White Paper

EMC ISILON MULTIPROTOCOL DATA ACCESS WITH A UNIFIED SECURITY MODEL

Abstract

This paper explains how the unified security model of EMC Isilon OneFS 6.5.5 resolves mismatches between the permissions of Windows and UNIX systems while preserving the security of files and satisfying the expectations of users.

August 2012

EMC²

Part Number H10920

Table of Contents

1. Introduction

The EMC® Isilon® OneFS® operating system delivers seamless multiprotocol data access over Server Message Block (SMB) and Network File System (NFS) with a unified security model. SMB is a file sharing protocol that gives computers running Microsoft® Windows® access to the OneFS file system. NFS is a distributed file system protocol that gives UNIX and Linux systems access to OneFS.

OneFS enables Windows and UNIX clients to interoperate: They can access the same file, regardless of the operating system from which the file originated and regardless of the protocol used to store the file. To provide secure cross-platform file access, OneFS maps user accounts across different identity management systems. For example, Windows users managed in Microsoft Active Directory are transparently mapped to their corresponding UNIX accounts managed in Network Information Service (NIS) or Lightweight Directory Access Protocol (LDAP).

More powerful, though, is the way in which OneFS preserves file permissions across operating systems and protocols. Although the security model used by Windows systems and the SMB protocol is different from the security model used by UNIX systems and the NFS protocol, an EMC Isilon cluster works with both the access control lists (ACLs) of Windows systems and the Portable Operating System Interface (POSIX) mode bits of UNIX systems.

By default, OneFS maintains file permissions regardless of the client's operating system, the user's identity management system, or the file sharing protocol. When OneFS must transform a file's permissions from ACLs to mode bits or vice versa, it merges the permissions into an optimal representation that uniquely balances user expectations and file security. This paper explains how OneFS unites permissions to provide seamless, secure multiprotocol data access.

This introduction presents an overview of the Windows and UNIX security models, and covers some common terminology for data access protocols, user accounts, and file permissions.

The second section of this paper describes the EMC Isilon unified security model and explains how OneFS resolves mismatches between the permissions of Windows and UNIX systems while preserving their security restrictions and satisfying the expectations of users. In most cases, the Isilon cluster's default settings handle permissions securely and effectively for networks that mix UNIX and Windows systems. Several examples demonstrate how the model manages security not only with ease and automation but also in a way that optimally resolves conflicts to preserve file security.

To deal with those cases when the default settings lead to unexpected behavior, the third section of this paper analyzes several use cases to illustrate how you can get the results you want, either by manually configuring OneFS with ACL policies or by using a preset configuration.

1.1. File systems and security models

The file systems of Windows and UNIX computers not only have different network access protocols—SMB for Windows and NFS for UNIX—but also have different security models for user accounts and file permissions.

To process a user account, a Windows computer retrieves a security identifier (SID) from an authority, typically a Windows domain controller, when a user logs in. The computer also retrieves security identifiers for the user's groups and generates an access token, which includes a user's group memberships. Most enterprises manage Windows user accounts and group memberships with Microsoft Active Directory.

To manage file permissions, a Windows computer uses a discretionary access control list (DACL) to list the users and groups that can gain access to a file and perform write, read, and execute actions. A DACL is frequently referred to simply as an ACL, and this paper follows that convention except in cases when it must be distinguished from its less well-known counterpart, a system access control list (SACL), which is used for logging and auditing access to system objects.

A DACL contains an ordered list of access control entries. Each access control entity (ACE) specifies whether access is allowed or denied for a given user or group. An ACE also lists the rights, such as the right to read a file, that are explicitly allowed or denied.

Together, a file owner's SID and an object's ACL form a security descriptor, a data structure that captures the security information for a securable Windows object, such as a file or directory.

Because the Windows security model is exposed through the SMB protocol, the file system of a server that works with SMB, such as OneFS, can process security descriptors. The descriptors enable the file system to control the way users and groups interact with files, directories, and other securable objects. When a user attempts to access a file protected with an ACL by using SMB, OneFS processes the permissions in the standard way used by a Windows file system. (The Windows file system is commonly known as New Technology File System, or NTFS.) An attempt by a Windows user to gain access to a file with an ACL over SMB proceeds as follows:

1. The user logs in to a system that works with SMB, such as OneFS. During login, the system generates an access token for the user. The token contains security identifiers for the user and group accounts to which the user belongs.
2. The user attempts to access a file, prompting the system to request a set of access rights for the file.
3. The system checks for ACEs in the ACL that apply to the user identified in the access token. The system examines each ACE in sequence until one of the following events occurs:
    - An access-denied ACE explicitly denies any of the requested access rights to the user.
    - One or more access-allowed ACEs explicitly grant all of the requested access rights.
    - All the ACEs have been checked and there remains at least one requested access right that has not been explicitly allowed for a case in which access is implicitly denied.

For more information on how Windows systems and the SMB protocol process access requests, see the MSDN section on access control.

In contrast to the Windows model, a UNIX user account sets a user's identity and group memberships with a user identifier (UID) and one or more group identifiers (GIDs). UNIX accounts are often managed in identity management systems such as LDAP, NIS, and, less frequently, Active Directory.

For file permissions, UNIX and Linux file systems typically use what are commonly called mode bits—a concatenated set of three read-write-execute triplets. As part of the set of standards known as the POSIX, the mode bits control the actions that the owner, the group, and everyone else can perform.

When a UNIX user gains access to a file protected with mode bits by using NFS, OneFS interprets the permissions in the same way as any other POSIX file system:

1. The system checks the file's permissions.
2. If the user is the file's owner, the system uses the owner mode bits to allow or deny access to the file.
3. If the user is not the file's owner, but is a member of the file's group, the system uses the group bits to allow or deny access to the file.
4. If the user is neither the file's owner nor a member of the file's group, the system uses the other bits to allow or deny access to the file.

The same process holds true for directories, symbolic links, and other system objects that can be secured with permissions. For simplicity, most of this paper focuses on files, but the same points frequently apply to directories and other securable system objects.

1.2. Multiprotocol data access

OneFS includes a multiprotocol file service that gives both UNIX and Windows users access to content over NFS and SMB regardless of the original protocol that was used to store it.

For NFS, OneFS works with versions 2 through 4 of the Network File System protocol (NFSv2, NFSv3, and NFSv4). The Isilon cluster's default export (/ifs) enables Linux and UNIX clients to remotely mount any subdirectory, including subdirectories created by Windows users. Linux and UNIX clients can also mount ACL-protected subdirectories that a OneFS administrator created.

For SMB, OneFS works with versions 1 and 2 (SMB1 and SMB2). The Isilon cluster's default file share (/ifs) gives Windows users access to file system resources over the network, including resources stored by UNIX and Linux systems.

With such capabilities, OneFS creates a heterogeneous environment with four primary types of network access to files. Keep in mind that the same kinds of access also apply to directories and other securable system objects.

- A UNIX client accessing a file stored on the Isilon cluster over NFS
- A Windows client accessing a file stored on the cluster over SMB
- A UNIX client connecting by NFS to a file that was stored on the cluster by a Windows client over SMB
- A Windows client connecting by SMB to a file that was stored on the cluster by a UNIX client over NFS

When a UNIX user requests a file protected with POSIX mode bits over NFS, OneFS controls access by using the file's POSIX permissions. The process is similar when a Windows user requests a file with an ACL: The rights in the user's access token are evaluated against the file's ACL under the Windows security model outlined in the previous section.

