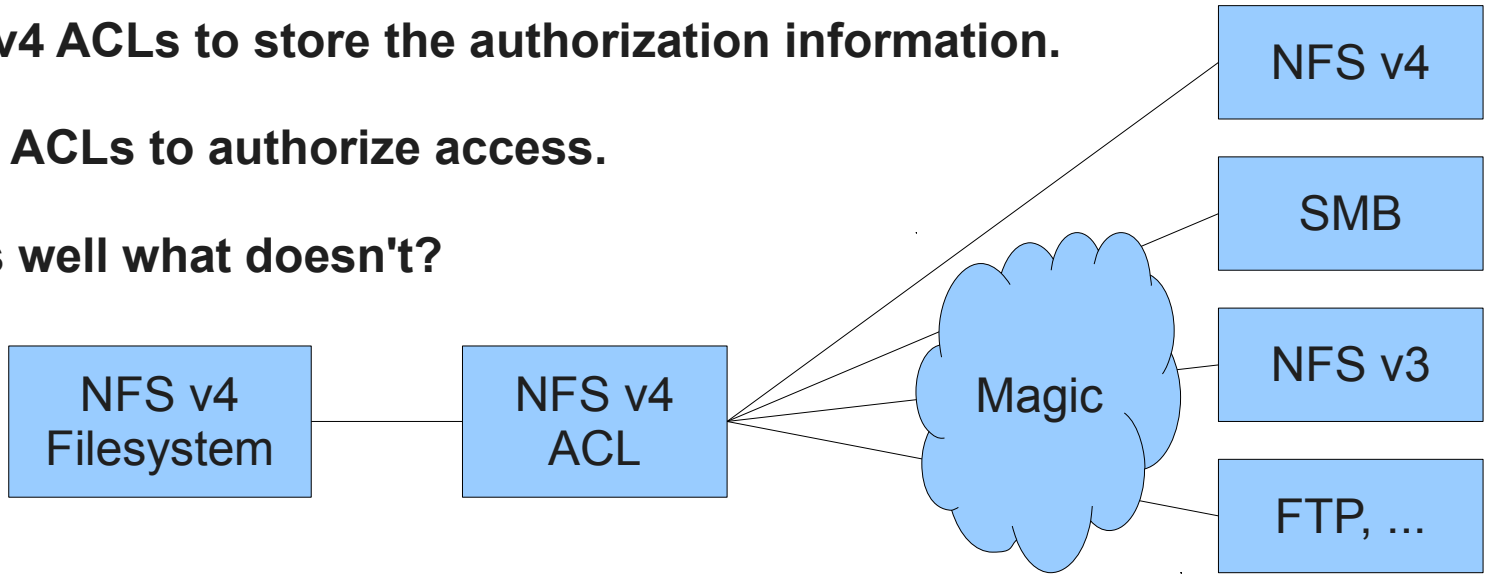
Using NFS v4 ACLs with Samba in a multiprotocol environment

