Qumulo Cross-Protocol Permissions

White Paper

August 2024

This document describes how Qumulo's cloud-native solution can flex from cost-effective cold storage to the most demanding high-performance computing and artificial intelligence workloads. The architecture's versatility allows it to accommodate any workload, providing the lowest total cost of ownership for cloud file services.

Introduction

Managing file access and permissions across multiple protocols presents significant challenges in today's diverse IT environments. The lack of industry standards for cross-protocol permissions often leads to inconsistencies, security vulnerabilities, and administrative complexities. This whitepaper delves into the intricacies of managing cross-protocol permissions (XPP) and highlights how Qumulo's innovative framework addresses these challenges. By offering a unified and efficient solution, Qumulo's XPP framework ensures secure and seamless data access across various platforms, thereby enhancing productivity and collaboration in mixed-protocol workflows. Real-world scenarios and detailed examples illustrate the practical applications and benefits of Qumulo's approach, demonstrating its effectiveness in maintaining robust and consistent permissions management in heterogeneous environments.

The Challenge with File Permissions Across Different Protocols

POSIX vs. ACL Permissions

POSIX permissions are represented through mode bits that define read, write, and execute rights for the file owner, group owner, and others. These are relatively straightforward but lack the granularity of ACLs. ACLs, used by SMB and NFSv4.1, consist of Access Control Entries (ACEs) that specify permissions for users or groups in a more detailed manner. Managing these two systems simultaneously requires a robust framework that can translate and reconcile differences without compromising security or functionality.

Integrating permissions and authentication based on different protocols is inevitable in modern data management. Traditional permission models often fall short with the rise of multi-protocol environments where files are accessed over NFSv3, NFSv4.1, SMB, and even S3 protocols.

Each protocol has its own permissions model, with distinct expectations and mechanisms for securing files and directories. NFSv3 primarily uses POSIX-mode bits, which offer basic read, write, and execute permissions for the file owner, group, and others. SMB and NFSv4.1 employ Access Control Lists (ACLs), which not only provide more granular control – offering up to 14 different permission settings that can be assigned or denied to specific users or groups – but which also make use of Kerberos authentication to deliver a much more robust security model.