When a Windows user attempts to access a UNIX file, a problem occurs because the permissions set with POSIX mode bits are incompatible with the Windows security model. Similarly, when a UNIX user requests access to a Windows file, the permissions set with an ACL do not work with the UNIX security model.

Because the two models are different and because the Windows model has a richer set of rights, there is no one-to-one mapping between the two types of permissions. As a result, a file's security could be compromised, or a user who expects to gain access to a file could be denied access.

Assuming the system can handle the comparatively easy task of mapping identities across SIDs and UIDs, the solution depends on how the multiprotocol file system resolves the permissions mismatch while satisfying the expectations of users and preserving the security of files.

The differences between the two models are compounded when you change the permissions of a file that came from a Windows computer by using commands such as chmod, chown, and chgrp on a UNIX client. Another problem stems from how UNIX and Windows systems process the rights of groups to access a file. The UNIX security model associates only a single group with a file, while the Windows model supports multiple group associations.

1.3. Approaches to permissions management

There are three primary approaches to addressing the permissions mismatch between UNIX and Windows systems:

- Maintain last-touched behavior across protocols
- Store permissions separately for each protocol
- Unify the permissions for both protocols

Maintaining last-touched behavior renders the security of directories and files unpredictable. The approach retains the permissions set during the last action taken on the file, permanently overwriting previous settings. You can continually overwrite restrictions without intending to do so. The result is often inconsistent permissions, which can lead to confused users. Although it is not recommended, you can apply a similar approach by configuring OneFS to replace ACLs with UNIX permissions.

Storing permissions separately entails two security models running at the same time—one for the NFS protocol and another for the SMB protocol. In the context of multiprotocol access, there are two different, irreconcilable representations of the same data, leading to problems of synchronization and consistency. For example, when you change the permissions of a file stored over NFS by using the chmod command, the changes are not typically applied to the version accessible over SMB—unless you remember to change the same file's permissions over SMB as well. With two independent representations of the same permissions data, the

permissions of the same file can differ. The expectations of users and the security of files can become sacrificed for ease of implementation.

Unifying permissions for both protocols balances user expectations with file security. It ensures that permissions persist across protocols, remain consistent, and retain their intended security restrictions.

2. The unified security model of OneFS

The OneFS solution unites the UNIX and Windows permissions models to provide seamless, secure file sharing across protocols and operating systems. In designing its solution, EMC Isilon aimed to optimally balance ease of access for all identity types, user expectations about gaining access and running commands, and uniform security across protocols. The technical solution is marked by the following design goals:

- Meet the expectations of users while maintaining consistent security settings across protocols
- Enforce the same permissions for all protocols, avoiding the pitfalls of maintaining separate per-protocol permissions that some other file systems use
- Retain the original security settings across protocols unless they have been explicitly overwritten, avoiding the pitfalls of the last-touched behavior that some other file systems use
- Store an authoritative set of identities and permissions so that administrators can directly view the actual permissions on the cluster
- Present protocol-specific views of permissions so that NFS exports display mode bits and SMB shares show ACLs
- Enhance standard UNIX tools—such as the chmod, chown, and chgrp commands—to conveniently and consistently manage not only mode bits but also ACLs
- Provide settings to control how permissions are managed, inherited, and mapped so that users can optionally customize the cluster to achieve the results they want

To meet these goals, OneFS must map identities and permissions across protocols. To do so, Isilon uses several constructs:

- An on-disk identity that transparently maps to the right identifier for the requesting protocol.
- An on-disk authoritative representation of an object's permissions. Only one set of permissions—either ACLs or mode bits—is authoritative. The other set is a virtual approximation based on its internal representation. The authoritative representation preserves the file's original permissions.
- An internal representation of a file system object's permissions, such as a directory or a file, that contains information from the POSIX mode bits, the ACL, or both. The internal representation, which is partly based on RFC 3530, mediates between the access rights of the Windows security model and the mode bits of the POSIX security model. The internal representation helps enforce the same permissions across protocols.

- A synthetic ACL that approximates the mode bits of a UNIX file for an SMB client. A synthetic ACL illustrates the difference between a file's authoritative representation and its internal representation. The authoritative representation of a UNIX file stored over NFS is its mode bits; the file's internal representation is an estimation of the mode bits that, based on RFC 3530, is used to generate a synthetic ACL for SMB clients.
- A policy engine that transforms the authoritative representation of an object's permissions into a form that the requesting access protocol can process.
- A default ACL policy that, when a chmod command is run on a file with an ACL, optimally merges the new permissions with the old ones.
- ACL policies that optionally enable users to configure, either manually or globally, how OneFS manages permissions for different environments.

2.1. On-disk identity

OneFS freely mixes Windows and UNIX identities. It uses an on-disk identity to transparently map the requesting protocol to the right identifier. A user connecting to the cluster over NFS with a UID and GID is mapped to a SID for access to files that another user stored by using SMB. In the same way, a user connecting to the cluster over SMB with a SID is mapped to a UID and GID to access files stored by a UNIX client. On OneFS, a security descriptor can contain not only a user and group's SID, but also a UID and GID.

OneFS works with multiple identity management systems, including the following:

- Microsoft Active Directory (AD), a popular directory service that uses domain controllers to authenticate users and authorize access to personal computers, servers, and other network resources. To work with UNIX and Linux systems, Active Directory includes support for RFC 2307.
- Lightweight Directory Access Protocol (LDAP), an application protocol for querying and modifying directory services over TCP/IP.
- Network Information Service (NIS), a client/server directory service for distributing system information such as user names.
- Local users and local groups that have both Windows and UNIX account properties.
- File provider for working with accounts in /etc/spwd.db and /etc/group files. The file provider enables you to add an authoritative third-party source of user and group information.

When a user connects to a cluster, OneFS expands the user's identity to include the user's other identities and groups. Based on the identity mappings, OneFS generates an access token. When a user attempts to access a file, OneFS compares the access token generated during the connection with the authorization data on the file. All user and identity mapping takes place when a token is generated; no mapping is performed when permissions are evaluated.

Even though OneFS expands an identity to include identities from other identity management systems, OneFS stores an authoritative version of it—the preferred on-disk identity. The on-disk identity types are UNIX, SID, and native. When upgrading from a previous version of OneFS, the on-disk identity is set to UNIX, which was the default the identity type on older versions of OneFS. The SID on-disk identity is for a homogeneous network of Windows systems managed with Active Directory.

On new installations, the on-disk identity is set to native. The native identity is likely to be best for a network with UNIX and Windows systems.

If the UNIX on-disk identity type is set, the system authentication daemon, known as lsassd, looks up incoming SIDs in the authentication sources. If a match is found, the SID is converted to either a UID or GID. If the identity is not on the cluster because, for example, the identity is local to the client or part of an untrusted AD domain, the SID is stored instead.

Similarly, for the SID on-disk identity, the authentication daemon searches the authentication sources for matches to an incoming UID or GID. If no match is found, the UNIX identity is stored on disk.

