DELL Technologies

Technical White Paper

Access Control Lists on Dell EMC PowerScale OneFS

Abstract

This document introduces access control lists (ACLs) on the Dell EMC PowerScale OneFS operating system, and shows how OneFS works internally with various ACLs to provide seamless, multiprotocol access. This document covers OneFS versions 8.0.x and later.

December 2021

Revisions

Date	Description
September 2018	Initial release
September 2019	Updated for OneFS 8.2.1; canonical ACL translation behavior is disabled for NFSv4 in OneFS 8.2.1 and later
June 2020	PowerScale rebranding
October 2021	Updated for HDFS ACL support

Acknowledgments

Author: Lieven Lin (lieven.lin@dell.com)

Table of contents

Executive summary

This document addresses three main topics:

Access control list introduction: This section provides an introduction to access control list (ACL) technology implementation on different platforms and protocols, including Microsoft Windows NTFS, SMB, NFSv4, Dell EMC PowerScale OneFS, and Linux. Examples are also demonstrated for each platform. Readers experienced with basic ACL concepts and operations may skip this section and go to section 2.

Unified file permission model in OneFS: This section introduces the OneFS unified permission model and discusses the internal functions of OneFS ACL permission mapping across different protocols. It also introduces ACL policy and permission repair job.

Considerations for permissions issues: This section discusses key considerations while troubleshooting permissions issues on OneFS, including OneFS RBAC privileges, share and export settings, and OneFS user mapping.

Audience

This document is intended for PowerScale storage administrators who want to understand the mechanism of ACL on OneFS and how file permissions are implemented for seamless multiprotocol access.

The document assumes readers have basic knowledge of the following:

- Network-attached storage (NAS) systems
- Windows operating system and Server Message Block (SMB) protocol
- Linux operating system and Network File System (NFS) protocol
- The OneFS distributed file system and multiprotocol access feature

For more information about the topics discussed in this paper, we recommend reviewing the following publications:

- Dell EMC PowerScale OneFS: A Technical Overview
- PowerScale OneFS Web Administration Guide
- PowerScale OneFS CLI Administration Guide
- Current PowerScale Software Releases

1 Introduction to ACL

This section introduces access control list (ACL) technology, and provides an overview and examples of ACL use with OneFS.

1.1 ACL overview

An ACL is a list of permissions associated with an object. It specifies which users or system processes have permissions to objects, and what operations are allowed on given objects. Each entry in a typical ACL, referred to as an access control entry (ACE), specifies a subject and an operation.

Many kinds of systems implement ACLs, for example, network hardware and file systems. On network hardware like routers and switches, each entry in an ACL specifies hosts and networks that are permitted to port numbers or IP addresses on a host that provides a service. In a file system, each entry in an ACL specifies individual user or group rights to a specific system object. This document focuses on file system ACLs.

For file systems, ACLs were first widely used in Microsoft Windows environments for NTFS and SMB. The NFSv4 ACL and OneFS ACL are derived from the Windows ACL. The expressive, Windows-style ACL is typically referred to as the **rich ACL**. This ACL defines similar permissions and inheritance. For details about ACLs on Windows NTFS and SMB, see the appendix A.1. For details about the NFSv4 ACL, see appendix A.2.

With the influence of expressive, Windows-style ACLs, a POSIX working group was formed to standardize an ACL permission model for POSIX systems. Although this initiative effort proved to be too ambitious and was abandoned, many UNIX style systems have implemented the last draft proposal, POSIX 1003.1e Draft 17, which is the POSIX ACL. When compared with Windows-style ACLs, the POSIX ACL is much less rich and only defines read, write, and execute permissions for a file-system object. For details about POSIX ACL, see appendix A.3. HDFS ACL is an Apache implementation of POSIX ACL.

1.2 OneFS ACL

OneFS provides a single namespace for multiprotocol access and it has its own internal ACL representation to perform access control. The internal ACL is presented as protocol-specific views of permissions so that NFS exports display POSIX mode bits for NFSv3, and shows ACL for NFSv4 and SMB.

When connecting to an PowerScale cluster with SSH, you can manage not only POSIX mode bits but also ACLs with standard UNIX tools such as **chmod** commands. In addition, you can edit ACL policies through the web administration interface to configure how OneFS manages permissions for networks that mix Windows and UNIX systems.

OneFS ACL design is derived from Windows NTFS ACL, and many concept definitions and operations are similar to the Windows NTFS ACL, such as ACE permissions and inheritance.

1.2.1 OneFS synthetic ACL and real ACL

To deliver cross-protocol file access seamlessly, OneFS stores an internal representation of a file-system object's permissions. The internal representation can contain information from the POSIX mode bits or the ACL.

OneFS has two types of ACLs to fulfill different scenarios:

- **OneFS synthetic ACL:** Under the default ACL policy, if no inheritable ACL entries exist on a parent directory, such as when a file or directory is created through NFS or SSH session on OneFS within the parent directory, the directory only contains POSIX mode bits permission. OneFS uses the internal representation to generate a OneFS synthetic ACL, which is an in-memory structure that approximates the POSIX mode bits of a file or directory for an SMB or NFSv4 client.
- **OneFS real ACL:** Under the default ACL policy, when a file or directory is created through SMB or when the synthetic ACL of a file or directory is modified through an NFSv4 or SMB client, the OneFS real ACL is initialized and stored on disk. The OneFS real ACL can also be initialized using the OneFS enhanced **chmod** command tool with the **+a**, **-a**, or **=a** option to modify the ACL.

1.2.2 OneFS access control entries

In contrast to the Windows DACL and NFSv4 ACL, the OneFS ACL access control entry (ACE) adds an additional identity type. OneFS ACEs contain the following information:

- **Identity name:** The name of a user or group
- **ACE type:** The type of the ACE (allow or deny)
- **ACE permissions and inheritance flags:** A list of permissions and inheritance flags separated with commas (details are provided in sections 1.2.3 and 1.2.4)

1.2.3 OneFS ACE permissions

Similar to the Windows permission level, OneFS divides permissions into the following three types:

- **Standard ACE permissions:** These apply to any object in the file system (see Table 1).
- **Generic ACE permissions:** These map to a bundle of specific permissions (see Table 2).
- **Constant ACE permissions:** These are specific permissions for file-system objects (see Table 3).

The standard ACE permissions that can appear for a file-system object are shown in Table 1.

Table 1 OneFS standard ACE permissions

ACE permission	Applies to	Description
std_delete	Directory or file	The right to delete the object
std_read_dac	Directory or file	The right to read the security descriptor, not including the SACL
std_write_dac	Directory or file	The right to modify the DACL in the object's security descriptor
std_write_owner	Directory or file	The right to change the owner in the object's security descriptor
std_synchronize	Directory or file	The right to use the object as a thread synchronization primitive
std_required	Directory or file	Maps to std_delete, std_read_dac, std_write_dac, and std_write_owner

The generic ACE permissions that can appear for a file system object are shown in Table 2.

Table 2 OneFS generic ACE permissions

ACE permission	Applies to	Description
generic_all	Directory or file	Read, write, and execute access. Maps to file_gen_all or dir_gen_all.
generic_read	Directory or file	Read access. Maps to file_gen_read or dir_gen_read.
generic_write	Directory or file	Write access. Maps to file_gen_write or dir_gen_write.
generic_exec	Directory or file	Execute access. Maps to file_gen_execute or dir_gen_execute
dir_gen_all	Directory	Maps to dir_gen_read, dir_gen_write, dir_gen_execute, delete_child, and std_write_owner.
dir_gen_read	Directory	Maps to list, dir_read_attr, dir_read_ext_attr, std_read_dac, and std_synchronize.
dir_gen_write	Directory	Maps to add_file, add_subdir, dir_write_attr, dir_write_ext_attr, std_read_dac, and std_synchronize.
dir_gen_execute	Directory	Maps to traverse, std_read_dac, and std_synchronize.
file_gen_all	File	Maps to file_gen_read, file_gen_write, file_gen_execute, delete_child, and std_write_owner.
file_gen_read	File	Maps to file_read, file_read_attr, file_read_ext_attr, std_read_dac, and std_synchronize.
file_gen_write	File	Maps to file_write, file_write_attr, file_write_ext_attr, append, std_read_dac, and std_synchronize.
file_gen_execute	File	Maps to execute, std_read_dac, and std_synchronize.

The constant ACE permissions that can appear for a file-system object are shown in Table 3.

Table 3 OneFS constant ACE permissions

ACE permission	Applies to	Description
modify	File	Maps to file_write, append, file_write_ext_attr, file_write_attr, delete_child, std_delete, std_write_dac, and std_write_owner
file_read	File	The right to read file data
file_write	File	The right to write file data
append	File	The right to append to a file
execute	File	The right to execute a file
file_read_attr	File	The right to read file attributes

ACE permission	Applies to	Description
file_write_attr	File	The right to write file attributes
file_read_ext_attr	File	The right to read extended file attributes
file_write_ext_attr	File	The right to write extended file attributes
delete_child	Directory or file	The right to delete children, including read-only files within a directory; this is currently not used for a file, but can still be set for Windows compatibility
list	Directory	List entries
add_file	Directory	The right to create a file in the directory
add_subdir	Directory	The right to create a subdirectory
traverse	Directory	The right to traverse the directory
dir_read_attr	Directory	The right to read directory attributes
dir_write_attr	Directory	The right to write directory attributes
dir_read_ext_attr	Directory	The right to read extended directory attributes
dir_write_ext_attr	Directory	The right to write extended directory attributes

1.2.4 OneFS ACL inheritance

Inheritance allows permissions to be layered or overridden as needed in an object hierarchy and allows for simplified permissions management. The semantics of OneFS ACL inheritance meanings are the same as the Windows ACL inheritance, and they are easily understandable to someone familiar with the Windows NTFS ACL inheritance. Table 4 shows the ACE inheritance flags defined in OneFS.

Table 4 OneFS ACE inheritance

ACE inheritance flag	Set on directory or file	Description
object_inherit	Directory only	Indicates an ACE applies to the current directory and files within the directory
container_inherit	Directory only	Indicates an ACE applies to the current directory and subdirectories within the directory
inherit_only	Directory only	Indicates an ACE applies to subdirectories only, files only, or both within the directory.
no_prop_inherit	Directory only	Indicates an ACE applies to the current directory or only the first-level contents of the directory, not the second-level or subsequent contents
inherited_ace	File or directory	Indicates an ACE is inherited from the parent directory

1.2.5 OneFS ACL examples

The following examples show how ACLs can be used with OneFS.

1.2.5.1 Create a OneFS synthetic ACL and real ACL

As mentioned in section 1.2.1, if a parent directory does not contain any inheritable ACL entries, a OneFS synthetic ACL will be generated in the following situations:

- A new file or directory is created within the parent directory through the SSH session to the cluster
- A new file or directory is created over NFS within the parent directory

The example shown in Figure 1 connects to a cluster with SSH and creates a parent directory that has no inheritable ACL entries.

```
Cluster01-1# ls -led parent-dir
drwxr-xr-x +  2 root  wheel  0 Jul 17 01:16 parent-dir
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: group:wheel allow dir_gen_read,dir_gen_execute
 2: everyone allow dir_gen_read,dir_gen_execute
```

Figure 1 Parent directory ACL

This example also creates a sub directory under the **parent-dir** directory using **mkdir** command, and checks the ACL of the sub directory. Shown in Figure 2, a OneFS synthetic ACL is generated according to the POSIX mode bits.

```
Cluster01-1# cd parent-dir
Cluster01-1# ls
Cluster01-1# pwd
/ifs/parent-dir
Cluster01-1# mkdir sub-dir
Cluster01-1# ls -led sub-dir
drwxr-xr-x    2 root  wheel  0 Jul 17 01:18 sub-dir
 OWNER: user:root
 GROUP: group:wheel
 SYNTHETIC ACL
 0: user:root allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: group:wheel allow dir_gen_read,dir_gen_execute
 2: everyone allow dir_gen_read,dir_gen_execute
```

Figure 2 OneFS synthetic ACL

As mentioned in section 1.2.1, a OneFS real ACL is initialized if the synthetic ACL is modified through SMB. The **sub-dir** directory ACL is edited through the Windows Explorer UI, shown in Figure 3. The first ACE is modified, and full-control permission is granted to the user.

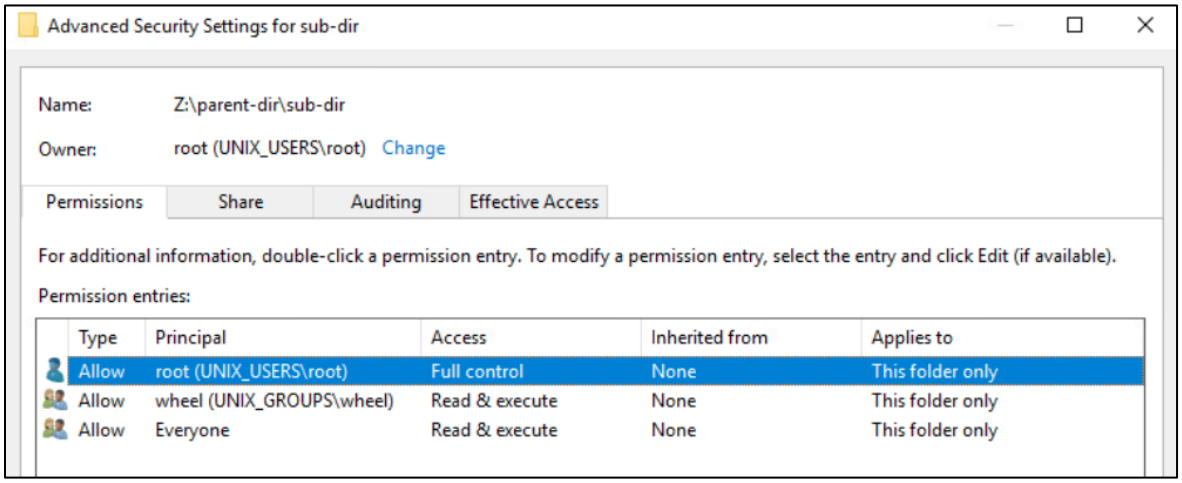

Figure 3 ACL on Windows UI

Back to cluster, the OneFS ACL is verified again using **ls –led** command. This command shows that the OneFS synthetic ACL is changed to a OneFS real ACL and the allowed permission is changed to **dir_gen_all,** shown in Figure 4.

```
Cluster01-1# ls -led sub-dir
drwxr-xr-x +  2 root  wheel  0 Jul 17 01:18 sub-dir
 OWNER: user:root
 GROUP: group:wheel
 CONTROL:dacl_auto_inherited
 0: user:root allow dir_gen_all
 1: group:wheel allow dir_gen_read,dir_gen_execute
 2: everyone allow dir_gen_read,dir_gen_execute
```

Figure 4 OneFS real ACL

You can also initialize the OneFS real ACL by modifying the OneFS synthetic ACL using the NFSv4 client tool (**nfs4_setfacl**) and the enhanced **chmod** command tool on OneFS.

1.2.5.2 Set, modify, and view a OneFS ACL

To manage and manipulate ACLs directly in OneFS, traditional **ls** and **chmod** command tools are retrofitted. They can be used to view and manage ACLs on OneFS, including adding, modifying, and deleting ACL entries. They also provide options to set ACL inheritance flags. This example shows how to use the retrofitted **ls** and **chmod** command tools to manage OneFS ACL.

To view the ACL of a file in OneFS, run the command **ls –le** or **ls –len**. To view the ACL of a directory in OneFS, run the command **ls –led** or **ls –lend.** The **–n** option in the command is used to display user and group IDs numerically rather than converting to a user or group name string.