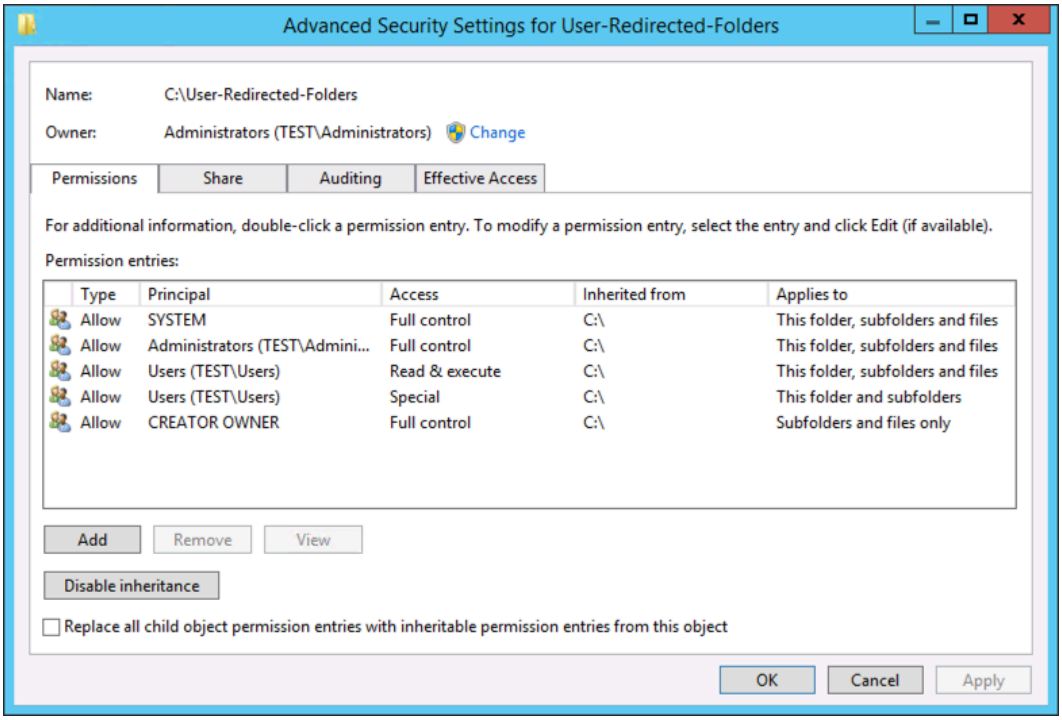

Figure 1: Managing ACL settings in Windows

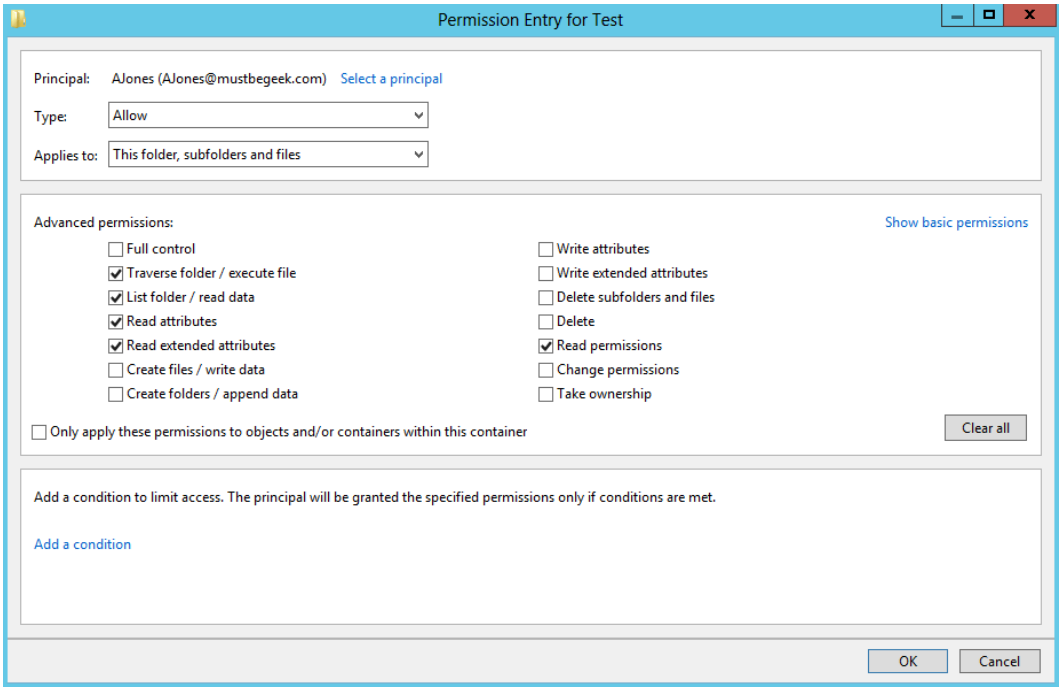

Figure 2: Managing permissions inheritability in Windows

Advanced permissions	Basic permissions				
	Full control	Modify	Read and execute	Read	Write
Traverse folder/execute file	✓	✓	✓		
List folder/read data	✓	✓	✓	✓	
Read attributes	✓	✓	✓	✓	
Read extended attributes	✓	✓	✓	✓	
Create folders/append data	✓	✓			✓
Write attributes	✓	✓			✓
Write extended attributes	✓	✓			✓
Delete subfolders and files	✓	✓			
Delete	✓	✓			
Read permissions	✓	✓	✓	✓	
Change permissions	✓				
Take ownership	✓				
Synchronize	✓				

Table 1: NTFS permissions

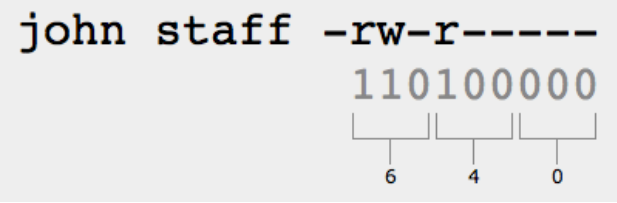

Figure 3: POSIX permission bit settings

In the POSIX world, there are three basic permission levels, and they mean different things when applied to files versus directories.

	Read	Write	Execute
Files	Read file	Append, Modify	Execute file
Directories	List directory	Create/rename/ delete children	Traverse directory (look at children)

Table 2: POSIX rights

These protocols also have very different and often conflicting concepts of object access control.

For example, when evaluating a user's right to access a file via SMB, that right might be granted by the permissions set directly in the object. In contrast, over NFS, the rights are often managed by the permissions set on the directory that contains the object.

The lack of compatibility and consistency across protocols has also often led to the unintended removal of previously applied permissions via user activity from a different protocol.

A classic manifestation is stripping inherited NTFS permissions from files and directories due to an NFS user's SETATTR operation, like **chmod** or **chown**.

Key challenges include:

- **Inconsistent Permissions:** Actions like modifying permissions using one protocol can unintentionally alter or invalidate permissions when accessed via another protocol.
- **Complex Management:** Admins must manually reconcile and manage permissions across protocols, often leading to increased administrative overhead and potential errors.
- **User Collaboration Issues:** Collaborative workflows are hampered when users encounter permission-related issues that prevent seamless access and modification of files.

Challenges in Cross-Protocol Environments

One of the main challenges in multi-protocol file data access is handling file permissions correctly. Different protocols expect different permissions models, which can lead to conflicts and potential security risks. For instance, NFSv3 users expect a POSIX model, whereas Windows users accessing files over SMB require ACLs. Qumulo's XPP framework addresses these issues by providing a unified approach to managing permissions across different protocols.

Other File Storage Vendors' Recommendations

Most storage vendors have historically strongly recommended the Storage Administrator choose to manage everything with a POSIX-only or NTFS-only approach or to store two completely separate and possibly conflicting sets of permissions specific to each protocol.

Some storage vendors provide more robust cross-protocol permissions support, which requires carefully applying complex and difficult-to-manage special flags and scenario-specific file system options.

NetApp:

NetApp offers the concept of unified permissions, attempting to harmonize permissions across protocols by implementing a mapping system. However, this often requires complex configurations and administrative intervention.

NetApp offers several best practices for managing file permissions effectively in a multiprotocol environment:

Choose the Appropriate Security Style

NTFS or Unix Security Style: To avoid complications, select either NTFS (ACL) or Unix (POSIX) as the security style. Mixed mode should generally be avoided, except in rare edge cases. This choice ensures consistent permissions application and reduces administrative overhead.

User Mapping

Name Mapping: Map NFS users to SMB(Windows) users or vice versa to ensure consistent permissions across different protocols. This mapping helps maintain access controls regardless of the protocol used to access the files.

Credential Cache: Utilize the WAFL credential cache (wcc command) to view and verify user mappings, which helps in troubleshooting and ensures that user identities are correctly mapped across protocols.

Export Rules and Unix Permissions

Export Rules: Define export rules clearly for new volumes, particularly specifying access controls for NFS clients using IPv4 addresses or subnets in CIDR notation. This ensures that only authorized clients can access the shared resources.

Unix Permissions: When creating new volumes, set appropriate Unix permissions to control access effectively. This is especially relevant for environments where Unix clients need to access the storage.

Limitations

Single Set of Permissions: Files on NetApp can have either SMB or NFS permissions but not both simultaneously. This limitation requires careful planning and user mapping to ensure appropriate access controls across different client systems.

Complexity in Mixed Protocol Environments: Configuring and managing permissions in a mixed protocol environment (e.g., both NFS and SMB) can be complex. Incorrect settings can lead to permission conflicts and access issues, necessitating thorough testing and validation during setup.

Performance Overhead: Name resolution and user mapping processes can impact system performance, especially in large environments. Proper configuration and optimization are necessary to minimize these impacts.

Administrative Burden: Managing permissions across different protocols often requires additional administrative efforts. This includes setting up and maintaining user mappings, export rules, and ensuring consistent permissions are applied across all access methods.

Dell EMC Isilon / PowerScale

Like NetApp, Isilon's best-practices recommendations for managing cross-protocol permissions is a complex endeavor. This complexity arises from the need to manage and reconcile POSIX permissions with ACL settings. Ensuring consistent access control and preventing permission conflicts require a deep understanding of both the underlying protocols and the way they're managed on an Isilon / PowerScale cluster.

Administrators must meticulously map user identities across both security models in order to secure data effectively. This involves configuring authentication sources, managing identity mappings, and applying appropriate permissions while ensuring compliance with security policies.

Limitations

Permission Inconsistencies: In mixed POSIX and ACL environments, managing consistent permissions can be challenging. Misconfigurations may lead to unexpected access issues or security vulnerabilities.