For the native on-disk identity type, the authentication daemon attempts to select the correct identity to store on disk by checking for the identity mapping types in the following order:

1. Algorithmic mapping: A SID that matches S-1-22-1-UID or S-1-22-2-GID is converted back to the corresponding UNIX identity and set as the on-disk identity.
2. External mapping: An object that has an explicit UNIX identity defined in an external source like AD, LDAP, or NIS has that identity set as the on-disk identity.
3. Mappings stored persistently in the identity mapper database: An identity with a persistent mapping in the identity mapper database uses the destination of that mapping as the on-disk identity, a situation that primarily occurs with manual mappings. For example, if there is a mapping of GID:10000 to S-1-5-32-545, a request for the on-disk storage of GID:10000 returns S-1-5-32-545.
4. If an object lacks a UID or GID, its SID is set as the on-disk identity. In addition, OneFS allocates a UID and GID from a preset range for accessing files over NFS.

You can view the on-disk identity by logging in to OneFS with ssh and running the authentication mapping command as in the following example:

```
isi auth mapping token -w --name=W2K3I-UNO\\jsmith

Initial name: W2K3I-UNO\jsmith
Final Token
-------------------------------------------------------
Primary uid: W2K3I-UNO\jsmith (1000000)
Primary user sid: W2K3I-UNO\jsmith
 (SID:S-1-5-21-3542649673-1571749849-686233814-1117)
Primary gid: W2K3I-UNO\domain users (1000000)
Primary group sid: SID:S-1-5-21-3542649673-1571749849-686233814-513
On-disk user identity: W2K3I-UNO\jsmith
 (SID:S-1-5-21-3542649673-1571749849-686233814-1117)
On-disk group identity: W2K3I-UNO\domain users
 (SID:S-1-5-21-3542649673-1571749849-686233814-513)
Additional Identities:
   W2K3I-UNO\marketing
 (SID:S-1-5-21-3542649673-1571749849-686233814-1109)
   W2K3I-UNO\marketing (GID:1000001)
```

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

For more information, see the topics in the OneFS User Guide on configuring the on-disk identity and configuring settings for identity mapping. If you change the on-disk identity, a best practice is to run the repair permissions job. For more information, see the OneFS User Guide. The EMC Isilon documentation portal includes additional best practices for setting up a cluster to work with multiple directory services.

2.2. On-disk permissions

Compared with storing an on-disk identity, storing permissions poses a more complex problem. Because each data access protocol requires the use of its own permissions model, OneFS must not only store an authoritative version of the original permissions for the file sharing protocol that stored the file, but also map the authoritative permissions to a form acceptable to the other protocol. In doing so, OneFS must maintain the file's security settings and meet user expectations for access. The result of the translation preserves the intended security settings on the file. Users and applications can continue to access the file consistently and predictably.

To deliver cross-protocol file access seamlessly, OneFS stores an internal representation of a file system object's permissions, such as a directory or a file. The internal representation, which can contain information from the POSIX mode bits, the ACL, or both, is based on RFC 3530. A guiding principal of RFC 3530 is that a file's permissions must not make it appear more secure than it really is. The internal representation if the permissions is used to generate a synthetic ACL, which is an in-memory structure that approximates the mode bits of a UNIX file for an SMB client. Because OneFS derives the synthetic ACL from mode bits, it can express only as much permission information as mode bits can.

Because each access protocol can process only its native permissions, OneFS transforms its representation of the permissions into a form that the access protocol can accept. But because there is no one-to-one mapping between the permissions models of the two protocols, there are some subtle differences in the way the security settings map across protocols. Because the ACL model is richer than the POSIX model, no permissions information is lost when POSIX mode bits are mapped to ACLs. When ACLs are mapped to mode bits, however, ACLs must be approximated as mode bits. As a result, some information may be lost, especially when the permissions of a file with an ACL are changed from a UNIX client with the chmod or chown commands, a case that is discussed in a later section. Of utmost importance is ensuring that the mappings do not make a file appear more secure than it really is.

2.3. Mapping windows permissions to UNIX mode bits

This section describes how OneFS maps for file and directory permissions across the SMB and NFS protocols when OneFS is running with its default settings. Table 1 maps the access rights of the Microsoft Windows security model for files and directories to the OneFS internal representation and to POSIX mode bits. The table shows how OneFS represents an ACL as POSIX mode bits when a client using NFS attempts to gain access to a file with an ACL. The table does not necessarily describe how the POSIX mode bits appear over NFS. See Section 2.8 for a description of how actual access rights can differ from mode bits.

The mapping rules that Isilon developed were influenced by two documents: RFC 3530, Network File System (NFS) version 4 Protocol, and a network working group Internet draft on mapping between NFSv4 and POSIX draft ACLs.

Table 1 reflects the mapping for client computers running Microsoft Windows XP or later and for servers running Windows Server 2003 or later. The Windows access rights for files and directories include five standard access rights, which are the last five rows in the table.

Table 1. Mapping windows access rights to OneFS permissions and POSIX mode bits

Windows access right	OneFS internal representation	Mode bits approximation
FILE_ADD_FILE	add_file	d-w-
FILE_ADD_SUBDIRECTORY	add_subdir	d-w-
FILE_ALL_ACCESS	dir_gen_all, file_gen_all	drwx or -rwx
FILE_APPEND_DATA	append, add_subdir	--w- or d-w-
FILE_DELETE_CHILD	delete_child	d-w-
FILE_EXECUTE	execute	---x
FILE_LIST_DIRECTORY	list	dr--
FILE_READ_ATTRIBUTES	file_read_attr	-r--
FILE_READ_DATA	file_read	-r--
FILE_READ_EA	file_read_ext_attr	-r--
FILE_TRAVERSE	traverse	d--x
FILE_WRITE_ATTRIBUTES	file_write_attr	--w-
FILE_WRITE_DATA	file_write	--w-
FILE_WRITE_EA	file_write_ext_attr	--w-
DELETE	std_delete	d-w- or --w-
READ_CONTROL	std_read_dac	dr-- or -r--
WRITE_DAC	std_write_dac	drwx or -rwx
WRITE_OWNER	std_write_owner	drwx or -rwx
SYNCHRONIZE	std_synchronize	NA

For descriptions of the Windows constants, see File and Directory Access Rights Constants at MSDN. For more information on standard access rights, see MSDN.

In addition, Windows has generic access rights for files and directories. By default, when OneFS finds generic access rights, it maps them to each specific constant, and then maps the constants to mode bits, as shown in Table 1. The Windows generic access rights are shown in Table 2.

Table 2. Windows generic access rights

Generic access right	Included constants
FILE_GENERIC_EXECUTE	FILE_EXECUTE FILE_READ_ATTRIBUTES STANDARD_RIGHTS_EXECUTE SYNCHRONIZE
FILE_GENERIC_READ:	FILE_READ_ATTRIBUTES FILE_READ_DATA FILE_READ_EA STANDARD_RIGHTS_READ SYNCHRONIZE
FILE_GENERIC_WRITE:	FILE_APPEND_DATA FILE_WRITE_ATTRIBUTES FILE_WRITE_DATA FILE_WRITE_EA STANDARD_RIGHTS_WRITE SYNCHRONIZE

For more information on generic access rights, see MSDN.

2.4. Mapping mode bits to ACLs

Because mode bits are a subset of the richer Windows ACL model, mapping mode bits to ACLs is simpler. Tables 3 through 5 demonstrate how OneFS processes mode bits to create a synthetic ACL when an SMB client attempts to access a file or a directory with mode bits. Because the security model for ACLs is richer than that of mode bits, no information is lost. The tables cover files and directories for user, group, and other.

