The Old New Thing

Moving a file does not recalculate inherited permissions

24 Aug 2006 10:00 AM | 39

Inherited permissions on an object are established when it is created. Once the object has been created, you can change the permissions of the parent and it won't have any effect unless you explicitly ask for the inheritable properties to be re-propagated to the child objects. (You may recall that the CREATOR_OWNER SID works in a similar way.) This rule applies to files, though that can lead to behavior that some people might consider non-intuitive.

Files are strange from a security perspective because a single file can have multiple parent folders, thanks to hard links. Suppose you have two directories `DirA` and `DirB`. `DirA` has an inheritable permission that gives `UserA` full access and denies access to `UserB`, while `DirB` does the same with the roles of the two users reversed. It's easy to tell who the parent of a file is when it is created, since a file is created by giving a path, and you can extract the parent from the path. For example, if the file `DirA\File` is created, it will naturally inherit permissions from `DirA`.

One the file is created, though, that's the end of it. Inheritable permissions don't have any effect any more.

This simple rule wouldn't normally cause any problems, except that files have properties unusual among most named objects: They can go by multiple names (thanks to hard links) and can change their names (via renaming). The apparently counter-intuitive behavior stems from confusing the object with its name.

For example, suppose we create a hard link to `DirA\File` under the name `DirB\File`. What effect does this have on the file's ACLs? Answer: None whatsoever. The inheritable ACLs on `DirB` aren't applied to the file since the file is not being created. Besides, you couldn't apply the inheritable ACLs from `DirB` even if you wanted to because it would create a paradox: The file resides in both `DirA` and `DirB`, but each of those directories contains contradictory inheritable permissions. What would the result be if you somehow managed to apply both of them?

All right, then. We have no choice but to decide that a file's ACLs don't change when you create a hard link. Now delete the original link `DirA\File`, leaving just `DirB\File`. Does this change the ACLs now? If you believe that it should, then you're saying that inheritable ACLs can take effect even when nothing got created! After all, we didn't create a hard link; we deleted one.

Okay, maybe you concede that deleting a hard link shouldn't affect the ACL. But what did we just do by creating a hard link and then deleting another one? The net effect is that we moved the file from `DirA\File` to `DirB\File`. Which brings us to our third example: Renaming/moving a file does not change its ACL.

We've just rediscovered the simple rule that inheritable ACLs take effect only when a file is created. Nothing special happens when a new hard link is created or when the file is moved.

Of course, that simple rule holds only when you look at the file system at a low level. Layers built on top of the low-level file system can end up complicating our simple rule.

When you move a file across volumes with the `MOVEFILE_COPY_ALLOWED` flag, you're saying that "move the file if possible; if not, then convert it to a copy/delete operation". The

copy operation **creates a new file**, which means that inheritable properties on the destination folder **do** take effect. But only if the file motion crosses volumes. If you're moving the file within the same volume, then the ACL remains unchanged. How confusing. When you pass the MOVEFILE_COPY_ALLOWED flag, you lose control of the ACL. (You actually lose control of much more than just the ACL. Since the file is being copied, none of the attributes from the original file are kept on the copy. The copy inherits its encryption and compression status from the new parent directory. The copy also gets a new owner, which has follow-on consequences for things like disk quota.)

Another layer built on top of the low-level file system operations is the shell's copy engine. If you use SHFileOperation to move a file and pass the FOF_NOCOPYSECURITYATTRIBUTES flag, then it will not preserve the original ACLs on the moved files but will rather recalculate them from the destination's inheritable properties. (If you want to do the same thing in your own code, you can call the SetNamedSecurityInfo function, specifying that you want an empty, unprotected DACL.) In Windows XP, the shell decides whether or not to use this "naive mode" ACL management based on the following algorithm:

- If the MoveSecurityAttributes policy is set, then the policy determines how ACLs are handled when files are moved. (ACLs are reset if set to "0" and are preserved if set to "1".)
- Otherwise, the "Use simple file sharing" setting controls how ACLs are handled when files are moved. (ACLs are reset if simple file sharing is enabled and are preserved if simple file sharing is disabled.)

Blog - Comment List MSDN TechNet

Comments

Larry Osterman [MSFT]
24 Aug 2006 10:04 AM
#
I got into a two day long screaming match with one of the testers in Exchange because he was upset about this behavior and insisted that inherited ACEs shouldn't change on a copy :)

Adam
24 Aug 2006 10:10 AM
#
Weird.