Figure 5 shows the ACL of a directory. In the output, the **+** sign is added after the POSIX mode bits, indicating that a file or directory contains a OneFS real ACL.

```
Cluster01-1# ls -led top-dir
drwxr-xr-x +  3 root  wheel  23 Jul 16 07:10 top-dir
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_all
 1: group:wheel allow dir_gen_read,dir_gen_execute
 2: everyone allow dir_gen_read,dir_gen_execute
Cluster01-1# ls -lend top-dir
drwxr-xr-x +  3 0  0  23 Jul 16 07:10 top-dir
 OWNER: user:0
 GROUP: group:0
 0: user:0 allow dir_gen_all
 1: group:0 allow dir_gen_read,dir_gen_execute
 2: SID:S-1-1-0 allow dir_gen_read,dir_gen_execute
```

Figure 5 ACL on a directory

This example adds an ACL entry to the directory using the **chmod** command with the **+a#** option. As Figure 6 shows, the ACL entry is placed in the ACL with index **1** and allows **user01** to have the permissions of **dir_gen_read** and **dir_gen_write** for the **top-dir** directory.

```
Cluster01-1# chmod +a# 1 user user01 allow  dir_gen_read,dir_gen_execute top-dir
Cluster01-1# ls -led top-dir
drwxr-xr-x +  3 root  wheel  23 Jul 16 07:10 top-dir
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_all
 1: user:user01 allow dir_gen_read,dir_gen_execute
 2: group:wheel allow dir_gen_read,dir_gen_execute
 3: everyone allow dir_gen_read,dir_gen_execute
```

Figure 6 Add an ACL entry

To modify the ACL entry added previously, use the **chmod** command with the **=a#** option. Some shells require **=** to be escaped with the **** character. As Figure 7 shows, the ACL entry is modified only to grant the permission of **dir_gen_read** to **user01.**

```
Cluster01-1# chmod \=a# 1 user user01 allow  dir_gen_read top-dir
Cluster01-1# ls -led top-dir
drwxr-xr-x +  3 root  wheel  23 Jul 16 07:10 top-dir
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_all
 1: user:user01 allow dir_gen_read
 2: group:wheel allow dir_gen_read,dir_gen_execute
 3: everyone allow dir_gen_read,dir_gen_execute
```

Figure 7 Modify an ACL entry

To delete the ACL entry modified previously, use the **chmod** command with the **–a#** option, followed by an index of the ACE, shown in Figure 8.

```
Cluster01-1# chmod -a# 1 top-dir
Cluster01-1# ls -led top-dir
drwxr-xr-x +  3 root  wheel  23 Jul 16 07:10 top-dir
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_all
 1: group:wheel allow dir_gen_read,dir_gen_execute
 2: everyone allow dir_gen_read,dir_gen_execute
```

Figure 8 Delete an ACL entry

1.2.5.3 Use ACE inheritance

This example adds an ACL entry with the **object_inherit** and **container_inherit** inheritance flags specified, which is similar to Windows (OI)(CI) flags. This applies the ACE to the current directory and propagates it to the subdirectories and files within the **top-dir** directory.

```
Cluster01-1# chmod +a# 1 user user01 allow  dir_gen_read,dir_gen_execute,object_inherit,container_inherit top-dir
Cluster01-1# ls -led top-dir
drwxr-xr-x +  3 root  wheel  23 Jul 16 07:10 top-dir
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_all
 1: user:user01 allow dir_gen_read,dir_gen_execute,object_inherit,container_inherit
 2: group:wheel allow dir_gen_read,dir_gen_execute
 3: everyone allow dir_gen_read,dir_gen_execute
```

Figure 9 ACL with inheritance flags

The next step created a new directory under **top-dir**. Figure 10 shows the ACE with the flags of **object_inherit** and **container_inherit** specified in the parent directory, which are propagated to the new directory. There is also an **inherited_ace** flag that indicates the ACE is inherited from parent directory and not explicit specified.

```
Cluster01-1# pwd
/ifs/top-dir
Cluster01-1# mkdir dir-1
Cluster01-1# ls -led dir-1
d---r-x---  +  2 root  wheel  0 Jul 16 15:00 dir-1
 OWNER: user:root
 GROUP: group:wheel
 CONTROL:dacl_auto_inherited,sacl_auto_inherited
 0: user:user01 allow inherited dir_gen_read,dir_gen_execute,object_inherit,container_inherit,inherited_ace
```

Figure 10 Inherited ACL on a new directory

2 Unified file permission model in OneFS

2.1 OneFS AIMA and file permission checking

OneFS authentication, identity management, and authorization (AIMA) provides OneFS cluster access control, and enables multiprotocol access with a unified permission. Figure 11 shows the anatomy of AIMA OneFS cluster access control.

When a user connects to a OneFS cluster through SmartConnect or node IPs in an IP pool, OneFS finds the associated access zone with the IP pool. Then, OneFS checks the directory services of the access zone. If OneFS finds an account that matches the user's login name, OneFS verifies the user's identity to authenticate the user. During authentication, OneFS creates an access token for the user and combines the user access token from different directory services based on user-mapping rules. The token contains the user's full identity including group memberships. OneFS uses the token later to check access to directories and files according to the file permissions on disk.

For details about SmartConnect, user mapping, ID mapping, and the access token, refer to the documents PowerScale SmartConnect and Identities, Access Tokens, and the PowerScale OneFS User Mapping Service. This document focuses on file permissions on disk.

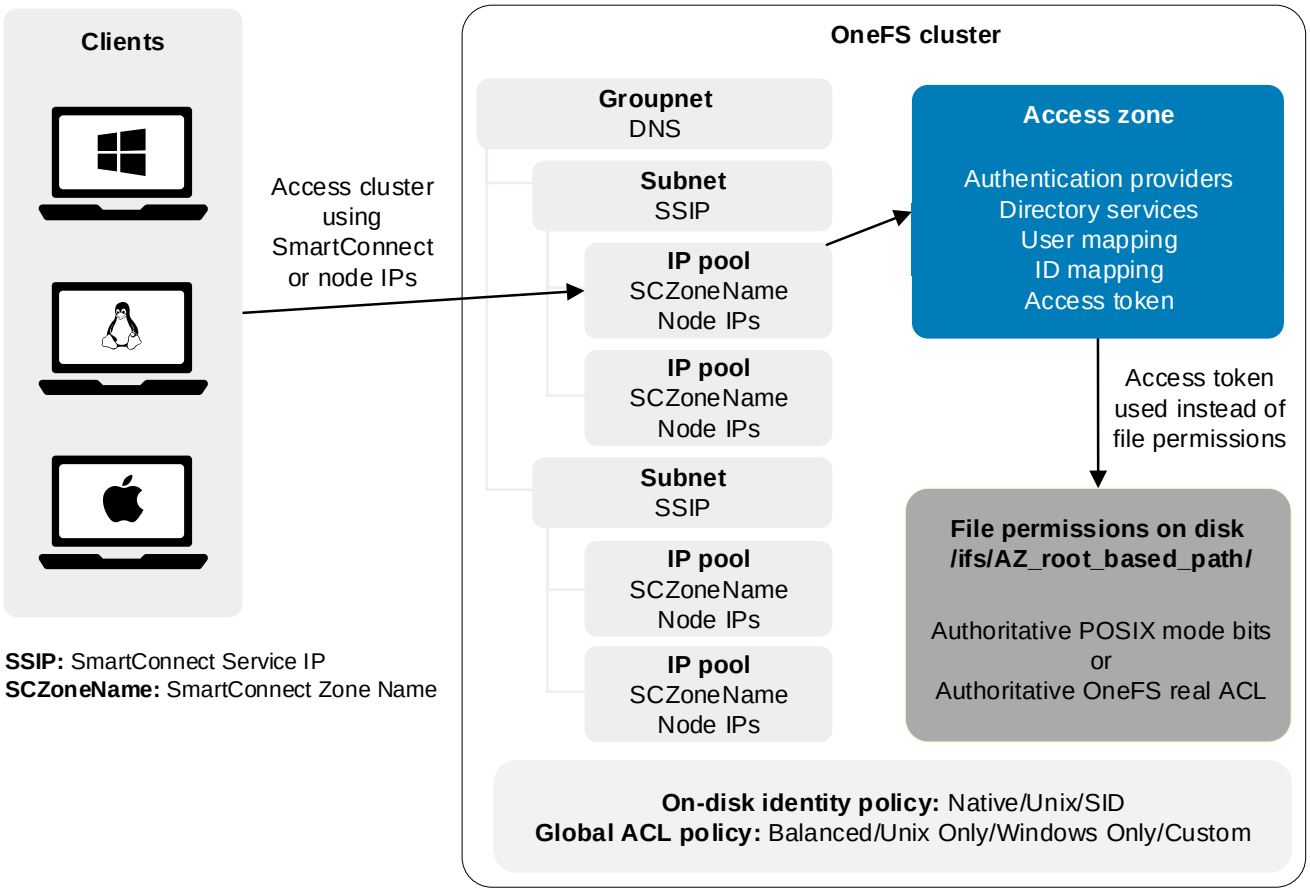

Figure 11 File permission checking

2.1.1 OneFS on-disk file permissions

OneFS unites permissions models of different protocols to provide seamless and secure file-sharing across protocols and platforms. To provide this capability, OneFS must be able to maintain unified file permissions on disk. It must also present a protocol-specific view of permissions, such as presenting POSIX mode bits to NFSv3 clients and the protocol ACL for NFSv4/SMB clients. To do this action, OneFS implements the following design for a unified-permissions model.

Only one set of permissions (POSIX mode bits or OneFS real ACL) is authoritative to preserve a file's original permissions. The view of file permissions from the PowerScale system can be in one of two states:

- Authoritative POSIX mode bits and OneFS synthetic ACL
- Authoritative OneFS real ACL and approximated POSIX mode bits for representation only

When a client accesses a OneFS cluster with different protocols, the OneFS internal file permissions are translated to a protocol-specific view of permissions for representation only. The access checking is always based on the OneFS internal file permission.

During the translation to SMB ACL, OneFS translates the internal ACL to a canonical ACL and sends it to SMB clients. A canonical ACL always places an explicit ACE before an inherited ACE, and always places a deny ACE before an allow ACE. The SMB ACL is implemented as a canonical ACL.

Since SMB clients are always presented with a reordered canonical ACL other than the actual ACL in OneFS, users need to be careful when editing ACLs through SMB.

The following example demonstrates editing an ACL through the Windows Explorer UI, and the permissions are **not** granted properly due to the order of the ACEs.

In this example, **user01** is a member of group **smbusers**. For the directory **top-dir**, **user 01** is given read, write, and execute permissions, the group **smbusers** is denied write permissions**,** and everyone else has read and execute permissions. The OneFS ACL shown in Figure 12:

```
Cluster01-1# ls -led top-dir
drwxrwxr-x +  2 root  wheel  0 Jul 20 03:32 top-dir
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: user:user01 allow dir_gen_read,dir_gen_write,dir_gen_execute
 2: group:smbusers deny add_file,add_subdir,dir_write_ext_attr,dir_write_attr
 3: everyone allow dir_gen_read,dir_gen_execute
```

Figure 12 Original on-disk ACL

As Figure 13 shows, when checking the ACL of the **top-dir** directory from Windows Explorer through the SMB protocol, the ACL representation is reordered and shown as a canonical ACL like SMB ACL. For now, the effective permission is still based on the actual OneFS ACL, and **user01** still has write permissions on the directory **top-dir.**

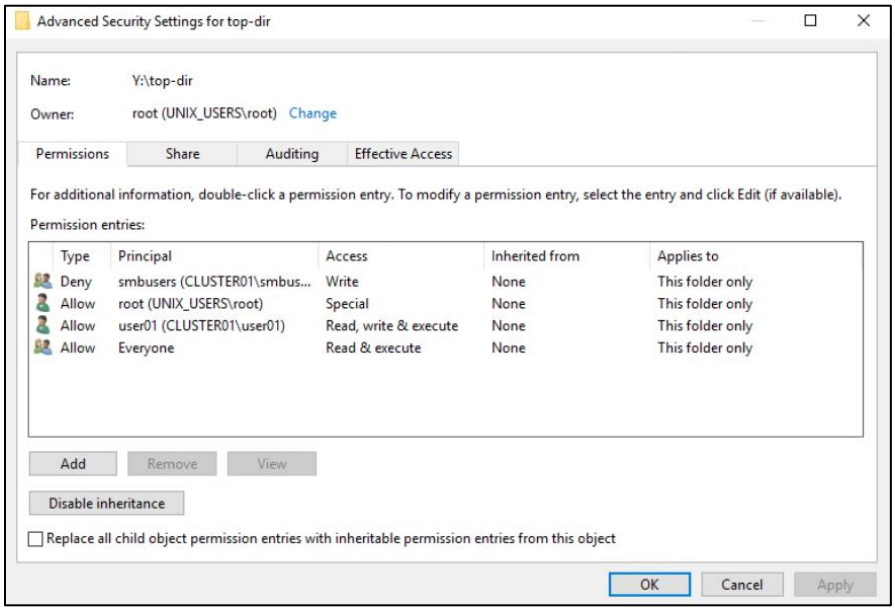

Figure 13 OneFS ACL over SMB

This example now shows modifying the ACL in Windows Explorer to remove execute permission for everyone, and keeping the rest of the ACEs as-is. The result shown in Figure 14.

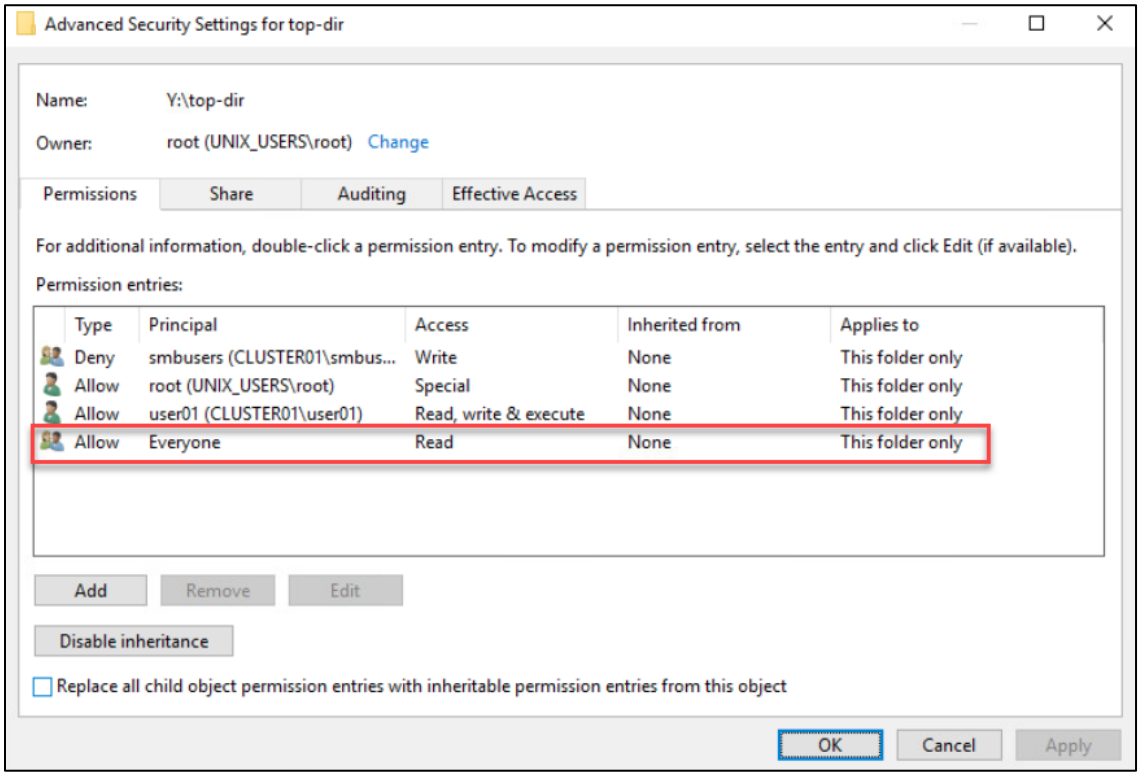

Figure 14 Modify ACL over SMB

The Windows client sends back the canonical ACL, and when checking the ACL on the OneFS cluster using the **ls –led** command, the ACE order has been changed as shown as Figure 15. The semantic of the new OneFS ACL is also changed, and it will deny write permissions for all **smbusers** group members, including user **user01.**

```
Cluster01-1# ls -led top-dir
drwxrwxr-- +  2 root  wheel  0 Jul 20 03:32 top-dir
 OWNER: user:root
 GROUP: group:wheel
 0: group:smbusers deny add_file,add_subdir,dir_write_ext_attr,dir_write_attr
 1: user:root allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 2: user:user01 allow dir_gen_read,dir_gen_write,dir_gen_execute
 3: everyone allow dir_gen_read
```

Figure 15 Reordered on-disk ACL

For OneFS 8.2.0 and earlier, when users use the **nfs4_setfacl** command to modify a file's ACL through NFSv4 on OneFS 8.2.0 and earlier, the canonical ACL translation behavior also exists for NFSv4 clients, but users can place the ACE in the original order using the **nfs4_setfacl –e** option. Be careful when manipulating a OneFS file's ACL through the SMB or NFSv4 protocol because OneFS will reorder on-disk permissions to be canonical before sending them to clients.