Table 3. Mapping for UNIX Read

Description	Permissions
UNIX permission	Read
OneFS representation	file_gen_read
Windows Effective Permissions	list folder/read data
Mapping to Windows access rights constants	FILE_LIST_DIRECTORY, FILE_READ_ATTRIBUTES, FILE_READ_DATA, FILE_READ_EA

Table 4. Mapping for UNIX Write

Description	Permissions
UNIX permission	Write
OneFS representation	file_gen_write
Windows Effective Permissions	create files/write data; create folders/append Data; delete subfolders and files
Mapping to Windows access rights constants	FILE_ADD_FILE, FILE_WRITE_DATA; FILE_ADD_SUBDIRECTORY, FILE_APPEND_DATA; DELETE, FILE_DELETE_CHILD, FILE_WRITE_ATTRIBUTES, FILE_READ_EA

Table 5. Mapping for UNIX Execute

Description	Permissions
UNIX permission	Execute
OneFS representation	file_gen_execute
Windows Effective Permissions	traverse folder/execute file
Mapping to Windows access rights constants	FILE_TRAVERSE, FILE_EXECUTE

In addition, the mode rwx is mapped to full control (FILE_ALL_ACCESS), which is represented on OneFS as file_gen_all. As such, a user, a group, or everyone with the mode bit set to rwx includes the following additional effective permissions: change permissions, take ownership, delete, and synchronize (WRITE_DAC, WRITE_OWNER, DELETE, and SYNCHRONIZE). A OneFS ACL policy enables you to retain rwx permissions without including full control. See Section 3 for information on ACL policies for mixed environments.

2.5. OneFS permissions for file system objects

Like the Windows permissions model, the Isilon system of representing permissions divides permissions into three related groups: standard permissions, which can apply to any object in the file system; generic permissions, which are logical wrappers for a bundle of more specific permissions; and constants, each of which is a specific type of permission. In addition, certain permissions apply only to a directory; others apply only to a nondirectory file system object.

The standard permissions that can appear on OneFS in the listing for a file system object are shown in Table 6.

Table 6. OneFS standard permissions

Permission	Description
std_delete	The right to delete the object
std_read_dac	The right to read the security descriptor, not including the SACL. (In OneFS, a superuser can list the SACL, but it is otherwise unsupported.)
std_write_dac	The right to modify the DACL in the object's security descriptor
std_write_owner	The right to change the owner in the object's security descriptor
std_synchronize	The right to use the object as a thread synchronization primitive. (On OneFS, this right has no effect.)
std_required	Maps to std_delete, std_read_dac, std_write_dac, and std_write_owner

Table 7 shows permissions that can appear only for a directory.

Table 7. OneFS directory permissions

Permission	Description
dir_gen_all	Read, write, and execute access
dir_gen_read	Read access
dir_gen_write	Write access
dir_gen_execute	Execute access
list	List entries
add_file	The right to create a file in the directory
add_subdir	The right to create a subdirectory
delete_child	The right to delete children, including read-only files
traverse	The right to traverse the directory
dir_read_attr	The right to read directory attributes
dir_write_attr	The right to write directory attributes
dir_read_ext_attr	The right to read extended directory attributes
dir_write_ext_attr	The right to write extended directory attributes

OneFS maps the generic permissions for directories to more specific permissions as shown in Table 8.

Table 8. Specific permissions for directories

Permission	Description
dir_gen_read	list, dir_read_attr, dir_read_ext_attr, std_read_dac, and std_synchronize
dir_gen_write	add_file, add_subdir, dir_write_attr, dir_write_ext_attr, std_read_dac, and std_synchronize
dir_gen_execute	traverse, std_read_dac, and std_synchronize
dir_gen_all	dir_gen_read, dir_gen_write, dir_gen_execute, delete_child, and std_write_owner

Table 9 shows permissions that can appear in the listing for a nondirectory file system object.

Table 9. OneFS permission for nondirectory objects

Permission	Description
file_gen_all	Read, write, and execute access
file_gen_read	Read access
file_gen_write	Write access
file_gen_execute	Execute access
file_read	The right to read file data
file_write	The right to write file data
append	The right to append to a file
execute	The right to execute a file
delete_child	This permission is not used for a file, but can be set for Windows compatibility.
file_read_attr	The right to read file attributes
file_write_attr	The right to write file attributes
file_read_ext_attr	The right to read extended file attributes
file_write_ext_attr	The right to write extended file attributes

OneFS maps the generic permissions for nondirectory file system objects to more specific permissions as shown in Table 10.

Table 10. Specific permissions for nondirectory objects

Permission	Description
file_gen_read	file_read, file_read_attr, file_read_ext_attr, std_read_dac, and std_synchronize
file_gen_write	file_write, file_write_attr, file_write_ext_attr, append, std_read_dac, and std_synchronize
file_gen_execute	execute, std_read_dac, and std_synchronize
file_gen_all	file_gen_read, file_gen_write, file_gen_execute, delete, std_write_dac, and std_write_owner

2.6. Anatomy of an ACL

On an Isilon cluster, each ACE in an ACL is presented as a single line prefaced by an index number, which starts at 0, and is followed by these parts:

- Identity: the identity to which the ACE applies
- Allow or deny: whether the ACE allows or denies the permissions listed as part of the ACE
- Permissions: a list of one or more permissions that are allowed or denied by the ACE
- Permission flags: flags that reflect the types of inheritance

The identity can be one of three types: user (listed as "user:"), group (listed as "group:"), or the special identity, everyone. For directories, it can also be one of two special template identities: creator_owner or creator_group. When present in the ACL of a containing directory, these template identities are replaced in the ACL of a newly created file system object with the specific user and group of the respective creator.

An ACE can optionally contain flags that specify whether it is inherited by child folders and files. Inheritance takes place when files and subdirectories are created; modifying an inherited rule affects only new files and subdirectories, not existing files and subdirectories. The following flags specify the types of inheritance for permissions in the ACE:

- object_inherit: Only files in this directory and its descendants inherit the ACE.
- container_inherit: Only directories in this directory and its descendants inherit the ACE.
- no_prop_inherit: This ACE will not propagate to descendants (applies to object_inherit and container_inherit ACEs).
- inherit_only: The ACE does not apply for permissions to this object, but will apply to descendants when inherited.
- inherited_ace: The ACE was inherited.

Here is an ACL listing from OneFS that shows some of these components. The listing was obtained by running the ls command with an option (le) that Isilon added to show the ACL and other information from the security descriptor. The option is available only on the Isilon cluster, not on a UNIX client that has mounted an export. See the OneFS man page for ls.

```
ls -le bar.txt
-rw-r--r-- + 1 root wheel 0 Apr 22 17:23 bar.txt
  OWNER: user:root
  GROUP: group:wheel
  0: group:Administrators allow
 std_read_dac,std_synchronize,file_read_ext_attr,file_read_attr
  1: user:root allow file_gen_read,file_gen_write,std_write_dac
  2: group:wheel allow file_gen_read
  3: everyone allow file_gen_read
```

OneFS provides methods to modify an ACL. Isilon added an option to the chmod command, for example, to manipulate ACL entries:

```
ls -lze
-rw-r--r--+ 1 juser wheel 0 Apr 28 14:06 file1
  0: user:guest deny file_read
  1: user:admin allow file_write

chmod +a# 1 group "domain users" deny file_read file1

ls -lze
-rw-r--r--+ 1 juser wheel 0 Apr 28 14:06 file1
  0: user:guest deny file_read
  1: group:"domain users" deny file_read
  2: user:admin allow file_write
```

As the examples illustrate, Isilon has added options to several commands that allow you to view and modify ACL permissions when you connect to OneFS by ssh:

```
ls -le
ls -lze
ls -len
chmod +a
```

The commands with the Isilon options appear in this paper to show how OneFS processes permissions. Keep in mind that the options are unavailable on Linux and UNIX systems that remotely mount a OneFS export. For more information about the options, see the OneFS man pages for ls and chmod.