Permission Propagation: When files or directories are copied or moved, their permissions may not always propagate as expected. For instance, copying files might not retain the original ACLs, leading to potential access control issues.

Performance Overheads: The process of mapping users and groups between different protocols can introduce delays, impacting performance, especially in large environments.

Complexity in Large Deployments: Managing permissions across a large number of files and directories, particularly in multi-protocol environments, requires significant administrative effort and careful planning.

The entire process, which demands a high level of expertise and careful attention to detail, with a high likelihood of either initial misconfiguration or gradual erosion of security standards over time, and with adverse consequences (e.g. regulatory noncompliance, unauthorized data access, etc.) for security lapses. Their best practices document is complex and long, over 100 pages.

Overview of Qumulo's Cross-Protocol Permissions

Qumulo's Cross-Protocol Permissions (XPP) model is designed to handle the inherent incompatibilities between POSIX and ACL permissions, greatly simplifying the process of managing data in heterogeneous environments and enabling users across both permissions models to collaborate seamlessly. XPP ensures that permissions are managed consistently, regardless of the protocol used.

Qumulo employs a proprietary, unified internal access-control tracking system, known as QACL, which maps closely to industry-standard permissions models while understanding the superset of rights encompassing both POSIX and ACL permissions. This approach allows Qumulo to internally maintain both permissions models independently for the same data, while also enabling the use of POSIX mode bits and ACL-based permissions, such as the ability to create and change files, change file permissions, and change file ownership, in a way that is consistent and predictable across different protocols.

Where both NetApp and Dell PowerScale offer only management complexity and inconsistent security models, Qumulo's XPP framework was explicitly designed to be simple, reliable, and powerful. It requires no configuration, works transparently and automatically, and does not require a tree walk to enable. This ease of use makes it highly recommended for most customers.

Qumulo invested over two years of engineering time developing XPP in close association with several of our most demanding customers, who provided us with real-world scenarios crucial to their daily workflows and then rigorously tested our code.

Qumulo's XPP delivers a method that allows Administrators to enforce the access controls needed to secure a system while honoring the intent of operations performed by end users and all the while requiring the least number of special knobs and settings.

Qumulo XPP is enabled by default and allows Administrators to choose the permissions management approach that best fits the task at hand on a directory-by-directory basis. It also allows administrators to change their minds at any time.

For example, you could initially manage Directory A with POSIX permissions while managing Directory B via ACL, and then subsequently enable Directory A with ACL access and Directory B with POSIX security without impacting existing permissions on either folder.

With Qumulo, users can collaborate safely without setting up elaborate permissions schemes. XPP leverages the full power of ACLs across both SMB and NFS, ensuring that one set of permissions applies universally. This significantly reduces the time spent cleaning up permissions and fixing broken inheritance, as typical NFS operations like CHMOD no longer disrupt SMB permissions.

SMB, NFSv3, NFSv4.1, S3, FTP or REST – all protocols and API endpoints use the same unified permissions set. For example, Qumulo XPP allows the application and enforcement of complex ACL-based permissions to files accessed via NFSv3 and the application of POSIX-specific operations like SetGID to SMB clients.

Moreover, Linux applications such as **rsync**, **cp**, or **vi** function properly with XPP, even if administrators do not grant file owners full rights, which is a scenario that many administrators may prefer. This ensures that administrative preferences and operational efficiency are maintained without compromising on security or functionality.

The unified approach to managing ACL and POSIX permissions also allows for managing rights that would not normally be possible via a single protocol.

Qumulo XPP enables:

- Seamless mapping of the standard 14 ACL permissions into the standard three POSIX rights
- Direct management of access permissions by file and directory owners, rather than systems administrators
- Consistent treatment of all users over all protocols
- Admin-level overrides to deny user change rights even for file owners
- Admin-managed permissions via ACL inheritance
- Reconciliation of POSIX user accounts from NFS clients with the same users' LDAP accounts from SMB and NFSv4.1 clients
- The ability to STAT a file and get the same mode back if you check it
- Display POSIX mode accurately showing User/Group/Owner rights

Key Components of Qumulo XPP

From a file storage perspective, authentication and authorization are two fundamental concepts that ensure secure access to files and resources.

Authentication is the process of verifying the identity of a user or system attempting to access the storage system. In the context of file storage, protocols like SMB (Server Message Block) require authentication to ensure that the user trying to access the files is indeed who they claim to be. This is typically done through mechanisms like usernames and passwords, Kerberos tickets, or digital certificates. For example, when a user attempts to access a network share over SMB, they must provide valid credentials that have been authenticated against a centralized directory service like Active Directory.

Authorization, on the other hand, determines what an authenticated user is allowed to do within the file system. Once a user's identity is confirmed through authentication, the system checks their permissions to see what actions they are permitted to perform on specific files or directories. File permissions govern this aspect and include rules that specify who can read, write, modify, or execute a file. These permissions are set and managed through Access Control Lists (ACLs) in the case of SMB, or using UNIX file permissions for NFS (Network File System).

Authentication Services

Authentication services ensure that only authorized users can access the file system. These services are available through protocols that support authentication mechanisms, such as SMB, NFSv4.1, S3, FTP, and REST. This broad compatibility allows for secure access across different network environments.

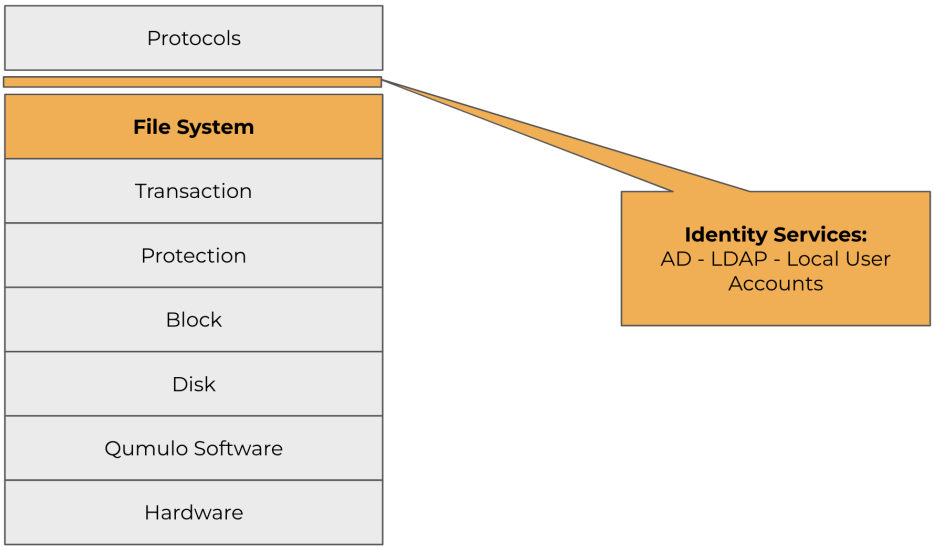

Figure 4: Identity and authentication as part of the data access stack

Qumulo supports a range of authentication systems to cater to different organizational needs. This includes:

- Active Directory
- Local User Accounts
- Kerberos (AD & Standalone)
- NTLMv2
- SAML SSO for REST API

It is important to note that positive user authentication is not possible over NFSv3 due to limitations within the protocol itself. This means user identities cannot be definitively verified when accessing files through NFSv3.

