Dell PowerScale OneFS: Authentication, Identity Management, and Authorization

January 2023

H13115.5

White Paper

Abstract

This white paper provides design considerations for configuring and troubleshooting user access and file management with the PowerScale scale-out NAS platform. It details the PowerScale OneFS Unified Permission Model and Authentication, Identity Management, and Authorization (AIMA) stack.

Dell Technologies

Copyright

Published in the USA January 2023 H13115.5.

Contents

Executive summary

Overview

This document provides design considerations for configuring and troubleshooting user access and file management with the Dell PowerScale scale-out NAS platform. Support for multiple protocols requires a model for ensuring that users are provided equal rights irrespective of the access protocol and authentication providers. Additionally, the model must define file permission management. To provide consistent, flexible, and secure access across supported protocols, PowerScale OneFS uses a Unified Permission Model combined with an Authentication, Identity Management, and Authorization (AIMA) stack. This document dissects and explains the Unified Permission Model and AIMA stack.

Traditionally, legacy NAS systems only provided support for a single protocol. However, PowerScale supports several protocols, introducing the challenges of multi-protocol support. While many vendors provide multi-protocol support on a single platform, each vendor implements a proprietary model to provide user access and file management in a multi-protocol environment. Given that multi-protocol support is not governed by an RFC or an open-source model, each vendor provides a different approach and implementation. The goal of this paper is to provide an understanding of PowerScale implementation of multi-protocol support, which is different from other vendors but is simple to apply once it is understood.

Note to readers

Before making changes on a production cluster, exercise extreme caution. Understand the concepts explained in this paper in their entirety before implementing significant file and permission updates. As with any major infrastructure update, testing changes in a lab environment is best practice. Once updates are confirmed in a lab environment, commence with a gradual roll-out to a production cluster.

Revisions

Date	Description
August 2016	Initial release
November 2018	Renamed from “PowerScale OneFS Multi-Protocol Security” to “PowerScale OneFS AIMA”; document completely updated with new content
March 2019	Updated to introduce new features in OneFS 8.2
June 2020	PowerScale rebranding
November 2021	Updated template
January 2023	Updated to introduce the SAML-based SSO for the WebUI; editorial updates

We value your feedback

Dell Technologies and the authors of this document welcome your feedback on this document. Contact the Dell Technologies team by email.

Authors: Aqib Kazi, Lieven Lin

Note: For links to other documentation for this topic, see the PowerScale Info Hub.

Legacy single-protocol environments

Introduction

To fully comprehend the implementation of a multi-protocol model, it is crucial to understand how user access and file permissions are handled in a legacy single-protocol environment.

Conventionally, legacy single-protocol environments supported either a Microsoft Windows or Linux architecture. In these environments, a clear separation of protocol and authentication existed. Microsoft users authenticated with Active Directory, while Linux users authenticated with LDAP. Microsoft users accessed files through CIFS or SMB and Linux users through NFS. In a single-protocol environment, cross-platform access was not an option because Microsoft users would not access files created through NFS and vice versa, as illustrated in the following figure:

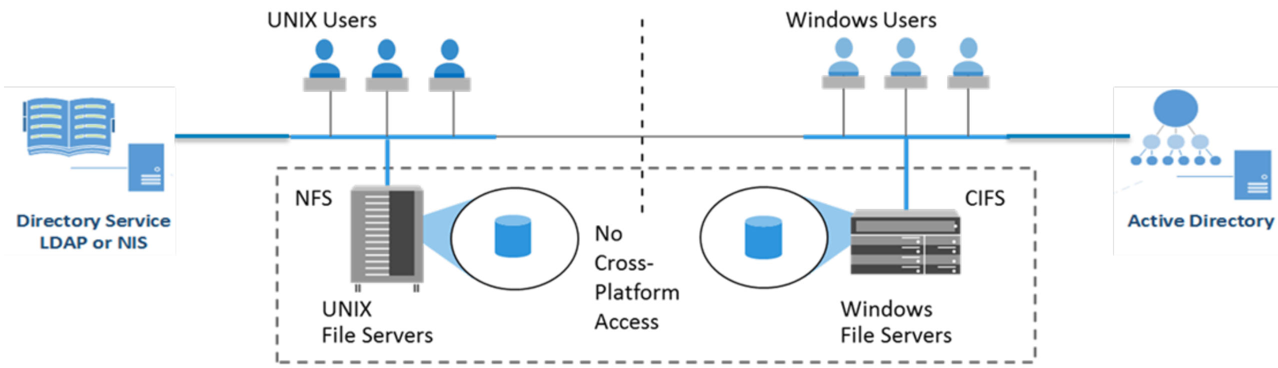


Figure 1. Legacy single-protocol environments

UNIX Identifier

In a single-protocol environment, the UNIX Identifier (UID) identifies a user with a positive integer assigned by a Lightweight Directory Access Protocol (LDAP) server. The UID maps to several Group Identifiers (GID) to determine access permissions. At login, the user ID is mapped to the matching UID and GID. The UID and GID for a user are displayed with an LDAP query in the following figure:

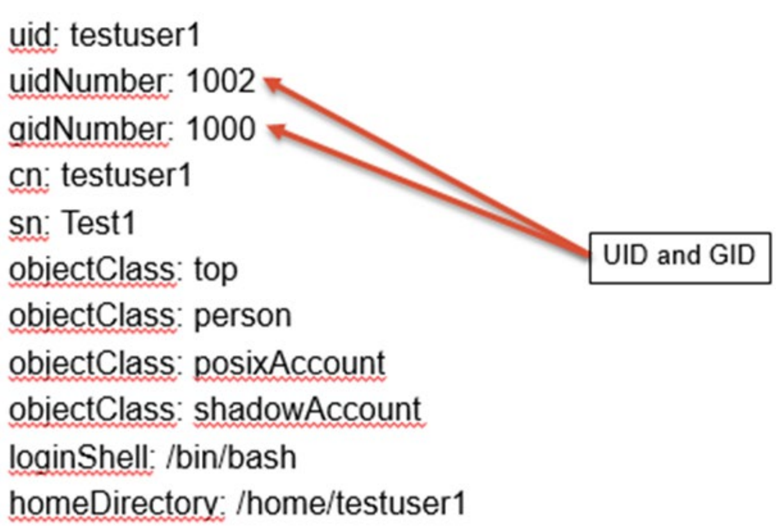


Figure 2. UNIX Identifier UID and GID

Microsoft Security Identifier (SID)

In a Microsoft environment, the Security Identifier (SID) is a unique value assigned to a user, group, and accounts. The SID is issued by a security authority, which in most cases is the domain controller, and is pulled from the Active Directory database. At login, an access token is generated. It is composed of the SID with group SIDs and a list of privileges, as illustrated in the following figure:

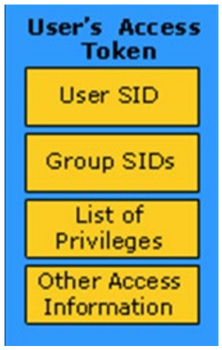


Figure 3. Microsoft user access token

The SID is written in the following format: (SID)-(revision level)-(identifier-authority)-(subauthority1)-(subauthority2)-(etc)

As an example, consider the following SID: S-1-5-21-1004336348-1177238915-682003330-512

- Revision level (1)
- Identifier authority (5, NT Authority)
- Domain identifier (21-1004336348-1177238915-682003330)
- Relative identifier (512, Domain Admins)

For more information about SIDs, see the following Microsoft article: What Are Security Identifiers?

Multiprotocol NAS

In contrast to a single-protocol environment, a multi-protocol system introduces new challenges and requires a planned approach to managing users and file permissions. One of the key facets of PowerScale scale-out NAS is the support of several protocols, leading to the elimination of silos and focusing on a single storage platform.

In a multi-protocol environment, UNIX and Windows users access the same file through the same directory structure, but through different protocols. The challenge is how identities are verified and what file permissions are used for authorization. Previously, each set of users only had a single authentication provider. A multi-protocol infrastructure might be composed of LDAP and Active Directory, connected to a single NAS. Additionally, the authentication provider might not be related to the client operating system. For example, a UNIX user could authenticate with Active Directory. Furthermore, users might have accounts in both Active Directory and LDAP, requiring mapping between those accounts and allowing OneFS to link the accounts.

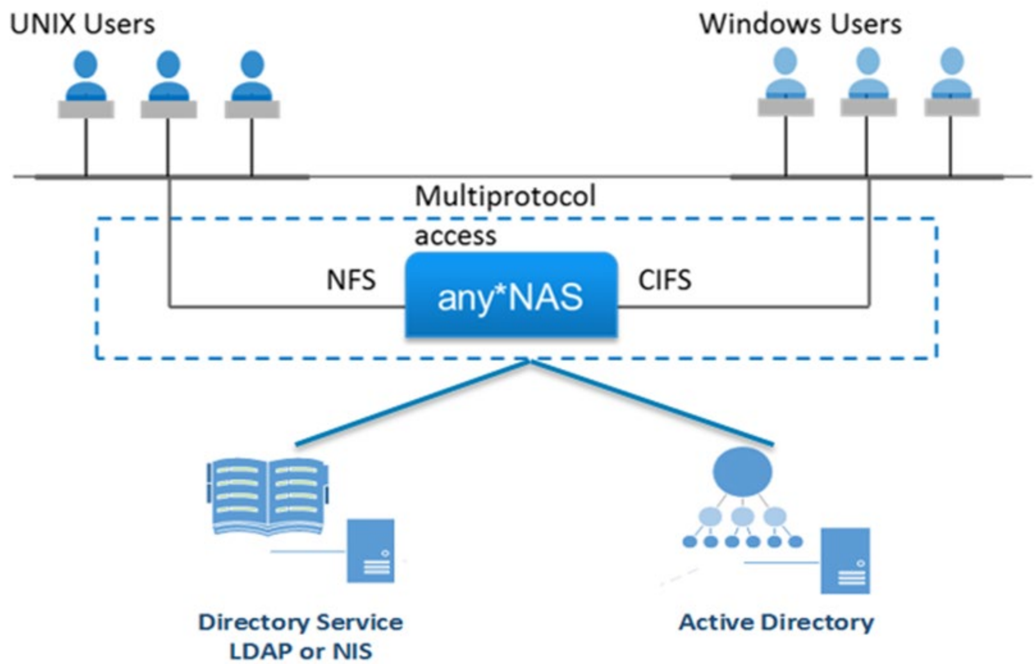


Figure 4. Multi-protocol NAS environment

Although PowerScale OneFS supports many protocols, it may be configured as a single-protocol system. Access zones and directories may be limited to a single protocol, while other areas are defined as multi-protocol.

OneFS Unified Permission Model

Multi-protocol environments introduce new challenges for managing user access and file permissions. Because multi-protocol environments are not governed by an open standard or RFC, each vendor implements multi-protocol with a different approach.

PowerScale OneFS developed the Unified Permission Model to implement multi-protocol support. Using the Unified Permission Model ensures that the permission model remains consistent irrespective of the access protocol. A single model simplifies multi-protocol integration because the access protocol is not considered when comparing users and permissions. From an administrative perspective, only a single model has to be ascertained, rather than several protocol-specific models.

Multi-protocol is not only limited to SMB and NFS. OneFS also supports HTTP, HDFS, S3, and FTP. It is essential to ensure that the permission model remains consistent across all these protocols. Further, the Unified Permission Model accounts for users from different systems with different IDs that may be the same or a different user.

The Unified Permission Model ensures that a common access token is generated for each user at login, representing the user's persona to the cluster. Once the token is generated, it is evaluated against file permissions to check for access.

Figure 5. Unified Permission Model overview

OneFS Authentication, Identity Management, and Authorization

Introduction

At the core of the Unified Permission Model is the OneFS Authentication, Identity Management, and Authorization (AIMA) model. Upon initial client connection to the cluster, information passes through the AIMA stack, which includes Directory Services, User Mapping, ID Mapping, Tokens, OnDisk ID, File Permissions, and ACL Policies, as illustrated in the following figure:

Figure 6. PowerScale OneFS AIMA

All the components of the AIMA stack work together to complete a user's access experience and determine if access is granted to a file. Understanding the AIMA stack provides the basis for the OneFS implementation of multi-protocol. The AIMA stack is the focus of this paper.

Authentication

Authentication refers to confirming an identity. Upon login, a user states an identity, and the authentication process ensures that the user is associated with the presented identity through a password. The authentication process takes place through providers such as Active Directory or MIT KDC. OneFS also offers a local provider, where users are manually added to OneFS but are only available on the local cluster. Another local option is file provider, where a file is uploaded with user information and could also contain UNIX user and group information from other systems. The same file could be uploaded to other clusters, but the file must be manually updated on each cluster.