2.7. How OneFS handles chmod permissions changes

When a Windows client changes the permissions of a file with an ACL, no information is lost because OneFS stores the original ACL and replaces it. The same holds true when a Windows client changes the permissions of a file with mode bits. Because OneFS maps the mode bits to a synthetic ACL and because the ACL model can capture the full range of POSIX permissions, no security information is lost. In such

cases, OneFS replaces the file's synthetic ACL with an actual ACL that is equivalent to the mode bits.

The situation is different when a chmod or chown command modifies the permissions of a file protected by an ACL. OneFS must map the permission changes between two noncorresponding security models. To do so, OneFS, in its default setting, merges the ACL with a patch derived from the change in mode bits.

As an example, say that you want to run the chmod command to change the permissions of a file with an ACL from 764 (rwxrw-r--) to 744 (rwxr--r--). If you are connected to the Isilon cluster by ssh, you can check the file's on-disk permissions beforehand by running the ls -le command with the root account.

In the following example, the actual permissions are listed with the generic access rights for files and directories because the file was stored by a Windows client over the SMB protocol. OneFS approximates the mode bits as 764:

```
ls -le
-rwxrw-r-- + 1 W2K3I-UNO\jsmith W2K3I-UNO\market 2056 Feb 2 10:18
 adocs.txt
 OWNER: user:W2K3I-UNO\jsmith
 GROUP: group:W2K3I-UNO\marketing
 0: user:W2K3I-UNO\jsmith allow
 file_gen_read,file_gen_write,file_gen_execute,std_write_dac
 1: group:W2K3I-UNO\marketing allow file_gen_read,file_gen_write
 2: everyone allow file_gen_read
```

When you run the chmod command to change the permissions from 764 to 744, OneFS applies the difference between 764 and 744 to the ACL, removing the write permission of the primary group (in this case, marketing) while preserving the other permissions. The actual permissions, again listed as generic access rights, change as follows:

```
chmod 744 adocs.txt

ls -le
-rwxr--r-- + 1 W2K3I-UNO\jsmith W2K3I-UNO\market 2056 Feb 2 10:18
 adocs.txt
 OWNER: user:W2K3I-UNO\jsmith
 GROUP: group:W2K3I-UNO\marketing
 0: user:W2K3I-UNO\jsmith allow
 file_gen_read,file_gen_write,file_gen_execute,std_write_dac
 1: group:W2K3I-UNO\marketing allow file_gen_read
 2: everyone allow file_gen_read
```

In most cases, a result like this preserves the ACL information and minimizes conflicts between actual and expected behavior.

2.8. How actual access rights can differ from mode bits

The following examples show how the actual permissions, as set on OneFS, can differ from the mode bits displayed on a Linux client. The examples assume that OneFS is running with its default ACL policies.

On a Linux client connected to an Isilon cluster with NFS, a file with full control for the owner approximates the ACL permissions as follows (the file was originally stored by a Windows system over SMB):

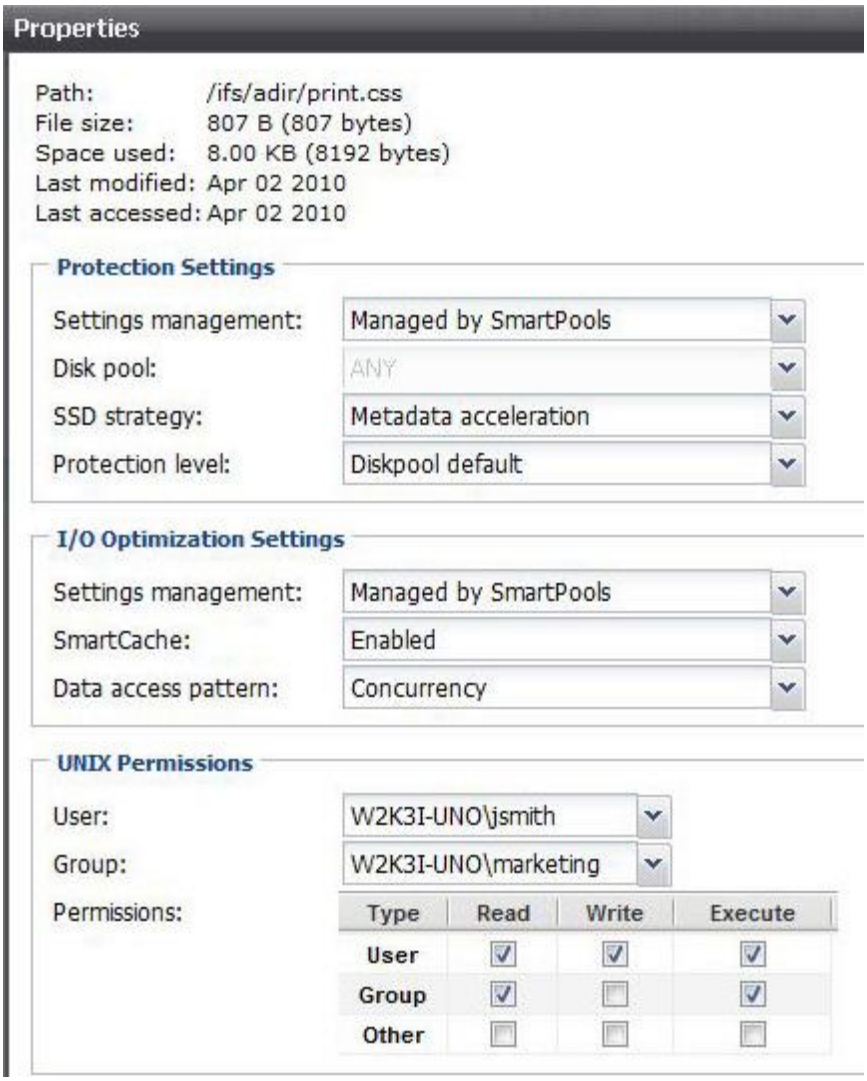
```
[root@rhel5d adir]# ls -l print.css
-rwxr-x--- 1 1000000 1000001 807 Apr 2 2010 print.css
```

By connecting to OneFS with the root account over ssh, you can see the same permissions by running the ls -l command:

```
ls -l print.css
-rwxr-x--- + 1 W2K3I-UNO\jsmith W2K3I-UNO\market 807 Apr 2 2010
 print.css
```

In the Isilon web administration interface, which is accessible over HTTPS with a web browser, the permissions shown in Figure 1 appear in their UNIX form.