Any idea why inherited attributes aren't calculated when the file is accessed?

That would eliminate the need for explicit requests for inherited attributes to be propagated, and remove the "file exists in 2 locations" dilemma, because even if it exists at 2 locations, it's only accessed by one of them at a time. If you accessed it through DirA, you'd inherit permissions from DirA, and through DirB likewise.

Jon Payne
24 Aug 2006 10:13 AM
#

Why does "Use simple file sharing" affect the ACLs of copied files? If I wanted to enable or disable this behaviour, I wouldn't have guessed I needed to click on that check box.

TKW
24 Aug 2006 10:41 AM
#
@Adam - speed, I'm sure, for one although there will be other, subtler, reasons as well.

余啊雷
24 Aug 2006 10:50 AM
#
Управление правами на файлы и папки в NTFS, то есть ACL, с самого начала задумыв

Asky
24 Aug 2006 10:51 AM
#
>> All right, then. We have no choice but to decide that a file's ACLs don't change when you create a hard link.

Which brings to the question : why the ACLs are on the files and not on "links" in unix style ?

Adam
24 Aug 2006 10:56 AM
#
TKW : Speed shouldn't be an issue, as when you try to access/execute/open a file, the kernel needs to check the ACLs of all the elements of the full path anyway to make sure you're allowed to access it.

Yes, you might have read permission on the file "c:\dir\readme.txt" according to that file's ACL, but if you're not allowed to read/list/access "c:\dir", the kernel has to enforce that.

Adam
24 Aug 2006 11:03 AM
#
Asky : In Unix & derivatives, attributes are on files, not links.

Damien Guard
24 Aug 2006 11:18 AM
#
Adam, they've have to store permissions in the link then and not the file itself. I guess this would be a big change.

What is more annoying is that VisualStudio.NET seems to use the file attributes of obj\ directories to create the binaries of bin\

I guess it actually creates the binaries in obj\ and then moves them to bin\ and keeps the obj\ permissions.

[]amien

KJK::Hyperion
24 Aug 2006 11:20 AM
#
Asky, you are wrong, UNIX stores mode bits in the vnode, just like how Windows filesystems cache the security descriptor in the FCB. It's so taken for granted that the specification doesn't even mention it, and in any case how do you suppose fstat and fchmod would work?

(also note: storing and retrieving the security descriptor is left to the discretion of the filesystem. The I/O subsystem works on FILE_OBJECTs, and how do these map to the underlying files isn't specified. So it COULD be done, if not for the blatant violation of the "least surprise" principle)

Carlos
24 Aug 2006 11:38 AM
#
Adam: "the kernel needs to check the ACLs of all the elements of the full path anyway to make sure you're allowed to access it"

The kernel doesn't usually bother. Users typically have the "Bypass traverse checking" privilege, which disables the check.

Jacobo
24 Aug 2006 11:45 AM
#
Permissions are stored in the file, not in the link, to keep people from accessing/modifying/deleting files by just creating hard links to them with the appropriate permissions :)

Adam
24 Aug 2006 12:40 PM
#
Carlos : *boggle*

sandman
24 Aug 2006 12:41 PM
#
"because a single file can have multiple parent folders, thanks to hard links"

Yikes. Are you saying the windows kernel supports hardlinking directories.

Unix doesn't allow this delibarately because it introduces to many corner cases - the worst being that you need prevent cycles in hardlinks hanging the kernel. The only easy way I can see to prevent this is a depth limit.

What does windows do? (Can anyone point me at a KB article on this?)

Mike
24 Aug 2006 12:56 PM
#

> Permissions are stored in the file, not in the link, to keep people from accessing/modifying/deleting files by just creating hard links to them with the appropriate permissions :)

This seems easily solvable with a "create hardlink" permission in the ACL. That might not even be necessary because the attacker would need access to the file to create the link in the first place.

You are right though that people would have to be careful what permissions were inherited otherwise it would be easy to "lose" them.

[*This imaginary "create hardlink" permission would be a security hole. If you granted me "create hardlink", I would create a hardlink, then edit the ACLs on my hardlink to grant me full control of the file. Bingo, I've elevated from "create hardlink" to "full control". -Raymond*]

Cooney
24 Aug 2006 2:01 PM
#

A file may live in multiple directories due to hard linking, but how would you trace all those links backwards? The reference is likely stored in the directory, and there's no indexed way to get at it from the file.