For OneFS 8.2.1 and above, no canonical ACL translation occurs for NFSv4 clients.

Note: For OneFS 8.2.0 and OneFS 8.1.2, Dell Technologies provides a patch to disable the canonical ACL translation behavior for NFSv4 clients. See the document Current PowerScale OneFS Patches for available patches.

2.2 ACL policy in OneFS

All of the previous examples are based on the default OneFS cluster ACL policy. The ACL policy is a cluster global setting and is not access-zone aware. It controls how permissions are processed and managed. For example, with default policy settings, OneFS ACL entries are not inheritable and can be created through the SMB or NFSv4 protocols. New permissions added by **chmod** command will be merged with a file's original ACL.

The ACL policy setting is divided into two categories: general ACL settings and advanced ACL settings.

OneFS has three default, predefined policies:

- **Balanced:** This default setting enables the OneFS cluster to operate in a mixed environment including UNIX and Windows.
- **UNIX only**: This policy provides UNIX semantics to OneFS.
- **Windows only:** This policy provides Windows semantics to OneFS.

Users can also configure both general and advanced ACL policy options manually to fulfill their requirements. However, it is **not recommended** to modify these options at will unless they are well understood and tests are conducted before applying changes to production environments. For the complete list of available options, refer to section 9 of the PowerScale OneFS Web Administration Guide. Table 5 shows the differences between the three predefined policies.

Table 5 Predefined ACL policies

ACL policy option	ACL policy option value	Balanced	UNIX only	Windows only
ACL creation through SMB	Allow ACLs to be created through SMB	✓		✓
	Do not allow ACLs to be created through SMB		✓	
Use the **chmod** command on files with existing ACLs	Merge the new permissions with the existing ACL	✓		
	Remove the existing ACL and set UNIX permissions instead		✓	
	Deny permission to modify the ACL			✓
Use the **chown** and **chgrp** commands on files with existing ACLs	Modify the owner or group and the ACL permissions	✓	✓	
	Modify only the owner or group			✓
Access checks (chmod, chown)	Allow the file owner and users with WRITE_DAC and WRITE_OWNER permissions to change the mode or owner of the file (Windows model)	✓		✓
	Allow only the file owner to change the mode or owner of the file (UNIX model)		✓	

2.2.1 ACL creation through SMB

When **ACL creation through SMB** is enabled, a OneFS real ACL is created when a new file is created over SMB. OneFS ACL can be modified with Windows Explorer commands. If creating ACLs through SMB is not allowed, when modifying the ACL through SMB in Windows, the modification will not be applied on the OneFS side.

This example uses the balanced default ACL policy and creates a new directory **smb_dir** through SMB in Windows using the account **demo\janed.** Shown in Figure 16, the OneFS real ACL is created.

```
Cluster01-1# ls -led smb_dir
drwx------ +  2 DEMO\janed  DEMO\domain users  0 Jul 23 00:38 smb_dir
 OWNER: user:DEMO\janed
 GROUP: group:DEMO\domain users
 0: user:DEMO\janed allow dir_gen_all
```

Figure 16 OneFS real ACL created through SMB

If not allowing ACLs to be created through SMB, repeat the prior procedure. The newly created directory only has POSIX mode bits and an associated synthetic ACL. See Figure 17.

```
Cluster01-1# ls -led smb_dir
drwx------    2 DEMO\janed  DEMO\domain users  0 Jul 23 00:56 smb_dir
 OWNER: user:DEMO\janed
 GROUP: group:DEMO\domain users
 SYNTHETIC ACL
 0: user:DEMO\janed allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: group:DEMO\domain users allow std_read_dac,std_synchronize,dir_read_attr
```

Figure 17 ACL creation through SMB disabled

2.2.1.1 Use chmod command on files with existing ACLs

This command has six options to control permissions when running traditional **chmod** command on files with a real ACL. Table 6 shows the options and descriptions.

Table 6 Chmod options on OneFS

Option	Description
Remove the existing ACL and set UNIX permissions instead	This option can cause information from ACLs to be lost, especially when SMB is used in the environment.
Remove the existing ACL and create an ACL equivalent to the UNIX permissions	This option also causes ACL information to be lost, although the option can create UNIX permissions instead.
Remove the existing ACL and create an ACL equivalent to the UNIX permissions, for all users or groups referenced in the old ACL	This option improves matters over the first two settings because it preserves the access of all the groups and users who were listed in the ACL.
Merge the new permissions with the existing ACL	This option merges the new permissions with the ACL, and is the recommended setting in a mixed environment because it best balances the preservation of security with the expectations of users.
Deny permission to modify the ACL	This option can cause the owner of a file to be unable to change its permission on the file.
Ignore the operation if the file has an existing ACL	Nothing will be modified on the exiting ACL.

Figure 18 shows the results of the default settings. When adding new permissions with **chmod**, the original ACL information is not lost and the new permission is merged.

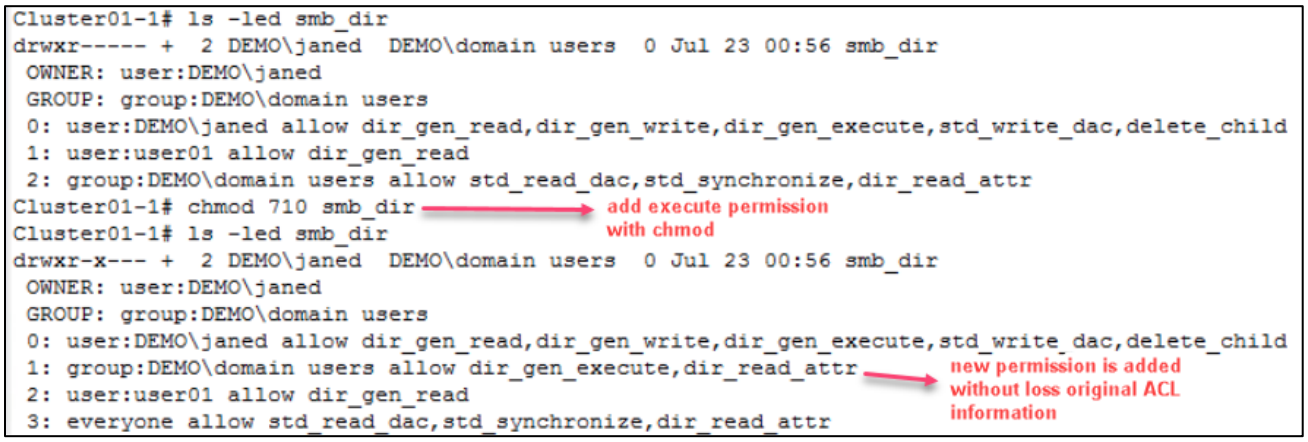

Figure 18 Merge newly added permission

Repeating the same procedure under the **UNIX only** policy, the original ACL information is removed as shown in Figure 19. If the same operation is performed under the **Windows only** policy, a **permission denied** prompt occurs.

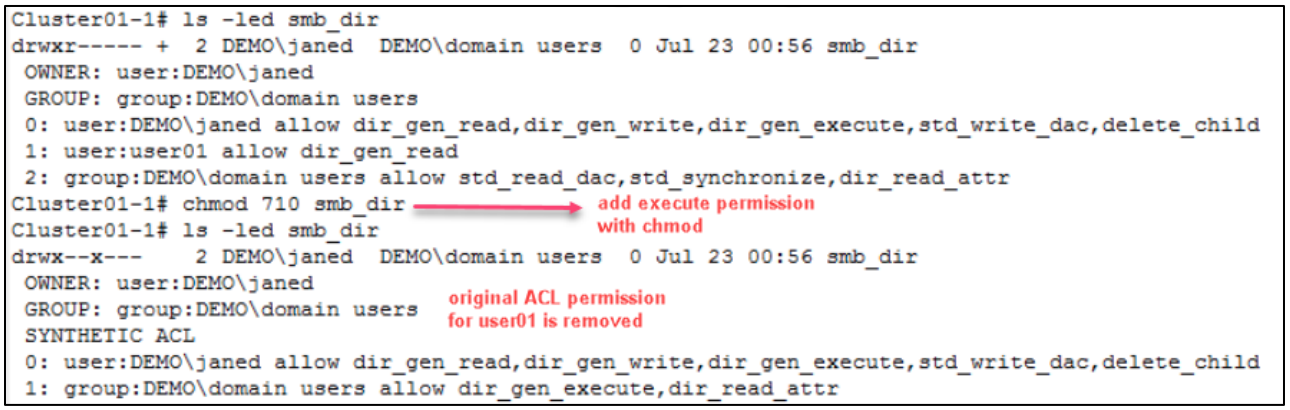

Figure 19 UNIX only behavior

2.2.1.2 Use chown and chgrp commands on files with existing ACLs

Traditionally, when the ownership of a file is modified in a Windows SMB environment, the ownership's associated ACEs are not modified and the original permissions are kept. In contrast to Windows, when attempting to change the ownership of a file in a NFS environment, the ownership's associated permissions are modified.

When running a **chown** or **chgrp** command over an NFS client on a file stored by SMB, the conflict between the two models becomes crucial. OneFS provides the option to modify the **chown** or **chgrp** behaviors to follow the Windows SMB model or NFS model. It also allows you to ignore the operation if the file has an existing OneFS real ACL.

As Figure 20 shows, under the balanced policy or UNIX only policy, OneFS changes a file's permission while changing the ownership.

```
Cluster01-1# ls -led nfsv4_dir
drwxr-xr-x +  2 user01  nfs_users  0 Jul 23 00:56 nfsv4_dir
 OWNER: user:user01
 GROUP: group:nfs_users
 0: user:user01 allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: user:user02 allow dir_gen_read
 2: group:nfs_users allow dir_gen_read,dir_gen_execute
 3: everyone allow dir_gen_read,dir_gen_execute
Cluster01-1# chown user02 nfsv4_dir
Cluster01-1# ls -led nfsv4_dir
drwxr-xr-x +  2 user02  nfs_users  0 Jul 23 00:56 nfsv4_dir
 OWNER: user:user02
 GROUP: group:nfs_users
 0: user:user02 allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: user:user02 allow dir_gen_read
 2: group:nfs_users allow dir_gen_read,dir_gen_execute
 3: everyone allow dir_gen_read,dir_gen_execute
```

Figure 20 Change ownership

Under the Windows only policy, OneFS will not change file permissions. Shown in Figure 21, OneFS only modifies the owner of the file and keeps the original ACL information.

```
Cluster01-1# ls -led nfsv4_dir
drwxr-xr-x +  2 user01  nfs_users  0 Jul 23 00:56 nfsv4_dir
 OWNER: user:user01
 GROUP: group:nfs_users
 0: user:user01 allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: user:user02 allow dir_gen_read
 2: group:nfs_users allow dir_gen_read,dir_gen_execute
 3: everyone allow dir_gen_read,dir_gen_execute
Cluster01-1# chown user02 nfsv4_dir
Cluster01-1# ls -led nfsv4_dir
drwxr-xr-x +  2 user02  nfs_users  0 Jul 23 00:56 nfsv4_dir
 OWNER: user:user02
 GROUP: group:nfs_users
 0: user:user01 allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: user:user02 allow dir_gen_read
 2: group:nfs_users allow dir_gen_read,dir_gen_execute
 3: everyone allow dir_gen_read,dir_gen_execute
```

Figure 21 Change ownership under Windows only policy

2.2.1.3 Synthetic deny ACEs

In Windows, the **everyone** identity includes every user and group including the owner and the owner group. In POSIX mode bits, the **other** identity strictly includes everybody except the owner and the owner group on the file. When the **other** identity has permissions which are not granted to the owner and the owner group, to accurately portray the POSIX mode bits semantics in a OneFS synthetic ACL, deny ACEs are needed for the owner or the owner group.

Due to the difference between everyone in Windows and other in POSIX mode bits, when generating a OneFS synthetic ACL using POSIX mode bits, this setting controls whether OneFS modifies the synthetic ACL by removing deny ACEs. This assumes there is a file containing POSIX mode bits **-rwxr-xrwx.** By default, OneFS will remove the deny write permission ACEs for the owner group, as shown in Figure 22.

```
Cluster01-1# ls -le testfile
-rwxr-xrwx    1 root  wheel  0 Sep  5 07:40 testfile
 OWNER: user:root
 GROUP: group:wheel
 SYNTHETIC ACL
 0: user:root allow file_gen_read,file_gen_write,file_gen_execute,std_write_dac
 1: group:wheel allow file_gen_read,file_gen_execute
 2: everyone allow file_gen_read,file_gen_write,file_gen_execute
```

Figure 22 Synthetic ACL with deny ACEs removed

This example changes the ACL policy setting to keep the deny ACEs for the owner group. Shown in Figure 23, the deny ACEs for the owner group are generated in the synthetic ACL.

```
Cluster01-1# ls -le testfile
-rwxr-xrwx    1 root  wheel  0 Sep  5 07:40 testfile
 OWNER: user:root
 GROUP: group:wheel
 SYNTHETIC ACL
 0: user:root allow file_gen_read,file_gen_write,file_gen_execute,std_write_dac
 1: group:wheel allow file_gen_read,file_gen_execute
 2: group:wheel deny file_write,append,file_write_ext_attr,file_write_attr
 3: everyone allow file_gen_read,file_gen_write,file_gen_execute
```

Figure 23 Synthetic ACL without deny ACEs removed

2.3 Permission repair job

Permission repair is a OneFS Job Engine job. This job provides three modes to help users fix or set a large set of file permissions. For details about how to run a permission-repair job and its concepts, refer to the Dell EMC PowerScale OneFS Permission Repair Job document.

This section shows typical use cases for each fix or set permission mode, and provides two additional examples that are not in the OneFS permission repair document. Table 7 shows the typical use cases for each mode.

Table 7 Permission repair job modes

Mode option	Use case
Clone mode	Clone mode should be used when a directory tree with many objects needs a new set of permissions such as switching from mode bits to ACLs.
Inherit mode	When access is checked in a Windows environment, this mode automatically traces all parent directories looking for inherited ACEs, adding them to the access check. In contrast to Windows, the PowerScale system handles ACL inheritance by explicitly adding the inherited ACE onto the security descriptor of the file or the directory and only newly created files can contain the inherited ACEs. The inherit mode should be used when an inherited ACE is added to a directory that contains objects.
Convert mode	The primary use case for convert mode is to update on-disk permissions after modifying the on-disk permission setting.