Figure 1. Permissions in the Isilon Web Administration Interface



You can, however, display the actual permissions stored on disk by using the ls -le command. On OneFS, adding the e option to -l prints the ACL from the security descriptor:

```
ls -le print.css
-rwxr-x--- + 1 W2K3I-UNO\jsmith W2K3I-UNO\market 807 Apr 2 2010
 print.css
 OWNER: user:W2K3I-UNO\jsmith
 GROUP: group:W2K3I-UNO\marketing
 0: user:W2K3I-UNO\jsmith allow file_gen_all
 1: group:W2K3I-UNO\marketing allow file_gen_read,file_gen_execute
 2: everyone allow std_read_dac,std_synchronize,file_read_attr
```

Because the file originated from a Windows computer over SMB and because ACL permissions map indirectly to POSIX mode bits, the actual permissions preserve the following Windows access controls for everyone else, even though the mode bits for everyone are unset:

allow std_read_dac, std_synchronize, file_read_attr

There is another difference that the mode bits cannot capture. In addition to std_write_owner, the owner has std_write_dac—the standard access right to modify the permissions in the object's security descriptor. It is included as part of file_gen_all.

Here is another example of a right that does not map directly from mode bits to ACLs. If you run the chmod command to change the permissions from 750 (rwxr-x---) to 650 (rw-r-x---), the resulting merge removes the right to modify the owner in the object's security descriptor, leaving the user with the standard right to modify the discretionary access control list in the object's security descriptor:

```
chmod 650 print.css

ls -le print.css
-rwxr-x--- + 1 W2K3I-UNO\jsmith W2K3I-UNO\market 807 Apr 2 2010
 print.css
 OWNER: user:W2K3I-UNO\jsmith
 GROUP: group:W2K3I-UNO\marketing
 0: user:W2K3I-UNO\jsmith allow
 file_gen_read,file_gen_write,std_write_dac
 1: group:W2K3I-UNO\marketing allow file_gen_read,file_gen_execute
 2: everyone allow std_read_dac,std_synchronize,file_read_attr
```

The standard access right to change the DACL is also given to an owner when a file from a UNIX client with read and write access is stored over NFS—but in such cases, the access right is part of the file's synthetic ACL that approximates the mode bits. When you run the ls -le command on OneFS for a file stored by a UNIX system, OneFS marks the file as having a synthetic ACL:

```
ls -le anewfile.txt
-rw-r--r-- 1 root nobody 0 Apr 26 16:32 anewfile.txt
 OWNER: user:root
 GROUP: group:nobody
 SYNTHETIC ACL
 0: user:root allow file_gen_read,file_gen_write,std_write_dac
```

```
 1: group:nobody allow file_gen_read
 2: everyone allow file_gen_read
```

You can also use the ls -len command to view the owner and group's SID:

```
-rwxr-x--- + 1 1000000 1000001 2056 Feb 2 10:18 adocs.txt
 OWNER: SID:S-1-5-21-3542649673-1571749849-686233814-1117
 GROUP: SID:S-1-5-21-3542649673-1571749849-686233814-1109
 0: SID:S-1-5-21-3542649673-1571749849-686233814-1117 allow
 file_gen_all
 1: SID:S-1-5-21-3542649673-1571749849-686233814-1109 allow
 file_gen_read,file_gen_execute,std_write_dac,file_write_attr
 2: everyone allow std_read_dac,std_synchronize,file_read_attr
```

The next section discusses how you can manage the outcomes of chmod, chown, and chgrp operations by using the ACL policies of OneFS—settings that enable you to alter the default patch-and-merge behavior on a file with an ACL to instead prohibit a chmod operation, replace the ACL with mode bits, or override the ACL with a new ACL derived entirely from mode bits.

3. ACL policies for mixed environments

The Isilon cluster includes ACL policies that control how permissions are processed and managed. By default, the cluster is set to merge the new permissions from a chmod command with the file's ACL, as several previous examples have shown. Merging permissions is a powerful method of preserving intended security settings while meeting the expectations of users. In addition, managing permissions policies manually gives you a range of options to respond to the kind of special cases that can surface in mixed environments.

An alternative is to set the Isilon cluster's global permissions policy to balanced mode—a mode designed to automate file sharing management for a network that mixes UNIX and Windows systems. In contrast to manually configuring ACL policies, balanced mode forces you to use a predetermined set of policies. If any of the balanced-mode policies are unsuitable for your mixed environment, it is recommended that you configure the policies manually.

When you configure policies manually, there are two categories to choose from: permission policies and advanced settings. The sections that follow look at the application of some of these policies in the context of a multiprotocol file server. For instructions on how to apply them, see the OneFS User Guide.

3.1. Permissions policies

This section examines the basic permissions policies. The examples assume that you have selected the policy that allows the Isilon cluster to create ACLs over SMB. In addition, when you manually configure permission policies, some policies apply only in the context of another setting. For details, see the OneFS User Guide.

3.1.1. Running chmod on a file with an ACL

There are five policy options to configure how OneFS processes a chmod command run on a file with an ACL. As discussed in a previous section, the option to merge the new permissions with the ACL is the recommended approach because it best balances the preservation of security with the expectations of users. Table 11 explains each of the four other options.

Table 11. Options for Configuring how OneFS Processes chmod

Setting	Discussion
Remove the existing ACL and set UNIX permissions instead	This option can cause information from ACLs, such as the right to write a DACL to a file, to be lost, resulting in a behavior that gives precedence to the last person who changed the mode of a file. As a result, the expectations of other users might go unfulfilled. Moreover, in an environment governed by compliance regulations, you could forfeit the rich information in the ACL, such as access control entries for allowing or denying access to specific groups, resulting in settings that might violate your compliance thresholds.
Remove the existing ACL and create an ACL equivalent to the UNIX permissions	This option can have the same effect as removing the ACL and setting UNIX permissions instead: Important security information that is stored in the original ACL can be lost, potentially leading to security or compliance violations.
Remove the existing ACL and create an ACL equivalent to the UNIX permissions for all the users and 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.
Deny permission to modify the ACL	This option can result in unexpected behavior for users who are owners of the file and expect to be able to change its permissions.

3.1.2. The inheritance of ACLs created on directories by chmod

The inheritance of ACLs that are created on directories by running the chmod command presents another point of contention. The two security models are at odds again. On Windows, the access control entries for directories can define fine-grained rules for inheritance; on UNIX, the mode bits are not inherited. The policy that makes directories not inheritable may be the more secure setting for tightly controlled environments. Such a setting, however, may deny access to some Windows users who would otherwise expect access. For a list of the types of inheritance of permissions in an ACE, see Section 2.6 on the anatomy of an ACL.

3.1.3. Chown on files with ACLs

The result of changing the ownership of a file is different over SMB and NFS. Over NFS, the chown command changes the permissions and the owner or owning group.

Consider a file owned by a user with rwx------ (700) permissions, signifying rwx permissions for the owner, but no permissions for anyone else. If you run the chown command to change ownership of the file to another user, the owner's permissions are still rwx, but they now represent the permissions of the new user. The previous owner will have lost all of his or her permissions.

On Windows systems and over SMB, changing file ownership of leaves ACEs in the ACL unmodified. The ACEs in the ACL associated with the old and new owner or group remain the same.

The conflict between the two models becomes crucial when you run a chown command over NFS on a file stored over SMB. The file has an ACL, which can include ACEs that explicitly allow or deny access to certain users and groups. The chown command could wipe out the ACEs.