Kuwanger
24 Aug 2006 2:41 PM
#

'[This imaginary "create hardlink" permission would be a security hole. If you granted me "create hardlink", I would create a hardlink, then edit the ACLs on my hardlink to grant me full control of the file. Bingo, I've elevated from "create hardlink" to "full control". -Raymond]'

So by your logic, "full control" is a security hole and not a permission setting?

Random Reader
24 Aug 2006 3:29 PM
#

Asky - you may be thinking of symlinks instead of hard links.

Adam - more on "bypass traverse checking":
http://www.sysinternals.com/blog/2005/10/bypass-traverse-checking-or-is-it.html

sandman - no, hard links are supported only for files. Read that again carefully: a _file_ can have multiple parent folders because it can be linked from two different places in the directory tree.

Kuwanger - the hypothetical scenario is that hard links have their own ACLs, controlling access to the underlying file through them, and the underlying file has a "create hardlink" privilege required for anyone to create a new link to it. In that situatation, being able to elevate your own permissions from "create hardlink" to "full control" on a file you do not have the ability to set an ACL on (by setting an ACL on the link instead) is a security hole.

microbe
24 Aug 2006 3:32 PM
#

It's a long, confusing article to express something as simple as "security attributes are stored with inodes, not file anmes".

What's the inode equivalent in Windows?

[*But that simple sentence doesn't actually explain why moving the inode doesn't update the security attributes. (The Windows term for inodes is just "files".) - Raymond*]

Adam
24 Aug 2006 4:00 PM
#

Random: Thanks for the link. Very useful.

I'm still unconvinced about whether "bypass traverse checking" is a security hole or not, but it sure sounds like an administrative nightmare. Given a heirarchy of folders and files with a mix of ACLs which, for a specific groups of users, include some files with read/write access, some with read-only access, and some with no access, imagine you want to temporarily revoke all access to the files to those users. (e.g. because you want to do an atomic snapshot of the tree, or even a restore, or because you suspect a break-in, whatever).

If users don't have the bypass-traverse checks, you can just deny them access to the top level directory for a time. Bingo.

If users do have bypass-traverse checks, how do you easily backup just the ACLs of every single file in that heirarchy so you can replace them with "no access", and then just restore all the orginal ACLs, whatever they were? Is there an easy way to do that with the administrative tools provided by the OS?

And you can't disble bypass-traverse checking for the users, as they might be relying on it for access to some other part of the OS which you don't want to change their access to.

And, AFAICT, there's no way to disable it for a portion of the OS.

So, how do Windows admins solve this? Anyone?

(With apologies for the topic-wandering)

dave
24 Aug 2006 4:51 PM
#

re: Why don't they store the ACL with each name rather than with the file?

Well, because for a start it would give you problems when opening a file in such a way that you present no name at all ;-)

The file system allows one to open a file 'by file id' (option FILE_OPEN_BY_FILE_ID in NtOpenFile).

This seems to be little used in Windows apps (I'm not even sure whether it can be expressed in Win32) although I used to use it all the time in VMS. The basic idea is that, once you've opened a file once, subsequent opens can be 'by id' and then you don't have to even think about what might happen if "foo.bar" turned out to refer to different objects between one open and the next.

Back to the main point though - it seems to me that protecting the name of an object rather than the object itself would be deeply confusing from a philosophical point of view.

dave
24 Aug 2006 4:54 PM
#

>I'm still unconvinced about whether "bypass
>traverse checking" is a security hole or not.

It is not actually a hole, in that you can always attempt to open the target file without using any path (open-by-id, see above) and thus you'd be traversing nothing.

I guess it might be a little easier to deal with names rather than guessing file ids, so it might have some practical ramifications.

Foolhardy
24 Aug 2006 5:14 PM
#

Just to clairify, if you have a directory A with a file B inside it and you use the shell's ACL editor to change inheritable ACEs on A, it'll silently update B's ACL (as long as B's DACL isn't marked protected) and any other children of A, recursively. You don't need to check the "Replace permission entries on all child objects..." option: that option also clears explicit entries and overrides protected ACLs.

This behavior can be seen with filemon, and is probably because the shell calls SetNamedSecurityInfo, which as documented "automatically propagates any inheritable access control entries (ACEs) to existing child objects, according to the rules of

inheritance."
Rules of inheritance: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthz/security/ace_inheritance_rules.asp

This is not true if you call SetKenelObjectSecurity or SetFileSecurity, which can be a point of confusion: these functions only edit the specified object or file (without propogating inheritable entries).