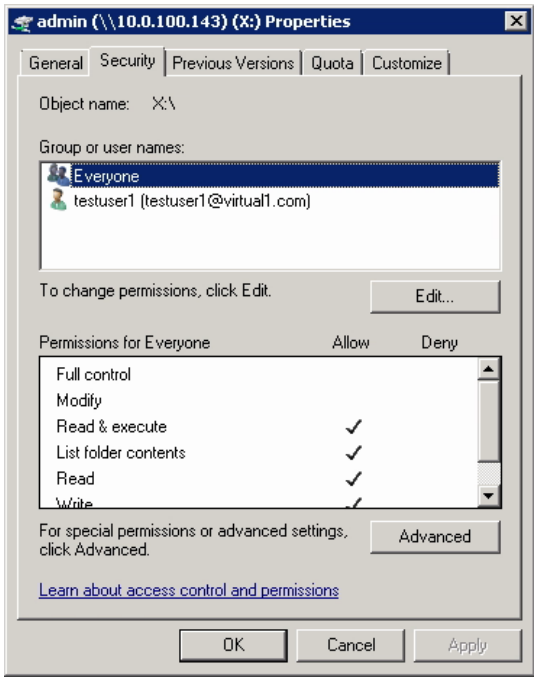
Alexander Werth IBM

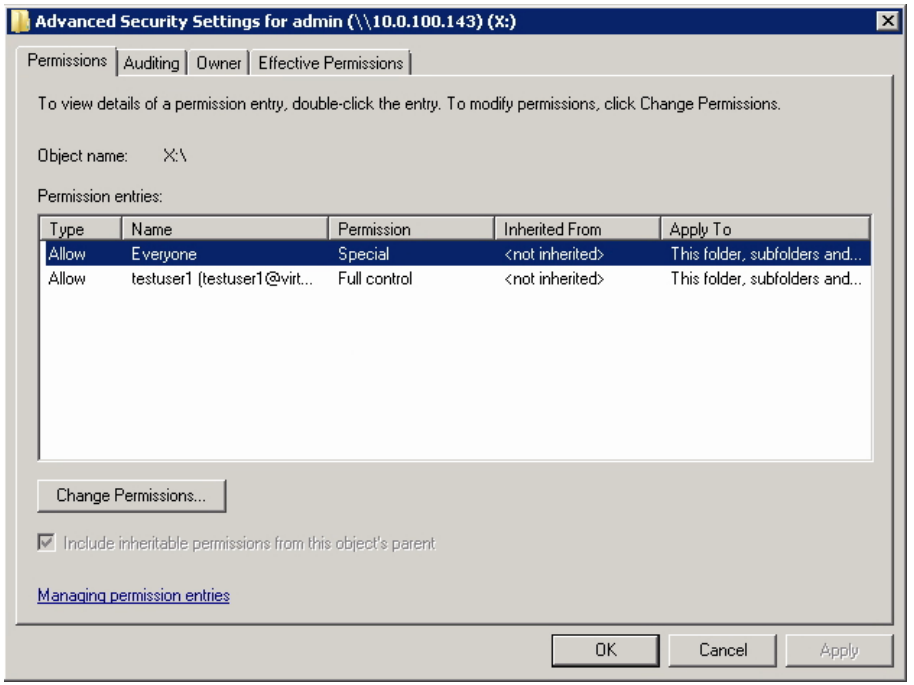
Using NFS v4 ACLs with Samba in a multiprotocol environment

- **Use multiple protocols with different authorization models to access the data.**
  - **SMB (via Samba)**
  - **NFS v4**
  - **NFS v3**
  - **FTP, SCP**
- **Using NFS v4 ACLs to store the authorization information.**
- **Use NFS v4 ACLs to authorize access.**
- **What works well what doesn't?**



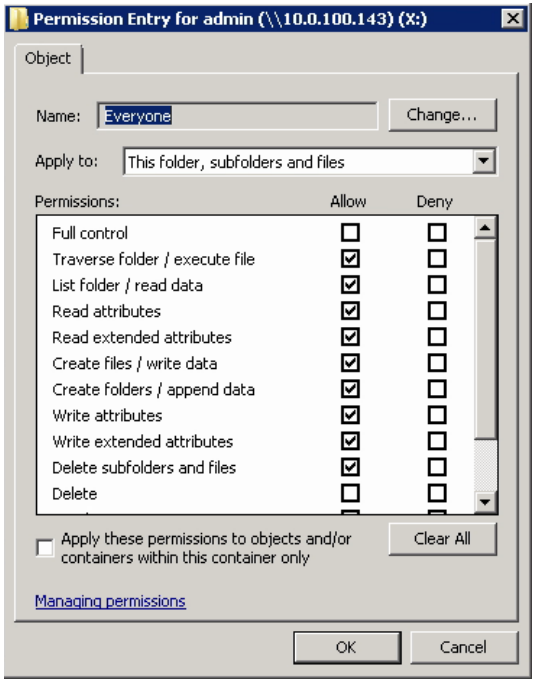
NTFS ACL





- **Default dialog to view the list of access permissions for individuals or groups**
- **ACL: Access control list**
- **ACE: Access control list entries**
- **SD: Security descriptor**
- **Type can be allow or deny**
- **Permission column contains a summary more on the next slide**
- **Inheritance to subfolders displayed in InheritedFrom, Apply To column**

NTFS ACL Permissions



- Traverse folder / execute file
- List folder / read data
- Read attributes
- Read extended attributes
- Create files / write data
- Create folders / append data
- Write attributes
- Write extended attributes
- Delete subfolders and files
- Delete
- Read permissions
- Change permissions
- Take ownership

- Fine grained control over individual permissions
- Some have a different meaning for files and folders
- Ability to select inheritance
- Checkbox Apply these permissions to objects ... within this container only