Identity management

Identity management is the process of associating memberships with a user. Once the identity is confirmed, OneFS identifies the user and checks for the access that the user has. Managing the identity typically takes place through Active Directory or LDAP but could also be through the OneFS local for file providers.

Authorization

Once a user is authenticated and memberships are associated with that user, OneFS can check to determine if the user has access to a specific file, based on the file permissions. At this step, the user is compared to the file permissions.

AIMA access hierarchy

Understanding AIMA requires an understanding of the OneFS network access hierarchy and how the AIMA hierarchy ties into the network hierarchy. The following figure illustrates the PowerScale OneFS network access hierarchy:

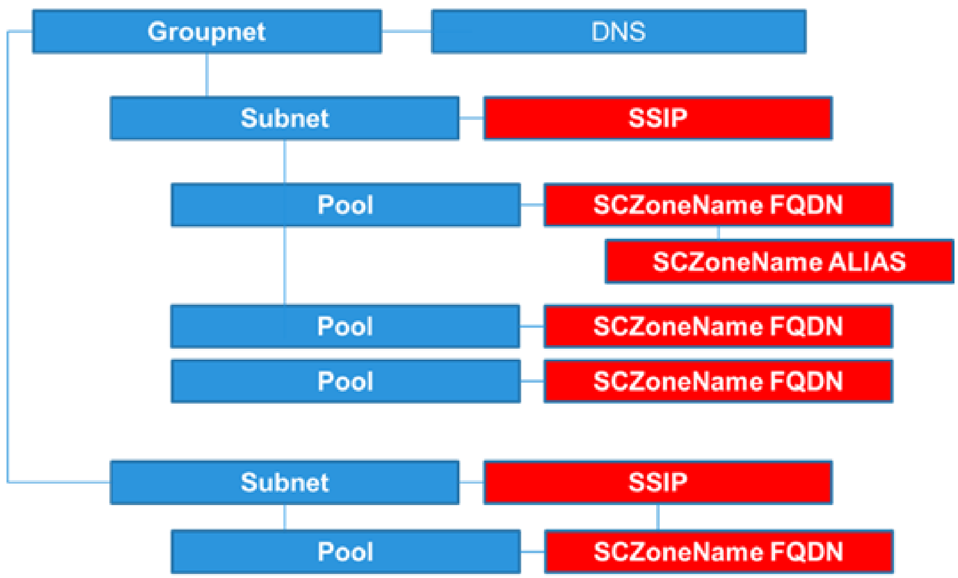


Figure 7. PowerScale OneFS network access hierarchy

As illustrated, each Groupnet has a specific DNS and supports multiple subnets. Each subnet supports a SmartConnect Service IP (SSIP) with multiple pools associated with each subnet and SmartConnect Zone Names. For more information about PowerScale network access hierarchy, see the *PowerScale: Network Design Considerations* white paper.

The AIMA hierarchy ties into the network hierarchy at different levels, as illustrated in the following figure:

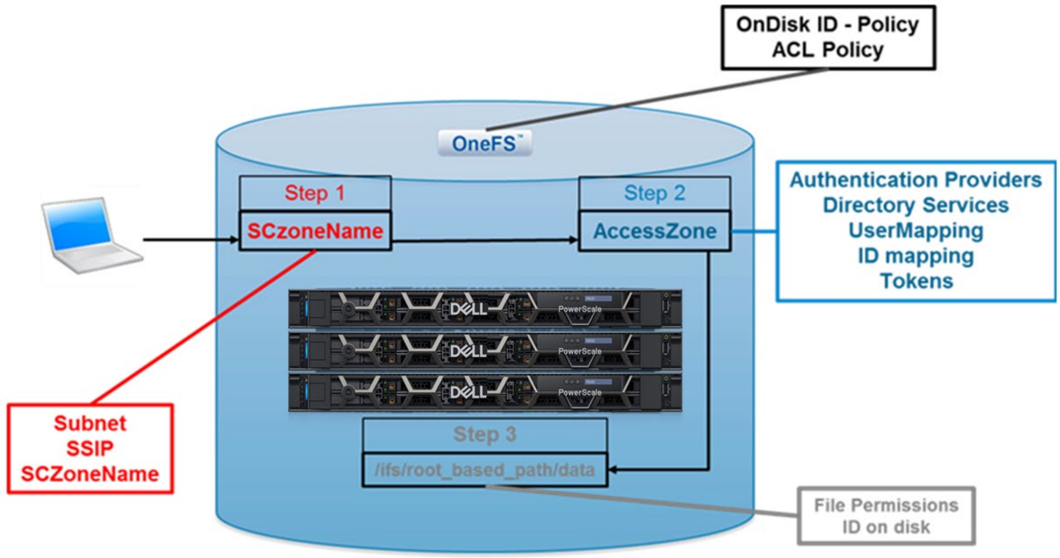


Figure 8. PowerScale OneFS AIMA hierarchy

When a client connects to a PowerScale cluster, AIMA plays a role at each level of the network access hierarchy. Recognizing the level at which each component resides is critical. The AIMA access hierarchy is as follows.

Note: Upon initial review, the terms in the figure and in the following description might seem confusing but will become more understandable as you proceed. We recommend that you review this section as each topic is explained and then again after you have reviewed this paper in its entirety.

---

1. The user connects to a SmartConnect Zone Name, which is tied to a subnet, and SSIP.
2. The SmartConnect Zone Name is mapped to an Access Zone. At the Access Zone level, authentication providers are defined. As the authentication providers are defined, Directory Services, User Mapping, ID mapping, and, ultimately, the user Token are generated.
3. For each Access Zone that is defined, a root-based path is required. The root-based path is where file permissions and the user's identity on disk are applicable.

At the overall cluster level, administrators define the OnDisk ID policy and set ACL policies.

Access zones and root-based paths

When Access Zones are configured, a root-based path must be defined to segment data into the appropriate Access Zone and enable the data to be compartmentalized. Access Zones carve out access to a PowerScale cluster creating boundaries for multi-tenancy or multi-protocol. They permit or deny access to areas of the cluster. At the Access Zone level, authentication providers are also provisioned.

For more information about Access Zone best practices, see the *PowerScale: Network Design Considerations* white paper.

PowerScale OneFS tokens and file permissions infographic

The following figure provides a high-level overview of tokens and file permissions. Review this figure when it is referenced elsewhere in this paper and as your knowledge of AIMA increases.

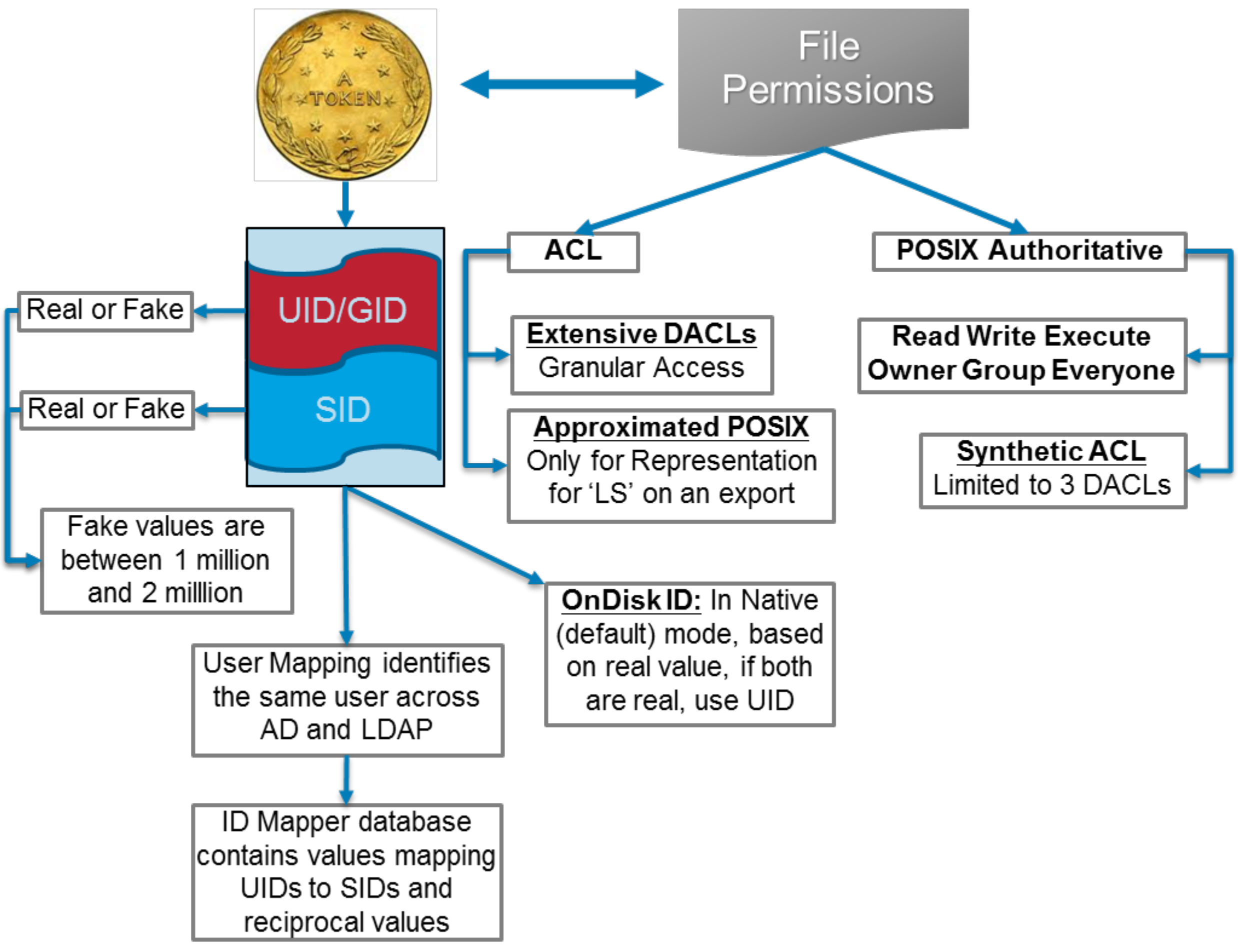


Figure 9. PowerScale OneFS token and file permissions

PowerScale OneFS tokens

Introduction

OneFS generates a token for each user upon initial connection to the cluster. In the Unified Permission Model, a token can be thought of as an identification system like a passport, confirming a user's identity while a visa for each country grants specific access levels.

The token is generated based on the information provided by authentication providers. If an authentication provider is not configured, OneFS locally generates a value, which is referred to as a "fake" value throughout this paper.

The PowerScale OneFS Unified Permission Model has a core requirement that every entity, user, or group has a UNIX component and a SID component. A token is composed of two parts, a UID with associated GIDs and a SID (introduced in Legacy single-protocol environments). Similar to how authentication occurs in a single-protocol environment, in the PowerScale AIMA model, OneFS reaches out to those same providers, if configured, to collect the UID and SID. However, under the Unified Permission Model, those UID and SID values are now combined into a single token.

Figure 10. PowerScale OneFS token

If authentication providers are not configured or unavailable, the fake UID/GID value is assigned, which, by default, is between 1 and 2 million. The default value is configurable, in case those values create a conflict in an existing environment.

During token generation, User Mapping occurs, which connects identities between LDAP and Active Directory to a single user. Mapping identities ensures that a user's token contains real values for both the UID and SID, as OneFS is aware that it is the same user.

Once a token is generated, the OnDisk ID of a user or group is selected. The OnDisk ID, described further in On-Disk Identity, is used when you create a file, set permissions, or change ownership of a file.

Authentication providers configured in OneFS assist with token generation by responding with values for the UID with associated GIDs and SIDs. The authentication provider's support for identifiers determines what is provided, as summarized in the following table:

Table 1. Authentication providers' supported identifiers

Authentication provider	UID/GIDs	SID	Ranking
Local Provider	Fake	Fake	Poor
File Provider	Fake	Fake	Poor
Active Directory	Fake	Real	Good
LDAP	Real	Fake	Good

Authentication provider	UID/GIDs	SID	Ranking
Active Directory mapping to LDAP	Real	Real	Best
Active Directory with RFC 2307*	Real	Real	Best

*For more information about RFC 2307, see Appendix A: Configuring Active Directory, LDAP, RFC 2307, and Kerberized NFS.