What's also confusing is that if you opt to replace child entries from the shell's editor, a progress dialog displaying every file modified will be shown, but if you just do normal propogation, no such dialog is shown, even when the same set of files is modified. It can make it seem like only the override option actaully touches the child files.

In any case, I appreciate any articles that help dispell some of the mystery of how NT's security system actually works.

Adam
24 Aug 2006 6:25 PM
#
Dave: But you point in your previous post that to open a file "by id", you need to have already opened it once by name first. Therefore you will have to have passed an ACL check already to have got the id to re-open the file with.

Yeah, it's not an issue if you try to open random files by their id, but - it seems weird on an FS that supports inherited permissions to allow a non-administrator to sidestep the very namespace (directory structure) that those permissions are inherited through.

Guess it's a good job I'm not in FS design. :)

Random Reader
24 Aug 2006 7:03 PM
#
The sysinternals comments also indicate that opening files by ID requires the Bypass Traverse Checking privilege anyway.

Adam, one common scenario for a hierachy with users at different privilege levels like that is with a Windows share. In that case you use the share permissions as if they were "all or nothing", and use filesystem DACLs to control things in a fine-grained way. Since everyone is accessing it through the share, when you need to temporarily block access to a specific group you just deny them at the share level.

That is, of course, specific to a Windows fileserver scenario. I'm not sure how you would approach the general case.

And yes, while the NT security system is quite flexible and powerful, keeping it all straight is one hell of an administrative nightmare.

microbe
24 Aug 2006 9:52 PM
#
"security attributes are stored with inodes, not file anmes".

: [But that simple sentence doesn't actually explain why moving the inode doesn't update the security attributes.

No it does. Moving an inode doesn't change the inode itself. It just changes directory entries (pathnames) that point to it.

Similar for hardlinks. When you create a hardlink you only creates a new directory entry, and updates the refcount on the inode, nothing else.

> [*That's explaining what happens, but not why. Why isn't the rule for inodes "When you move an inode, it updates the security attributes"? -Raymond*]

Dean Harding
24 Aug 2006 9:52 PM
#
Adam: As FoolHardy says, if you change the permission on the root folder, then everything which is set to "inherit" permissions is updated as well at that point. It's only files/folder which AREN'T set to inherit that won't be updated.

Besides, in order to get to C:\Foo\Bar.txt, you've usually got to navigate into C:\Foo first, so if you don't have permission to read that, then being able to read C:\Foo\Bar.txt isn't going to help if you don't already know the name.

Larry Osterman [MSFT]
24 Aug 2006 10:27 PM
#
Btw, full control IS a security hole. The security guys file bugs regularly on ACLs that grant full control to resources (I know).

Granting Creator-Owner full control makes sense (after all, that's the user that created the object), as does granting it to the administrator and SYSTEM (they have it already by virtue of having the take ownership privilege).

Beyond that, it doesn't necessarily make a ton of sense, certainly not for random objects. Read Howard, Leblanc and Viega's 19 Deadly Sins of Software Security, Sin #12 (Failing to Store and Protect Data Securely).

Stefan Kanthak
24 Aug 2006 10:30 PM
#
[But that simple sentence doesn't actually explain why moving the inode doesn't update the security attributes. (The Windows term for inodes is just "files".) -Raymond]

The "inode" but is not moved or changed, only it's directory entry is moved. (Since you explicitly use creation and deletion of hardlinks to model the move this operation can't cross volumes, therefore the file itself is not moved.

BTW: KB article 310316 describes ForceCopyACLWithFile and MoveSecurityAttributes. The latter is evaluated from Explorer.exe only.

James
25 Aug 2006 4:53 AM
#

Raymond: "moving an inode" has no meaning - inodes have no location or name, they're just blocks of data with permissions and timestamps. A directory is a set of name->inode mappings, which will tell you "raymond.pdf" is inode number 3892. Inode #3892, however, has no name information - and indeed may not have *any* name corresponding to it!

Cooney's wrong about there being no way to reverse this mapping (on NTFS) though: unlike Unix inodes, NTFS File Records (which are the equivalent of inodes) *do* contain names (often multiple names, thanks to the existence of 8.3 names) along with the number of the parent directory's file record (record number 5, if it's in the root directory). In addition, there is no need to open a file by name first to determine its file ID.