NFS v4 ACL

A list of the following permissions for individual users

LIST_DIRECTORY / READ_DATA	**(share the same bit)**
ADD_FILE / WRITE_DATA	**(share the same bit)**
ADD_SUBDIRECTORY / APPEND_DATA	**(share the same bit)**
READ_NAMED_ATTRS	
WRITE_NAMED_ATTRS	
EXECUTE	
DELETE_CHILD	
READ_ATTRIBUTES	
WRITE_ATTRIBUTES	
WRITE_RETENTION	**(new in NFS v4.1)**
WRITE_RETENTION_HOLD	**(new in NFS v4.1)**

DELETE	
READ_ACL	
WRITE_ACL	
WRITE_OWNER	
SYNCHRONIZE	**(optional)**

Source: RFC 5661, Network File System (NFS) Version 4 Minor Version 1 Protocol

Trivial NFSv4 ACL to NTFS ACL permission mapping

Designed to fit the same features as NTFS ACLs have into a unix environment

LIST_DIRECTORY / READ_DATA	**-**	**List folder / read data**
ADD_FILE / WRITE_DATA	**-**	**Create files / write data**
ADD_SUBDIR. / APPEND_DATA	**-**	**Create folders / append data**
READ_NAMED_ATTRS	**-**	**Read extended attributes**
WRITE_NAMED_ATTRS	**-**	**Write extended attributes**
EXECUTE	**-**	**Traverse folder / execute file**
DELETE_CHILD	**-**	**Delete subfolders and files**
READ_ATTRIBUTES	**-**	**Read attributes**
WRITE_ATTRIBUTES	**-**	**Write attributes**
WRITE_RETENTION		
WRITE_RETENTION_HOLD		
DELETE	**-**	**Delete**
READ_ACL	**-**	**Read permissions**
WRITE_ACL	**-**	**Change permissions**
WRITE_OWNER	**-**	**Take ownership**
SYNCHRONIZE		

Obvious mapping problem 1

WRITE_RETENTION
WRITE_RETENTION_HOLD

- **optional for NFS v4.1**
- **permission to modify retention may also be controlled by write attribute**

Map a set write attribute to also have the write retention permission set
- **We can't represent a case where this bit is set differently from write attribute.**
- **If someone uses the windows dialog to modify the ACL they will lose these bits.**

SYNCHRONIZE

A similar permission is actually definied for the NTFS ACE but not exposed.
Description indicates it can be used to toggle a particular server behavior.
Just map to the corresponding non exposed NTFS permission.
Right now that bit doesn't work due to some ZFS issue. Move that to ZFS code.

Non obvious mapping problem 1

READ_ATTRIBUTES - Read attributes

Same permission name but a sightly different semantic

NFS v4: Stat call is allowed and NFS v4.1 attributes can be read
Windows: Dos attributes (hidden, archive, etc) and timestamps can be read.

On windows the read attribute permission is not required to figure out if a file is really a file or a directory.

In windows someone with the List folder permission on the current folder can learn if something in that folder is a file or directory.

Useability vs. Security
If someone doesn't set this permission on NFS v4. Does he really want to hide if something is a file or a directory?

Different users might expect different things.

Non obvious mapping problem 2

WRITE_ATTRIBUTES - Write attributes

On NFSv4 write attribute implies the modification and creation time of a file can be changed. On window this permission doesn't allow to change these timestamps.

Also only NFS v4.1 specifies that this permission also extends to a hidden and system attribute of files.

NFS v4 recommends to deny setting of useless ACLs

The NFS v4 spec notes that some permission combinations might be hard to implement. For example the distinction between append and write.
NFS v4 recommends that a set ACL call with these invalid permission combinations get's denied.

In windows arbitrary and even invalid data can be stored in an ACL.
Windows doesn't expect that it can't store all combinations.

Unexpected problems when migrating data with invalid permission combinations.
Instead of a partial failure or limitation the entire ACL gets rejected.

But the windows GUI places restrictions on the ACLs that can be created and viewed. Getting these possible ACLs to work is a reasonable goal.

Default permission

NFS v4.1 and NTFS explicitely states that a permission that allowed or denied for a user with an ACL entry is denied by default.

The NFS v4 spec states that it's up to the implementation if a particular permission is denied or allowed by default.