User Identity Mapping

Qumulo employs various methods to map AD/Kerberos-style identities (typically used in Windows environments) to POSIX-style identities (common in UNIX/Linux systems). This mapping ensures that permissions and access controls are maintained consistently across different systems.

Preferred Method

Active Directory with POSIX Attributes: The most efficient method involves using Active Directory (AD) with POSIX attributes. This utilizes the RFC2307 standard, which allows AD to store UNIX attributes like uidNumber and gidNumber, providing a unified identity management system.

Alternative Methods

Other methods include using local user accounts specific to the Qumulo cluster and LDAP to AD user-defined mappings, which involve creating a custom mapping between LDAP and AD users.

Active Directory with POSIX Attributes

Qumulo's XPP system maps user identities across different protocols to ensure consistent access control. By using Active Directory with POSIX attributes as the preferred method, Qumulo can efficiently manage user identities. This method leverages RFC2307 values, which are integrated into the default schema of Active Directory domains since Windows 2003, making it easy to manage and scale.

The attributes **uidNumber** and **gidNumber** for each user and gidNumber for each group must be populated in Active Directory for successful AD to POSIX user mapping.

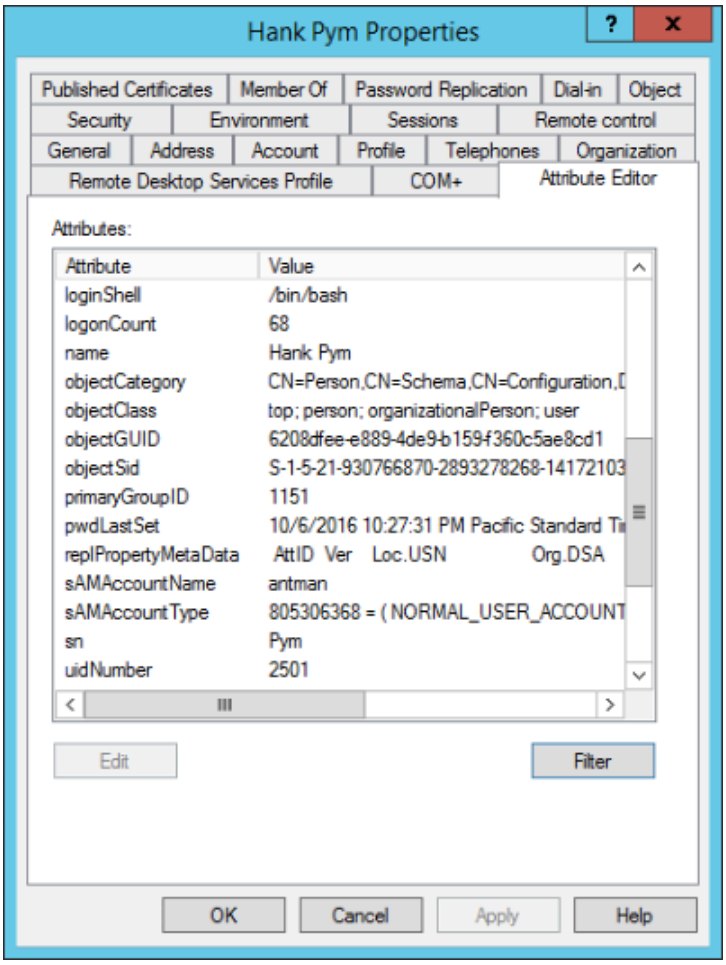

Figure 5: Mapping a local POSIX user to an AD account in Windows

Local User Accounts

Qumulo's XPP framework can manage "orphan" or unassigned POSIX identities by mapping them to local cluster accounts. This feature is particularly useful for integrating users without a corresponding entry in an Active Directory (AD) or other centralized identity management system.

The local user accounts feature can be used alongside Active Directory, regardless of whether POSIX attributes are enabled in AD. This flexibility allows administrators to manage centrally and locally authenticated users within the same environment.

> **Note:** *Local user accounts are unique to each Qumulo instance. This means that user accounts created within one instance will not exist in another instance, even if both instances are part of the same environment. Additionally, local user accounts are not replicated to disaster recovery (DR) instances. This ensures that DR clusters remain clean and free from unnecessary local account clutter. A successful failover strategy in the event of a disaster should rely instead on centralized identity management systems for user authentication and access control.*

LDAP to AD User-Defined Mappings

Qumulo's Cross-Protocol Permissions (XPP) framework includes a feature that allows for the mapping of LDAP "uid"(username) attributes to Active Directory (AD) usernames using a flat-file JSON document. This method is particularly useful in environments where LDAP is used for user management but needs to be integrated with AD for unified identity management.

Flat File JSON Document for Mapping:

The mapping process requires the creation of a flat file in JSON format. This type of document can be used to define the relationship between LDAP "uid" attributes, which represent the usernames in the LDAP directory, and the corresponding user names in Active Directory.

Example Structure:

```
[
  {
    "down_level_logon_name": "DOMAIN\\Alice.Cooper",
    "ldap_name": "acoop"
  },
  {
    "down_level_logon_name": "DOMAIN\\Joan.Jett",
    "ldap_name": "jjett"
  }
]
```

In this example, the LDAP username "acoop" is mapped to the AD user "DOMAIN\Alice.Cooper", and "jjett" is mapped to "DOMAIN\Joan.Jett".

By incorporating LDAP to AD user-defined mappings, Qumulo's XPP framework offers flexibility for environments where LDAP directories need to integrate with AD. This method ensures that all user identities can be managed effectively, even when using different directory services, providing a cohesive and secure permissions management system.