As an aside, this makes it possible (with sufficient access privileges!) to scan the MFT (master file table) directly looking for a particular name, then convert that into a full path, as my little TFind utility does. All the speed of Unix's 'locate' - but using the current index, not one built overnight!

So in fact, yes, you *could* calculate the inherited permissions for a given file, even when opening by ID - except for the problem Raymond pointed out about "the current user": a file can have multiple names, each in different directories!

> [*I know what an inode is; I was just asking the same questions my article was trying to answer from the Windows side. Saying "Oh, the security is in the inode" doesn't explain why the security isn't updated when the inode is "moved". (Where "moved" here is from the end-user's viewpoint of moving a file from one directory to another.) -Raymond*]

Neil
25 Aug 2006 4:58 AM
#

So if you have a doubly-linked file, and you tick the "Inherit from parent" checkbox, I assume that API just uses the parent as defined by the path that you used to update the security settings on that file?

James
25 Aug 2006 8:49 AM
#

Neil: the GUI does strange things on top of the API, as Foolhardy mentions - it tries to "inherit" things its own way, by copying ACLs where it thinks they would have been inherited from the thing you're editing. It's not a filesystem thing, or related to the real inheritance mechanism - think of it as like a search and replace on method names in your IDE, as opposed to actually inheriting methods.

dave
25 Aug 2006 9:49 AM

#

>What's the inode equivalent in Windows?

A record in the MFT (file "\$Mft").

The mapping is fairly exact - the record contains or points to the file metadata, the record is identified by an index number, a directory entries contains that index number, etc.

microbe
25 Aug 2006 3:34 PM

#

[That's explaining what happens, but not why. Why isn't the rule for inodes "When you move an inode, it updates the security attributes"? -Raymond]

Because it's consistent with any other attributes stored with an inode, such as timestamps, owner, size, content, and almost everything else.

When you move something you are just renaming it, not changing the object itself. It's like you go to change your legal name and your gender, age and hair color doesn't change. :-)

It's just a natural way to do things. I don't think anybody would have a hard time understand it once they realize name (dir entry) and inode (file object) are totally separate entities.

> [*Yet when the entirely analogous thing happens in Windows, people are confused. Go figure. -Raymond*]

Evan Anderson
26 Aug 2006 12:52 AM

#

The frustration I have w/ inherited ACEs moving when a file is moved mostly relates to the "Advanced" security dialog. The dialog shows the inherited permissions as still coming from "Parent Object", even though they came from an object that is, potentially, no longer a parent.

I guess I feel like the UI design is trying to "have it's cake and eat it too", since permissions aren't calculated at the time an access is performed, but all of the UI dialogs seem to be geared toward giving the user the impression that such a realtime calculation really is happening. The behaviour of the product is convincing enough to fool you into believing that permissions are calculated in realtime until you hit a gap in the facade like moving a file and having "inhereited" ACEs stick to it.

I don't belive that your statment "One (sic) the file is created, though, that's the end of it. Inheritable permissions don't have any effect any more." is strictly true, though. You can always add a new "inheritable" ACE at a parent folder that should alter the ACLs of child objects. This behaviour seems to be the source of another bit of frustration and curiosity to me.

I've had several cases where I've needed to add an ACE such as "Authenticated Users - List Folder Contents - This Folder Only" to a parent folder w/ several hundred-thousand

subfolders and files in Windows 2000 and later version of Windows. (I should preface by saying that I haven't tried to use any utilities to figure out what Windows is actually doing behind-the-scenes when this happens.) My workflow is to open the properties dialog on the parent folder, move to the security tab, click "Add" and enter "Authenticated Users" in the "Select Users and Groups" dialog, modify the permission in the lower portion of the security dialog to reflect "List Folder Contents", click "Advanced" to get into the "Advanced Security Settings" dialog, locate and highlight the newly-added (but not yet "Applied") "Authenticated Users - Read and Execute" permission, click "Edit" and alter the "Apply onto" line to read "This folder only", click "OK" to return to the "Advanced Security Settings" dialog, and click "OK" again to apply the new settings to the filesystem and return to the parent folder's properties sheet. When I do this I'm greeted by the properties sheet UI hanging for several minutes while, presumably, something is being done to ACLs of child objects. I've verified that the delay varies in proportion to the number of subfolders and files below the parent folder to which I make this modification. It makes no sense to me, though, since I'm applying the new ACE to "This folder only", as to why the number of subfolders or files should matter.