ID mapping database

Once a user has connected to the cluster and the token is generated, the SID and the corresponding UID with GID must be placed in the ID mapping database. Although it is easy to confuse user mapping with ID mapping, user mapping is the process of identifying users across authentication providers to generate a token. After the token is generated, the mappings of SID to UID are placed in the ID mapping database. The ID mapping database also contains a reciprocal entry for each user. Because PowerScale is zone aware, both the UID and SID are required for each user, regardless of whether the values are real or fake, as shown in the following figure:

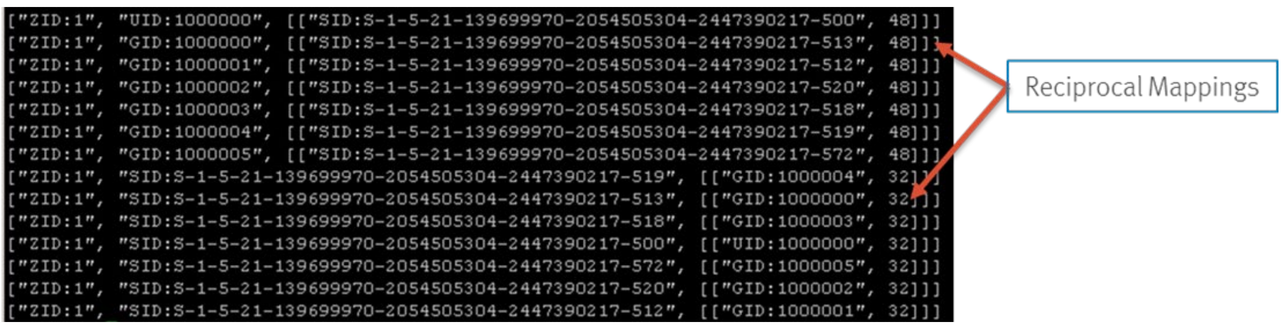


Figure 11. ID mapping database

In the figure, the column on the right delineates which of the values are real or fake. `32` indicates the left is real; right is fake. `48` indicates the right is real, left is fake. `128` and `144` (not shown in the figure) indicate both the left and right values are real.

The command `isi auth mapping list` produces a list of the ID mapping database. The `isi auth mapping` command allows for manually deleting or adding entries. To see a list of the possible commands, enter `isi auth mapping --help`.

ID ranges cannot overlap

In networks with multiple identity sources, ensure that the UID and GID ranges do not overlap. If UIDs and GIDs overlap, certain users could gain access to directories or files that were not intended. Additionally, do not use UIDs and GIDs below 1000. They are reserved for system accounts.

OneFS user-mapping options

OneFS offers many options for configuring user mapping and ensuring that tokens contain the appropriate information reflecting the workflow and environment. For more information about the user-mapping options, see the *PowerScale OneFS User Mapping* white paper.

Checking access tokens

You can check an access token by dumping it on the PowerScale CLI. You can view a token by username, UID, GID, or Kerberos-principal, paired with an access zone. The following figure displays a token by username, with corresponding SIDs, UID, and GIDs:

```
moby1-b-1# isi auth mapping token --user=foo\\administrator
                   User
                       Name: FOO\administrator
                        UID: 1000000
                        SID: S-1-5-21-139699970-2054505304-2447390217-500
                    On Disk: S-1-5-21-139699970-2054505304-2447390217-500
                        ZID: 1
                       Zone: System
             Privileges: -
          Primary Group
                       Name: FOO\domain users
                        GID: 1000000
                        SID: S-1-5-21-139699970-2054505304-2447390217-513
                    On Disk: S-1-5-21-139699970-2054505304-2447390217-513
Supplemental Identities
                       Name: FOO\domain admins
                        GID: 1000002
                        SID: S-1-5-21-139699970-2054505304-2447390217-512

                       Name: FOO\group policy creator owners
                        GID: 1000001
                        SID: S-1-5-21-139699970-2054505304-2447390217-520

                       Name: FOO\schema admins
                        GID: 1000003
                        SID: S-1-5-21-139699970-2054505304-2447390217-518

                       Name: FOO\enterprise admins
                        GID: 1000004
                        SID: S-1-5-21-139699970-2054505304-2447390217-519

                       Name: FOO\denied rodc password replication group
                        GID: 1000005
                        SID: S-1-5-21-139699970-2054505304-2447390217-572

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

                       Name: Administrators
                        GID: 1544
```

Figure 12. Viewing an access token

In the figure, a UID of a 1000000 is displayed, signifying a fake value. The value of 1000000 is assigned to the first user requiring a fake value and increments for each corresponding GID and as more users join. Additionally, a GID is generated for each SID group that was pulled from Active Directory.

Another option to view an access token is `isi auth id`, which lists IDs, privileges, and zone information for the logged in user, as shown in the following example:

```
Ids
------------------------------
ID Type           ID
------------------------------
UID               UID:0
User SID          SID:S-1-22-1-0
GID               GID:0
Group SID         SID:S-1-22-2-0
On Disk User ID   UID:0
On Disk Group ID  GID:0
Additional ID     SID:S-1-5-11
                  GID:5
                  GID:10
                  GID:20
                  GID:70
------------------------------
Total: 7

Privileges
-------------------------------------------------------
ID                      Name                Access
-------------------------------------------------------
ISI_PRIV_LOGIN_CONSOLE  Console             Read only
ISI_PRIV_LOGIN_PAPI     Platform API        Read only
ISI_PRIV_LOGIN_SSH      SSH                 Read only
ISI_PRIV_SYS_SHUTDOWN   Shutdown            Read only
```

Figure 13. Abridged example of isi auth id

Importance of user mappings

If user mappings are not available or are not configured correctly, asymmetrical tokens are created. An asymmetrical token is where a user has access to a file from a certain protocol but is denied access from the other protocol. The inconsistent access is because the user's access token is different when accessing the file from each protocol, as shown in the following example:

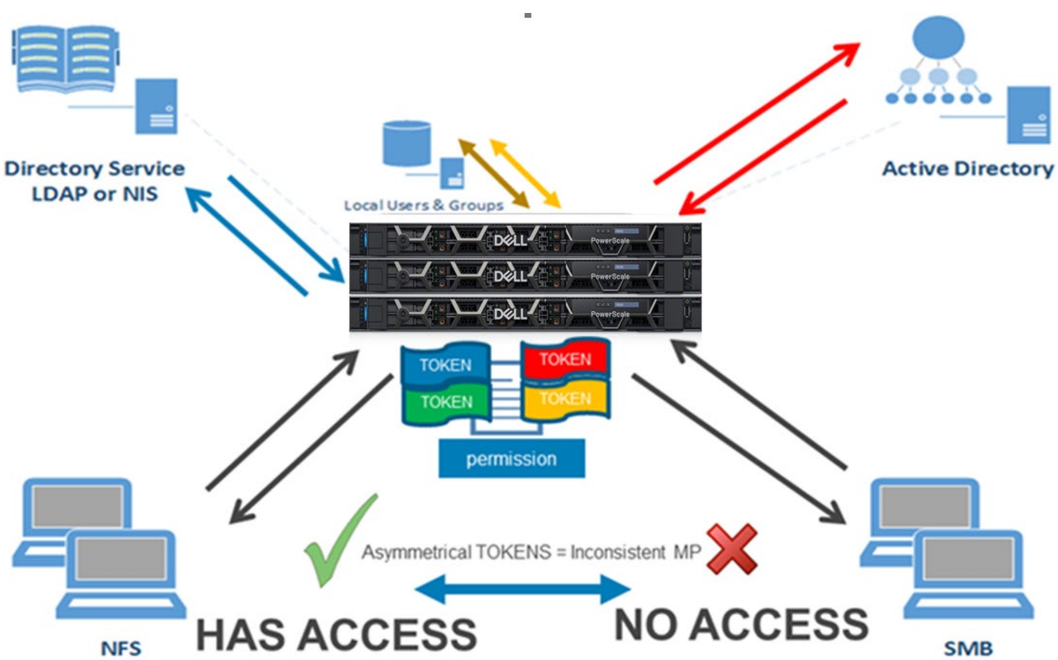


Figure 14. Asymmetrical tokens

Asymmetrical tokens lead to asymmetrical access. In the preceding example, for illustration purposes, the file permission is the “blue permission identity” and requires the blue ID portion of the access token. The blue portion is notating the UID with GIDs. When the same user tries to access the file from NFS and SMB, the following events occur:

- **NFS**: The user logs in to access the file with the blue permission. At login, OneFS reaches out to LDAP. The user is found, and the blue half of the token is generated with a real UID and associated GIDs. Because mapping is not configured, a lookup in Active Directory fails, and a fake green portion is assigned for the SID. However, if the file permission is based on the UID and the user has a real UID in the token, the user is granted access.
- **SMB**: The user logs in to access the file with the blue permission. At login, OneFS reaches out to Active Directory. The user is found, and the red half, or SID, of the token, is generated with a real SID. Because mapping is not configured, a lookup in LDAP fails, and a fake yellow portion is assigned for the UID and corresponding GIDs. Because the file permission is based on the UID and the user has a fake UID in the token, access is denied.

On the contrary, configured user mapping leads to symmetrical tokens, irrespective of the access protocol. A symmetrical token grants equal access to the same user, which is the basis of the Unified Permission Model. OneFS user mapping provides a consistent multi-protocol experience when accessing the same file from NFS or SMB, as illustrated in the following figure:

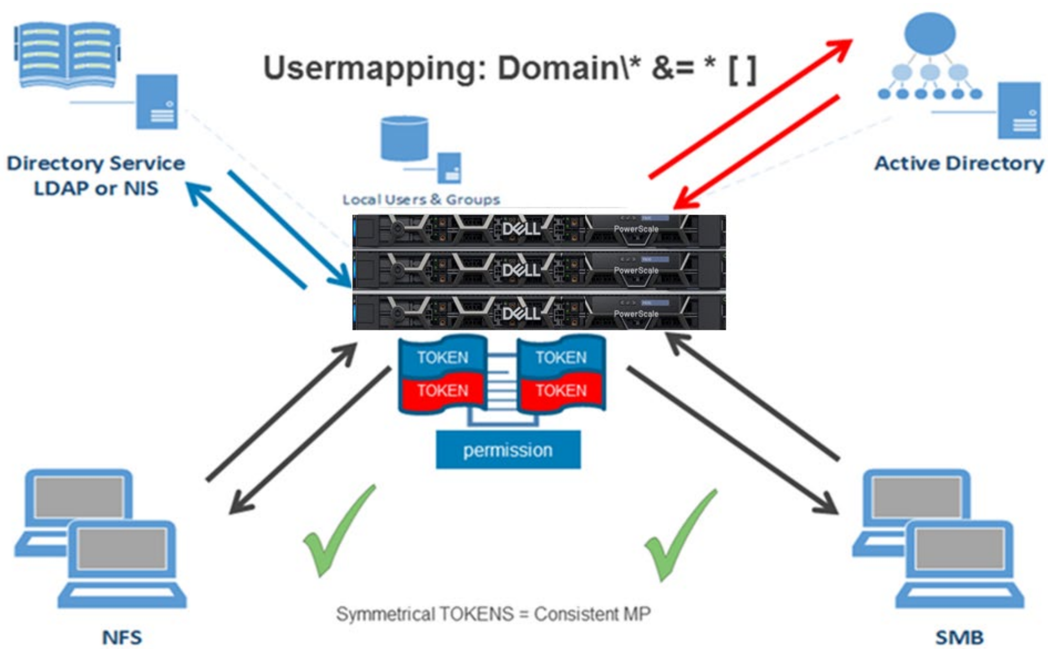


Figure 15. Symmetrical tokens

Symmetrical tokens lead to symmetrical access. In the preceding example, the file permission is again the “blue permission” and requires the “blue ID” portion of the access token. The blue is notating the UID with GIDs. When the same user tries to access the file from NFS and SMB, the following events occur:

- **NFS**: The user logs in to access the file with the blue permission. At login, OneFS reaches out to LDAP. The user is found, and the blue half of the token is generated with a real UID and associated GIDs. Because mapping is configured, a lookup in Active Directory occurs, and a real red portion is assigned for the SID. The file permission is based on the UID, and the user has a real UID and SID in the token. Thus, permission is granted.
- **SMB**: The user logs in to access the file with the blue permission. At login, OneFS reaches out to Active Directory. The user is found, and the red half, or SID, of the token is generated with a real SID. Because mapping is configured, a lookup in LDAP occurs, and a real blue portion is assigned for the UID and corresponding GIDs. Because the file permission is based on the UID and the user has a real UID in the token, access is granted.

On-Disk Identity

Once a token is generated for a user, OneFS uses the token’s contents to assign an On-Disk Identity. The identity is used when the file is created or when file ownership changes, affecting the file permissions. In a single-protocol environment, determining the On-Disk Identity is simple because Windows uses SIDs and Linux uses UIDs. However, in a multi-protocol environment, only one identity is stored, and the challenge is determining which one is stored.

