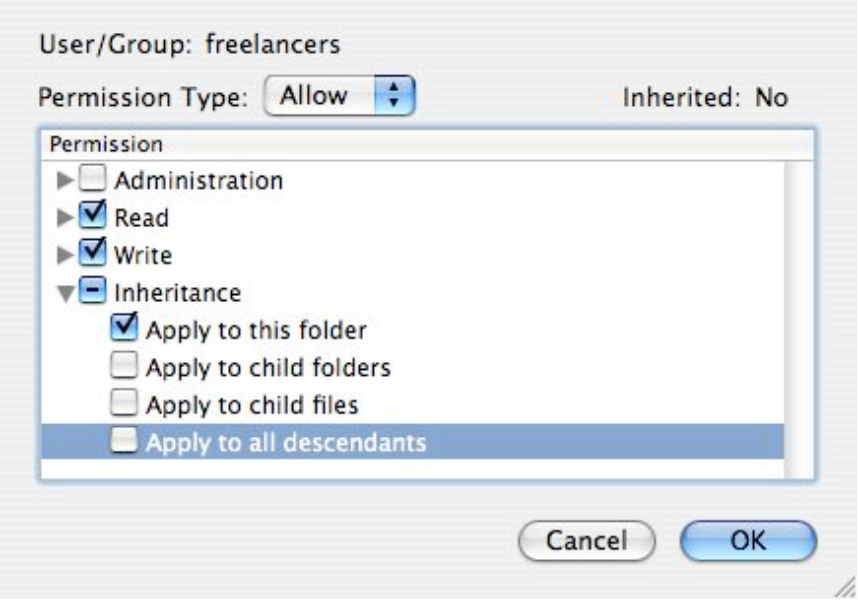


Figure 13. Controlling inheritance by editing an ACE.

Now that we've defined an explicit Read and Write for the Freelance folder for the *Freelancers* group use the EPI to check Joel's access again. Surprisingly he will still have no access and the results as seen in Figure 12 still apply! What's going on here?

Any access granted to a user on a folder is useless if he can't reach that folder!

Remember when I said that Joel couldn't even access the AFP_Design sharepoint? The EPI is smart enough to evaluate the entire directory structure and see that, even if it isn't clear to us from just eyeballing the ACLs. In this case we need to grant rights for everyone to read the sharepoint. The easiest way to do this is to add a read only ACE for our Global group to the root of the sharepoint and not propagate it into the directory structure. Once that is done we can re-evaluate Joel's access to the Freelance folder. Figures 14 shows the ACE and 15 and the results.

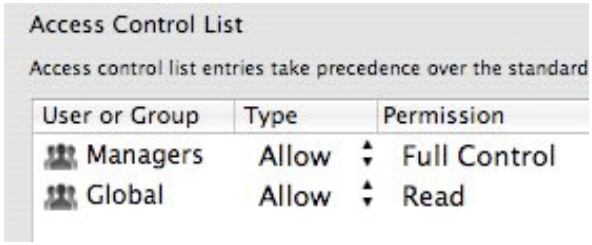


Figure 14. Adding the Global ACE to the share root.