Authentication ID

When a new user or group identity is encountered through an incoming protocol request, it is mapped to a unique internal Authentication ID (Auth ID). These Auth IDs are then written to the metadata of each file or directory object, ensuring that permissions and access controls are correctly applied. Auth IDs are unique to each Qumulo cluster, meaning the same domain user will have different Auth IDs in each cluster within that domain.

```
uidNumber 2001 = AuthID 25769804289
gidNumber 128876 = AuthID 34562241898
SID S-1-5-21-930766870-2893278268-1417210345-513 = Auth ID 25769804886
```

A Qumulo cluster will use the designated User Identity Mapping method, whether it's Active Directory with POSIX attributes, local user accounts, or LDAP to AD user-defined mappings. This method is employed to create a table of Related Auth IDs.

This table includes mappings of POSIX identities to NTFS identities, encompassing all relevant group memberships. By maintaining this comprehensive mapping, Qumulo ensures that permissions and access controls are consistently applied across different protocols.

The related identities table is used to map POSIX identities (common in UNIX/Linux environments) to their corresponding NTFS identities (used in Windows environments). This mapping includes not only the user identities but also all relevant group memberships, ensuring that group-based permissions are accurately enforced. This comprehensive mapping allows Qumulo to enforce consistent access control policies across different protocols, facilitating seamless collaboration in mixed-protocol environments.

Example of Active Directory with POSIX Attributes in Qumulo XPP

Qumulo's XPP framework effectively integrates Active Directory with POSIX attributes to manage cross-protocol identity mappings. This example illustrates how a Qumulo cluster handles a new user identity encountered over NFSv3 and how related identities are expanded and cached.

Scenario: Integrating a New POSIX User Identity

1. **Encountering a New uidNumber over NFSv3:** When a new POSIX user identity, represented by the uidNumber 2001, is detected over NFSv3, the Qumulo cluster must integrate this identity into its cross-protocol permissions framework.

The Qumulo cluster queries Active Directory to determine if the uidNumber 2001 has been assigned to any AD user accounts.

The Active Directory Domain Controller returns the Security Identifier (SID) of the user associated with uidNumber 2001. Additionally, it provides the SIDs of all groups to which this user belongs and any gidNumbers assigned to those groups.

This process ensures that the POSIX user identity (uidNumber) is accurately mapped to the corresponding NTFS identity (SID) within Active Directory.

2. **Expanding Related Identities Using the qq Command:**

The command qq auth_expand_identity is used to look up and expand all related NTFS and POSIX identities that the Qumulo cluster has identified. Performing this lookup manually will add the retrieved values to the local node's Identity Cache. This helps in optimizing subsequent access and reducing latency. This operation, also known as Credential or Identity Expansion, ensures that all relevant user and group identities are accurately reflected in the cluster's identity management system.

```
qq auth_expand_identity --sid S-1-5-21-930766870-2893278268-1417210345-1110
Identity:    QUMULOTEST\spiderman (S-1-5-21-930766870-2893278268-1417210345-1110)
Type:        User
NFS Mapping: uid:2001
SMB Mapping: QUMULOTEST\spiderman (S-1-5-21-930766870-2893278268-1417210345-1110)
Equivalent Identities:
Name  ID
====  ========
      uid:2001
Group Membership:
Name                           ID
=============================  ============================================
                               gid:10000
                               gid:11000
                               gid:16552
                               gid:5139
BUILTIN\Remote Desktop Users   S-1-5-32-555
BUILTIN\Users                  S-1-5-32-545
QUMULOTEST\Domain Users        S-1-5-21-930766870-2893278268-1417210345-513
QUMULOTEST\MoreSecret          S-1-5-21-930766870-2893278268-1417210345-5163
QUMULOTEST\Sales               S-1-5-21-930766870-2893278268-1417210345-1316
QUMULOTEST\nfsusers            S-1-5-21-930766870-2893278268-1417210345-1113
```

The command qq auth_find_identity can be used to look up a POSIX or NTFS identity's internal Auth ID

```
qq auth_find_identity --sid S-1-5-21-930766870-2893278268-1417210345-1110
domain: ACTIVE_DIRECTORY
name: QUMULOTEST\spiderman
auth_id: 25769804886
SID: S-1-5-21-930766870-2893278268-1417210345-1110
```

> **Note:** *If a cluster loses connectivity to the Directory Service it has been bound to then the cluster will stop any new requests for service across all protocols. Some services will continue to work for users whose attributes are already in the serving node's local cache until that cache expires.*

Storing Permissions

Internal QACL Format

This section explains how Qumulo's Cross-Protocol Permissions (XPP) framework synthesizes Qumulo ACL entries from POSIX and NTFS permissions, and how they're presented to users via standard permissions display tools available on client machines.

QACLs are synthesized or translated into a format that these client-side tools can understand and display. This involves converting the internal QACL representation into the appropriate POSIX mode bits for UNIX/Linux systems, or the standard NTFS permissions format for Windows systems. Common tools include the ls command on UNIX/Linux systems and the **Windows Security** dialog on Windows systems.

Example of Permissions Display:

UNIX/Linux (Using ls): On a UNIX/Linux system, a synthesized QACL would be displayed using the **ls -l command**, which shows file permissions in a human-readable format. The synthesis process converts QACLs into the familiar **rwx** (read, write, execute) format used by POSIX.

Windows (Using Security Dialog): On a Windows system, the synthesized QACLs are presented through the Security dialog, accessible via file properties. This dialog displays the detailed NTFS permissions, including access control entries (ACEs) and inheritance settings.

The command **qq fs_get_acl** can be used to look up an object's internal QACL.

```
qq fs_get_acl --path /splunk
Control: Present
Posix Special Permissions: None

Permissions:
Position  Trustee             Type     Flags  Rights
========  ==================  =======  =====
================================================
1         local:admin         Allowed         Delete child, Execute/Traverse, Read,
Write file
2         local:System Group  Allowed         Execute/Traverse, Read
3         Everyone            Allowed         Execute/Traverse, Read
```

On Linux:

On Linux, the **ls -l** command provides a simplified view of the permissions, generally showing the most permissive settings to reflect the broad allowances in the QACL.

```
ls -ld /splunk
drwxrwxrwx 34 root nogroup 20992 Mar 26 12:52 /splunk
```

The NFSv3 toolset cannot display the complexity of NTFS ACLs. Therefore, Qumulo XPP defaults to displaying the most permissive mode bits (777) when a non-representable user has elevated rights to an object. An unrepresentable user is any extra user or group that cannot be displayed in the NFSv3 "User, Group Owner, Others" mode bit sets.