For an example of clone mode, refer to the Dell EMC PowerScale OneFS Permission Repair Job document. The following sections include examples for inherit mode and convert mode.

2.3.1 Inherit mode

With inherit mode, the current permissions of directories or files in a target are written over (not appended) by only the inheritable permissions of the template.

This example creates a directory and file hierarchy on OneFS as shown Figure 24, and all the files or directories under **dir1** have the same ACL permissions as their parent directory.

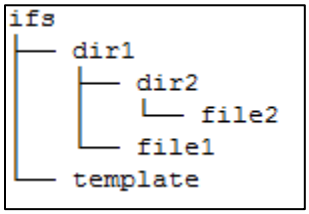

Figure 24 Directory hierarchy

The **template** directory is used as the permissions template for permission repair job, and the **template** directory ACL contains an inheritable ACE for **user01** with the flags of **object_inherit** and **container_inherit.** The ACL permissions of these files and directories are shown in Figure 25 and Figure 26.

```
Cluster01-1# ls -led template
drwxr-xr-x +  2 root  wheel  0 Jul 23 09:35 template
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: user:user01 allow dir_gen_read,object_inherit,container_inherit
 2: group:wheel allow dir_gen_read,dir_gen_execute
 3: everyone allow dir_gen_read,dir_gen_execute
```

Figure 25 Template directory

```
Cluster01-1# ls -led dir1
drwxrwxr-- +  3 root  wheel  45 Aug 24 08:54 dir1
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: group:wheel allow dir_gen_read
 2: user:user01 allow dir_gen_all
 3: everyone allow dir_gen_read
Cluster01-1# ls -led dir1/dir2
drwxrwxr-- +  2 root  wheel  23 Aug 24 08:12 dir1/dir2
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: group:wheel allow dir_gen_read
 2: user:user01 allow dir_gen_all
 3: everyone allow dir_gen_read
Cluster01-1# ls -le dir1/file1
-rwxrwxr-- +  1 root  wheel  0 Aug 24 08:12 dir1/file1
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow file_gen_read,file_gen_write,file_gen_execute,std_write_dac
 1: group:wheel allow file_gen_read
 2: user:user01 allow file_gen_all
 3: everyone allow file_gen_read
Cluster01-1# ls -le dir1/dir2/file2
-rwxrwxr-- +  1 root  wheel  0 Aug 24 08:12 dir1/dir2/file2
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow file_gen_read,file_gen_write,file_gen_execute,std_write_dac
 1: group:wheel allow file_gen_read
 2: user:user01 allow file_gen_all
 3: everyone allow file_gen_read
```

Figure 26 ACLs of target directories and files

The example now starts the permission repair job using the command **isi job jobs start,** and waits for the job to complete as shown in Figure 27.

```
Cluster01-1# isi job jobs start PermissionRepair --mode=inherit --template=/ifs/template/ --paths=/ifs/dir1
Started job [222]
Cluster01-1# isi job view 222
               ID: 222
             Type: PermissionRepair
            State: Succeeded
           Impact: Low
           Policy: LOW
              Pri: 5
            Phase: 1/1
       Start Time: 2018-07-23T09:43:59
     Running Time: 4s
     Participants: 1, 2, 3, 4
         Progress: Processed 2 files and 2 directories;0 errors
Waiting on job ID: -
      Description:
```

Figure 27 Run job with inherit mode

Checking the ACL permissions of the files and directory under **dir1**, the inherit mode of the permission repair job clones all permissions of the template to the parent directory **dir1**, and only the inheritable permission for **user01** is propagated and written over the existing permissions of the subdirectories and files.

```
Cluster01-1# ls -led dir1
drwxr-xr-x +  3 root  wheel  45 Aug 24 08:54 dir1
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: user:user01 allow dir_gen_read,object_inherit,container_inherit
 2: group:wheel allow dir_gen_read,dir_gen_execute
 3: everyone allow dir_gen_read,dir_gen_execute
Cluster01-1# ls -led dir1/dir2
d---r----- +  2 root  wheel  23 Aug 24 08:12 dir1/dir2
 OWNER: user:root
 GROUP: group:wheel
 0: user:user01 allow dir_gen_read,object_inherit,container_inherit
Cluster01-1# ls -le dir1/file1
----r----- +  1 root  wheel  0 Aug 24 08:12 dir1/file1
 OWNER: user:root
 GROUP: group:wheel
 0: user:user01 allow file_gen_read
Cluster01-1# ls -le dir1/dir2/file2
----r----- +  1 root  wheel  0 Aug 24 08:12 dir1/dir2/file2
 OWNER: user:root
 GROUP: group:wheel
 0: user:user01 allow file_gen_read
```

Figure 28 ACLs after the job has completed

2.3.2 Convert mode

Convert mode converts all file and directory on-disk identities under the target to the type specified, including **global**, **sid**, **unix,** or **native**. No template required in this mode.

The **ls** command is used to view the file and directory on-disk identity type of **user01**. The current on-disk identity for **user01** is its UID of **2002**.

```
Cluster01-1# ls -lend dir1
drwxr-xr-x +  3 0  0  45 Jul 23 09:30 dir1
 OWNER: user:0
 GROUP: group:0
 0: user:0 allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: user:2002 allow dir_gen_read,object_inherit,container_inherit,inherit_only
 2: group:0 allow dir_gen_read,dir_gen_execute
 3: SID:S-1-1-0 allow dir_gen_read,dir_gen_execute
Cluster01-1# ls -lend dir1/dir2
d---r----- +  2 0  0  23 Jul 23 09:30 dir1/dir2
 OWNER: user:0
 GROUP: group:0
 0: user:2002 allow dir_gen_read,object_inherit,container_inherit
Cluster01-1# ls -lend dir1/file1
----r----- +  1 0  0  0 Jul 23 09:30 dir1/file1
 OWNER: user:0
 GROUP: group:0
 0: user:2002 allow file_gen_read
Cluster01-1# ls -lend dir1/dir2/file2
----r----- +  1 0  0  0 Jul 23 09:30 dir1/dir2/file2
 OWNER: user:0
 GROUP: group:0
 0: user:2002 allow file_gen_read
```

Figure 29 On-disk identity: UID/GID

This example starts the permission-repair job using the command **isi job jobs start,** and waits for the job to complete as shown in Figure 30.

```
Cluster01-1# isi job jobs start PermissionRepair --mode=convert --paths=/ifs/dir1 --mapping-type=sid --zone=System
Started job [238]
Cluster01-1# isi job view 238
               ID: 238
             Type: PermissionRepair
            State: Succeeded
           Impact: Low
           Policy: LOW
              Pri: 5
            Phase: 1/1
       Start Time: 2018-07-24T02:46:48
     Running Time: 4s
     Participants: 1, 2, 3, 4
         Progress: Processed 2 files and 2 directories;0 errors
Waiting on job ID: -
      Description:
```

Figure 30 Run job with convert mode

Checking the file and directory permissions again, the on-disk permission for **user01** is converted into SID mode. The identities of **root** with UID **0** and **everyone** with SID **S-1-1-0** are special identities, and they are not converted in the converted mode.

```
Cluster01-1# ls -lend dir1
drwxr-xr-x +  3 0  0  45 Jul 23 09:30 dir1
 OWNER: user:0
 GROUP: group:0
 0: user:0 allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: SID:S-1-5-21-1644682095-960612969-1253432751-1003 allow dir_gen_read,object_inherit,container_inherit,inherit_only
 2: group:0 allow dir_gen_read,dir_gen_execute
 3: SID:S-1-1-0 allow dir_gen_read,dir_gen_execute
Cluster01-1# ls -lend dir1/dir2
dr--r----- +  2 0  0  23 Jul 23 09:30 dir1/dir2
 OWNER: user:0
 GROUP: group:0
 0: SID:S-1-5-21-1644682095-960612969-1253432751-1003 allow dir_gen_read,object_inherit,container_inherit
Cluster01-1# ls -len dir1/file1
-r--r----- +  1 0  0  0 Jul 23 09:30 dir1/file1
 OWNER: user:0
 GROUP: group:0
 0: SID:S-1-5-21-1644682095-960612969-1253432751-1003 allow file_gen_read
Cluster01-1# ls -len dir1/dir2/file2
-r--r----- +  1 0  0  0 Jul 23 09:30 dir1/dir2/file2
 OWNER: user:0
 GROUP: group:0
 0: SID:S-1-5-21-1644682095-960612969-1253432751-1003 allow file_gen_read
```

Figure 31 Converted on-disk identity

2.4 Mapping permissions in OneFS

When a client connects to a OneFS cluster with NFS or SMB, permission checking is based on the on-disk OneFS internal permission (POSIX mode bits or OneFS ACL). In OneFS, it is required to present a protocol-specific view of every file to clients. SMB clients can see the on-disk permissions with the SMB ACL, and NFSv4 clients can see the on-disk permissions with the NFSv4 ACL, so OneFS will map its internal permission to a protocol-specific view while the permission checking is still based on its internal permission representation. The following subsections show the permission-inheritance flag mapping and permission mapping between protocols and the OneFS ACL.

When a file contains an authoritative OneFS real ACL, the POSIX mode bits are only for representation and it are not expressive enough to represent the actual OneFS real ACL permissions on disk. When an NFSv3 client checks the POSIX mode bits of a file, if the file contains a OneFS real ACL, it is not possible to see the actual permission of the file from the client side.

2.4.1 Mapping permission-inheritance flags in OneFS

The **rich** ACL, including OneFS ACL, SMB ACL, and NFSv4 ACL, has its own inheritance flags defined, but they have the same function to enable ACL inheritance. Table 8 shows the mapping between the flags.

Table 8 Permission-inheritance flag mapping

OneFS keyword	Windows icacls tool keyword	Linux keyword for NFSv4	Flag set on file or directory	Description
object_inherit	(OI)	f	Directory only	Indicates that an ACE will apply to the current directory and files within the directory
container_inherit	(CI)	d	Directory only	Indicates that an ACE will apply to the current directory and subdirectories within the directory
inherit_only	(IO)	i	Directory only	Indicates that an ACE will apply to subdirectories only, files only, or both within the directory
no_prop_inherit	(NP)	n	Directory only	Indicates that an ACE will apply to the current directory or only the first-level contents of the directory, not the second-level or subsequent contents
inherited_ace	(I)	N/A	File or directory	Indicates that an ACE is inherited from the parent directory

2.4.2 Mapping Windows permissions to NFSv4 permissions and POSIX mode bits

Under the OneFS default ACL policy settings, when configuring permissions from Windows Explorer, Table 9 shows the permissions a file will contain on the OneFS side, and how the permissions are mapped to NFSv4 ACL permissions and POSIX mode bits.

Table 9 Windows permission mapping

Windows Explorer option	Icacls tool keyword	OneFS internal	Linux keyword for NFSv4	POSIX mode bits approximation
Full control	F	dir_gen_all	rwaDdxtTnNcCoy	rwx
		file_gen_all	rwadxtTnNcCoy	
Modify	M	dir_gen_read dir_gen_write dir_gen_execute std_delete	rwadxtTnNcy	rwx
		file_gen_read file_gen_write file_gen_execute std_delete		
Read and execute	RX	dir_gen_read dir_gen_execute	rxtncy	r-x

Windows Explorer option	Icacls tool keyword	OneFS internal	Linux keyword for NFSv4	POSIX mode bits approximation
		file_gen_read file_gen_execute		
Read	R	dir_gen_read	rtncy	r--
		file_gen_read		
Write	W	add_file add_subdir dir_write_ext_attr dir_write_attr std_synchronize	waTNy	-w-
		file_write append file_write_ext_attr file_write_attr std_synchronize		
Delete	D	std_delete std_synchronize	dy	---
Read permissions	RC, S	std_read_dac std_synchronize	cy	---
Change permissions	WDAC, S	std_write_dac std_synchronize	Cy	---
Take ownership	WO, S	std_write_owner std_synchronize	oy	---
List folder/read data	RD, S	file_read std_synchronize	ry	r--
		list std_synchronize		
Create files/write data	WD, S	file_write std_synchronize	wy	-w-
		add_file std_synchronize		
Create folders/ append data	AD, S	append std_synchronize	ay	-w-
		add_subdir std_synchronize		

Windows Explorer option	Icacls tool keyword	OneFS internal	Linux keyword for NFSv4	POSIX mode bits approximation
Read extended attributes	REA, S	file_read_ext_attr std_synchronize	ny	---
		dir_read_ext_attr std_synchronize		
Write extended attributes	WEA, S	file_write_ext_attr std_synchronize	Ny	---
		dir_write_ext_attr std_synchronize		
Traverse folder/ execute file	X, S	execute std_synchronize	xy	-x-
		traverse std_synchronize		
Delete subfolders and files (directory only)	DC, S	delete_child std_synchronize	Dy	-w-
Read attributes	RA, S	file_read_attr std_synchronize	ty	---
		dir_read_attr std_synchronize		
Write attributes	WA, S	file_write_attr std_synchronize	Ty	---
		dir_write_attr std_synchronize		---

2.4.3 Mapping NFSv4 permissions to Windows permissions and POSIX mode bits

Under the OneFS default ACL policy settings, when configuring permissions using the **nfs4_setfacl** command tool, Table 10 shows the permissions a file will contain on the OneFS side, and how the permissions are mapped to Windows NTFS ACL permissions and POSIX mode bits.

Table 10 NFSv4 permission mapping

Linux keyword for NFSv4	RFC3530 standard constant	OneFS internal ACE permission	Windows Explorer option	POSIX mode bits approximation
d	ACE4_DELETE	std_delete	Delete	---
c	ACE4_READ_ACL	std_read_dac	Read permissions	---
C	ACE4_WRITE_ACL	std_write_dac	Change permissions	---
o	ACE4_WRITE_OWNER	std_write_owner	Take ownership	---
y	ACE4_SYNCHRONIZE	std_synchronize	N/A	---
r	ACE4_ACE4READ_DATA	file_read	List folder/read data	r--
	ACE4_LIST_DIRECTORY	list		
w	ACE4_WRITE_DATA	file_write	Create files/write data	-w-
	ACE4_ADD_FILE	add_file		
a	ACE4_APPEND_DATA	append	Create folders/ append data	-w-
	ACE4_ADD_SUBDIRECTORY	add_subdir		
x	ACE4_EXECUTE	execute	Traverse folder/ execute file	--x
		traverse		
t	ACE4_READ_ATTRIBUTES	file_read_attr	Read attributes	---
		dir_read_attr		
T	ACE4_WRITE_ATTRIBUTES	file_write_attr	Write attributes	---
		dir_write_attr		
n	ACE4_READ_NAMED_ATTRS	file_read_ext_attr	Read extended attributes	---
		dir_read_ext_attr		
N	ACE4_WRITE_NAMED_ATTRS	file_write_ext_attr	Write extended attributes	---
		dir_write_ext_attr		
D (directory only)	ACE4_DELETE_CHILD	delete_child	Delete subfolders and files	-w-

2.4.4 Mapping POSIX mode bits to Windows and NFSv4 permissions

Mapping POSIX mode bits to ACLs is simpler because the mode bits are a subset of the **rich** ACL model, so no security information is lost. Table 11 shows how OneFS processes the POSIX mode bits to be mapped to OneFS synthetic ACL, Window NTFS ACL permissions, and NFSv4 ACL permissions.

Table 11 POSIX mode bits permission mapping

POSIX mode bits	OneFS internal ACE permission	Windows Explorer option	Linux keyword for NFSv4
r	dir_gen_read	Read	rtncy
	file_gen_read		
w	dir_gen_write, delete_child, dir_read_attr	Write, Read attributes, Delete subfolders and files, Read permissions	waDtTNcy
	file_gen_write, file_read_attr	Write, Read attributes, Read permissions	watTNcy
x	dir_gen_execute, dir_read_attr	Traverse folder/execute file, Read attributes, Read permissions	xtcy
	file_gen_execute, file_read_attr		

2.4.5 Mapping HDFS ACL permissions to OneFS permissions

HDFS ACL is an Apache implementation of POSIX ACL. It provides more flexible access control than the traditional POSIX-mode-bit permission model. Starting from OneFS 9.3.0, OneFS supports HDFS ACL to improve compatibility with HDFS. By default, HDFS ACL is disabled, and the following configuration is required to get consistent HDFS ACL semantics in OneFS.