There is no good way to convert an ACL with some permissions enabled by default into a NTFS ACL with the same expected behavior. Adding an entry that allows particular permissions at the end of the NTFS ACL fails because the windows UI places restrictions on the order of ACLs.

Permission implementation 1

Samba performs it's own checks checks on operations based on the SD.

That's very important for non NFS v4 filesystems with a very different authorization scheme.

Even for NFS v4 filesystems the Samba checks are useful because Samba has to map operations to system calls and there frequently isn't a system call with exactly the same semantic required by a compareable Windows call.

Checking the permissions in Samba requires to read the permissions even if the user doesn't have the “Read permissions“ permission. Read permissions only controls when permissions can be returned and not when they are evaluated.

Samba will have to read the permissions for internal use regardless of the corresponding permission bit.
→ become_root() for reading the permissions will be required.

A similar construct might be required around other calls.

Permission implementation 2

Windows allows to open a file without the read, write or execute permission. For example to read the permissions of a file.

The posix open call requires at least one of these to be specified at open.

Right now the NFS v4 read permission is required for pretty much everything.

Using become_root() around open() is a security nightmare.

Read data and read permissions is almost always required

ACL permission behavior Operation	Traverse folder / execute file [6]	List folder / read data	Read attributes	Read extended attributes [7]	Create files / write data	Create folders / append data	Write attributes	Write extended attributes	Delete subfolder and files	Delete	Read permissions	Write permissions	Take ownership
Execute file	X	X									X		
List folder		X	F										
Read data from file		X	X	X							X		
Read attributes			X										
Create file					P								
Create folder						P							
Write data to file		X	X		X	X	X	X			X		
Write file/folder attributes		X					X				X		
Delete or rename file/folder		X,P	X		P				P or X	X,P			
Read file/folder permissions		X									X		
Write file/folder permissions		X									X	X	
Take file/folder ownership		X									X		X

Requirements

What criteria can we use to determine what's right and what not?

- **Perfect ACL compatibility within one protocol**
  - **Using just SMB the behavior should match a windows server**
  - **Using just NFS v4 the behavior should match the NFS v4 spec**
  - **Using just NFS v3 the behavior should match a typical NFS v3 server**
- **Intuitive and secure mixed protocol access**
  - **Common operations should have the same meaning on all protocols.**
  - **People should have access to approximately the same operations.**
- **Support for specific common legacy features**
- **Performance should be acceptable**

Requirements Breakdown

- **Using just SMB the behavior should match a windows server**
  - **All SDs generated by the windows advanced ACL editor can be stored.**
  - **All permission bits can be stored individually**
  - **All parts of a SD that are evaluated by a windows server can be stored.**
  - **All SDs that can be stored through windows API calls can be stored and reproduced.**
  - **The effect of each permission is identical to the behavior of windows.**
- **Using just NFS v4 the behavior should match the NFS v4 spec**
  - **Trivial if the underlying filesystem is NFS v4.**
- **Using just NFS v3 the behavior should match a typical NFS v3 server**
  - **The NFS v4.1 spec already addresses how NFS v4 options are mapped to Nfs v3.**
- **Other protocols like FTP typically don't specify how to interact with ACLs.**

Applying these criteria to permissions

How good is a trivial mapping of NTFS permissions to NFS v4 permissions?

Using just SMB the behavior should match a windows server

- **All NTFS permissions can be stored individually**
  - **The filesystem might reject some combinations. They should avoid that.**
- **All SDs generated by the windows advanced ACL editor can be stored.**
- **All parts of a SD that are evaluated by a windows server can be stored.**
- **We can't store arbitrary data in the ACL.**
- **We could store the arbitrary data in an xattr since it's not relevant for NFS v4.**
- **The effect of each permission is identical to the behavior of windows.**
  - **Read permissions, Read data, ...**

Intuitive and secure mixed protocol access

- **All permissions in NTFS have a counterpart in NFS v4.**
- **But due to the implementation of the permissions it doesn't work well.**

Support for specific common legacy features

Performance should be acceptable

More than permissions in the ACL

A NTFS ACL for permissions is also called DACL
This stands for Discretionary Access Control List

An ACE in a DACL consists of:
- **The permissions.**
- **A type that describes if the ACE denies or allows operations.**
- **Information about inheritance.**
- **SID of the user or group the ACE applies to.**