NTFS ACLs can be viewed and edited by NFSv4.1 clients with the appropriate access level.

On Windows:

On a Windows system, the ACLs can be viewed through the Security tab in the file properties dialog. The detailed permissions will reflect the entries shown in the QACL.

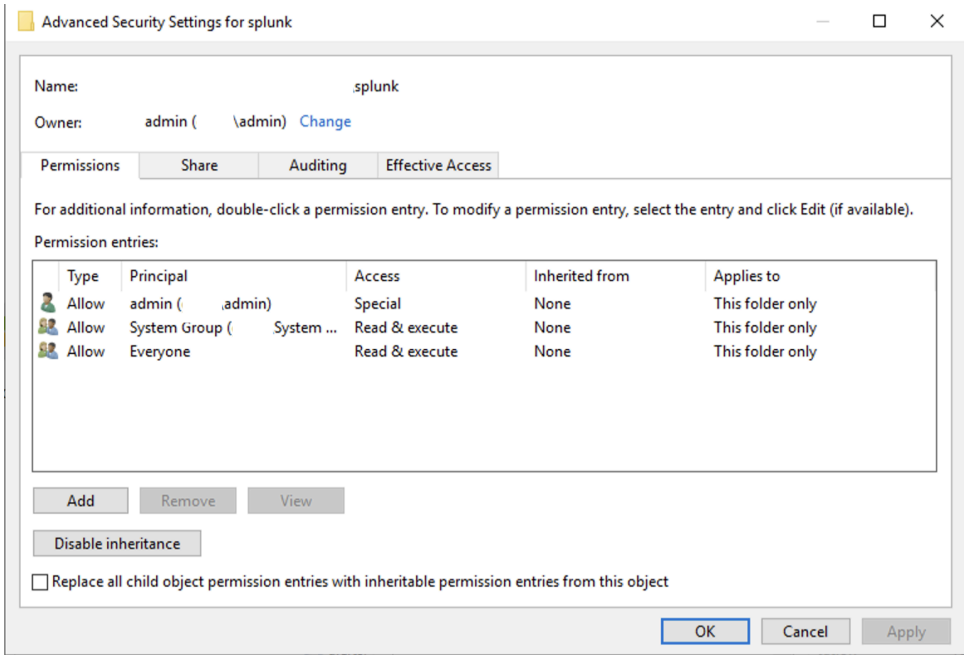

Figure 6: Viewing access-control entries in Windows

Some attributes, such as the object owner and primary group owner are not stored in the QACL, but are stored instead in the object's attributes. The command **qq fs_file_get_attr** may be used to retrieve those stored attributes.

```
qq fs_file_get_attr --path /splunk
{
    "access_time": "2023-02-17T22:52:43.105972129Z",
    "blocks": "1",
    "change_time": "2023-05-19T00:29:48.434017382Z",
    "child_count": 1,
    "creation_time": "2023-02-17T22:52:43.105972129Z",
    "data_revision": null,
    "datablocks": "0",
    "directory_entry_hash_policy": "FS_DIRECTORY_HASH_VERSION_FOLDED",
    "extended_attributes": {
        "archive": false,
        "compressed": false,
        "hidden": false,
        "not_content_indexed": false,
        "offline": false,
        "read_only": false,
        "sparse_file": false,
        "system": false,
        "temporary": false
    },
    "file_number": "1206000007",
    "group": "2",
    "group_details": {
        "id_type": "LOCAL_GROUP",
        "id_value": "System Group"
    },
    "id": "1206000007",
    "major_minor_numbers": {
        "major": 0,
        "minor": 0
    },
    "metablocks": "1",
    "mode": "0755",
    "modification_time": "2023-05-19T00:29:48.434017382Z",
    "name": "splunk",
    "num_links": 3,
    "owner": "500",
    "owner_details": {
        "id_type": "LOCAL_USER",
        "id_value": "admin"
    },
    "path": "/splunk/",
    "size": "512",
    "symlink_target_type": "FS_FILE_TYPE_UNKNOWN",
    "type": "FS_FILE_TYPE_DIRECTORY",
    "user_metadata_revision": "0"
}
```

XPP Logic and Special Case Handling

The XPP logic is designed to handle different protocols' unique characteristics and requirements, providing a versatile and comprehensive approach to managing cross-protocol permissions. This includes applying the appropriate tools and methods for each protocol to ensure consistent access control and permissions enforcement across diverse storage environments. The XPP logic seamlessly integrates with multiple protocols, such as SMB, NFS, and FTP, to facilitate a unified and coherent permissions model.

Special cases and scenarios, such as handling POSIX rights within an NTFS-based system, are also managed effectively to maintain security and functionality. The XPP logic can translate and map permissions between different protocols, ensuring that users have the correct access rights regardless of the underlying file system. This capability is crucial for environments where mixed-protocol access is common, as it prevents permission conflicts and ensures that security policies are uniformly applied.

Moreover, XPP logic incorporates advanced mechanisms for handling complex permission inheritance and propagation, ensuring that changes in permissions are consistently and accurately reflected across all protocols. It can dynamically adapt to changes in the environment, such as updates in directory services or modifications in user roles, to provide real-time permissions management.

In addition, the system includes robust auditing and logging features to track and record access and modifications across all protocols. This enhances security by providing visibility into user activities and helping to identify potential security breaches or misconfigurations. These logs can be integrated with external security information and event management (SIEM) systems for comprehensive monitoring and analysis.

By addressing the intricacies of cross-protocol permissions and incorporating special case handling, the XPP logic ensures a secure, reliable, and seamless access control experience. It enables organizations to leverage a unified storage solution that supports multiple protocols while maintaining rigorous security standards and operational efficiency.

Common Scenarios

Mode Bit Display

When users access Qumulo over NFS and want to see mode bits for a file, Qumulo generates these from the QACL. This can be straightforward when the QACL aligns with POSIX users/groups, but complications arise with additional trustees. Qumulo addresses this by incorporating an ID equivalence check to determine if an ACE is redundant or necessary, thereby ensuring accurate mode bit display without significant performance impacts.

For instance, consider the following QACL:

```
ALLOW file owner          read
ALLOW file group owner    read
ALLOW Everyone            read
ALLOW alice               read, execute, write file
```

Handling the additional ACE (Access Control Entry) for “alice” requires determining if “alice” is the file owner or belongs to the file’s group. This process, which involves recursive group membership checks and identity source queries, can be resource-intensive and slow down the common operation of displaying POSIX mode.