- Modify the default ACL policy, and ensure that your NFS/SMB workload is not impacted by changing the following settings.

  ```
  # isi auth settings acls modify --calcmode-group=group_aces --calcmode-traverse=ignore --group-owner-inheritance=creator
  ```

- Enable HDFS ACL support.

  ```
  # isi hdfs settings modify --hdfs-acl-enabled=true --zone=System
  ```

- If your Hadoop client version is lower than 3, apply the following setting.

  ```
  # isi hdfs settings modify --hadoop-version-3-or-later=false
  ```

Table 12 shows the final on-disk permissions when applying POSIX ACL permissions from the Hadoop client with the command **hdfs dfs -setfacl**. OneFS always adds a deny ACE explicitly after the allow ACE when applying HDFS ACL.

Table 12 HDFS ACL permissions

HDFS ACE permission	Apply to	OneFS internal ACE permission
rwx	Directory	allow dir_gen_read,dir_gen_write,dir_gen_execute,delete_child deny
	File	allow file_gen_read,file_gen_write,file_gen_execute deny
rw-	Directory	allow dir_gen_read,dir_gen_write,delete_child deny traverse
	File	allow file_gen_read,file_gen_write deny execute
r-x	Directory	allow dir_gen_read,dir_gen_execute deny add_file,add_subdir,dir_write_ext_attr,delete_child,dir_write_attr
	File	allow file_gen_read,file_gen_execute deny file_write,append,file_write_ext_attr,file_write_attr
r--	Directory	allow dir_gen_read deny add_file,add_subdir,dir_write_ext_attr,traverse,delete_child,dir_write_attr
	File	allow file_gen_read deny file_write,append,file_write_ext_attr,execute,file_write_attr
-wx	Directory	allow dir_gen_write,dir_gen_execute,delete_child,dir_read_attr deny list,dir_read_ext_attr
	File	allow file_gen_write,file_gen_execute,file_read_attr deny file_read,file_read_ext_attr
-w-	Directory	allow dir_gen_write,delete_child,dir_read_attr deny list,dir_read_ext_attr,traverse
	File	allow file_gen_write,file_read_attr deny file_read,file_read_ext_attr,execute
--x	Directory	allow dir_gen_execute,dir_read_attr deny list,add_file,add_subdir,dir_read_ext_attr,dir_write_ext_attr,delete_child, dir_write_attr
	File	allow file_gen_execute,file_read_attr deny file_read,file_write,append,file_read_ext_attr,file_write_ext_attr,file_write_attr
---	Directory	allow std_read_dac,std_synchronize,dir_read_attr deny list,add_file,add_subdir,dir_read_ext_attr,dir_write_ext_attr,traverse,delete_child, dir_write_attr
	File	allow std_read_dac,std_synchronize,file_read_attr deny file_read,file_write,append,file_read_ext_attr,file_write_ext_attr,execute, file_write_attr

3 Considerations for permissions issues

3.1 Data backup and restore privileges

OneFS provides a role-based access control (RBAC) model to delegate administrative tasks to specific users. RBAC roles are assigned privileges and then assigned members. All users that are assigned to a role must connect to the system access zone to configure and manage the cluster. OneFS includes several integrated roles with predefined privileges and cannot be modified. You can also create custom roles to fit your environment.

Most privileges in the RBAC system pertain to allowing changes to cluster configuration. Two privileges (**ISI_PRIV_IFS_BACKUP** and **ISI_PRIV_IFS_RESTORE**) were added in OneFS 7.1.1 and later, which allow access to cluster data. When a user is a member of a role with either or both **ISI_PRIV_IFS_BACKUP** or **ISI_PRIV_IFS_RESTORE** privileges, the user can circumvent traditional file access checks (POSIX mode bits or ACLs) to back up or restore file data. These two privileges are supported over the following protocols: SMB, NFS, OneFS API, FTP and SSH.

Circumventing traditional file access checks is done by adding a combination of privileges and flags to the user's access token during authentication. For NFS, it requires the export enable **--map-lookup-uid** option. For SMB, these two OneFS privileges emulate the Windows privileges **SE_BACKUP_NAME** and **SE_RESTORE_NAME**, which means that normal file opens are still protected by file-system permissions. To enable the backup and restore privileges over the SMB protocol, files must be opened with the **FILE_OPEN_FOR_BACKUP_INTENT** option. When users employ Windows backup software such as Robocopy, the option is applied automatically. When users perform general file browsing using Windows File Explorer, the option is not applied.

You can assign the **BackupAdmin** role to a user in the system access zone by using the WebUI (shown in Figure 32) or the **isi auth role** command.

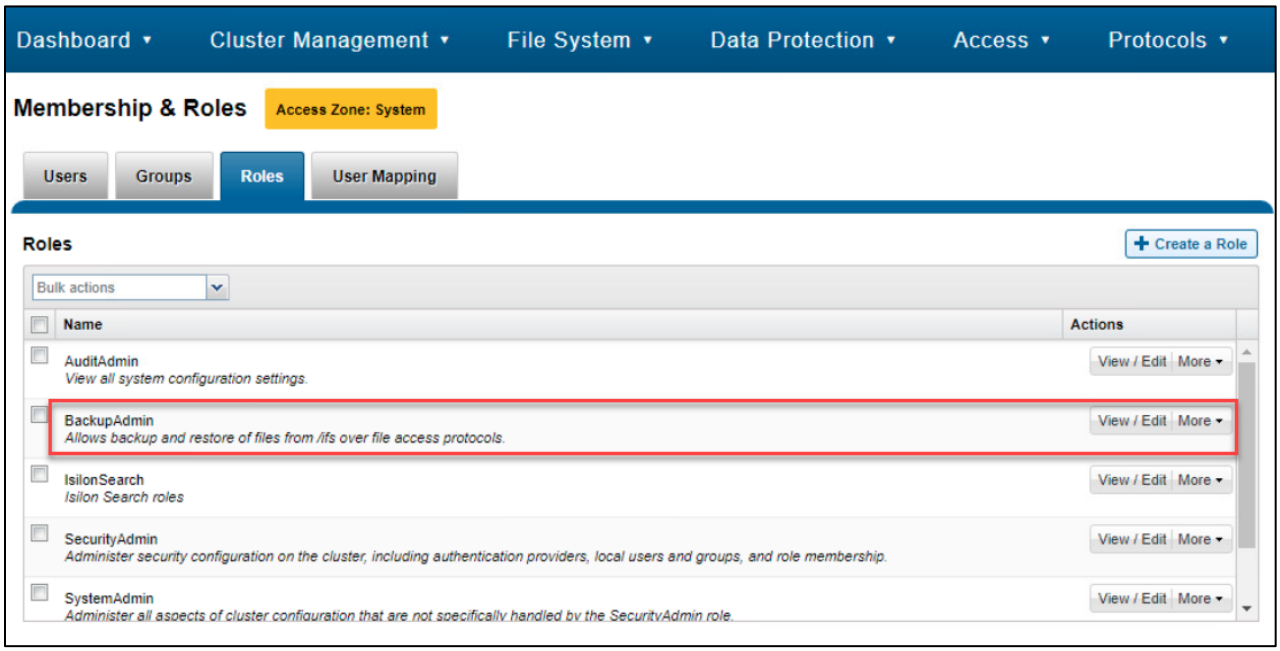

Figure 32 Role configuration WebUI

To verify if a user has the **ISI_PRIV_IFS_BACKUP** and **ISI_PRIV_IFS_RESTORE** privileges, use the command **isi auth user mapping token**. An example is shown in Figure 33.

```
Cluster01-1# isi auth mapping token user01
                  User
                      Name: user01
                       UID: 2002
                       SID: S-1-5-21-1644682095-960612969-1253432751-1003
                   On Disk: 2002
                   ZID: 1
                  Zone: System
              Privileges: ISI_PRIV_IFS_BACKUP
                          ISI_PRIV_IFS_RESTORE
          Primary Group
                      Name: backupusers
                       GID: 2000
                       SID: S-1-5-21-1644682095-960612969-1253432751-1002
                   On Disk: 2000
Supplemental Identities
                      Name: Authenticated Users
                       SID: S-1-5-11
```

Figure 33 Verify user privileges

3.2 NFS export and SMB share settings

The following settings are checked before on-disk permission checking and can impact a user's effective permission on a file.

- Root or non-root user mapping and map lookup UID options of a NFS export
- SMB share permission settings which restrict users to accessing a SMB share

3.2.1 NFS export map options

3.2.1.1 Root or non-root user mapping

By default, an NFS export on OneFS maps the root user to the nobody user, an unprivileged user account. In this way, a remote root user on the client does not get the root permission on the file system. This behavior is also known as **root squash**. OneFS also allows mapping other non-root users to a specific user that is used to access the files. By default, this setting is disabled. When enabling the non-root user mapping, all the permission checking is based on the new user that you specify, so when a client user gets the wrong permission to the export files, it is likely that the export user mapping setting is enabled.

Viewing and changing the NFS export mapping settings is performed through the Web UI (shown in Figure 34) or the **isi nfs exports modify** command.

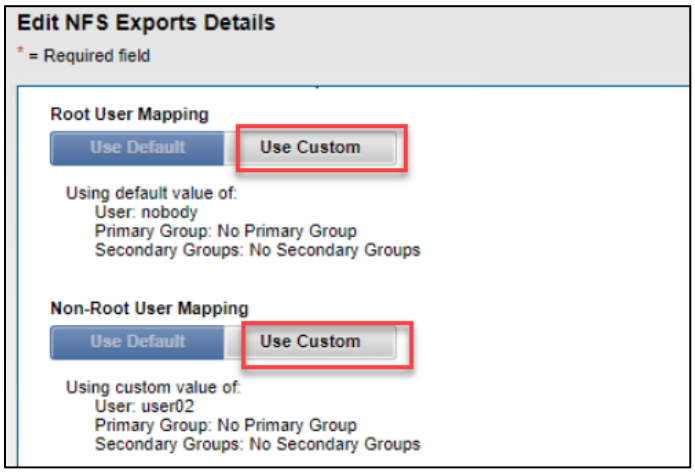

Figure 34 Export user mapping

3.2.1.2 Map lookup UID option

When an NFS client sends a series of calls to a server, the NFS RPC header contains the following information:

- User UID
- One primary GID
- Up to 15 supplemental GIDs

By default, when a server receives this list of GIDs, it trusts the GIDs are complete and checks the file-system permissions using only these GIDs.

When using the **NFS AUTH_SYS** authentication security type and a user is a member of more than 16 groups, the permissions may be improperly applied. This is a limitation of NFS in which a request is sent to a server from a client, the GID list sent by the client is truncated, and only a portion of group information is sent to the server. For example, **userA** belongs to **group20**, and **group20** has access to a file. **Group20** is truncated because it exceeds the 16-group limit and **userA** is denied access by the NFS server. Conversely, if the group is denied access to the file, the user would be allowed to access the file by accident.

To eliminate this limitation, OneFS provides an NFS export option called **map-lookup-uid**. By default, it is disabled. Once it is enabled, OneFS will not trust the GID information received from client. It will only look at the user's UID and ignore the GIDs. Then, OneFS performs a lookup of the UID in the connected authentication providers (local, NIS, LDAP, or RFC-2307-enabled AD) to determine group membership. Finally, OneFS performs the permission checking based on the result of the lookup.

The **map-lookup-uid** option is enabled using the WebUI or **isi** command. It is more convenient to enable it using the following command: **isi nfs exports modify exportID –-map-lookup-uid=true.**

3.2.2 SMB share permission

As a user traverses an SMB path on an PowerScale cluster, the first location that the user can be denied access is at the share level. When the share permission is combined with on-disk permission (POSIX mode bits or ACLs), the more restrictive permission always applies. For example, if the share permission is set to

allow everyone read access, and the on-disk permission allows users to read and write. Finally, the share permission applies and the user is not allowed to change a file.

To view the share permission, use either the WebUI or **isi** command. As Figure 35 shows, everyone on the default **ifs** SMB share has full control at the share level, and the effective permission is determined by on-disk permissions.

```
Cluster01-1# isi smb shares permission list ifs
Account  Account Type  Run as Root  Permission Type  Permission
-------------------------------------------------------------------
Everyone wellknown     False        allow            full
-------------------------------------------------------------------
Total: 1
```

Figure 35 SMB share permission

Note: OneFS allows administrators to enable the **run-as-root** option on an SMB share permission. This option is disabled by default. When enabled, a user has root access to files and directories under the share.

3.3 OneFS user mapping

When OneFS authenticates a user, it generates an access token and triggers the user-mapping service on the cluster. The user-mapping service combines access tokens from different directory services into a single, final access token. By default, OneFS automatically maps user accounts with the exact same name from different directory services. For example, the user-mapping service maps a user **DEMO\janed** from Microsoft Active Directory to a user **janed** from LDAP, and generates an access token that combines the user and group membership information from the two accounts.

A OneFS administrator can also define their own user-mapping rules to control the user-mapping behavior. It is important to know the following information when creating user-mapping rules:

- User-mapping rules are applied at the access-zone level. You can only create user-mapping rules for specific access zone if you are connected to the OneFS System access zone.
- When changing user-mapping rules on one node, OneFS will propagate the change to the other nodes.
- OneFS reloads the configuration when user-mapping rule changes occur.

There are five operations involved when creating user-mapping rules:

- **Insert fields from a user:** This modifies an existing access token by adding fields to it. Fields specified in the options list (user, group, or groups) are copied from new identity and inserted into the identity in the token.
- **Append field from a user:** This also modifies an access token by adding fields to it. The mapping service appends the fields that are specified in the list of options (user, group, or groups) to the first identity in the rule. All appended identifiers become members of the supplemental groups list.
- **Replace one user with a different user:** This removes the token and replaces it with a new token that is identified by the second username.
- **Remove supplemental groups from a user:** This modifies an access token by removing the supplemental groups.
- **Join two users together:** This inserts the new identity into the token. This operation combines two user access tokens together into a single access token containing the information from both users.

For details on configuring user-mapping rules, refer to the PowerScale OneFS Web Administration Guide.

The example in Figure 36 shows the result after joining two users together. In this example, **user01** from PowerScale local is joined with **demo\janed** from Active Directory.

```
Cluster01-1# isi auth mapping token --user=user01
                  User
                      Name: user01
                       UID: 2002
                       SID: S-1-5-21-1644682095-960612969-1253432751-1002
                   On Disk: 2002
                   ZID: 1
                  Zone: System
              Privileges: -
           Primary Group
                      Name: Isilon Users
                       GID: 1800
                       SID: S-1-5-21-1644682095-960612969-1253432751-800
                   On Disk: 1800
Supplemental Identities
                      Name: Authenticated Users
                       SID: S-1-5-11

                      Name: DEMO\janed
                       UID: 1000002
                       SID: S-1-5-21-2835929079-83856256-991020882-7601

                      Name: DEMO\domain users
                       GID: 1000006
                       SID: S-1-5-21-2835929079-83856256-991020882-513

                      Name: Users
                       GID: 1545
                       SID: S-1-5-32-545
```

Identity information from demo\janed

Figure 36 OneFS user mapping

The next example in Figure 37 assumes there is a directory which does not allow **user01** to access the directory, but allows **demo\janed** to read the directory.

```
Cluster01-1# ls -led dir01
drwxr----- +  2 root  wheel  0 Aug  3 14:41 dir01
 OWNER: user:root
 GROUP: group:wheel
 0: user:root allow dir_gen_read,dir_gen_write,dir_gen_execute,std_write_dac,delete_child
 1: user:DEMO\janed allow dir_gen_read
```

Figure 37 ACL of the directory

Because there is a user-mapping rule to join the two users together, using the **isi** command to check permissions of **user01** on the directory shows that **user01** also has permission to read the directory (see Figure 38). This command result only evaluates the ACL permission, and does not involve the share-level permissions.

```
Cluster01-1# isi auth access --user=user01 --zone=System dir01
              User
                  Name: user01
              File
       File Permissions: user:user01 allow dir_gen_read
```

Figure 38 Effective permissions of user01

A ACL technical background

A.1 ACL on Windows NTFS and SMB

Windows has an access-control model that determines who can access resources in the operating system. Windows applications call access control functions to set who can access specific resources, or to control access to resources provided by the application. See the Microsoft article on Access Control for more details.