By default, the policy configured for On-Disk Identity is Native mode. Under Native mode, OneFS selects the real value between the SID and UID. If both the SID and UID are real values, OneFS selects UID. The On-Disk Identity policy is configurable from the user interface, as shown in the following figure:

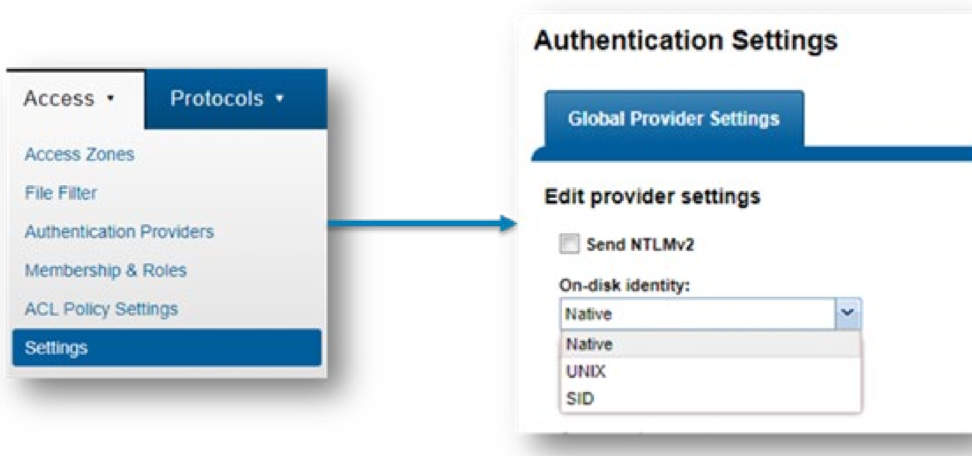
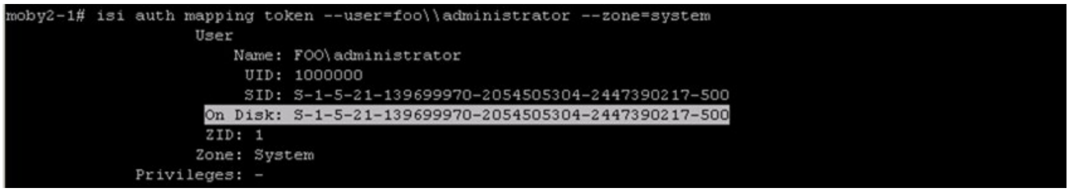


Figure 16. Configuring the On-Disk Identity policy

The On-Disk Identity should typically remain in Native mode, which is the best option for most environments. If a fake, locally generated value is used for a file permission, once that file is sent to another cluster, the original user no longer has access to the file. On the contrary, another user would have access to the file, as the fake tokens are distributed again starting from 1 million. When the real value is used for the On-Disk Identity, file access remains relative to the authentication provider, ensuring that a file is portable and providing a consistent experience.

Reverting to the previous example of viewing an access token, the On-Disk Identity is also visible, as shown in the following figure:



```
moby2-1# isi auth mapping token --user=foo\\administrator --zone=system
                  User
                      Name: FOO\administrator
                       UID: 1000000
                       SID: S-1-5-21-139699970-2054505304-2447390217-500
                   On Disk: S-1-5-21-139699970-2054505304-2447390217-500
                   ZID: 1
                  Zone: System
            Privileges: -
```

Figure 17. On-Disk Identity example

In this figure, the UID is set to 100000, which is a fake, locally generated value. The SID contains a full string and is a real value from Active Directory. Because the policy is configured for Native mode, the real value of the UID and SID is used as the On-Disk Identity, ensuring file portability and consistency as the file moves to another cluster or system.

The following table lists the On-Disk Identity in Native mode for various authentication providers:

Table 2. On-Disk Identity in Native Mode

Authentication provider	SID	UID	On-Disk Identity
Active Directory	Real-AD	Fake	SID
LDAP	Fake	Real-LDAP	UID
Active Directory Mapping LDAP	Real-AD	Real-LDAP	UID
Active Directory with RFC 2307	Real-AD	Real-LDAP	UID

A file's On-Disk Identity is confirmed using the PowerScale CLI commands, `ls -le <filename>` and `ls -len <filename>`. The `ls -le` function lists the usernames, and `ls -len` lists the actual On-Disk UID or SID identities. The following figure shows an example of an On-Disk UID:

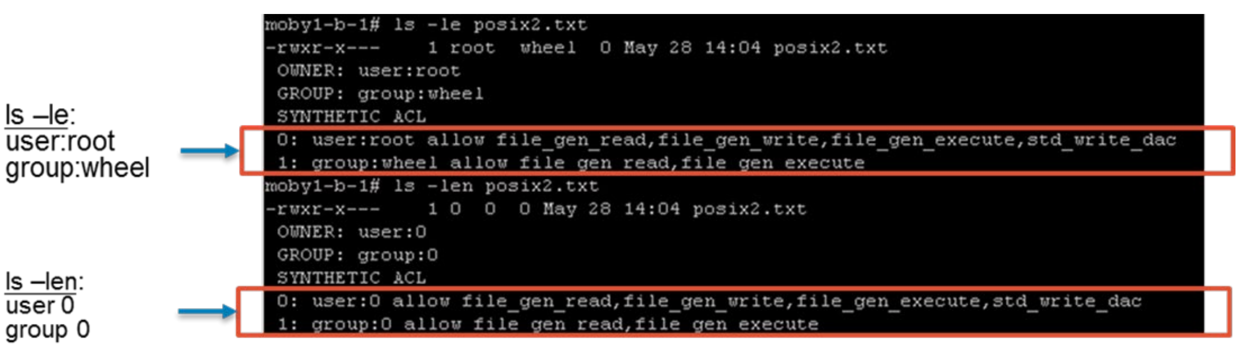


Figure 18. On-Disk UID

The following figures shows an example of an On-Disk SID:

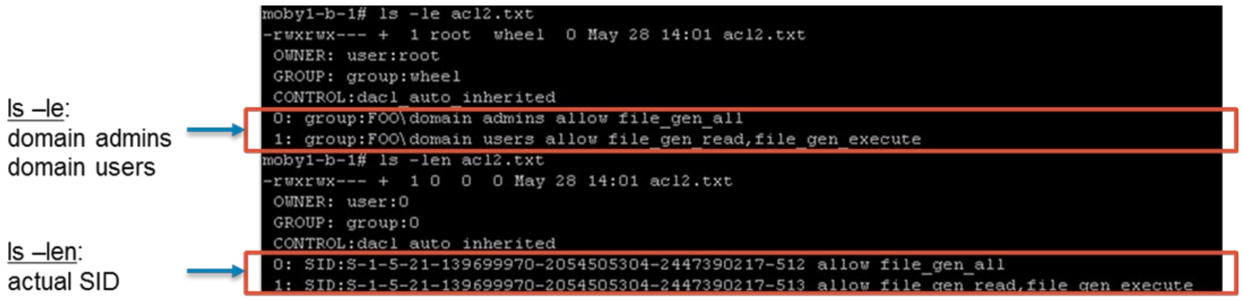


Figure 19. On-Disk SID

OneFS file permissions

Introduction

Once a user's token is generated, the next step is to compare the token to the file permissions. For a summary, see PowerScale OneFS tokens and file permissions infographic. In a multi-protocol environment, the Unified Permission Model is designed to support basic POSIX mode bits and ACLs. Therefore, two file permission states are designated:

- POSIX Mode Bit Authoritative with a Synthetic ACL
- ACL

However, a file can only be in one of these states at a time. The permissions on the file are the same, regardless of the state. The following figure illustrates file permission states and options:

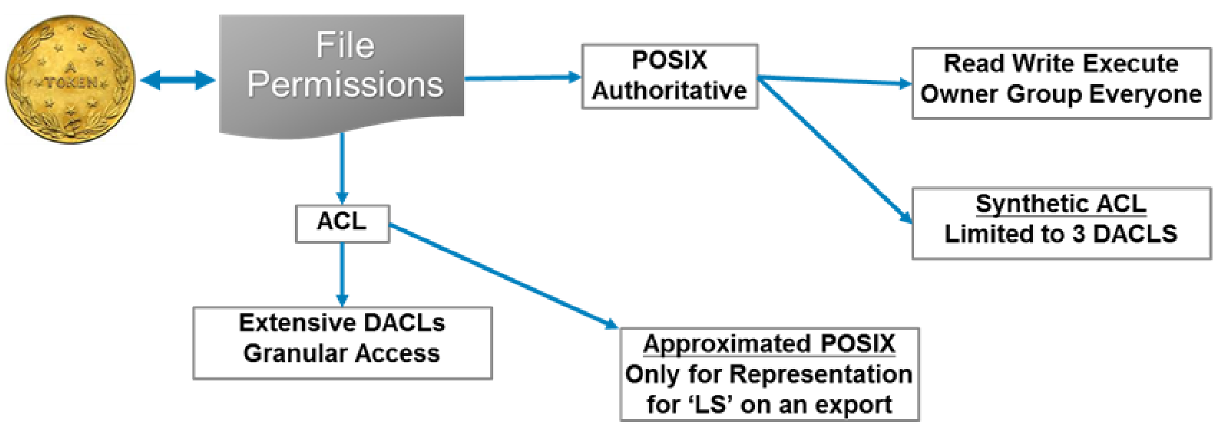


Figure 20. OneFS file permissions

NFS client file access

When a file is requested for access from a client, a sequence of events occurs. The events vary depending on the type of client and the state of file permissions. In the following figure, for example, an NFS client connects to the cluster and accesses a file in each state:

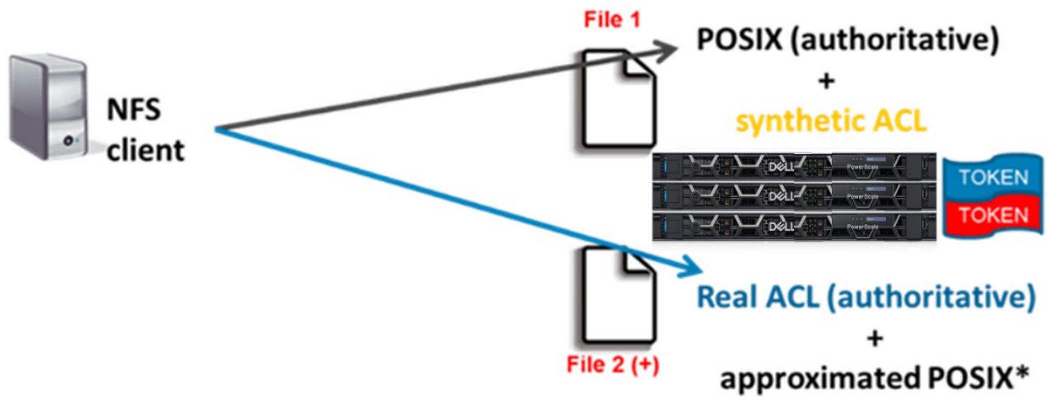


Figure 21. NFS client file access sequence

In the preceding figure, the NFSv3 client connects to the PowerScale cluster, and OneFS issues a token. It is assumed that the token has the correct UID and SID. The NFS client accesses File 1, which is in a POSIX file state. The NFS client understands POSIX bits natively, so the token is compared directly to the POSIX bits to validate access, as in a standard UNIX environment.

Next, the NFS client accesses File 2, which is in a Real ACL file state. The NFS client does not understand ACL. A file with a Real ACL has a richer set of permissions, with several complicated DACLs and subheadings. OneFS must respect the granularity of the ACL file permissions. Thus, the NFS client's token is compared directly to the Real ACL. However, if the NFS client issues an `ls` over an NFS export for File 2, OneFS must return POSIX bits, but these bits are for representation only and are not indicative of the actual file permissions. In fact, the permissions may even look more permissive than they are, because OneFS must approximate representing many ACLs into the six POSIX bits.