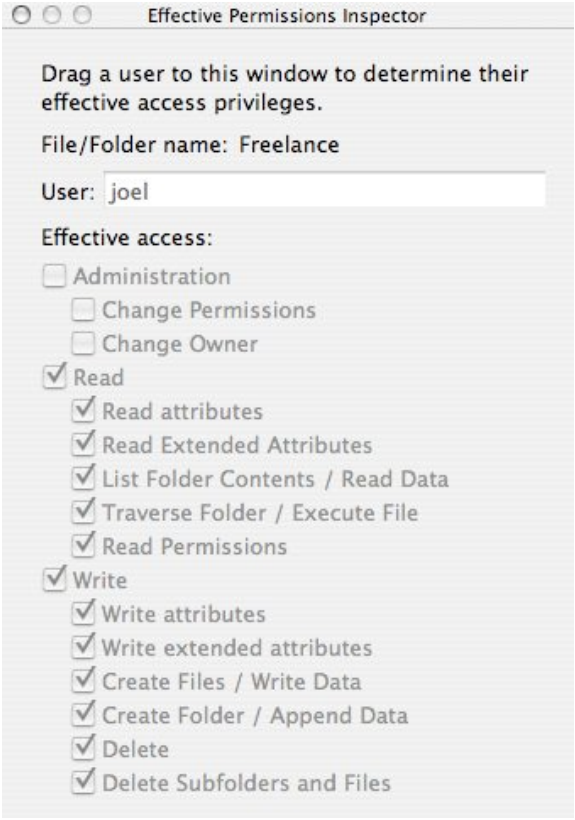


Figure 14. Joel now has access to the Freelance folder. Yea!

Notice that the default Read and Write permission gives Joel the ability to delete the Freelance folder and its subfolders. Since we don't want this you should further edit the ACE to remove the deletion rights. However since we want the *Freelancers* to be able to work freely inside this folder you should leave the Delete Subfolders and Files enabled. Check the EPI again and confirm that the deletion rights have been removed. This still isn't quite what we want though as the members of *Freelancers* can now delete the DropBox folder.

In order for a user to rename a file or folder they must have deletion rights to it.

If we enable the ability for users to delete subfolders and files of the Freelance folder then the DropBox folder is subject to that policy as a subfolder of Freelance. You would think that an explicit deny ACE for the Delete permission in the DropBox ACL would work but it won't. This is because the Delete Subfolders and Files permission is a policy of the enclosing folder. The only real solution is to remove the privilege again and add another folder, we'll call it "Projects", into the Freelance folder. Then we can restrict the *Freelancers* group rights on the Freelancer folder to Read Only, and just give them Read and Write access to the Projects folder. I would also remove the Delete rights from the Projects folder for the restricted group. If this seems like a lot of work, it is! The point we are trying to reach here is a tightly restricted system of directories to prevent those pesky contractors from messing anything up.

While we are working on the *Freelancers* group's access let's go ahead and define an ACE that gives them read only access to the Jobs folder so that they can pick up work. You should allow the ACE to inherit to all descendants of the Jobs folder to make sure that they will always have read access by default.

Now there is one last bit of work to complete the *Freelancers* group's privileges. When they login they can see the Managers folder. They can't get into it, but it's there, calling them with its siren's song of payroll reports and performance reviews. Like kids who are told they can't have something, it will gnaw at them until one day, someone will try to get in. At this point we can use the deny bomb to remove it from their, and the *Designers*', sight.

Select the Managers directory and add a Full Control Deny ACE for both the *Freelancers* and *Designers* groups. The EPI will now show us that anyone not in the *Managers* group has no access at all. None. If we connect in the Finder as Joel we will not even see the Managers folder; sorta. The Finder will attempt to show us the folder, but it will blink out of existence, and stay that way, if the user clicks on it. Figure 15 shows the Deny ACEs and Figures 16 an 17 show the effects in the Finder. (Poor, stupid Finder...)

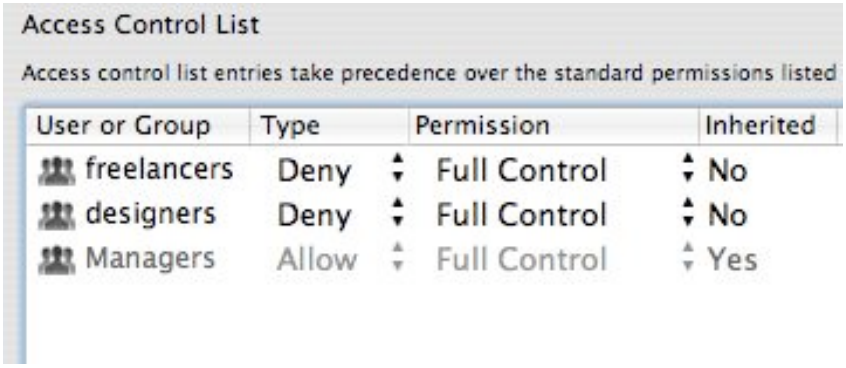


Figure 15. Adding deny ACEs to keep the Managers folder secret.

