Moving a file does not recalculate inherited permissions

devblogs.microsoft.com/oldnewthing/20060824-16

August 24, 2006

Raymond Chen

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.)

Raymond Chen

Follow