SMB client file access sequence

An SMB client connects to a PowerScale cluster and accesses files in both states, as illustrated in the following figure:

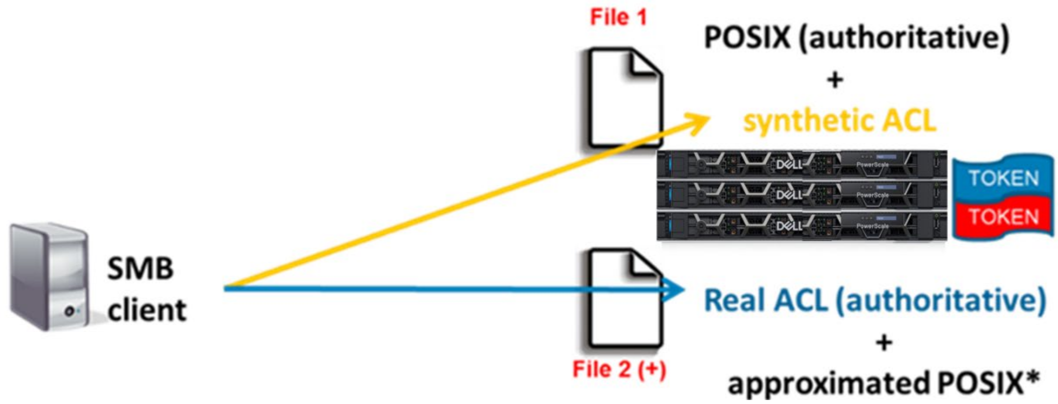


Figure 22. SMB client file access sequence

In the preceding figure, the SMB client connects to the PowerScale cluster, and OneFS issues a token. It is assumed that the token has the correct UID and SID. The SMB client accesses File 1, which is in a POSIX file state. The SMB client does not understand POSIX bits, but the token can still be evaluated against the bits as an access check. To represent the POSIX bits for the SMB client to view File 1 permissions, OneFS generates a synthetic ACL, which is a direct representation of the POSIX bits in ACL form. The synthetic ACL is not saved by OneFS and is only generated when the SMB client accesses the POSIX file.

Next, the SMB client accesses File 2, which is in Real ACL state. The SMB client supports ACL and understands the ACL permissions. In this case, OneFS makes a direct comparison of the SMB client's token with the Real ACL permissions, as would take place in a single-protocol Microsoft environment.

How file state is determined

How a file is created determines the initial state of the file. For example, a file created in an NFS export will likely be POSIX, while a file created in an SMB share will be in ACL. If a file was migrated to PowerScale OneFS, the migration tool affects the file state. Additionally, configured policies also determine the file state.

Initially, the file state is dictated by how it is created or copied, but the directory structure can also affect the file state. Thus, the file state depends on the workflow and directory structure. For example, if the parent directory has inherited ACL(+) and the file is created through NFS, the file will be in ACL state. The same also applies if the parent directory is POSIX mode. If a file is created through SMB, the file state is POSIX mode.

POSIX file state

The POSIX file state is best for environments with heavy read and writes from NFS or HDFS clients. If SMB clients are accessing the file, it is for read-only, and the synthetic ACL is created at access time. In the POSIX file state, the limiting factor is DACLs because only Owner, Group, and Everyone are available. Therefore, the POSIX file state supports a maximum of three DACLs.

ACL file state

The ACL file state offers granular access through extensive DACLs. The granular access might be required for heavy read and write activity from NFS, HDFS, and SMB clients. The token of all clients is compared to the ACL, while considering the many DACLs in the file permission. An approximated POSIX permission is created based on the ACL only for display purposes when an NFS or HDFS client issues an `ls` over an NFS export. The approximated POSIX is representation only and might look more permissive than the ACL actually is.

Viewing file permissions

You can check PowerScale OneFS file permissions from the PowerScale CLI. The following table shows the available extended `ls` commands:

Table 3. OneFS file permission commands

Command	Applicable to filename or directory	Description
ls –le <filename>	Filename	Displays the file permission state, ACLs, owner, and group information
ls –len <filename>	Filename	Displays the file permission state, ACLs, owner, and group information numerically
ls –led <directory>	Directory	Displays the directory permission state, ACLs, owner, and group information
ls –lend <directory>	Directory	Displays the directory permission state, ACLs, owner, and group information numerically

The following figure shows the output of `ls -le` for a POSIX file:

```
rip1-1% ls -le README.txt
-rw-r--r-- 1 yarn  hadoop  1029 Feb 16 08:17 README.txt
 OWNER: user:yarn
 GROUP: group:hadoop
 SYNTHETIC ACL
 0: user:yarn allow file_gen_read,file_gen_write,std_write_dac
 1: group:hadoop allow file_gen_read
 2: everyone allow file_gen_read
```

Figure 23. POSIX file ls –le example

In this example, the file permission state, ACLs, owner, and group information are displayed. The POSIX bits are used for access checks and can be returned to an NFS client correctly. The SMB clients view the synthetic ACL representation of those POSIX mode bits.

The following figure shows the output of `ls -len` for a POSIX file:

```
rip1-1% ls -len README.txt
-rw-r--r-- 1 507  500  1029 Feb 16 08:17 README.txt
 OWNER: user:507
 GROUP: group:500
 SYNTHETIC ACL
 0: user:507 allow file_gen_read,file_gen_write,std_write_dac
 1: group:500 allow file_gen_read
 2: SID:S-1-1-0 allow file_gen_read
```

Figure 24. **POSIX file ls –len example**

In this example, with the file permission state of `ls -len`, ACLs, owner, and group information are displayed numerically. User `yarn` maps to UID 507 and group `Hadoop` maps to GID 500.

The following figure shows the output of `ls -le` for an ACL file:

```
moby1-b-1# ls -le acl2.txt
-rwxrwx--- +  1 root  wheel  0 May 28 14:01 acl2.txt
 OWNER: user:root
 GROUP: group:wheel
 CONTROL:dacl_auto_inherited
 0: group:FOO\domain admins allow file_gen_all
 1: group:FOO\domain users allow file_gen_read,file_gen_execute
```

Figure 25. ACL file ls –le example

In this example, an ACL file with `ls -le` is displayed, with a total of 2 DACLs associated. The + indicates that the file has a real ACL. This information is only viewable through the PowerScale CLI. The POSIX bits are only for representation and might seem more permissive than the actual permission. They are a OneFS best estimate of representing the ACLs in POSIX bits. Because the file has a real ACL, all access checks for all clients will be against the ACL.

The following figure shows the output of `ls -len` for an ACL file:

```
moby1-b-1# ls -len acl2.txt
-rwxrwx--- +  1 0  0  0 May 28 14:01 acl2.txt
 OWNER: user:0
 GROUP: group:0
 CONTROL:dacl_auto_inherited
 0: SID:S-1-5-21-139699970-2054505304-2447390217-512 allow file_gen_all
 1: SID:S-1-5-21-139699970-2054505304-2447390217-513 allow file_gen_read,file_gen_execute
```

Figure 26. ACL file ls –len example

In this example, the same ACL file as in the previous example is displayed, now with `ls -len`. The user and group are translated to numeric values, and the DACLs show the actual SID values.

POSIX file permissions management

Managing file permissions for POSIX files is much like managing them in a single-protocol UNIX environment. A `chmod` can be run over an NFS export or from the PowerScale CLI with the same result. The `chmod` options are the same as in a standard UNIX environment. As the POSIX permission changes after a `chmod`, the synthetic ACL changes automatically, with a maximum of three DACLs, as shown in the following figure:

```
moby1-b-1# ls -le posix2.txt
-rwxr-x---     1 root  wheel  0 May 28 14:04 posix2.txt
 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
moby1-b-1# chmod 755 posix2.txt
moby1-b-1# ls -le posix2.txt
-rwxr-xr-x     1 root  wheel  0 May 28 14:04 posix2.txt
 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_execute
```

Figure 27. POSIX file permissions management

In the figure, after the `chmod 755` is run, the POSIX permission and synthetic ACL are updated as an additional DACL is displayed.

ACL file permissions management

ACL file permissions are managed through either an SMB share or the PowerScale CLI. As a user connects through an SMB share, the permissions can be updated through the file properties in the **Security** tab. However, from the PowerScale CLI, a `chmod +a` toggle is available, replicating the options from the file properties over an SMB share. The `chmod +a` option changes a file's mode bits to add a new ACE, inserting the ACE into the ACL. A DACL is added to an existing ACL file, as shown in the following figure:

```
moby1-b-1# ls -le acl2.txt
-rwxrwx--- +  1 root  wheel  0 May 28 14:01 acl2.txt
 OWNER: user:root
 GROUP: group:wheel
 CONTROL:dacl_auto_inherited
 0: group:FOO\domain admins allow file_gen_all
 1: group:FOO\domain users allow file_gen_read,file_gen_execute
moby1-b-1# chmod +a group wheel  allow dir_gen_all,object_inherit,container_inherit,inherit_only acl2.txt
moby1-b-1# ls -le acl2.txt
-rwxrwx--- +  1 root  wheel  0 May 28 14:01 acl2.txt
 OWNER: user:root
 GROUP: group:wheel
 CONTROL:dacl_auto_inherited
 0: group:wheel allow file_gen_all,object_inherit,container_inherit,inherit_only
 1: group:FOO\domain admins allow file_gen_all
 2: group:FOO\domain users allow file_gen_read,file_gen_execute
moby1-b-1# ls -len acl2.txt
-rwxrwx--- +  1 0  0  0 May 28 14:01 acl2.txt
 OWNER: user:0
 GROUP: group:0
 CONTROL:dacl_auto_inherited
 0: group:0 allow file_gen_all,object_inherit,container_inherit,inherit_only
 1: SID:S-1-5-21-139699970-2054505304-2447390217-512 allow file_gen_all
 2: SID:S-1-5-21-139699970-2054505304-2447390217-513 allow file_gen_read,file_gen_execute
```

Figure 28. ACL file permissions management

Note: If a file state is unknown or an administrator does not understand how PowerScale implements multi-protocol, running `chmod` or `chown`, or updating file properties in Windows Explorer, could lead to unexpected results. These results might be limited through OneFS ACL policies, which are cluster-wide policies. For more information, see OneFS ACL policies. Ensure that you have a thorough understanding of changing file states before making changes on a production cluster. We recommend that you test file state changes on the simulator.

File permission state changes

As previously noted, a file can be in only one state at a time. However, the file permission state of the file may be changed. If a file is in POSIX, it can be changed to an ACL file by opening it in Windows Explorer and modifying the permissions. If a file is in ACL, it can be changed to a POSIX file by running the following command on the PowerScale CLI:

```
chmod -b XXX <filename>
```

The `XXX` specifies the new POSIX permission. A bulk file state change from ACL to POSIX for all files and directories requires a custom script. Exercise extreme caution because this action affects all files and directories.

Note: The following script has an impact across an entire cluster. Ensure that you fully understand the impact of the script before running it. As with any significant change to a production cluster, first run the script in a lab environment to completely understand the repercussions.

To convert every file and directory from ACL to POSIX, run the following script:

```
find /ifs/path/to/change | while read FILENAME
do
PERM=$(stat -f "%OLp" "$FILENAME")
echo "$FILENAME: $PERM"
chmod -b "$PERM" "$FILENAME"
done
```

Another option for bulk file changes is the OneFS Permission Repair job. For more information about Permission Repair jobs, see the *PowerScale OneFS Permission Repair Job* white paper.

File state permission change example

This example illustrates a file starting in POSIX mode with a synthetic ACL, then changing to an ACL file, and finally changing back to a POSIX mode with a synthetic ACL.