An ACL is a key part of the Windows access model. In Windows, ACLs can provide access control to an Active Directory service object. Routines for ACL creation and modification of ACLs are included in Active Directory Service Interfaces (ADSI). The Windows ACL can also provide access control to NTFS file system objects, and the ACLs associated with these file system objects can be managed through Windows Explorer and the command-line tool.

Because SMB is designed for the Windows environment, the SMB ACL is compatible and has the same concepts and definitions as the Windows NTFS ACL. This section discusses the Windows NTFS ACL to show how ACL technology is designed and implemented on the Windows platform.

A.1.1 Security descriptor

A security descriptor contains the security information associated with a file system object, including the owner and primary group, DACL, and SACL of an object.

Run the **get-acl** PowerShell cmdlet to view the security descriptor of a file-system object.

```
PS C:\> Get-Acl -Audit "C:/test/acl-test.txt" | Format-List

Path   : Microsoft.PowerShell.Core\FileSystem::C:\test\acl-test.txt
Owner  : VLAB\jane
Group  : VLAB\Domain Users
Access : BUILTIN\Administrators Allow  FullControl
         NT AUTHORITY\SYSTEM Allow  FullControl
         BUILTIN\Users Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\Authenticated Users Allow  Modify, Synchronize
Audit  : VLAB\jane Success  DeleteSubdirectoriesAndFiles, Modify, ChangePermissions, TakeOwnership
Sddl   : O:S-1-5-21-854245398-1972579041-362288127-1767896G:DUD:AI(A;ID;FA;;;BA)(A;ID;FA;;;SY)(A;ID;0x1200a9;;;BU)
         (A;ID;0x1301bf;;;AU)S:AI(AU;SA;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;S-1-5-21-854245398-1972579041-362288127-1767896)
```

Figure 39 Windows security descriptor

Figure 39 displays the following details:

- **Path:** This lists the Windows PowerShell path to the file (<provider>::<resource-path>).
- **Owner and Group:** This is the owner and security group of the file.
- **Access:** The discretionary access control list (DACL) contains a list of the access control entries for the file. The DACL list is controlled by the owner.
- **Audit:** The system access control list (SACL) contains a list of entries.
- **Sddl:** The security descriptor of the resource is displayed in a single text string in the Security Descriptor Definition Language (SDDL) format.

A.1.2 DACL and SACL

A Windows security descriptor contains two types of ACLs (DACL and SACL) for a file system object.

- **Discretionary access control list (DACL):** This specifies the allowed or denied access rights to a user or group. When a process tries to access a system object, the system checks the ACEs in the object's DACL to determine whether to grant access to it.
- **System access control list (SACL):** This allows administrators to audit access attempts to a file-system object. An ACE in a SACL can generate audit events when an access attempt fails, when it succeeds, or both. For more information about how to use SACLs on Windows, see the Microsoft articles on Configuring Audit Policies and how to Apply a basic audit policy on a file or folder.

A.1.1 Windows ACE

The ACE is an element in an ACL, and an ACL can have zero or more ACEs. Each ACE controls access to a file-system object by a specified user or group. There are two types of ACEs in a DACL: the first is the access-allow ACE and the second is the access-deny ACE, and both contain the following information:

- **User/group:** A security identifier (SID) that identifies the trustee to which the ACE applies
- **ACE inheritance:** A set of inheritance flags that indicate whether the subdirectory and files can inherit the ACE
- **ACE type:** A flag that indicates the type of ACE (allow or deny)
- **Access right:** The access rights specified by the ACE

ACE inheritance

An ACE can be explicitly set on an object or it can be inherited. Inheritance allows permissions to be layered or overridden as needed in an object hierarchy and allows for simplified permission management.

There are five flags to indicate ACE inheritance. An ACE only applies to the current file or directory if no ACE inheritance flag is specified. Table 13 shows the five ACE inheritance flags and meanings.

Table 13 Windows ACE inheritance

ACE inheritance flag keyword	Flag set on file or directory	Description
(OI) Object inherit	Directory only	Indicates that an ACE applies to a current directory and files within the directory
(CI) Container inherit	Directory only	Indicates that an ACE applies to a current directory and subdirectories within the directory
(IO) Inherit only	Directory only	Indicates that an ACE applies to subdirectories only, files only, or both within the directory
(NP) No propagate inherit	Directory only	Indicates that an ACE applies to the current directory, only the first-level contents of the directory, or both, but not to the second-level or subsequent contents
(I) Inherited	File or directory	Indicates that an ACE is inherited from the parent directory

Windows Explorer provides a UI to configure ACE inheritance, as highlighted in Figure 40. To access the UI, right-click a directory and click **Properties** > **Security** > **Advanced** > **Permissions** > **Add**.

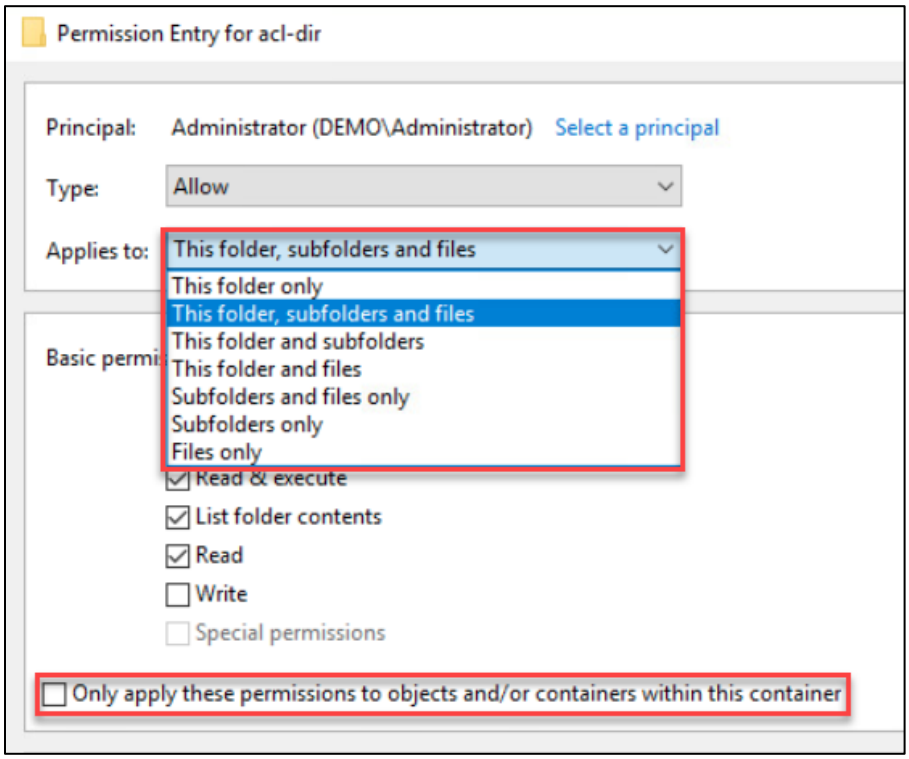

Figure 40 Configuring ACE inheritance flags

Table 14 shows the relationship between each UI option and the exact flags contained.

Table 14 Inheritance UI options

Windows Explorer UI option	Contained ACE inheritance flags
This folder only	No flags, an ACE applies only to the current directory
This folder, subfolders, and files	(OI) (CI)
This folder and subfolders	(CI)
This folder and files	(OI)
Subfolders and files only	(OI) (CI) (IO)
Subfolders only	(CI) (IO)
Files only	(OI) (IO)
Only applies these permissions to objects, containers within this container, or both	(NP)

Order of ACEs in a DACL

When users or groups access a file-system object, the system iterates through the ACEs in the object's DACL until it finds ACEs that allow or deny the requested access.

The access rights that a DACL allows a user could vary depending on the order of ACEs in the DACL. In Windows, a specific ordering of ACEs (a canonical ordering) is enforced. The ordering impacts how permissions are applied, and the rules for ordering are as follows:

1. All explicit ACEs are placed before any inherited ACEs.
2. Within the group of explicit ACEs, access-denied ACEs are placed before access-allowed ACEs.
3. Inherited ACEs are placed in the order in which they are inherited. ACEs inherited from the child object's parent come first, then ACEs inherited from the grandparent, and so on, up the tree of objects.
4. For each level of inherited ACEs, access-denied ACEs are placed before access-allowed ACEs.

Access rights

An access right corresponds to a set of operations that a user or group can have on a file-system object. The Windows DACL can be associated with many different Windows system objects, not just files and directories.

Table 15 to Table 17 show Windows NTFS access rights and their associated options in the Windows Explorer UI. To access this UI, right-click a directory and click **Properties** > **Security** > **Advanced** > **Permissions** > **Add**.

Table 15 NTFS generic access right

Constant	Keyword	Windows Explorer UI option	Description
GENERIC_ALL	GA	Full control	All possible access rights; maps to Full Control (F)
GENERIC_EXECUTE	GE	N/A	Execute access; maps to RC, S, X, RA
GENERIC_READ	GR	Read	Read access (R); contains permissions RD, RC, RA, REA
GENERIC_WRITE	GW	Write	Write access (W); contains permissions WD, AD, WA, WEA

Table 16 NTFS standard access rights

Constant	Keyword	Windows Explorer UI option	Description
DELETE	DE	Delete	The right to delete the object
READ_CONTROL	RC	Read permissions	The right to read the information in the object's security descriptor, not including the information in the system access control list (SACL); Read permissions
SYNCHRONIZE	S	N/A	The right to use the object for synchronization, which enables a thread to wait until the object is in the signaled state; some object types do not support this access right
WRITE_DAC	WDAC	Change permissions	The right to modify the discretionary access control list (DACL) in the object's security descriptor; change permissions

WRITE_OWNER	WO	Take ownership	The right to change the owner in the object's security descriptor. Take ownership.

Table 17 File-system object access rights

Constant	Keyword	Windows Explorer UI option	Description
FILE_READ_DATA	RD	Read data	The right to read data from the file
FILE_LIST_DIRECTORY		List folder	The right to list the contents of a directory
FILE_WRITE_DATA	WD	Write data	The right to write data to the file
FILE_ADD_FILE		Create files	The right to create a file in a directory
FILE_APPEND_DATA	AD	Append data	The right to append data to the file.
FILE_ADD_SUBDIRECTORY		Create folders	The right to create a subdirectory in a directory
FILE_READ_EA	REA	Read extended attributes	The right to read extended attributes
FILE_WRITE_EA	WEA	Write extended attributes	The right to write extended attributes
FILE_EXECUTE	X	Execute file	The right to execute a file
FILE_TRAVERSE		Traverse folder	The right to traverse a directory
FILE_DELETE_CHILD	DC	Delete subfolders and files	The right to delete a directory and all the files it contains (its children), even if the files are read-only
FILE_READ_ATTRIBUTES	RA	Read attributes	The right to read file attributes
FILE_WRITE_ATTRIBUTES	WA	Write attributes	The right to change file attributes

Windows Explorer UI basic and advanced permissions mapping

For the convenience of routine permissions management, Windows defines several basic permissions that contain a set of individual advanced permissions. To view these options through the Windows Explorer UI, right-click a file and click **Properties** > **Security** > **Advanced** > **Permissions** > **Add**. Table 18 shows the mapping between the permissions.

Table 18 Basic and advanced permissions in the Windows Explorer UI

Advanced permissions	Basic permissions				
	Full control	Modify	Read and execute	Read	Write
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	✓				
Synchronous	✓	✓	✓	✓	✓

A.1.2 Examples

The following examples use the Windows Explorer UI to set permissions and the **icacls** command-line tool to check the exact permissions applied.

Example 1: Windows NTFS ACL's ACE inheritance

This example removes all inherited ACEs from the top-folder and adds four new ACEs, shown in Figure 41**Error! Reference source not found.**. The following ACEs are added:

- **ACE 0:** Grants group **Administrators** the Full control permission on the folder, and propagates permissions to all subfolders and files. This means the ACE contains inheritance flags (OI)(CI).
- **ACE 1:** Grants user **janed** the Read permission on the folder only. No inheritance flag is set on this ACE.
- **ACE 2**: Grants user **johnd** the Read permission on all subfolders and files under the folder, not the folder itself. This means the ACE contains inheritance flags (OI)(CI)(IO).
- **ACE 3:** The last ACE has the option enabled (highlighted with the red rectangle), which means the ACE will only apply to the first-level content within the folder. This ACE only grants user **joed**the Read permission on first-level contents of the folder. This means the ACE contains inheritance flags (OI)(CI)(NP)(IO).

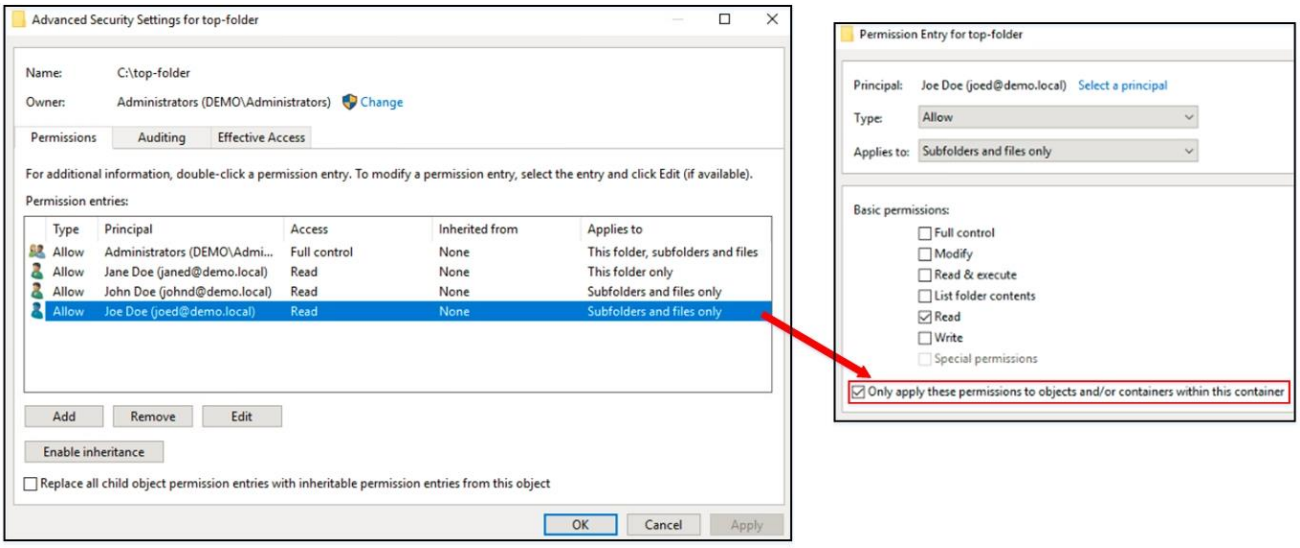

Figure 41 ACEs of top-folder

Use the **iscacls** command to check the inheritance flags on the folder's ACL, as shown in Figure 42:

```
c:\>icacls c:\top-folder
c:\top-folder BUILTIN\Administrators:(OI)(CI)(F)
              DEMO\janed:(R)
              DEMO\johnd:(OI)(CI)(IO)(R)
              DEMO\joed:(OI)(CI)(NP)(IO)(R)

Successfully processed 1 files; Failed processing 0 files
```

Figure 42 Inheritance flags

Now, verify the ACE inheritance on the first-level folder and second-level folder under the **top-folder.**

As Figure 43 shows, ACE 1 is not inherited to the **folder-level-1.** The rest of ACEs are inherited as expected.