A NFS v4 ACE consists of
- **The permissions.**
- **A type that can describe if the ACE denies or allows operations.**
- **Information about inheritance.**
- **A UID/GID or special entry that describes to whom the ACE applies to.**

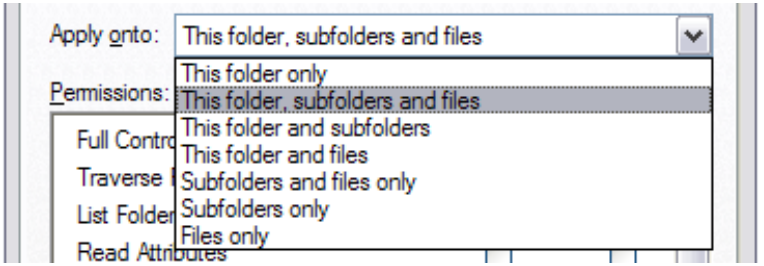
Inheritance information in ACEs

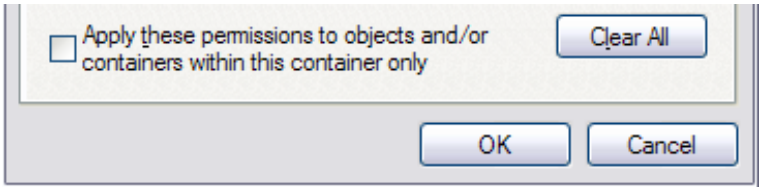
The following inheritance related information is stored on each ACE of a DACL

- **If this ACE inherits down to and applies to subfolders.**
- **If this ACE inherits down to subfolders and files and applies to files.**
- **If this ACE applies only after it has been inherited at least once.**
- **If this ACE is inherited just once.**
- **If this ACE has been inherited at all.**

There is a 1:1 mapping to similar behaving NFS v4 flags for each which is great.

In both windows and NFS the ACLs are static ACLs.
That means the ACLs are created at file creation and just the ACLs of the current file or folder have to be considered when evaluating the ACLs.
If the inheritance information is adjusted through windows it is automatically applied to each subdirectory by the windows client.





Order of ACEs in ACLs

The windows client automatically orders the ACEs of each ACL in the following way:

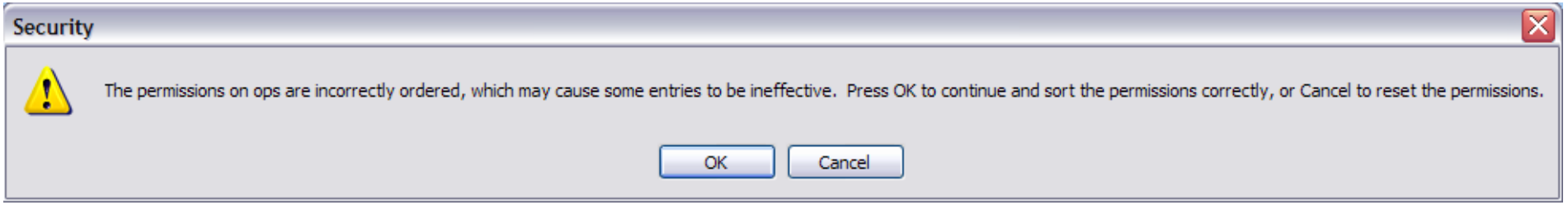
- **Deny ACEs that are new at the current file or folder.**
- **Allow ACEs that are new at the current file or folder.**
- **ACEs inherited from the parent folder in the order there.**

Because of that the order in which ACEs are added or removed doesn't matter.

NFS v4 doesn't place a restriction on the order of the ACEs in an ACL.

People creating NFS v4 ACLs are unlikely to manually create the right order. Using only allow ACEs works fine and should be recommended.

If an ACL is in the wrong order windows will already complain and suggests to reorder the ACL when the ACL is viewed and not just when it is edited.



Acting on GIDs and UIDs instead of SIDs

- **The NTFS ACE contains a SID of the user or group the ACL applies to.**
- **The SIDs of the ACE are compared to the SIDs of the security token obtained when the connection is established.**

- **The NFS v4 ACE contains either a UID or a GID or a special ID**

  → At the time a NTFS ACL is created a mapping from SID to UID/GID must exist.