The following figure illustrates a file named `This_is_PowerScale.txt` in POSIX mode with a synthetic ACL:

```
rip2-1# ls -le This_is_PowerScale.txt
-rwx------     1 hdfs  hadoop  0 Nov 29 05:45 This_is_PowerScale.txt
 OWNER: user:hdfs
 GROUP: group:hadoop
 SYNTHETIC ACL
 0: user:hdfs allow file_gen_read,file_gen_write,file_gen_execute,std_write_dac
 1: group:hadoop allow std_read_dac,std_synchronize,file_read_attr
```

Figure 29. File with POSIX mode and synthetic ACL

When the file is opened in Windows Explorer, the file permissions are modified. The file state changes to ACL, as shown in the following figure:

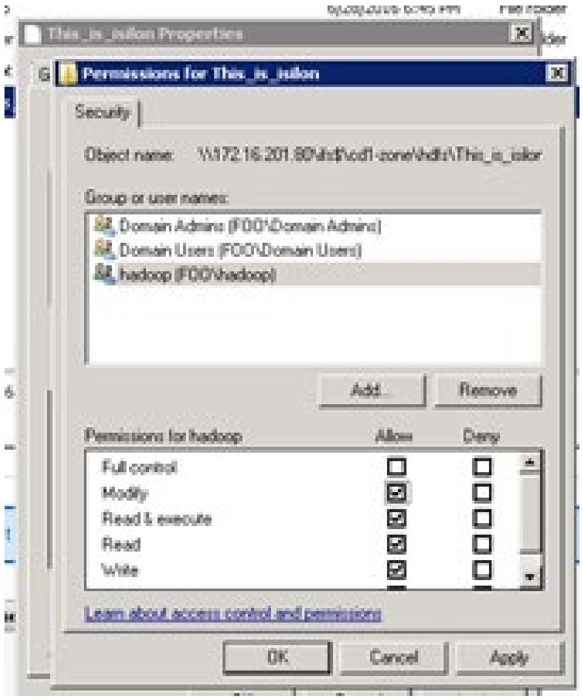


Figure 30. File permissions are modified with Windows Explorer

To confirm the file state, run `ls -le`:

```
rip2-1# ls -le This_is_PowerScale.txt
-rwxrwx--- +   1 hdfs  hadoop  0 Nov 29 05:45 This_is_PowerScale.txt
 OWNER: user:hdfs
 GROUP: group:hadoop
 CONTROL:dacl_auto_inherited
 0: group:Domain Admins allow file_gen_all
 1: group:hadoop allow std_read_dac,std_synchronize,file_read_attr
 2: user:hdfs allow file_gen_read,file_gen_write,file_gen_execute,std_write_dac
 3: group:Domain Users allow file_gen_read,file_gen_execute
```

Figure 31. File in ACL mode

Now that the file is in ACL mode, changing it back to POSIX mode with a synthetic ACL requires running the `chmod -b` command. Run the following command to change the file back to POSIX mode with a synthetic ACL and a 755 permission:

```
chmod -b 755 This_is_PowerScale.txt
```

To confirm the file state, run `ls -le`:

```
rip2-1# ls -le This_is_PowerScale.txt
-rwxr-xr-x     1 hdfs  hadoop  0 Nov 29 05:45 This_is_PowerScale.txt
 OWNER: user:hdfs
 GROUP: group:hadoop
 SYNTHETIC ACL
 0: user:hdfs allow file_gen_read,file_gen_write,file_gen_execute,std_write_dac
 1: group:hadoop allow file_gen_read,file_gen_execute
 2: everyone allow file_gen_read,file_gen_execute
```

Figure 32. File in POSIX mode with a synthetic ACL

Note: If a file state is unknown or an administrator does not understand how PowerScale implements multi-protocol, running `chmod` or `chown`, or updating file properties in Windows Explorer, it could lead to unexpected results. The results might be limited through OneFS ACL policies, which are cluster-wide policies. For more information, see OneFS ACL policies. Ensure that you have a thorough understanding of changing file states before making changes on a production cluster. We recommend that you test file state changes on the simulator.

ACLs and OneFS

For more information about ACLs, see the *Access Control Lists on Dell PowerScale OneFS* white paper.

SID history

OneFS 8.0.1 introduced support for SID history. SID history is an Active Directory attribute that maintains a history of previous SID values if an object is moved from another domain. SIDs are prefixed with a unique domain identifier. If users and groups are migrated from one Active Directory domain to another domain, each migrated object will have a new SID with a domain identifier of the new domain. When migrated users to the new domain attempt to access older files, access would be denied because the file permission would have the new SID. SID history retains the old SIDs, allowing them to be used for access checks.

Note: Historical SIDs cannot be used to add users to new groups or roles. Modify users or add them to a role or group only through the current object SID as defined by the domain.

Before OneFS 8.0.1, historical SIDs were not included in the access token because they were not recognized. In OneFS 8.0.1 and later versions, information from the Active Directory PAC is no longer discarded. For LDAP, OneFS queries the SIDHistory field to add the historical SIDs. If OneFS has a historical SID, then an RPC lookup is performed to find the current SID. Next, another RPC lookup is performed for SID to name resolution.

Historical SIDs may be viewed with the following commands:

```
isi auth users view <user>
isi auth groups view <group>
isi auth mapping token <user>
```

From an administrative perspective, additional configuration is not required. The SID history attribute is now recognized. Files that have historical SIDs can now be accessed by users with those historical SIDs. The previous domain users still have access to the file because the historical SIDs are left on disk.

Access checks with tokens and file permissions

When a user tries to access a file, OneFS compares the identities in the user's access token with the file permissions. If the file permission contains an allow Access Control Entry (ACE) for the identity and does not contain a deny ACE for the identity, OneFS grants access to the identity. As an example, the token and a file permission are displayed here:

```
isi auth mapping token --user=MAINE-UNO\jsmith
      User
            Name : MAINE-UNO\jsmith ❶
            UID : 1000000
            SID : S-1-5-21-3542649673-1571749849-686233814-1117
            On Disk : S-1-5-21-3542649673-1571749849-686233814-
            1117
            ZID: 1
            Zone: System
            Privileges: -
Primary Group
      Name : MAINE-UNO\domain users
      GID : 1000000
      SID : SID:S-1-5-21-3542649673-1571749849-686233814-513
Supplemental Identities
      Name : MAINE-UNO\marketing ❷
      GID : 1000001
      SID : SID:S-1-5-21-3542649673-1571749849-686233814-1109
      Name : Users
      GID : 1545
      SID : S-1-5-32-545
```

Here is the file permission for a file on the cluster:

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

The following items 1 through 4 refer to the labels in the preceding examples:

1. In the token, the user's identity is MAINE-UNO\jsmith, which is an Active Directory account.
2. The token also shows that the user is a member of the marketing group.

3. In the file permission, an access control entry shows that MAINE-UNO\jsmith is allowed to access the file. The ACE also lists the user's permissions, such as file_gen_write, which is permission to write to the file.
4. This ACE shows that members of the marketing group are allowed to read the file.

OneFS ACL policies

PowerScale OneFS ACL policies are cluster-wide policies for controlling how permissions are processed and managed. Each policy provides an option to reflect a specific environment or workflow affecting permission change interactions. The previous sections discussed how to change a permission state and how an unintended permission change could have unexpected results. The unexpected results are essentially where ACL policies affect the behavior.

For example, the impacts of ACL policies control the use of `chmod` for files and `chmod +a/-a` for setting DACLs. Settings are available for each environment, and the fine-tuning of each setting is also possible. When a different environment is selected, the other ACL settings also change. For most workflows, the **Balanced** setting is the best option. Because the ACL policies are cluster-wide, the **Balanced** setting supports a multi-protocol environment evenly, considering both UNIX and Windows.

By default, the environment is set for **Balanced**, as shown in the following figure:

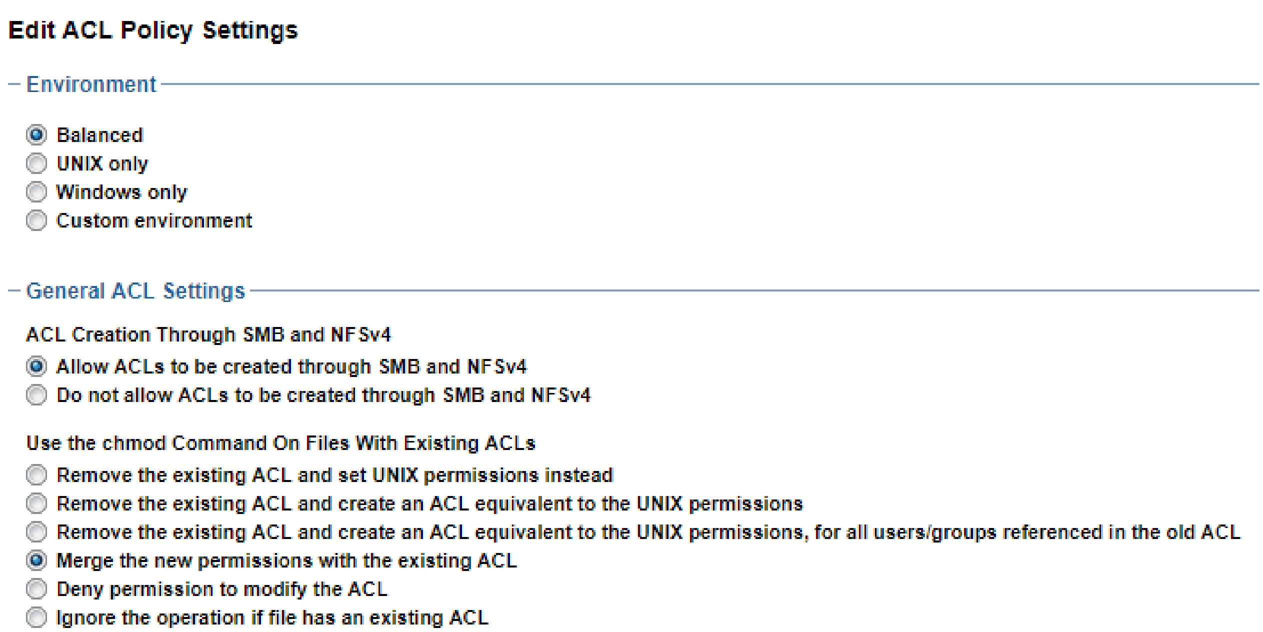


Figure 33. OneFS ACL policy settings

The options in the top two fields under **General ACL Settings** change automatically to reflect the environment option selected. For an explanation of each field, see the *PowerScale OneFS Web Administration Guide*. The focus of this section is on the first two fields under **General ACL Settings**, which provide the following options:

- **ACL Creation Through SMB and NFSv4**
  - **Allow ACLs to be created through SMB and NFSv4**: This option is preselected with the **Balanced** and **Windows only** environment options. It allows ACL creation external to a PowerScale cluster. For SMB, it allows ACL creation through File Explorer in an SMB share. For NFSv4, it allows ACL creation over an NFSv4 export. It is not applicable to NFSv3.
  - **Do not allow ACLS to be created through SMB and NFSv4:** This option is preselected with the **UNIX only** environment option. It disallows any ACL creation over an SMB share or NFSv4 export.
- **Use the chmod Command On Files With Existing ACLs**
  - **Remove the existing ACL and set UNIX permissions instead**: This option is preselected with the **UNIX only** environment option. If a file has an existing ACL on it, running a `chmod` removes the existing ACL and sets the new UNIX permissions that are specified.
  - **Merge the new permissions with the existing ACL:** This option is preselected with the **Balanced** environment option. If a `chmod` is run on a file with existing ACLs, the new permissions are merged, creating additional ACLs for the file.
  - **Deny permission to modify the ACL**: This option is preselected with the **Windows only** environment option. If a `chmod` is run on a file with existing ACLs, the operation is denied.

Note: As with any major update on a PowerScale cluster, practice extreme caution while updating these options because they are cluster-wide and could lead to unexpected and undesired results. Before updating a production cluster, configure a lab environment to learn how ACL policies would affect a specific workflow.

OneFS Permission Repair

To support the PowerScale Unified Permission Model, OneFS provides options for permission and on-disk repairs. The Permission Repair job supports three different modes: Clone, Convert, and Inherit. As an alternative to making changes through an NFS export or SMB share, the Permission Repair job runs directly on the cluster and across nodes. The job can be scheduled and is part of the OneFS Job Engine, providing reliable and consistent results.

For more information about the Permission Repair job, see the *PowerScale OneFS Permission Repair Job* white paper.

Role Based Access Control

Role Based Access Control (RBAC) enables PowerScale administrators to delegate administrative tasks to cluster-authenticated users. Roles can be assigned to users and groups to control administrative access. By default, only the `root` and `admin` users have access to the CLI and the web interface. The `root` and `admin` users can add administrative privileges to other users or create custom roles. As the role of the cluster grows across departments, you must ensure that each additional user has the minimum required access levels to enforce security and better maintain accountability.

Before OneFS 8.2.0, roles and privileges could be created and assigned only from the System access zone. All administrators, including administrators who own privileges by being a member of a role, must connect to the System access zone to configure the cluster. When these administrators log in to the cluster through the WebUI, SSH, or API interface, they can view and modify all access zones in the cluster based on the granted privileges. For more information about RBAC, see the *OneFS CLI Administration Guide*.

Beginning with OneFS 8.2.0, Zone-aware Role Based Access Control (ZRBAC) provides a more granular cluster administration. Administrators might want to delegate a user to perform administrative tasks in a specific access zone only, but disallow the user to have control over other access zones. ZRBAC supports this requirement by enabling roles and a subset of privileges to be assigned on a per-access-zone level. A user in the System access zone can still view and modify all non-System access zones. There are two roles, ZoneAdmin and ZoneSecurityAdmin, for zone-specific administration. Administrators from non-System access zones can connect to a cluster only through the WebUI or API interface. The following table outlines the access methods supported by RBAC and ZRBAC:

Table 4. Access methods supported by RBAC and ZRBAC

	Zone	WebUI access	API access	SSH access
RBAC (before OneFS 8.2.0)	System access zone	√	√	√
	Non-System access zone			
ZRBAC (OneFS 8.2.0 and later)	System access zone	√	√	√
	Non-System access zone	√	√	

Privileges available in non-System access zones

The following table shows the privileges that are allowed in non-System access zones for ZRBAC, which are also known as zone-based privileges. For information about all supported privileges, see the *OneFS CLI Administration Guide*.