Because the security models differ at such a crucial point, OneFS lets you choose the approach to changing file ownership that works best for you. For a mixed environment with multiprotocol file sharing, leaving the ACL unmodified is the recommended approach because it preserves the ACEs that explicitly allow or deny access to specific users and groups. Otherwise, a user or group who was explicitly denied access to a file or directory might be able to gain access, possibly leading to security or compliance violations.

This setting does not affect chown commands performed over NFS on files with UNIX permissions, and it does not affect ownership changes to Windows files over SMB.

3.1.4. Access checks (chmod, chown)

With the UNIX security model, only the file owner or the superuser has the right to change the mode or the owner of a file—rights to which UNIX users are accustomed.

With the Windows model, users other than the file owner can also have the change-permissions right (WRITE_DAC) and the take-ownership right (WRITE_OWNER)—rights that they expect to be able to use. In addition, in Windows Server 2008 and Windows Vista, you can curtail the default permissions (WRITE_DAC and READ_CONTROL) that an owner of an object automatically receives by adding the OwnerRights security principal to an object's ACL. With version 6.5.4 or later, OneFS works with the OwnerRights SID to curtail the powerful WRITE_DAC right. According to MSDN, using the OwnerRights SID helps ensure that users who create files and folders cannot change the intended access control policy.

The asymmetry between the two models that drives the access check policy is as follows:

POSIX file systems

- chmod: only owner and root
- chown: only root
- chgrp: only owner and root, and only to groups they are in

NTFS

- Change ACL: requires WRITE_DAC (or owner)
- Change owner: requires WRITE_OWNER, which can be taken but not given away
- Change group: requires WRITE_OWNER, which can be given to any group

If you configure permissions settings manually, the access check policy enables you to apply the approach most likely to meet the expectations of users in your environment.

3.2. Advanced settings

This section examines the advanced ACL settings. For instructions on how to set these options, see the OneFS User Guide.

3.2.1. Treatment of rwx permissions on chmod

When you chmod a file that was stored over NFS with its mode bit set to rwx for owner, group, or other, a full control ACE with all access rights is granted by default to the owner, group, or everyone else. The access control entry includes the following effective permissions for SMB clients: change permissions, take ownership, delete, and synchronize. On Windows systems, these effective permissions correspond to the following standard access rights: WRITE_DAC, WRITE_OWNER, DELETE, and SYNCHRONIZE.

In the following file, which was stored by a Windows client over SMB, the rwx mode bit for the owner maps to full control, represented by OneFS as the file_gen_all access right, which includes the following rights:

file_gen_read, file_gen_write, file_gen_execute, delete_child, and std_write_owner

The owner retains delete_child as well as std_write_owner—the standard access right to modify the security descriptor. (The access right for synchronizing is included as part of file_gen_execute.)

The following example illustrates how a file retains full control when the advanced ACL policy for rwx permissions is set to treat rwx as full control:

```
-rwxr--r-- 1 1000000 1000001 0 Apr 26 15:56 newfile.txt
 OWNER: SID:S-1-5-21-3542649673-1571749849-686233814-1117
 GROUP: SID:S-1-5-21-3542649673-1571749849-686233814-1109
 SYNTHETIC ACL
 0: SID:S-1-5-21-3542649673-1571749849-686233814-1117 allow
 file_gen_all
 1: SID:S-1-5-21-3542649673-1571749849-686233814-1109 allow
 file_gen_read
 2: everyone allow file_gen_read
```

After changing the policy to retain rwx permissions and changing the file mode so the new setting takes effect, the file drops the access right for full control—file_gen_all—and replaces it with a less permissive set:

```
chmod 777 newfile.txt

ls -le newfile.txt
-rwxrwxrwx 1 1000000 1000001 0 Apr 26 15:56 newfile.txt
```

```
OWNER: SID:S-1-5-21-3542649673-1571749849-686233814-1117
GROUP: SID:S-1-5-21-3542649673-1571749849-686233814-1109
SYNTHETIC ACL
0: SID:S-1-5-21-3542649673-1571749849-686233814-1117 allow
file_gen_read,file_gen_write,file_gen_execute,std_write_dac
1: SID:S-1-5-21-3542649673-1571749849-686233814-1109 allow
file_gen_read,file_gen_write,file_gen_execute
2: everyone allow file_gen_read,file_gen_write,file_gen_execute
```

3.2.2. Group owner inheritance from a parent folder

Operating systems tend to work with group ownership and permissions in two different ways: BSD inherits the group owner from the file's parent folder; Windows and other Linux systems inherit the group owner from the file creator's primary group. By default, OneFS uses the Linux and Windows method when a file has an ACL; otherwise, because the OneFS operating system is based on FreeBSD, it uses the BSD method.

If you are expecting to see inheritance based on one of these methods, you might get unexpected results. You can select the method that you want the cluster to use, or you can select to always use the Windows and Linux method regardless of whether a file has an ACL. If you enable a setting that causes the group owner to be inherited from the creator's primary group, it can be overridden on a per-folder basis by running the chmod command to set the set-group-ID-on-execution bit. See the OneFS man page for the chmod command.

In an environment governed by compliance regulations, inheriting the group owner from the parent folder can have unintended consequences. Membership in the group that owns the parent folder might give a user access to a file that should otherwise be accessed only by the group that owns it.

3.2.3. Running chmod 007 on files with ACLs

After you mount OneFS on a UNIX or Linux system over NFSv2 and NFSv3, you can view only a file's POSIX mode bits. The ls -le command is unavailable on the client. You cannot see a file's ACL with the ls -l command unless you are using NFSv4. But there are cases in which an ACL that you cannot view can deny an operation that you would otherwise expect to succeed. There are also cases in which you might need to remove an ACL from a file that was stored over SMB.

Running chmod 007 on a file is a method that Isilon developed to let you remove the file's ACL and then reset the permissions with POSIX mode bits. The following example assumes that the advanced ACL setting for chmod 007 is set to the option that removes the ACL.

The following file, viewed by connecting to the cluster with ssh and running the ls -le command, has these ACL permissions:

```
-rwxrw-r-- + 1 W2K3I-UNO\jsmith W2K3I-UNO\market 0 Apr 26 15:56
 newfile.txt
 OWNER: user:W2K3I-UNO\jsmith
 GROUP: group:W2K3I-UNO\marketing
 0: user:W2K3I-UNO\jsmith allow file_gen_all
 1: group:W2K3I-UNO\marketing allow file_gen_read
 2: user:root allow file_gen_read,file_gen_write,std_write_dac
```

```
 3: group:nobody allow file_gen_read
 4: everyone allow file_gen_read
```

When you view the same file from a Linux computer by running the ls -l command, however, it looks like this:

```
-rwxrw-r-- 1 1000000 1000001 0 Apr 26 15:56 newfile.txt
```

If you were to change the mode bits on the file by running the chmod command as root, the mode bits would be changed, but the new permissions would, by default, be merged with the ACL. As a result, several of the ACLs would continue to govern the file, and the mode bits would continue to be an approximation of the ACL.

Although the cluster is configured to merge permissions, you can run the chmod 007 command to remove the ACLs entirely, and then you can run the command again to set the actual UNIX permissions that you want:

```
chmod 007 newfile.txt
chmod 744 newfile.txt
```

On the cluster, the file, viewed with the ls -le command, now looks like this:

```
-rwxr--r-- 1 W2K3I-UNO\jsmith W2K3I-UNO\market 0 Apr 26 15:56
 newfile.txt
 OWNER: user:W2K3I-UNO\jsmith
 GROUP: group:W2K3I-UNO\marketing
 SYNTHETIC ACL
 0: user:W2K3I-UNO\jsmith allow file_gen_all
 1: group:W2K3I-UNO\marketing allow file_gen_read
 2: everyone allow file_gen_read
```

The file's original ACL has been replaced by a synthetic ACL generated from the new UNIX permissions. The POSIX mode bits represent the actual permissions; the synthetic ACL is a virtual approximation of them.