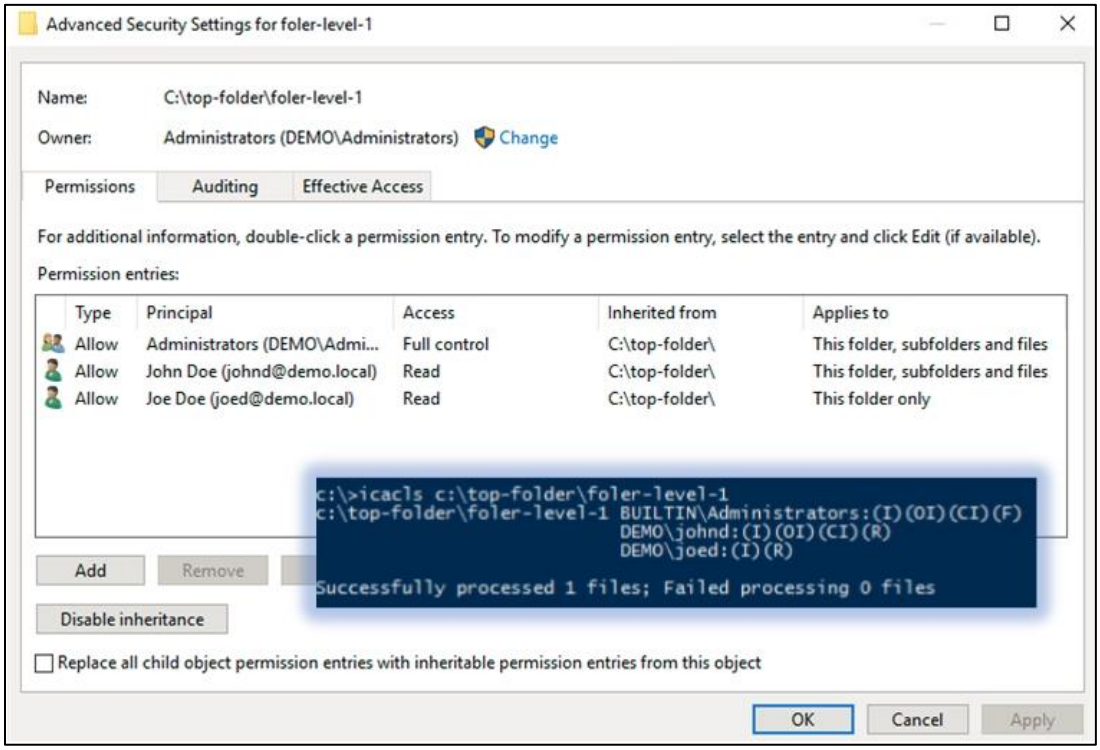

Figure 43 ACEs on first-level folder

As Figure 44 shows, ACE 1 is not inherited to the **folder-level-2** and ACE 3 is also not inherited because it is applied a (NP) flag on **top-folder.** The rest of ACEs are inherited as expected.

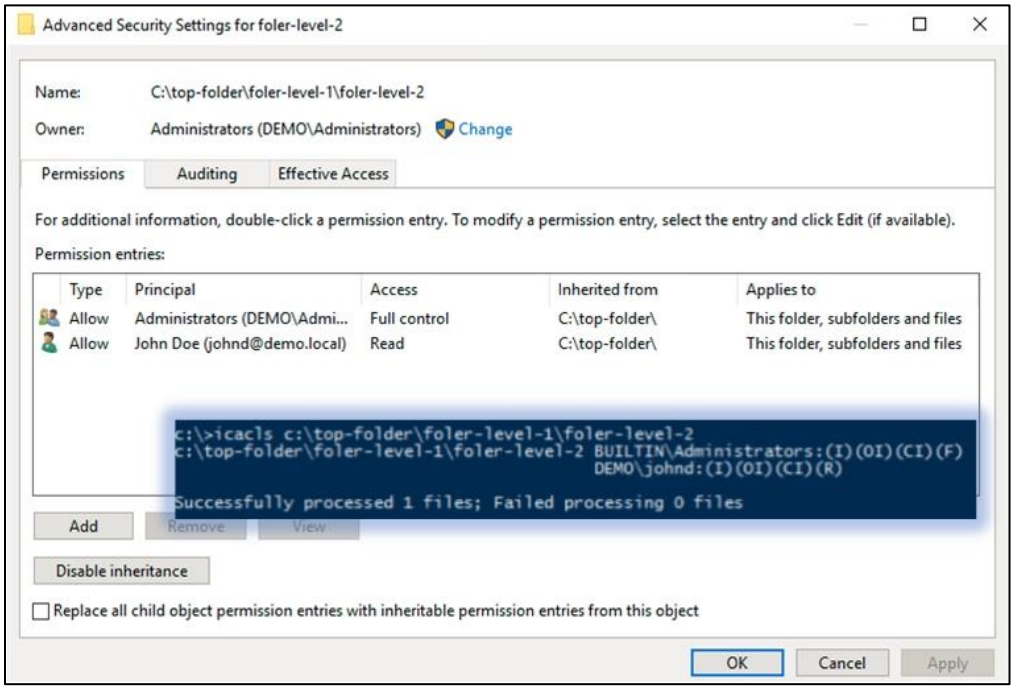

Figure 44 ACEs on second-level folder

Example 2: Order of ACEs in a DACL

Add access-allow ACEs and access-deny ACEs on both **top-folder** and **folder-level-1** as shown in Figure 45

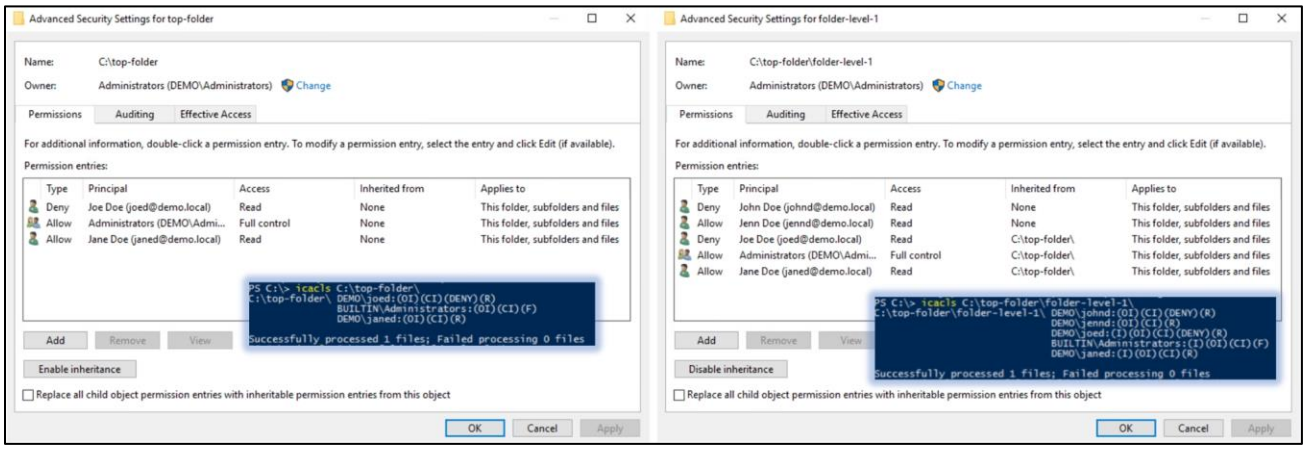

Figure 45 ACEs on top-folder and folder-level-1

As mentioned in section A.1.1, the order of ACEs on **folder-level-2** is as expected. Shown in Figure 46, a group of all the explicit ACEs (highlighted in red) are placed before any other inherited ACEs, and then a group of ACEs (highlighted in blue) inherited from the parent folder **folder-level-1** come first, then ACEs (highlighted in yellow) are inherited from the grandparent folder **top-folder.** Meanwhile, in each group of ACEs, access-deny ACEs are always placed before access-allow ACEs.

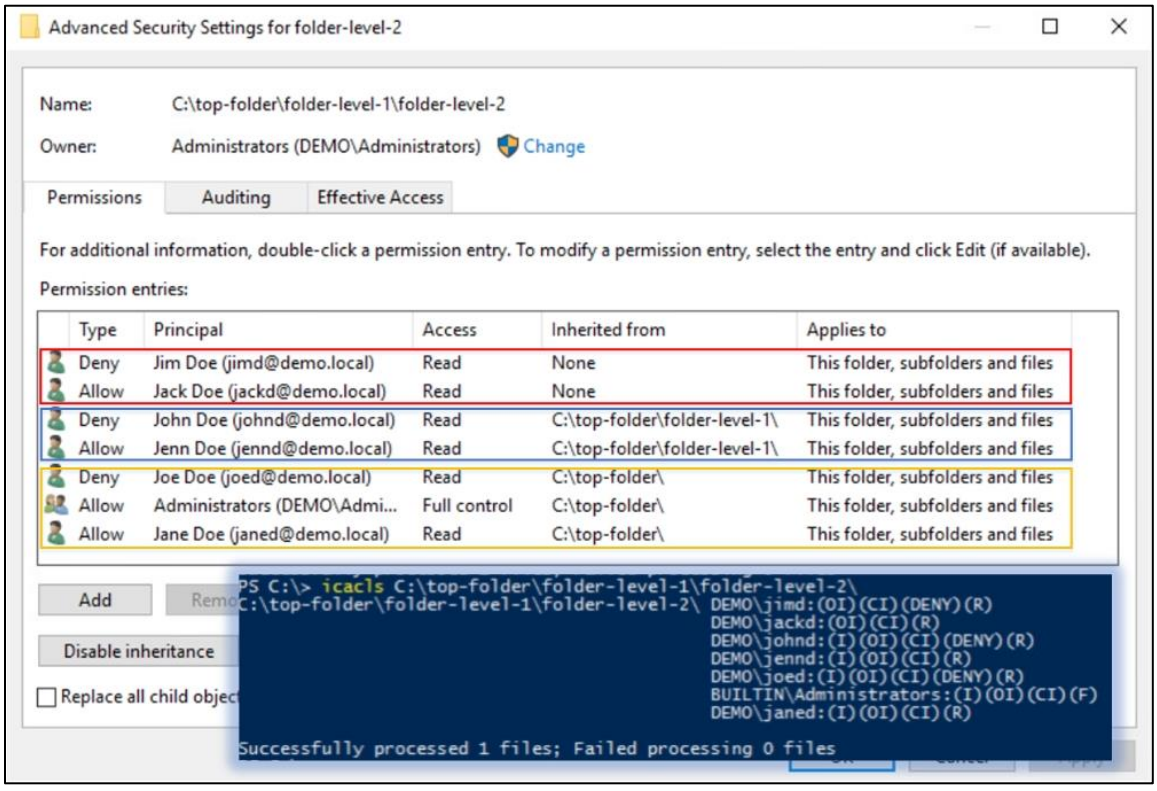

Figure 46 Order of ACEs on folder-level-2

A.2 NFSv4 ACL

ACLs are not supported nor defined in any RFC document for NFSv3 or earlier. They are influenced by the granular and expressive access control in Windows NTFS ACL. NFSv4 has defined its own ACLs and operations in RFC 3530.

The NFSv4 ACL defines allow and deny ACE types for access control which are similar to the Windows NTFS DACL, and also defines audit and alarm ACE types for system-access logging attempts, which is similar to the Windows NTFS SACL. This section discusses allow and deny ACEs of the NFSv4 ACL.

A.2.1 NFSv4 ACE

An NFSv4 ACE contains the following information:

- **ACE type:** This indicates the type of the ACE (allow or deny).
- **ACE principal:** This indicates a user or group. NFSv4 uses variable-length string names of the form *user/group@domain*. An ACE principal could be a named user or group (for example, janed@nfsdomain.local) or one of three special principals: OWNER@, GROUP@, and EVERYONE@. These are, respectively, analogous to the POSIX mode bits user, group, or other.
- **ACE permissions:** This specifies the permissions that the ACE applies to the file or directory. See section A.2.2 for more details.
- **ACE inheritance flag:** This can only be used in a directory object. See section A.2.3 for more details.

A.2.2 NFSv4 ACE permissions

There is a variety of ACE permissions, each which is represented by a single character in Linux. The ACE4_DELETE_CHILD permission can only be applied to directories. An ACE should have one or more of the permissions specified in Table 19.**Error! Reference source not found.**

Table 19 NFSv4 ACE permissions

RFC3530 constant	Linux keyword	Description
ACE4_READ_ACL	c	Permission to read the ACL
ACE4_WRITE_ACL	C	Permission to write the ACL
ACE4_WRITE_OWNER	o	Permission to change the owner
ACE4_SYNCHRONIZE	y	Permission to access file locally at the server with synchronous reads and writes
ACE4_DELETE	d	Permission to delete the file/directory.
ACE4_DELETE_CHILD	D	Used for a directory only, the permission to delete a file or directory within the directory
ACE4_READ_DATA	r	Permission to read the data of a file
ACE4_LIST_DIRECTORY		Permission to list contents of a directory

RFC3530 constant	Linux keyword	Description
ACE4_WRITE_DATA	w	Permission to modify the data of a file
ACE4_ADD_FILE		Permission to add a new file to a directory
ACE4_APPEND_DATA	a	Permission to append data to a file
ACE4_ADD_SUBDIRECTORY		Permission to create a subdirectory to a directory
ACE4_EXECUTE	x	For a file, the permission to execute the file For a directory, the permission to traverse the directory
ACE4_READ_ATTRIBUTES	t	Permission to read basic attributes (non-acls) of a file
ACE4_WRITE_ATTRIBUTES	T	Permission to change basic attributes (non-acls) of a file
ACE4_READ_NAMED_ATTRS	n	Permission to read the named attributes of a file
ACE4_WRITE_NAMED_ATTRS	N	Permission to change named attributes of a file

A.2.3 NFSv4 ACE inheritance flag

ACEs can be inherited from the parent directory's ACL when a file or subdirectory is created. The ACE inheritance flags can be used only on directories. The inheritance behavior is similar to Windows.

Table 20 NFSv4 ACE inheritance

RFC3530 constant	Linux keyword	Description
ACE4_FILE_INHERIT_ACE	f	Indicates that an ACE applies to the current directory and files within the directory
ACE4_DIRECTORY_INHERIT_ACE	d	Indicates that an ACE applies to the current directory and subdirectories within the directory
ACE4_NO_PROPAGATE_INHERIT_ACE	n	Indicates that an ACE applies to subdirectories only, files only, or both within the directory.
ACE4_INHERIT_ONLY_ACE	i	Indicates that an ACE applies to the current directory, only the first-level contents of the directory, or both but not the second-level or subsequent contents.

A.2.4 Examples

To manage and manipulate NFSv4 ACL, the Linux nfs4-acl-tools package is required. You can install this package using command **yum install nfs4-acl-tools** or **apt-get install nfs4-acl-tools** depending on your Linux distribution.