- **Typically this mapping is part of a LDAP directory or AD with SFU.**

  But such a mapping might still not be available when an ACL is created:
- **SID to UID/GID mapping has not been distributed through forest of domains.**
- **The SID is a buildin SID and isn't stored in the AD.**
- **The SID is not part of the AD.**
  - **Typically this implies that the SID can't be used to authenticate with the file server and as such is irrelevant for authentication. But not for backups.**

Creator Owner

The NTFS ACE can contain SIDs that belong to creator owner or creator group.

- **These ACEs must be inherited.**
- **These ACEs can't apply to the current folder.**
- **On folder creation the server splits these ACEs:**
  - **One part is the original inheriting creator owner or creator group ACE. This is still inheriting and doesn't apply to the current folder.**
  - **A non inheriting ACE for the file creator that applies to the current file.**
- **On file creation the server just creates a ACE for the user creating a file.**
- **When trying to create a creator owner entry that applies to the current folder the windows GUI splits the entry in a way similar to the one described above.**
-

The creator group entries apply to the primary group of the file creator.

This behavior of a creator owner entry is very similar to the NFS v4 special owner@ and group@ entries with the InheritOnly flag set.
The NFS v4 filesystem doesn't split the ACE on file creation. But the inherited special owner@ ACE behaves similar to both of the split NTFS ACEs.
So it's sufficient to split these only when displaying the NFS v4 ACL through SMB.

Mode bits rwxrwxrwx

The NFS v4 spec recommends that the mode bit's are calculated based on the NFS v4 special IDs owner@, group@, and everyone@ that apply to the current file.

The mode bits for Everyone are easy to explain.

The InheritOnly owner@ and group@ special IDs are already used for creator owner.
→ This implies that for files created without SMB the mode bits will be determined by the creator owner and creator group NTFS ACEs.
→ In particular the mode bits will match the permissions of the non inheriting ACL entries of the file owner.

Consistency suggests to always store the non inheriting ACEs of the file owner as NFS v4 special IDs.

Mode bits of the owner and group match the permissions of the non inheriting ACEs for the file owner and group.

Inherited mode bits can be controlled through creator owner and creator group.

NFS v3 and mode bits rwxrwxrwx

How does NFS v3 handle ACLs?

NFS v3 implements an access call asking the file server if a user has access to a particular file.

The file server can return success or access denied.

The actual ACL implementation is transparent to NFS v3.

But: Some NFS v3 clients take the answer and most of it away.
With the exception of the mode bits.

So mode bits are important for NFS v3 access to files with NFS v4 ACLs.

More than the ACL in the SD

A windows Security Descriptor consists of more that just the DACL:

- **The DACL**
- **A SACL (System ACL that contains audit and alarm entries)**
- **A SID indicating the file owner**
- **A SID indicating the file owning group**
- **Control access bit flags**

SACL and DACL in SD vs audit ACLs

A windows SD can contain two ACLs, each with different ACE types:
- **DACL can contain allow and deny ACEs.**
- **SACL can contain audit and alarm ACEs.**

A NFS v4 ACL can contain four ACE types:
Allow, Deny, Audit, and Alarm ACEs

Storing the SACL and DACL in the same NFS v4 ACL and splitting it up again when reading the ACL seems to work.

A minor issue would be that one couldn't distinguish between an empty SACL and no SACL at all. Windows allows SACLs with size 0. But that doesn't result in a functional distinction.

And windows uses operations that will only update the DACL and not the SACL. To emulate this with NFS v4 ACLs one has to read the NFS v4 ACL to check if it contains any SACL entries that need to be preserved and write back the updated NFS v4 ACL. This is slower than just writing the ACL.

Acting on the SACL or the NFSv4 audit and alarm entries is a different matter.

SACL and DACL in SD vs audit ACLs

A windows SD can contain two ACLs, each with different ACE types:
- **DACL can contain allow and deny ACEs.**
- **SACL can contain audit and alarm ACEs.**

A NFS v4 ACL can contain four ACE types:
Allow, Deny, Audit, and Alarm ACEs

Storing the SACL and DACL in the same NFS v4 ACL and splitting it up again when reading the ACL seems to work.

A minor issue would be that one couldn't distinguish between an empty SACL and no SACL at all. Windows allows SACLs with size 0. But that doesn't result in a functional distinction.

And windows uses operations that will only update the DACL and not the SACL. To emulate this with NFS v4 ACLs one has to read the NFS v4 ACL to check if it contains any SACL entries that need to be preserved and write back the updated NFS v4 ACL. This is slower than just writing the ACL.