Qumulo uses ID equivalence checks to determine if an extra ACE is actually redundant. If not, the extra trustee’s permissions are folded into the Everyone bit. Assuming “alice” is not the file owner, the POSIX mode would be displayed as 447, including “alice’s” permissions in the Everyone bit.

This method ensures that users are aware that someone, if not everyone, has the permissions denoted. For instance, showing a mode of 444 might incorrectly suggest no one has write or execute permissions, ignoring “alice’s” rights and creating a potential security risk.

Changing POSIX Mode

A **chmod** operation over NFS can alter permissions in ways that might conflict with existing ACLs. Qumulo’s algorithm manipulates the QACL to reflect the requested POSIX mode while preserving the original ACL's inheritance structure. This ensures that changes made via one protocol do not inadvertently disrupt permissions required by another.

For example, consider a QACL before a **chmod 555** operation:

```
DENY   alice              read, take ownership (Object inherit)
DENY   charlie                  execute/traverse
ALLOW charlie             read, write file
ALLOW charlie’s group     read
ALLOW Everyone            read contents
ALLOW bob                 read, write (Inherit-only)
```

After applying **chmod 555**, the QACL is adjusted to:

```
DENY   alice              take ownership
DENY   alice              read, take ownership (Object inherit, inherit-only)
ALLOW  charlie            read, execute/traverse
ALLOW charlie’s group     read, execute/traverse
ALLOW Everyone            read, execute/traverse
ALLOW bob                 read, write (Inherit-only)
```

Changes include:

- The previous DENY entry for “charlie” (the file owner) was removed because it conflicted with the chmod.
- Two ACEs for “alice”: One preserves the non-POSIX right (take ownership), and the other, marked inherit-only, ensures the ACE is passed to children without affecting the current file.
- “bob’s” inherit-only ACE remains unchanged.

Explain Permissions Tools

Explain Permissions Tools is a suite of utilities which can examine a given file or directory, and break down how permissions sets were derived. The tool performs an annotated ACE-by-ACE ACL evaluation and mode evaluation.

fs_acl_explain_posix_mode

This command explains how Qumulo produced the displayed POSIX mode for a file or directory. The command breaks down the ACL, annotating each entry and showing how it contributes to the final POSIX mode assigned.

```
qq fs_acl_explain_posix_mode --path /splunk
      Permissions Mode: Cross protocol
      Owner: admin
      Group: System Group

      ==== Current ACL ====
      Control: Present
      Posix Special Permissions: None

      Permissions:
      Position  Trustee             Type     Flags  Rights
      ========  ==================  =======  =====
      ================================================
      1         local:admin         Allowed         Delete child, Execute/Traverse,
      Read, Write file
      2         local:System Group  Allowed         Execute/Traverse, Read
      3         Everyone            Allowed         Execute/Traverse, Read

      Mode derivation from ACL for "/splunk/":

      ==== 1 ====
      Allowed      local:admin  Delete child, Execute/Traverse, Read, Write file
          Matched: OWNER
          Cumulative rights allowed:
```

```
            Owner: Delete child, Execute/Traverse, Read, Synchronize, Write file
            Group: None
            Other: None
        Mode Bit Contribution: rwx------

    ==== 2 ====
    Allowed      local:System Group  Execute/Traverse, Read
        Matched: GROUP
        Potentially affects rights for: OWNER
        Cumulative rights allowed:
            Owner: Delete child, Execute/Traverse, Read, Synchronize, Write file
            Group: Execute/Traverse, Read, Synchronize
            Other: None
        Mode Bit Contribution: r-xr-x---

    ==== 3 ====
    Allowed      Everyone     Execute/Traverse, Read
        Matched: EVERYONE
        Cumulative rights allowed:
            Owner: Delete child, Execute/Traverse, Read, Synchronize, Write file
            Group: Execute/Traverse, Read, Synchronize
            Other: Execute/Traverse, Read, Synchronize
        Mode Bit Contribution: r-xr-xr-x

    Final Derived Mode: drwxr-xr-x
```

fs_acl_explain_chmod

This command runs a 'what-if' simulation of the effects of a chmod on the ACL of a given file. The command takes the desired POSIX mode, then produces a step-by-step explanation of how the file's ACL would be affected by the chmod. This command provides a preview, but does not actually change the permissions of the file in question.

```
qq fs_acl_explain_chmod --path /splunk --mode 0744
    Permissions Mode: Cross protocol
    Owner: admin
    Group: System Group

    ==== Current ACL ====
    Control: Present
    Posix Special Permissions: None

    Permissions:
    Position  Trustee             Type     Flags  Rights
    ========  ==================  =======  =====
    ================================================
    1         local:admin         Allowed         Delete child, Execute/Traverse,
    Read, Write file
```

```
2         local:System Group  Allowed         Execute/Traverse, Read
3         Everyone            Allowed         Execute/Traverse, Read

Mode 0744 translates to rights:
    owner: Delete child, Execute/Traverse, Read, Synchronize, Write file
    group: Read, Synchronize
    other: Read, Synchronize

Steps for applying mode 0744 to original permissions:
==== 1 ====
Action:             Insert entry
New entry:          Allowed      local:admin  Delete child, Execute/Traverse,
Read, Write file
Reason:             Add rights granted by requested mode.

==== 2 ====
Action:             Insert entry
New entry:          Allowed      local:System Group  Read
Reason:             Add rights granted by requested mode.

==== 3 ====
Action:             Insert entry
New entry:          Allowed      Everyone     Read
Reason:             Add rights granted by requested mode.

==== 4 ====
Action:             Remove entry
Source entry:       Allowed      local:admin  Delete child, Execute/Traverse,
Read, Write file
Trustee match:      'local:admin' matches POSIX owner
Reason:             Remove old POSIX trustee ACE to replace with rights from
the requested mode.

==== 5 ====
Action:             Remove entry
Source entry:       Allowed      local:System Group  Execute/Traverse, Read
Trustee match:      'local:System Group' matches POSIX group owner
Reason:             Remove old POSIX trustee ACE to replace with rights from
the requested mode.

==== 6 ====
Action:             Remove entry
Source entry:       Allowed      Everyone     Execute/Traverse, Read
Trustee match:      'Everyone' matches POSIX others
Reason:             Remove old POSIX trustee ACE to replace with rights from
the requested mode.

==== Resulting ACL ====
```

```
Control: Present
Posix Special Permissions: None

Permissions:
Position  Trustee             Type     Flags  Rights
========  ==================  =======  =====
================================================
1         local:admin         Allowed         Delete child, Execute/Traverse,
Read, Write file
2         local:System Group  Allowed         Read
3         Everyone            Allowed         Read
```

fs_acl_explain_rights

This command explains the rights a given user has to a specified file. The command takes a file ID or path and a user/group identifier, then breaks down the ACL, explaining how each ACE affects the rights of the user/group in question.