Example 1: Set, modify, or view an NFSv4 ACL

When a file or directory is created over NFSv4, the NFSv4 ACL contains three ACEs for OWNER@, GROUP@, and EVERYONE@, if there is no inherited ACL from parent directory. As Figure 47 shows, the **nfs4_getfacl** command gets the NFSv4 ACL of a file or directory.

```
[root@linuxclient nfs]# nfs4_getfacl top-dir/
A::OWNER@:rwaDxtTnNcCy
A::GROUP@:rxtncy
A::EVERYONE@:rxtncy
```

Figure 47 Original NFSv4 ACL

To add an ACL entry, use the **nfs4_setfacl** command with the **-a** option. Specify an index (**2** in the example) to indicate the ACL entry index, as shown in Figure 48. The default is **1**.

```
[root@linuxclient nfs]# nfs4_setfacl -a A::user01@demo.local:rxtncy 2 top-dir
[root@linuxclient nfs]# nfs4_getfacl top-dir
A::OWNER@:rwaDdxtTnNcCoy
A::user01@demo.local:rxtncy
A::GROUP@:rxtncy
A::EVERYONE@:rxtncy
```

Figure 48 Add an NFSv4 ACL entry

To remove an ACL entry, use the **nfs4_setfacl** command with **-x** option as shown in Figure 49.

```
[root@linuxclient nfs]# nfs4_getfacl top-dir
A::OWNER@:rwaDdxtTnNcCoy
A::user01@demo.local:rxtncy
A::GROUP@:rxtncy
A::EVERYONE@:rxtncy
[root@linuxclient nfs]# nfs4_setfacl -x A::user01@demo.local:rxtncy top-dir
[root@linuxclient nfs]# nfs4_getfacl top-dir
A::OWNER@:rwaDdxtTnNcCoy
A::GROUP@:rxtncy
A::EVERYONE@:rxtncy
[root@linuxclient nfs]# nfs4_setfacl -a A::user01@demo.local:rxtncy 2 top-dir
[root@linuxclient nfs]# nfs4_setfacl -x 2 top-dir
[root@linuxclient nfs]# nfs4_getfacl top-dir
A::OWNER@:rwaDdxtTnNcCoy
A::GROUP@:rxtncy
A::EVERYONE@:rxtncy
```

remove an entry by specifying the complete entry

remove an entry by specifying the index of an entry

Figure 49 Remove an NFSv4 ACL entry

To modify an ACL entry, use the **nfs4_setfacl** command with the **-m** option as shown in Figure 50.

```
[root@linuxclient nfs]# nfs4_getfacl top-dir/
A::OWNER@:rwaDdxtTnNcCoy
A::user01@demo.local:r
A::GROUP@:rxtncy
A::EVERYONE@:rxtncy
[root@linuxclient nfs]# nfs4_setfacl -m A::user01@demo.local:r A::user01@demo.local:rxtncy top-dir
[root@linuxclient nfs]# nfs4_getfacl top-dir/
A::OWNER@:rwaDdxtTnNcCoy
A::user01@demo.local:rxtncy
A::GROUP@:rxtncy
A::EVERYONE@:rxtncy
```

Figure 50 Modify an NFSv4 ACL entry

Besides modifying an ACL using a command, you can also use **nfs4_setfacl** with the **-e** option to open an editor program and edit the ACL as text.

Example 2: ACE inheritance

Add an ACL entry with **f** and **d** inheritance flags specified, which is similar to Windows (OI)(CI) flags. This will makes the ACE apply to the current directory and propagate to subdirectories and files within the **top-dir** directory.

```
[root@linuxclient nfs]# nfs4_setfacl -a A:fd:user01@demo.local:rxtncy 2 top-dir
[root@linuxclient nfs]# nfs4_getfacl top-dir/
A::OWNER@:rwaDdxtTnNcCoy
A:fd:user01@demo.local:rxtncy
A::GROUP@:rxtncy
A::EVERYONE@:rxtncy
```

Figure 51 NFSv4 ACL with inheritance flags

Now, create a new directory under **top-dir**. As Figure 52 shows, the ACE with flags **f** and **d** are specified in the parent directory and have been propagated to the new directory.

```
[root@linuxclient top-dir]# pwd
/mnt/nfs/top-dir
[root@linuxclient top-dir]# mkdir dir-1
[root@linuxclient top-dir]# nfs4_getfacl dir-1
A:fd:user01@demo.local:rxtncy
```

Figure 52 Inherited ACL entry

A.3 Linux POSIX ACL and NFSv3

On common question is if NFSv3 is compatible with POSIX ACL. There is no ACL specification defined in the NFSv3 RFC document, so OneFS does not support POSIX ACL for NFSv3. In OneFS, you can only use POSIX mode bits while using NFSv3.

This section introduces how POSIX ACL works and includes examples.

Traditionally, Linux uses a simple file-system permission model. The model defines three classes of users: owner, group, and other. Each user class is granted a set of permissions. The permissions defined are read (r), write (w), and execute (x). This model is usually referred as POSIX mode bits. The **ls -l** command displays

the permissions. For example, **-rw-rw-r--** indicates a file with read and write access for owner, read and write access for group, and read access only for other.

For a more granular access control, many Linux operating systems have implemented ACLs based on the POSIX 1003.1e/1003.2c Draft Standard 17. This ACL is known as POSIX ACL. These ACLs were added to Linux kernel version 2.5.46 in November 2002. This is an extended scheme for the traditional mode-bits permission model. Each ACL consists of a list of ACEs. Each of the three classes of users in POSIX mode bits is represented by an ACE. Additional ACEs can be added to grant permission for additional users or groups.

A.3.1 POSIX ACL types

There are two types of POSIX ACL:

- **Access ACL:** This defines the current access permissions of a file-system object and is used in the file-system object access-check process.
- **Default ACL:** This defines the inherited permissions from the parent directory when a file system object is created. The default ACL is not used in the access-check process. Under a directory that has a default ACL, a newly created directory inherits the parent directory's default ACL both as its access ACL and default ACL, and a newly created nondirectory object inherits the default ACL of the parent directory as its access ACL only.

A.3.2 POSIX ACL entries and POSIX mode bits

Each POSIX ACL entry is formed with the format **type:qualifier:permissions,** and the qualifiers are undefined for **owner**, **group**, **mask**, and **other**. As Table 21 shows, there are six types of POSIX ACL entries.

Table 21 POSIX ACL entries

Entry type	Text format
owner	user::rwx
named user	user:username:rwx
owning group	group::rwx
named group	group:groupname:rwx
mask	mask::rwx
other	other::rwx

POSIX ACLs consist of three entries (**owner, group, other**) that are called minimal ACLs, and they are equivalent with POSIX mode bits. POSIX ACLs with more than the three entries are called extended ACLs. Extended ACLs consists of additional **mask** entry, **named user,** and **named group** entries.

As Figure 53 shows, **named user** and **named group** entries belong to the group class, which already contains the owning group entry. In the extended ACL, the group class permissions alone are not expressive enough to represent all the ACL entries permissions it contains. The **mask** entry is defined to solve this problem. It redefines the group class to represent an upper bound of the permissions. With minimal ACLs, the group class permissions map to the owning group entry permissions. With extended ACLs, the group class permissions are mapped to the mask entry permissions, whereas the owning group entry still defines the owning group permissions.

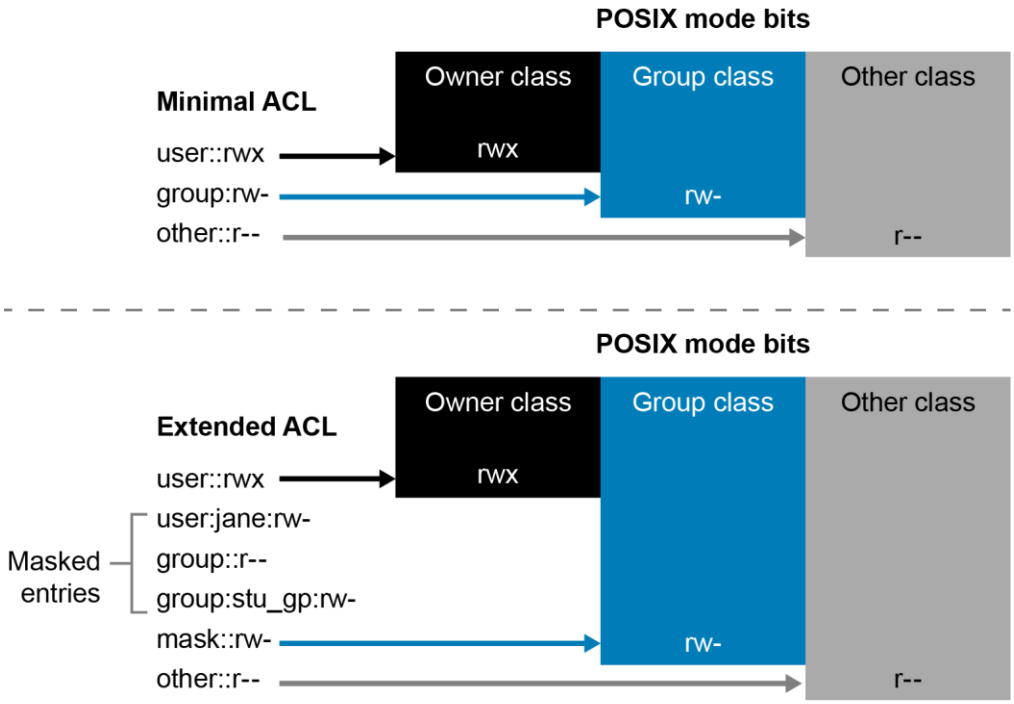

Figure 53 POSIX ACL entries and POSIX mode bits

A.3.3 Examples

The default ACL is used to determine the newly created file and directory permissions, and the **umask** and **mode** parameters are also used to calculate their initial permissions. The mode parameter contains nine permission bits that stand for the permissions of the owner, group, and other class permissions. By default, the **umask** value is **022**, and the **mode** value is **666** for **touch** command and **777** for **mkdir** command. Here is the algorithm for the initial access ACL of a new file or directory.

- If the parent directory has a default ACL, the effective permissions of each class are set to the intersection of the permissions defined for this class in the default ACL and specified in the **mode** parameter.
- If the parent directory has no default ACL, the effective permissions are set to the permissions defined in the **mode** parameter, minus the permissions set in the current **umask**. The **umask** has no effect if a default ACL exists.

The following examples illustrate POSIX ACL and POSIX mode bits.

Example 1: Effective permissions when the parent has no default ACL

Create a parent directory without a default ACL, as shown as Figure 54.

```
[root]# umask
022
[root]# mkdir p_dir
[root]# ls -dl p dir
drwxr-xr-x 2 root 6 Jul  8 22:14 p dir/
[root]# getfacl --omit-header p_dir
user::rwx
group::r-x
other::r-x
```

Figure 54 Parent directory creation

Create a sub-directory under **p_dir**. Because the parent directory has no default ACL, the initial access ACL for the **sub_dir** is based on result of permission defined in **mode** (777), minus the permissions defined in **umask** (022). The initial permission is 755, which stands for **rwx** for owner, **r-x** for group, and **r-x** for other. See Figure 55.

```
[root]# umask
022
[root]# mkdir sub_dir
[root]# ls -dl sub_dir
drwxr-xr-x 2 root 6 Jul  8 23:33 sub dir/
[root]# getfacl --omit-header sub_dir
user::rwx
group::r-x
other::r-x
```

Figure 55 Initial access ACL for a directory

Now, add a named user ACL entry to give read, write, and execute permissions to user **jane** on directory **sub_dir**. To do this, use the **setfacl** command with the **–m** option. Meanwhile, the mask ACL entry is autogenerated as needed. See Figure 56.

```
[root]# setfacl -m user:jane:rwx sub_dir
[root]# getfacl --omit-header sub_dir
user::rwx
user:jane:rwx
group::r-x
mask::rwx
other::r-x
```

Figure 56 Add a named-user ACL entry

Using the **ls** command to check the POSIX mode bits again, the **mask** ACL entry maps to the group class permissions. However, the permissions for the owning group are still **r-x**. An additional **+** character is displayed after the permissions of all files that have extended ACLs.

```
[root]# ls -dl sub_dir
drwxrwxr-x+ 2 root 6 Jul  8 23:33 sub_dir/
```

Figure 57 Checking the result

Now, use the **chmod** command to modify the group class permissions as shown in. If no mask entry exists, **chmod** modifies the permissions of the owning group entry as it does traditionally. This example removes write access from the group class as shown in Figure 58. If an ACL entry contains permissions that are disabled by the mask entry, **getfacl** adds a comment that shows the effective set of permissions granted by that entry.

```
[root]# chmod 755 sub_dir
[root]# getfacl --omit-header sub_dir
user::rwx
user:jane:rwx                   #effective:r-x
group::r-x
mask::r-x
other::r-x
```

Figure 58 Modify permissions with the chmod command

Example 2: Parent directory default ACL

Create a parent directory that has a default ACL as shown in Figure 59. Use the **setfacl** command with the **–d** option to modify default ACL. This example only adds an ACL entry for the **stu_gp** group in the **setfacl** command. The other entries require entries for owner, owning group, and others are automatically copied from the current access ACL to the default ACL.

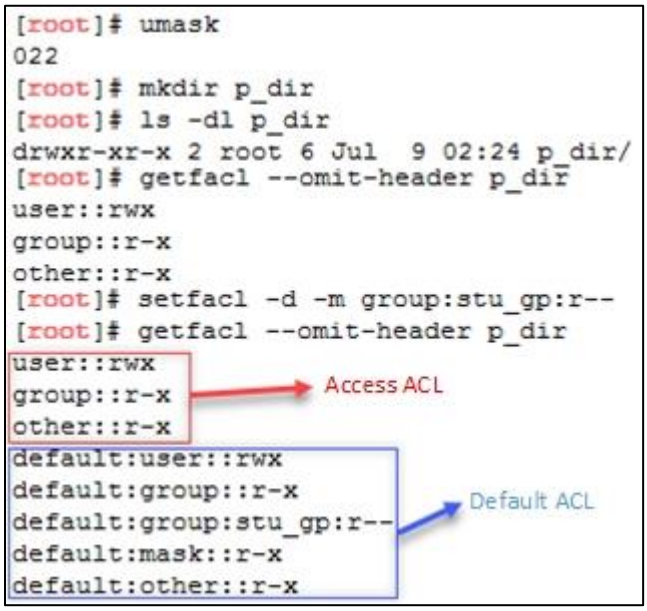

Figure 59 Parent directory with the default ACL

Now, create a new directory under the parent directory **p_dir**. When creating new directory with **mkdir**, the command uses **777** as the mode value by default, so the parent directory's default value is inherited as the initial access ACL and default ACL of the **sub_dir** directory.

```
[root]# mkdir sub_dir
[root]# ls -dl sub_dir
drwxr-xr-x+ 2 root 6 Jul  9 02:47 sub_dir/
[root]# getfacl --omit-header sub_dir
user::rwx
group::r-x
group:stu_gp:r--
mask::r-x
other::r-x
default:user::rwx
default:group::r-x
default:group:stu_gp:r--
default:mask::r-x
default:other::r-x
```

Figure 60 Sub-directory creation

Similarly, create a new file under the parent directory **p_dir**. When creating the new file with the **touch** command, the **mode** value is **666** by default, so the effective permissions of each class are set to the intersection of the permissions defined in the default ACL and the permissions defined in the **mode** parameter.

```
[root]# touch sub_file
[root]# ls -l sub_file
-rw-r--r--+ 1 root 0 Jul  9 03:06 sub_file
[root]# getfacl --omit-header sub_file
user::rw-
group::r-x                      #effective:r--
group:stu_gp:r--
mask::r--
other::r--
```

Figure 61 File creation with the default ACL

B Technical support and resources

Dell.com/support is focused on meeting customer needs with proven services and support.

See the following additional resources for more information:

- PowerScale Info Hub
- Dell EMC PowerScale OneFS: A Technical Overview
- PowerScale OneFS Web Administration Guide
- PowerScale OneFS CLI Administration Guide
- Current PowerScale Software Releases
- OneFS Security Configuration Guide