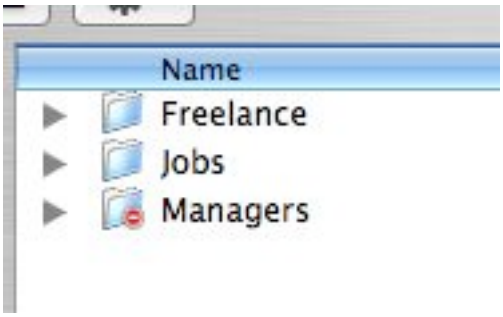


Figure 16. Without the deny ACE.

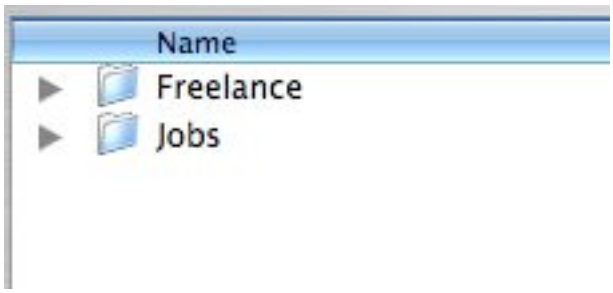


Figure 17. With the deny ACE.

Wrapping up the ACE Definitions
So now that the *Freelancers* group is finally secure the *Designers* group is easy. We can use the power of propagation to make it even easier.

The *Designers* group needs Read and Write access to everything except the Managers folder and the AFP_Design folder itself. Simply add a Read and Write ACE for the Designers group to the AFP_Design folder, then remove the Apply to This Folder inheritance style and propagate the ACL into the folder structure. This action may cause the extra propagation of some ACEs, as Figure 18 shows.

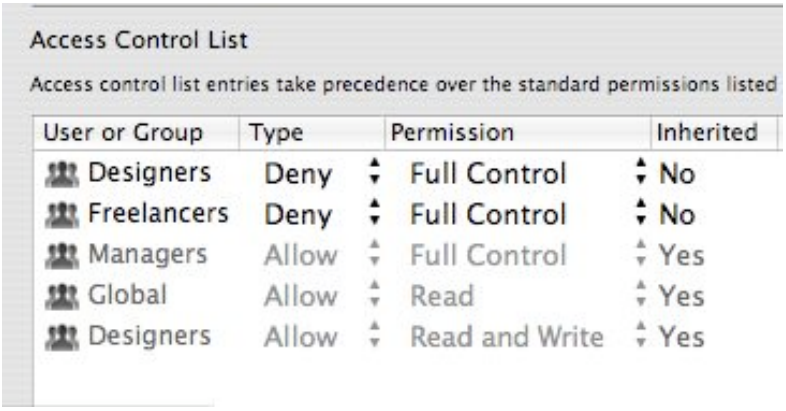


Figure 18. Extra inherited ACEs.

This happens because the ACL propagation is absolute and occurs regardless if an ACE for a user or group already exists. Luckily this is easy to work around. In the case of our Managers folder the Full Control deny ACE will take precedence, although it's probably best to remove unneeded entries from the ACL. You can remove all inherited entries with the action button, but in our case you should simply select the ones that you don't want and delete them with the minus button.

You should think carefully before removing inherited ACLs though. While you can simply reapply them to file, you can not do the same to a folder. Make sure you understand where in the folder structure the inherited entries came from and how to get them back if needed.

Test, Test, Test
Now that our ACLs are defined you should examine the folders in detail with the EPI. Really get down in there and check users from every group to make sure the permissions look like you expect. Then login with a workstation and try creating files and folders. If you are using inheritance then give it a workout as well. After creating a file or folder switch back to Workgroup Manager and take a look with the EPI tool to make sure that the permissions that were picked up were what you intended. For example, after all of our work on the *Freelancers* access it's still not right as the DropBox won't function. By using the EPI we can determine that we need to add the Traverse Folder permission to the *Freelancers* group ACE. After adding this right we can confirm the

Drop Box behavior in the Finder. WIth that last change in place our entire directory structure functions as expected.