On the Linux client, it looks like this:

```
-rwxr--r-- 1 1000000 1000001 0 Apr 26 15:56 newfile.txt
```

The permissions contained in the mode bits seen from the Linux client are now the actual permissions, not an approximation. On the cluster, the actual permissions are also encoded as POSIX mode bits.

On OneFS, you can also remove the ACL and replace it with the mode bits that you want by running the chmod command with the b option. See the OneFS man page for chmod.

3.2.4. Owner and group permissions

The advanced setting for owner and group permissions affects only how permissions are displayed in mode bits. For both the owner and group permissions, the option that approximates mode bits by using all possible owner or group ACEs displays a more permissive set of permissions than the actual mode bits.

If, for example, you are an owner who has only read access to a file but might be in a group with write and execute access to it, the owner's permissions are displayed as rwx on a UNIX client as well as on OneFS. Checking all the group access control entries before displaying the permissions is, in terms of computing resources, too expensive.

Viewed from an export mounted on a UNIX client, the mode bits of the following file show rwx for group:

```
[root@rhel5d adir]# ls -l print.css
-rwxrwx--- 1 1000000 1000001 807 Apr 2 2010 print.css
```

Although the mode bits are also set to rwx for group on OneFS, running the ls -le command reveals that the group does not explicitly have permission to write to the file:

```
ls -le print.css
-rwxrwx--- + 1 W2K3I-UNO\jsmith W2K3I-UNO\market 807 Apr 2 2010
 print.css
 OWNER: user:W2K3I-UNO\jsmith
 GROUP: group:W2K3I-UNO\marketing
 0: user:W2K3I-UNO\jsmith allow file_gen_all
 1: group:W2K3I-UNO\marketing allow file_gen_read,file_gen_execute
 2: user:root allow file_gen_all
 3: user:nobody allow file_gen_all
 4: everyone allow std_read_dac,std_synchronize,file_read_attr
```

OneFS displays rwx for group because either the root user or the nobody user might be a member of the marketing group.

The permissiveness compensates for applications and NFS clients that incorrectly inspect the UNIX permissions when determining whether to attempt a file system operation. By default, this setting ensures that such applications and clients proceed with the operation to allow the file system to properly determine user access through the ACL.

In an environment where there are no applications or clients that incorrectly inspect UNIX permissions, and most users are accustomed to seeing a more literal representation of their owner or group permissions, you can set the ACL policy to approximate the owner or group mode bits by using only the ACE with the owner or group identity. Keep in mind, though, that it might result in access-denied problems for UNIX clients. In addition, although the mode bits will correctly represent the permissions in nearly every case, they might occasionally make the file appear to be more secure than it really is, a situation that, according to RFC 3530, should be avoided. It is therefore recommended that you use the default setting for mixed environments.

3.2.5. No deny ACEs

For files with mode bits, OneFS generates a synthetic ACL to control the access of Windows users. A synthetic ACL can contain both allow and deny entries, which either allow or deny access to specific users and groups.

By default, OneFS removes the deny entries from synthetic ACLs. Although their removal can lead to representations that are less restrictive than the corresponding POSIX mode bits, there are only two fringe cases in which less restrictive representations occur:

- When a group is explicitly allowed access but a user who is a member of the group is explicitly denied access.
- When everyone is allowed access but a group or a user is explicitly denied access.

Here is an example. Suppose that a file stored by a UNIX system has the following mode bits and access control entries:

```
-r--rw-rwx
 SYNTHETIC ACL
 0: user:W2K3I-UNO\jsmith allow file_gen_read
 1: user:W2K3I-UNO\jsmith deny file_gen_write, file_gen_execute
 2: group:W2K3I-UNO\marketing allow file_gen_read, file_gen_write
 3: group:W2K3I-UNO\marketing deny file_gen_execute
 4: everyone allow file_gen_read, file_gen_write, file_gen_execute
```

In the example, the user jsmith is specifically denied write and execute access. If, however, jsmith is in the marketing group, jsmith will obtain write access to the document—an unexpected, but unlikely, consequence of the default setting.

Such a consequence may be unacceptable in an environment governed by tight security policies or compliance regulations. In such cases, you can set OneFS to preserve the deny ACEs. But doing so can have detrimental side effects. Windows systems require that deny ACEs come before allow ACEs. When a Windows user retrieves a file with a synthetic ACL from the cluster and modifies its ACL, the Windows ACL user interface rearranges the ACEs into the Windows canonical ACL order—all the deny entries are put before the allow entries. When the user returns the file to the Isilon cluster, the new ACE ordering makes the file more restrictive than it was intended to be.

Consider the following example:

```
-rwxrw-rwx
 SYNTHETIC ACL
 0: user:0 allow file_gen_all
 1: group:0 allow file_gen_read,file_gen_write
 2: group:0 deny execute
 3: everyone allow file_gen_all
```

When the Windows ACL user interface sets the ACEs in canonical order, it puts the deny entries before the allow entries—a change that in effect denies execute access to the user and to everyone, even though both were explicitly allowed file_gen_all, which includes the execute access right.

3.2.6. Utimes for access and modification of files

The advanced setting for utimes configures OneFS to let either owners or owners and users with write access change utimes to client-specific times. Utime is the access and modification times of a file. Allowing only owners to change utimes complies with the POSIX standard—an approach that is probably familiar to administrators of UNIX

systems. Allowing owners as well as users with write access to modify utimes is a less restrictive approach, one that is probably familiar to Windows administrators.

4. Conclusion

OneFS delivers multiprotocol data access with a unified security model that preserves, with ease and automation, a file's permissions across UNIX and Windows systems and the NFS and SMB protocols. OneFS unites the disparate security models of Windows and UNIX systems to meet the expectations of users and to maintain the security of directories and files, even when commands such as chmod and chown are run from a UNIX client on a file from a Windows system.

The unified security model of OneFS includes the following features:

- An on-disk identity that transparently maps accounts across multiple directory services
- An on-disk authoritative representation of a file's permissions that retains the file's original security settings and can be viewed directly on the cluster
- Protocol-specific views of permissions that enable NFS exports to display POSIX mode bits and SMB shares to show ACLs
- An internal representation of the permissions of a file system object, such as a directory or a file, that mediates between the access rights of the Windows security model and the mode bits of the POSIX security model to enforce the same permissions for all protocols
- A synthetic ACL that approximates the mode bits of a UNIX file for an SMB client
- A policy engine that transforms the authoritative representation of an object's permissions into a form that satisfies the requirements of the requesting protocol
- Standard UNIX commands such as chmod and chown to conveniently and consistently manage not only mode bits but also ACLs
- A default policy that, when a chmod command is run on a file with an ACL, optimally merges the new permissions with the old
- Policy settings that optionally enable you to customize how OneFS manages permissions for different environments

By combining these features into an automated solution with a single point of administration, OneFS uniquely balances interoperability, transparency, user expectations, administrative control, and file security.