I'm hypothesizing that we're running thru a code path in SetNamedSecurityInfo that is processing (and, presumably, "recalculating") ACLs on all of child objects, regardless of whether or not the newly-made change should actually change the child object ACLs. That would explain the delay I'm seeing even when the new ACE is set to apply to "This folder only". (I would _love_ if there were a method to add the ACE to the parent folder w/o having to wait for this "recalculation" on child objects...)

As a fun test scenario, I moved a couple of files with inherited ACLs into a subfolder of a parent folder that had an ACL with different "inheritable" permissions than the files' original parent folder. The files original "inherited" ACLs did "stick" like I expected them to. I then added a "This folder only" ACE to the top level folder, and verified that the moved files ACLs appeared to be reset to reflect the inheritable permissions of their present location, rather than the ACLs that "stuck" from when they were moved. This seems to validate my hypothesis regarding this "recalculation" of ACLs that happens when the ACL on a parent object is changed.

The document <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/secauthz/security/automatic_propagation_of_inheritable_aces.asp> seems to say all of this, but it doesn't clearly spell out, in my opinion, what happens to these inherited ACLs when an ACE is added to a parent object. It also seems a bit non-orthagonal to me that adding an ACE to a parent object should cause these "stuck" inherited ACLs to be modified if the act of moving the file causes the inherited ACL to "stick" to the file in the first place. That just feels backwards.

Finally, it appears that Active Directory doesn't operate the same way as NTFS w/ respect to inheritance of ACL's when objects move between different OU's that have different inheritable permissions set. W/o seeing the source, of course, I can only speculate wildly, but it does appear that AD is actually calculating the permission in realtime. In contrast to NTFS, that also could be a source of confusion.

James
26 Aug 2006 4:50 AM
#

Raymond: the complexity arises from two things - first, NTFS MFT record *do* have filenames and locations stored inside them, unlike Unix inodes, so 'moving a file record' isn't entirely meaningless, unlike 'moving an inode'; secondly, the GUI's behaviour differs from the underlying API, as Evan has discovered.

Evan: When you move a file, you're seeing the file system's own behavior; when you change permissions through Explorer, you're seeing the behavior Explorer explicitly requests, overriding the underlying mechanisms. If you just added an ACE through the API, rather than using the GUI, you would see different results. Much as you'd get if you granted new rights to 'CREATOR OWNER', AIUI.

Norman Diamond
27 Aug 2006 11:01 PM
#

> [Yet when the entirely analogous thing
> happens in Windows, people are confused.
> Go figure. -Raymond]

Here's part of the reason it's confusing:

It's a checkbox that sets an option on a file (or folder), like the checkbox that sets the "readonly" option. This lends an impression that the option will be obeyed each time the file is accessed.

A button that can be clicked to do an action once, such as "do some inheriting now", might help make the effect more obvious.

Meanwhile the facts described by Foolhardy ought to make it pretty obvious why the facts are pretty confusing.

In the scenario described by Evan Anderson, a few times I didn't even guess that Explorer was working overtime to screw up my hard disk, I just thought Explorer hanged. When Explorer didn't close by itself, I let the crash report tool send dumps to Microsoft. (Maybe it hasn't been changed because the crash dumps weren't sufficiently anonymous ^_?)

Thursday, August 24, 2006 10:27 PM by LarryOsterman
> Granting Creator-Owner full control makes
> sense (after all, that's the user that
> created the object),

I still find that much confusing, and it's not obvious to me that it makes sense. Usually it's reasonable for the current owner to have full control. When the current owner isn't the creator, usually it isn't reasonable for the creator to have full control, but the creator and current owner might have agreed to transfer ownership while not transfering full control. Which of these people is the creator-owner and which of these people is something else, and what is that something else, beats me.

Proponents of DRM want to deny most rights to both the creator and the owner and grant all rights to the marketer.

Dean Harding
27 Aug 2006 11:39 PM
#

> Proponents of DRM want to deny most rights to both the creator
> and the owner and grant all rights to the marketer.

Hahaha, that's awesome! Perfect description of DRM :)

余啊雷
11 Mar 2008 10:15 AM
#

PingBack from http://blog.delphi-jedi.net/2008/03/11/moving-a-file-does-not-recalculate-inherited-permissions/