Conclusion

ACLs are here to stay and it's best to get ahead of the curve on them now. As you explore the capabilities you will find the following advice useful.

- Keep it simple. If you don't need ACLs don't use them.
- When defining ACEs do it at a group level. Reserve user level ACEs for special cases such as deny rules.
- Speaking of deny rules, you should hold off on the deny bomb until you really need it.
- When defining the ACLs you should consult the EPI often. It is your friend.
- Use propagation to your advantage. You don't need to go crazy setting permissions on each folder when the system can do it for you.
- Make sure that you document the ACL structure that you put in place. Take screenshots in Workgroup Manager or redirect `ls -leR <sharepoint>` (See Appendix A.) to a file. This way you have a reference for working on things later.

If you follow these guidelines you can cut back on the frustrations that ACLs have brought to so many. In this case a good plan of attack is the best defense. This paper is just the beginning and only touches on the GUI tools, but in the future it will be updated to include the CLI tools as well.

As always, read the man pages and have fun!

Appendix A

ACL accounting

A very important step in ACL setup is documenting what ACEs have been applied to the files and folders. This way you have a reference to look at if things go wrong later. You can take screenshots in Workgroup Manager, but the easiest way to quickly document your setup is to use `le -leR` and redirect the output to a file. Below is the output from this command on our example directory tree. Future versions of this document will look at command line manipulation in detail.

```
sudo ls -leR AFP_Design > AFP_Design_ACLs

total 0

drwx------   4 tigeradm  admin  136 Jan 18 01:01 Freelance

drwx------   2 tigeradm  admin   68 Jan 17 22:41 Jobs

drwx------   2 tigeradm  admin   68 Jan 17 22:41 Managers



AFP_Design//Freelance:

total 0

drwx------ + 3 tigeradm  admin  102 Jan 18 01:40 DropBox

 0: group:freelancers allow
add_file,search,add_subdirectory,writeattr,writeextattr,limit_inherit

 1: group:managers inherited allow
list,add_file,search,delete,add_subdirectory,delete_child,readattr,writeattr,
readextattr,writeextattr,readsecurity,writesecurity,chown,file_inherit,direct
ory_inherit

 2: group:designers inherited allow
list,add_file,search,delete,add_subdirectory,delete_child,readattr,writeattr,
readextattr,writeextattr,readsecurity

drwx------ + 2 tigeradm  admin   68 Jan 18 01:40 Projects

 0: group:freelancers allow
list,add_file,search,add_subdirectory,delete_child,readattr,writeattr,readext
attr,writeextattr,readsecurity,file_inherit,directory_inherit

 1: group:managers inherited allow
list,add_file,search,delete,add_subdirectory,delete_child,readattr,writeattr,
```

```
readextattr,writeextattr,readsecurity,writesecurity,chown,file_inherit,direct
ory_inherit

 2: group:designers inherited allow
list,add_file,search,delete,add_subdirectory,delete_child,readattr,writeattr,
readextattr,writeextattr,readsecurity



AFP_Design//Freelance/DropBox:

total 0

drwxr-xr-x + 2 joel  admin  68 Jan 18 01:38 The big project

 0: group:freelancers inherited allow
list,add_file,search,add_subdirectory,delete_child,readattr,writeattr,readext
attr,writeextattr,readsecurity,file_inherit,directory_inherit

 1: group:managers inherited allow
list,add_file,search,delete,add_subdirectory,delete_child,readattr,writeattr,
readextattr,writeextattr,readsecurity,writesecurity,chown,file_inherit,direct
ory_inherit



AFP_Design//Freelance/DropBox/The big project:



AFP_Design//Freelance/Projects:



AFP_Design//Jobs:



AFP_Design//Managers:
```

Version History

1.0 Initial internal release

1.1 Initial public release

1.2 Corrected error (Changed “subfolders” to “files” in list item 3.) on page 9.

Clarified language about unintended ACE inheritance on page 18. (Changed “double propagated” and “double inherited” to “extra propagated” and “extra inherited”.

Cleaned up image spacing to header on page 14.

Added version history.