Acting on the SACL or the NFSv4 audit and alarm entries is a different matter.

Owner and owning group SID in SD

The windows SD contains fields for

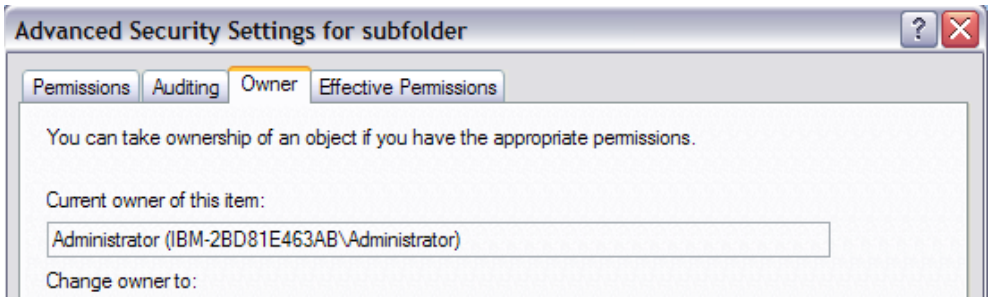
- **A SID indicating the file owner**
- **A SID indicating the file owning group**

Only one of these fields will be filled when the windows dialogs are used.

There is no corresponding field in a NFS v4 ACL but there is a file owner and file group in unix filesystems.

Standard Samba 4.0 mechanism to map these fields to the file owner and file group.

Use trick to map each ID only once for UID and GID and then store this ID in both UID and GID fields. Otherwise we don't know if the user or group mapping to SIDs should be used.



Control access bit flags

Also known as ACL flags

These contain information about how inheritance is to be computed.
Most information seems to be redundant.
Probably used for sensible recovery from errors in the ACL inheritance.

No similar data available in the NFS v4 spec.
It's up to the filesystem to store this data or use a xattr.

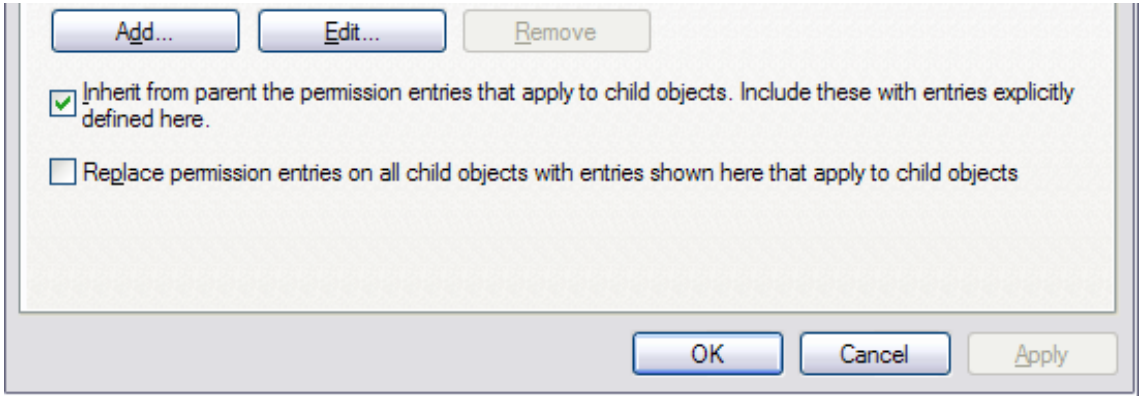
DACL Protected

The DACL Protected bit in the Control access bit flags is actually useful:

Without this bit set windows applies a heuristic to determine if an ACE is inherited from the parent folder.
If all inheriting ACEs from the parent folder show up in the ACL of the current folder windows assumes the ACL has been generated through inheritance and checks the ACL entries inherited from parent checkbox in it's ACL dialog.
That's even when the ACEs don't have the inherited bit set.

→ Breaking inheritance by unchecking the box and copying all entries doesn't work.

If the DACL Protected bit is set the box ACLs inherited from parent is unchecked.



Summary

- **Permission map very well between NFS v4 and NTFS ACLs.**
- **But not everything can be mapped perfectly between NFS v4 and NTFS ACLs.**
- **Enough can be mapped to have a useful and sensible multiprotocol behavior.**
- **More improvements are possible.**

Thanks for listening

- **Questions?**