Table 5. Privileges available in non-System access zones

Privilege	Description
ISI_PRIV_AUDIT	• Add/remove your zone from list of audited zones • View/modify zone-specific audit settings for your zone • View global audit settings
ISI_PRIV_AUTH	• View/edit your access zone • Create/modify/view users, groups, and local authentication providers • View/modify auth mapping settings for your zone • View global settings related to authentication • View global authentication providers and add them to your zone
ISI_PRIV_FILE_FILTER	Configure file filtering settings in your own zone
ISI_PRIV_HDFS	Configure HDFS settings in your own zone
ISI_PRIV_NFS	• View NFS global settings, but not modify them. • Configure NFS zone settings and export settings in your own zone.
ISI_PRIV_SMB	• View SMB global settings, but not modify them. • Configure SMB zone settings and share settings in your own zone.
ISI_PRIV_SWIFT	Configure SWIFT settings in your own zone
ISI_PRIV_ROLE	Create roles and assign privileges in your own zone
ISI_PRIV_VCENTER	Configure VMware for vCenter in your own zone
ISI_PRIV_LOGIN_PAPI	Log in to the Platform API and the WebUI in your own zone
ISI_PRIV_IFS_BACKUP	Bypass file permission checks and grant all read permissions inside the zone base path
ISI_PRIV_IFS_RESTORE	Bypass file permission checks and grant all write permissions inside the zone base path
ISI_PRIV_NS_TRAVERSE	Traverse and view directory metadata inside the zone base path
ISI_PRIV_NS_IFS_ACCESS	Access directories inside the zone base path through RAN

Authentication provider behavior between RBAC and ZRBAC

Before OneFS 8.2.0, authentication providers were created in System access zones and accessible by any access zones in a cluster. Each non-System access zone contains only its own local provider and uses other providers in the System access zone, as shown in the following figure. A local provider is created implicitly in each access zone.

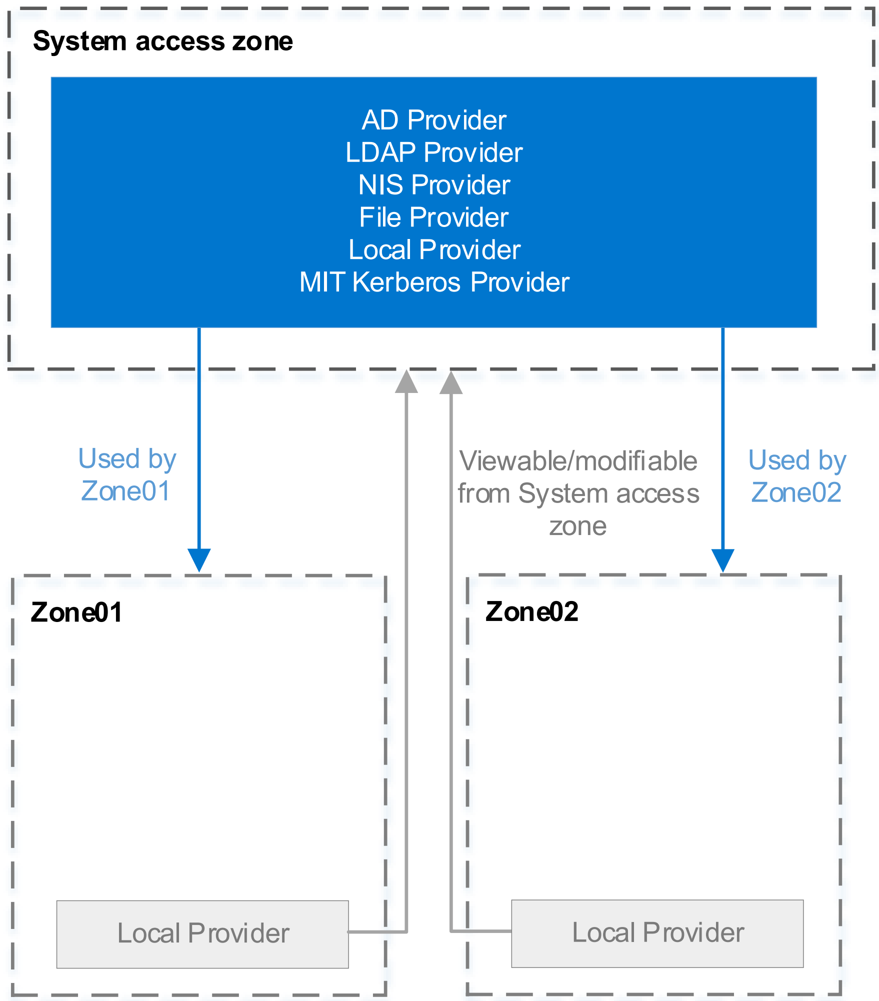


Figure 34. Authentication provider behavior in RBAC

Starting with ZRBAC in OneFS 8.2.0, when an authentication provider is created from an access zone, it is implicitly associated with the access zone. As shown in Figure 35, an authentication provider has following behavior based on that association.

- An authentication provider created from System access zone:
  - Can be viewed and used by all access zones
  - Can be modified/deleted only from System access zone
- An authentication provider created from a non-System access zone:
  - Can only be used by that specific non-System access zone
  - Cannot be used by other access zones, including System access zone

- Can be viewed/modified/deleted only from that specific access zone and System access zone

- The MIT Kerberos provider can only be created from System access zone and used by all access zones.
- A local provider in a non-System access zone:
  - Can only be used by that specific non-System access zone
  - Cannot be used by other access zones, including System access zone
  - Can be viewed/modified only from that specific access zone and System access zone

---

Note: The name of an authentication provider must be unique globally. For example, you cannot create an LDAP provider named “ldap01” in two different access zones.

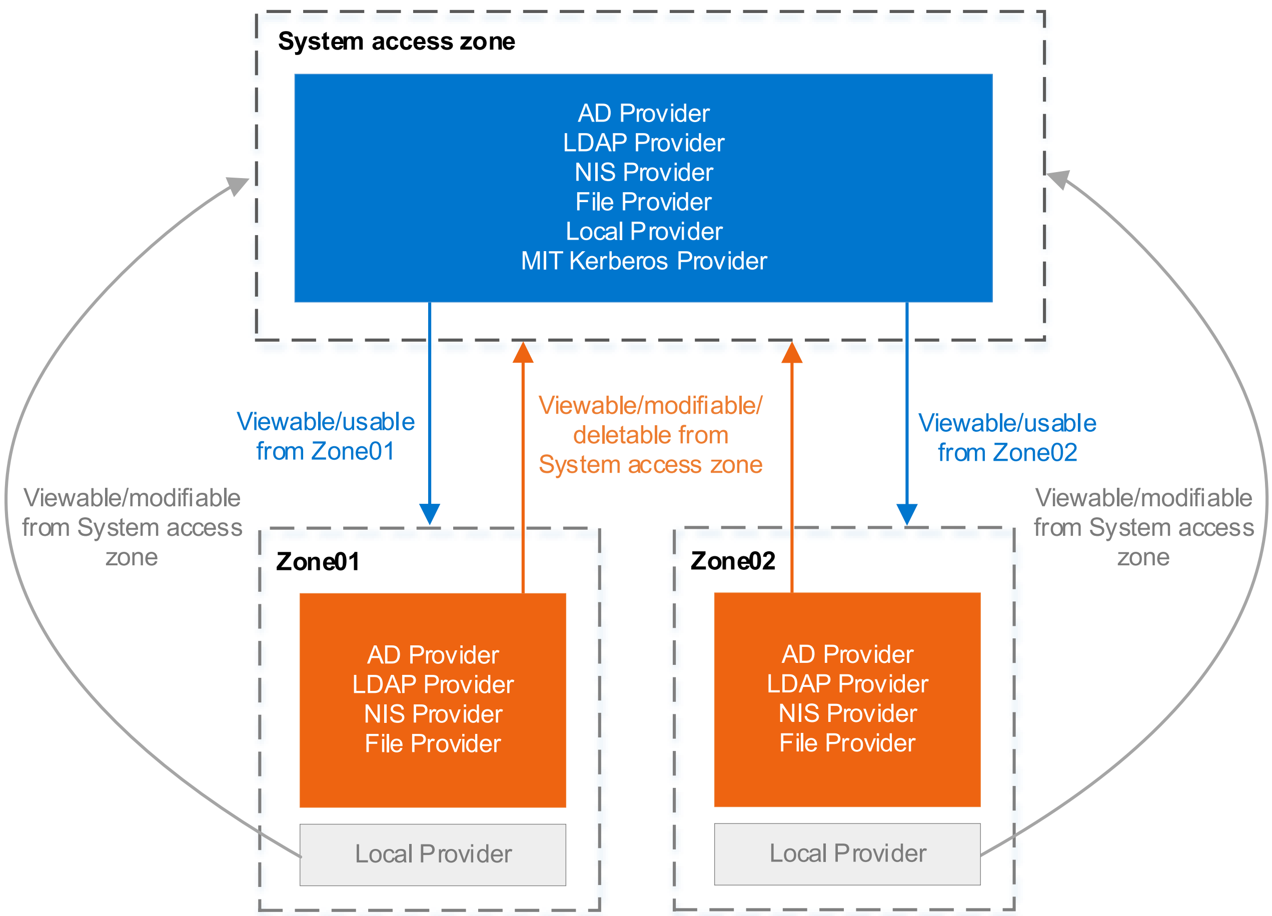


Figure 35. Authentication provider behavior in ZRBAC

Multi-instance Active Directory authentication provider

Previously, only one connection to a Microsoft Active Directory domain was allowed, and the name of the Active Directory provider had to be the same as the domain name. For the ZRBAC in OneFS 8.2 and later, a multi-instance Active Directory authentication provider allows multiple connections to a same Active Directory. Multiple access zones can create their own Active Directory provider, which connects to the same Active Directory, but each access zone can still have at most one Active Directory provider.

To use the multi-instance Active Directory provider, you must take the following actions when configuring the Active Directory provider in an `isi` CLI command or the WebUI:

- Create a provider instance name for the provider.
- Create a machine account in Active Directory for the provider.

In the `isi` CLI command, specify `--instance` and `--machine-account` options when using `isi auth ads create`. For example:

```
# isi auth ads create --name=example.com --user=administrator
--instance=example01 --machine-account=ad-zone01
```

The following figures shows the WebUI fields for specifying the instance name and machine account name:

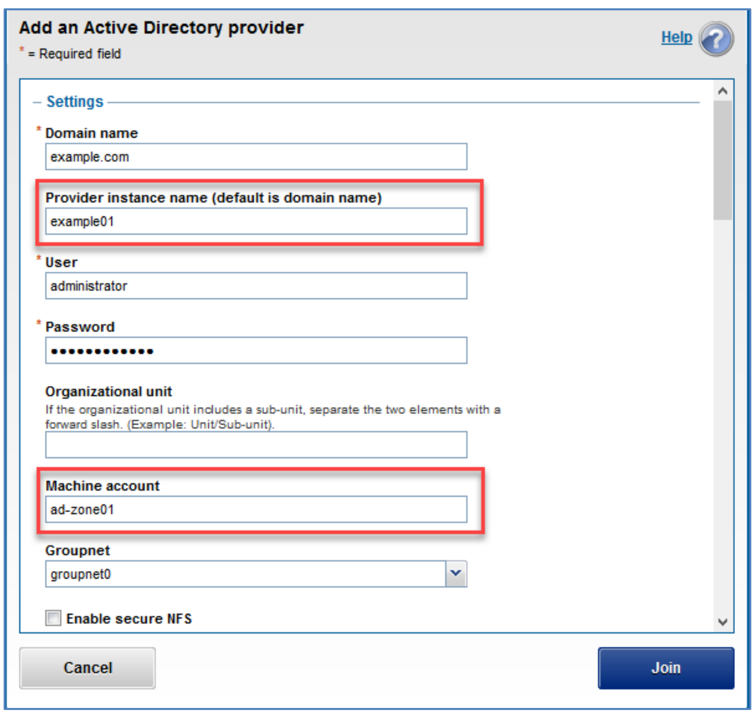


Figure 36. Multi-instance Active Directory provider configuration

SSH multifactor authentication with Duo

Duo is a vendor of cloud-based multifactor authentication (MFA) services. MFA enables security to prevent a hacker from masquerading as an authenticated user. Duo allows an administrator to require multiple options for secondary authentication. With multifactor authentication, even when stealing the username and password, a hacker cannot be easily authenticated to a network service without a user's device.

