Inherited access control entries are captured when the child object is created

devblogs.microsoft.com/oldnewthing/20160412-00

April 12, 2016

Raymond Chen

In the discussion of how to change permissions as fast as Explorer does it, it appears that it was not clear to people how inherited access control entries work, so there were a lot of suggestions based on faulty mental models. So let me explain.

Inherited permissions on an object are established when the object 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 child objects. (You may recall that the `CREATOR_OWNER` SID works in a similar way.)

Let's say this again. Each file has a security descriptor. To determine whether you can access a file, the only thing that is consulted is the security descriptor on the file being accessed. The security descriptors on the parent folders do not enter the picture, except possibly to determine whether traversal is allowed.[1]

Now, let's talk about these alternate theories from the comments.

"I suspect the `false` [second parameter to `ObjectSecurity.SetAccessRuleProtection`] is the problem."

The security descriptor on a file can be marked as "protected". This means "If somebody changes the security attributes on a parent folder and tries to propagate inheritable attributes to children, do not apply the inheritable attributes to me." You can think of it as an "renounce inheritance" bit on a security descriptor. If you set the bit, then you're saying, "I know my grandmother left me a collection of antique creamers shaped like animals, but I don't want them."

The `ObjectSecurity.SetAccessRuleProtection` method lets you set or clear the "renounce inheritance" bit, and if you choose to renounce the inheritance, you can set the second parameter to `false` to say that you want to renounce the access control entries that you already inherited. (Say, because they remind you of grandma's unpleasant smell.)

While that last parameter is interesting, it has nothing to do with the problem at hand, since that bit controls the security attributes of a single file and does not perform any propagation.

"Does it take Explorer two minutes to change parent directory permissions when the ACL specifies that the changes are supposed to be inherited from the parent directory? That would surprise me. (Does inheritance work by altering every descendant rather than the permission check testing ancestors?)"

When you change the security attributes of a parent folder, you are typically asked whether you want to propagate inheritable attributes to children. If you say Yes, then the security attributes of the children are *overwritten* with values that are consistent with the security attributes of the parent. It is this step (rewriting security descriptors of all children) that takes a long time.

(The Advanced Security Settings dialog assumes you always say Yes, becaus it goes hunting up the parent tree looking for the source of your inheritable ACEs, and it it doesn't find one, it just shrugs.)

Walking up the directory tree is a tricky proposition in practice for a variety of reasons. First of all, the existence of hard links means that a file can have multiple parent directories. Do you have to walk up all of them? Or only the one that was used to open the file? (In which case, it means that the effective security attributes of a file can vary depending on which path you use to open it.) And what if you open the file by using its GUID instead of a file name or path? What is the "parent directory"?

Windows takes the position that the security descriptor is an attribute of the object itself, not its containers. The container can influence the default security descriptor at the time a child object is created, but once that's done, the child object can exercise its own free will and make its own choices.

Bonus chatter: In practice, you may have a lot of files, but you almost never have a lot of unique security descriptors. For example, all the files in a directory typically have the same security descriptor, and a directory typically has the same security descriptor as its parent. Therefore, NTFS keeps all the security descriptors in a single table, and all the files with the same security descriptor share an entry in the table.

[1] Security checks on traversal are normally disabled, so by default, you can access anything whose security descriptor grants you access, as long as you know its path. You can take advantage of this: Create a directory, let's call it `Parent`, that denies *List Folder Contents* to everyone. Inside this directory, create subdirectories that grant access only to certain people, and give the names of those subdirectories to those people. For example, you might grant Bob read access to the `Manhattan` subdirectory. Bob can `cd` into `Parent\Manhattan` to see what's in it, but if Bob tries to `cd` into `Parent` and do a `dir`, he sees nothing.

Raymond Chen

Follow