Starting with version 8.2.0, OneFS supports SSH MFA with the Duo service through SMS, phone callback, and push notification through the Duo Mobile app. SSH MFA does not bypass any existing access-check process on OneFS. A user must have a valid password or public-private key and the RBAC SSH privilege. Currently, the SSH MFA configuration supports only CLI commands (no WebUI support). The following CLI commands allow you to view and configure related exposed settings:

- SSH access settings: `isi ssh view/modify`
- Duo service settings: `isi auth duo view/modify`

To use Duo with OneFS, an administrator must have a Duo account to configure the following settings in the Duo service:

- Create an "UNIX Application" entry to represent the OneFS cluster. OneFS will use the information contained in the "UNIX Application," including the Duo service API hostname, integration key, and secret key.
- Create user accounts that use Duo. Duo does not have any access to data or configuration on the cluster. All users in the OneFS cluster that will use SSH MFA with Duo must be added to the Duo service.

Note: By default, the Duo username normalization is not Active Directory aware, which means that it will alter incoming usernames before trying to match them to a user account. For example, `DOMAIN\username`, `username@domain.com`, and `username` are treated as the same user.

For configuration steps, see Configure SSH MFA on OneFS 8.2 Using Duo.

SAML-based SSO for WebUI

Security Assertion Markup Language (SAML) is an open standard for sharing security information about identity, authentication, and authorization across different systems. SAML is implemented with the Extensible Markup Language (XML) standard for sharing data. SAML defines three categories of entities:

- **End user**—A person who must be authenticated before being allowed to use an application.
- **Service provider (SP)**—Any system that provides services, for example, PowerScale cluster.
- **Identity provider (IdP)**—A special type of service provider that administers identity information, for example, Active Directory Federation Services (ADFS).

Beginning with version 9.5.0.0, OneFS supports SAML-based SSO for the WebUI by using ADFS. You can enable or disable the OneFS WebUI SSO feature at the access zone level. OneFS also allows you to configure one IdP for each access zone.

PowerScale migrations and permissions

Understanding how a migrated file affects file state is essential. NFS-based tools such as rsync provide POSIX permissions. SMB-based tools such as emcopy create ACL permissions.

If SMB files from another vendor's storage system are migrated to a PowerScale cluster, the shares, file data, security metadata, security identifiers, ACLs, ACEs, and inheritance attributes must all be migrated.

Datadobi greatly simplifies migrations by migrating all file permissions over to a PowerScale cluster.

Troubleshooting and commands

To troubleshoot access issues, review each of the components shown in the PowerScale OneFS tokens and file permissions infographic. Confirm the contents of the components as follows:

- Confirm the expected permission of a user based on a specific path to file:

  ```
  # isi auth access --path=<path_to_file>  --
  user=<user_to_view> -v
  ```

- List the privileges for the issuing user, including memberships, On-Disk, RBAC, and active sessions:

  ```
  # isi auth id
  ```

- Check authentication providers, confirming that they are online and operational. A –d is optional for verbose output:

  ```
  # isi_auth_expert
  ```

- Confirm file permissions:

  ```
  # ls –le & ls –len
  ```

- Review user-mapping information:

  ```
  # isi auth mapping list --zone
  ```

Anatomy of a cross-platform access token

If user-mapping rules are not configured, a user authenticating with one directory service receives full access to the identity information in other directory services when the account names are the same. For example, a user who authenticates with an Active Directory domain `york\jane` automatically receives identities for the corresponding UNIX user account for Jane from LDAP or NIS.

In the most common scenario, OneFS is connected to two directory services, Active Directory and LDAP. In such a case, the default mapping provides a user with a UID from LDAP and a SID from the default group in Active Directory. The user's groups come from Active Directory and LDAP. The user's home directory, gecos, and shell come from Active Directory.

The following examples demonstrate how OneFS builds an access token for a Windows user who authenticates with Active Directory but has a corresponding account with the same name in LDAP. User-mapping rules are not in place.

First, view a user's token from only Active Directory by running the following command and targeting the user's Active Directory domain account. The output shown is abridged to remove some immaterial information.

```
isi auth users view --user=york\\stand --show-groups
      Name: YORK\stand
      DN: CN=stand,CN=Users,DC=york,DC=hull,DC=example,DC=com
      DNS Domain: york.hull.example.com
      Domain: YORK
      Provider: lsa-activedirectory-provider:YORK.HULL.EXAMPLE.COM
      Sam Account Name: stand
      UID: 4326
      SID: S-1-5-21-1195855716-1269722693-1240286574-591111
      Primary Group
      ID : GID:1000000
      Name : YORK\york_sh_udg
      Additional Groups: YORK\sd-york space group
      YORK\york_sh_udg
      YORK\sd-york-group
      YORK\sd-group
      YORK\domain users
```

Next, view a user's token from only LDAP by running the following command and targeting the user's LDAP account. The output is abridged.

```
isi auth user view --user=stand --show-groups
      Name: stand
      DN:
      uid=stand,ou=People,dc=colorado4,dc=hull,dc=example,dc=com
      DNS Domain: -
      Domain: LDAP_USERS
      Provider: lsa-ldap-provider:Unix LDAP
      Sam Account Name: stand
      UID: 4326
      SID: S-1-22-1-4326
      Primary Group
      ID : GID:7222
      Name : stand
      Additional Groups: stand
      sd-group
      sd-group2
```

When there are no mapping rules, and when the user logs in to the cluster over SMB, OneFS authenticates the user with Active Directory and builds an access token. It prioritizes the account information from Active Directory, but appends the supplemental groups from the UNIX LDAP token to the end of the final token:

```
isi auth mapping token --user=york\\stand
      User
      Name : YORK\stand ❶
      UID : 4326 ❷
      SID : S-1-5-21-1195855716-1269722693-1240286574-591111 ❸
      On Disk : 4326
      ZID: 1
      Zone: System
      Privileges: -
Primary Group
      Name : YORK\york_sh_udg ❹
      GID : 1000000
      SID : S-1-5-21-1195855716-1269722693-1240286574-66133
Supplemental Identities
      Name : YORK\sd-york space group ❺
      GID : 1000002
      SID : S-1-5-21-1195855716-1269722693-1240286574-579109
      Name : YORK\sd-york-group
      GID : 1000004
      SID : S-1-5-21-1195855716-1269722693-1240286574-475739
      Name : YORK\sd-workers
      GID : 1000003
      SID : S-1-5-21-1195855716-1269722693-1240286574-169779
      Name : YORK\domain users
      GID : 1000001
      SID : S-1-5-21-1195855716-1269722693-1240286574-513
      Name : Users ❻
      GID : 1545
      SID : S-1-5-32-545
      Name : sd-group ❼
      GID : 100001
      SID : S-1-22-2-100001
      Name : sd-group2
      GID : 100002
      SID : S-1-22-2-100002
```

The following items 1 through 7 refer to the labels in the preceding example:

1. The primary username is from Active Directory.
2. The user's UID is from LDAP.
3. The user's SID is from Active Directory.
4. The primary group is from Active Directory.
5. These supplemental identities are from Active Directory, as indicated by the name of the domain before the name of the group.

6. The group named Users and its GID of 1545 is a OneFS local group that comes from the OneFS local provider. It appears in the token by default because the OneFS operating system adopts the standard Microsoft Windows practice of adding the Domain Users group to the local users group when the system is joined to an Active Directory domain.
7. These last two groups are appended to the token from the UNIX LDAP token.

The mapping service omits the user's LDAP primary group. Add the primary group from LDAP to the final token by creating a user-mapping rule.

By default, when you run the `isi auth mapping` command with a UNIX username, OneFS looks up the UNIX user's information from LDAP without mapping it to the UNIX user's Active Directory account information. Why? Because OneFS gives preference to using a UID to maximize NFS performance. If OneFS showed the information from Active Directory as well, the results of the command would have visual symmetry with the result of an `isi auth mapping` request for an Active Directory user, which includes the information from LDAP. However, the visual symmetry would come at the expense of NFS performance.

Anatomy of a cross-platform file permission

On a PowerScale cluster, each ACE in a file permission 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 in 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 subdirectories and files. Inheritance takes place when files and subdirectories are created; modifying an inherited rule affects only new files and subdirectories, not existing ones. 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 to this object but will apply to descendants when inherited. For example, when this flag is set on a directory, the ACE for the directory will not apply to the directory but will apply to its subdirectories.
- inherited_ace: The ACE was inherited.

The following file permission shows some of these components. The listing was obtained by running the `ls` command with an option (`le`) that PowerScale added to show the ACL. The option is available only on the PowerScale cluster, not on a UNIX client that has mounted an export. See the OneFS man page for the `ls` command. The plus sign that follows the POSIX mode bits indicates that the file contains an actual ACL, not a synthetic ACL.

```
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
```

Appendix A: Configuring Active Directory, LDAP, RFC 2307, and Kerberized NFS

Documentation

For more information about configuring Microsoft Active Directory, LDAP, or Kerberized NFS, see the following documents:

- OneFS Web Administration Guide
- Integrating OneFS with Kerberos Environment for Protocols
- How to configure OneFS and Active Directory for RFC2307 compliance

Note: An active Dell Support account is required to access the documents.

Required fields for LDAP

If you are using an LDAP server, such as OpenLDAP, the following fields are required:

- ldap-uid
- ldap-user-filter
- ldap-group-filter
- ldap-loginshell
- ldap-homedirectory

Microsoft Active Directory with RFC 2307

RFC 2307 was initially created to use LDAP as a Network Information Service. As enterprise requirements have evolved, Active Directory and RFC2307 have also evolved.

For more information about RFC 2307, see the official RFC: https://www.ietf.org/rfc/rfc2307.txt

RFC 2307 support for Active Directory first launched with Windows Server 2003 and exists today in Windows Server 2016. However, the implementation has been through several iterations. The initial release of Active Directory with RFC 2307 was referred to as "Services for UNIX." It was later renamed to "Identity Management for UNIX" and enables:

- Management of user accounts and passwords on Windows and UNIX systems through Server for Network Information Service (NIS)
- Automatic synchronization of passwords between Windows and UNIX operating systems

From a OneFS perspective, integrating RFC 2307 with Active Directory simplifies the management of users in a multi-protocol environment because only a single authentication provider is required to collect the SID and UID with associated GIDs. In this architecture, Active Directory stores the user credentials, and RFC2307 stores UIDs and GIDs. OneFS does not require the NIS authentication component because only the UID/GIDs are used. Active Directory with RFC 2307 maps SIDs with UID/GIDs, eliminating the need for mapping in OneFS, which further simplifies management.

Through Windows Server 2003, 2008, 2012, and 2016, the RFC 2307 support has varied significantly, not only from a cosmetic perspective, but by the overall implementation. For more information about the changes throughout the years, see this Microsoft Technet blog post. Although the RFC 2307 implementation has changed throughout the releases, RFC 2307 attributes (GID, UID, and so on) in Active Directory continue to exist, which is all that is required for simplifying a multi-protocol implementation with OneFS.

Required fields for Windows Services for UNIX

If you use Microsoft Active Directory with Windows Services for UNIX and RFC 2307 attributes to manage Linux, UNIX, and Windows systems, the following fields are required in Active Directory:

- uid
- uidNumber
- gidNumber
- loginShell
- UNIXHomeDirectory

Permission mapping

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.

Windows access rights to OneFS to mode bits

Table 6. Windows access rights to OneFS to mode bits

Windows access rights	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

Mapping mode bits to ACLs

Because mode bits are a subset of the richer Windows ACL model, mapping mode bits to ACLs is simpler. 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.

Table 7. Mapping for UNIX Read

Description	Permissions
UNIX Permission	Read
OneFS	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_DATA, FILE_READ_EA

Table 8. Mapping for UNIX Write

Description	Permissions
UNIX Permission	Write
OneFS	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 9. Mapping for UNIX Execute

Description	Permissions
UNIX Permission	Execute
OneFS	file_gen_write
Windows Effective Permissions	traverse folder / execute file
Mapping to Windows Access Rights Constants	FILE_TRAVERS, 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).

OneFS permissions for file system objects

Similar to the Windows permissions model, the PowerScale 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. Also, certain permissions apply only to a directory; others apply only to a nondirectory file system object.

Table 10. OneFS standard permissions

Description	Permissions
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 DAC L 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 11. OneFS directory permissions

Description	Permissions
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

Table 12. Specific permissions for directories

Description	Permissions
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 13. OneFS permission for nondirectory objects

Description	Permissions
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

Table 14. Specific permissions for nondirectory objects

Description	Permissions
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

Appendix B: Additional resources

See the following resources for more information:

- PowerScale OneFS Web Administration Guide
- PowerScale OneFS CLI Administration Guide
- Dell PowerScale OneFS: Security Considerations
- Dell PowerScale: Network Design Considerations
- PowerScale OneFS User Mapping