qq fs_acl_explain_rights --path /splunk -u auth_id:500

```
admin has 1 equivalent IDs and is a member of 2 groups.
File Owner: admin
File Group Owner: System Group

ACL evaluation steps for 'admin':
==== 1 ====
    Entry:           Allowed 'local:admin' Delete child, Execute/Traverse,
Read, Write file
    Trustee Matches: True
    Allowed so far:  Delete child, Execute/Traverse, Read, Synchronize, Write
file
==== 2 ====
    Entry:           Allowed 'local:System Group' Execute/Traverse, Read
    Trustee Matches: False
==== 3 ====
    Entry:           Allowed 'Everyone' Execute/Traverse, Read
    Trustee Matches: True
    Allowed so far:  Delete child, Execute/Traverse, Read, Synchronize, Write
file

Implicit Rights for 'admin':
    Administrator rights:    Delete, Delete child, Execute/Traverse, Read,
Synchronize, Take ownership, Write ACL, Write file
    File Owner rights:       Read, Synchronize, Write ACL, Write EA, Write
attr, Write group
    Parent Directory rights: Delete, Read attr

Rights that would be granted to 'admin':
```

```
Read contents     (Read file data or list directory)
Read EA           (Read extended attributes)
Read attr         (Read attributes)
Read ACL          (Read access control list)
Write EA          (Write extended attributes)
Write attr        (Write attributes)
Write ACL         (Write access control list)
Change owner      (Change file owner)
Write group       (Change file group-owner)
Delete            (Delete this object)
Execute/Traverse (Execute file or traverse directory)
Write data        (Modify file data)
Extend file       (Append to file)
Delete child      (Delete any of a directory's immediate children)
Synchronize       (Meaningless, exists for compatibility)
```

Case Studies and Examples

This next section describes some real-world scenarios in which Qumulo customers have used XPP to ensure that data shared over both SMB and NFS is appropriately secured and protected without interfering with workflows from either client base.

Case Study #1: Multi-Protocol Access in a Media Production Environment

A media production company uses Qumulo to manage files accessed by both NFS and SMB protocols. This dual-protocol environment is critical for supporting the diverse workflows within the company. Editors on Windows require granular ACLs for detailed control over file access, enabling them to specify precise permissions for different users and groups. This fine-grained control ensures that sensitive media files are securely managed, with specific team members granted varying levels of access based on their roles and responsibilities.

On the other hand, the company's render farms, which run on Linux, need efficient POSIX permissions to process large volumes of media files swiftly. POSIX permissions are simpler and faster to process, which is essential for the high-performance computing demands of rendering tasks. The render farms require consistent and reliable access to the files, without the complexity of managing ACLs, to maintain high throughput and minimize rendering times.

Qumulo's XPP framework provides a seamless solution for this mixed environment, allowing for the smooth access and modification of files across both Windows and Linux systems. XPP ensures that permission changes made by one team do not disrupt the workflow of another.

For example, an editor on a Windows machine might set an ACL to grant read/write permissions to specific team members, ensuring that only authorized personnel can modify the media files. When a Linux-based render farm accesses the same files, Qumulo's XPP framework translates these ACLs into the appropriate POSIX mode bits. This translation ensures that the render farm can read the files without altering the original ACL settings.

This process maintains the integrity and security of the file permissions, allowing editors and render farms to work concurrently without interference. Editors can continue to manage detailed permissions using ACLs, while the render farms operate efficiently with POSIX permissions. The ability of Qumulo's XPP framework to harmonize these two different permission models is crucial for the company's productivity, ensuring that all teams can access the resources they need without compromising security or performance.

Additionally, Qumulo's real-time analytics provide visibility into file usage and access patterns, enabling administrators to monitor and optimize performance across both protocols. This insight helps in fine-tuning the system for better efficiency and in identifying any potential issues before they impact the workflow.

Case Study #2: Research Data Management in Higher Education

A university research department stores large datasets on Qumulo, accessed by researchers using various operating systems, including Windows, Linux, and macOS. The need for secure cross-protocol access is critical to maintaining data integrity and to ensure compliance with stringent data protection regulations. These datasets often contain sensitive information, such as personal data from research participants or proprietary experimental results, making it essential to have robust access controls in place.

With Qumulo's XPP framework, the department can enforce consistent permission policies across all user groups, regardless of the access protocol they use. This unified approach to permissions management ensures that all data is protected consistently, simplifying administrative tasks and enhancing security.

In practice, this means a researcher accessing data via NFS on a Linux machine can set POSIX permissions to control access to their files. The XPP framework then automatically translates these POSIX permissions into ACLs for Windows-based collaborators. This translation process is seamless and ensures that the permissions are enforced uniformly, regardless of whether a file is accessed via NFS, SMB, or another protocol.

For example, a researcher might set POSIX permissions to grant read-only access to a dataset for a specific group of users while retaining full control over the data themselves. When a collaborator using a Windows system accesses the same dataset, the XPP framework ensures that these permissions are

respected and enforced through corresponding ACLs. This mechanism prevents unauthorized access and modifications, safeguarding the integrity of the research data.

The consistent enforcement of permission policies across different operating systems and access methods ensures that sensitive data remains protected and accessible only to authorized users. Researchers can collaborate effectively without compromising data security, as Qumulo's XPP framework handles the complexities of cross-protocol permission management.

Case Study #3: Healthcare Data Compliance

A healthcare provider must comply with strict data protection regulations such as HIPAA, which mandate rigorous security controls to protect patient information. Qumulo's XPP framework ensures that medical records are securely accessed by both administrative staff on Windows systems and medical personnel on Linux-based devices. This mixed-environment access is crucial for the efficient operation of healthcare facilities, where different departments rely on various operating systems to perform their duties.

Qumulo's XPP framework is designed to enforce consistent and accurate permissions across different protocols, ensuring that access to sensitive data is tightly controlled and compliant with regulatory standards. One of the key features of XPP is its ability to perform ID equivalence checks and QACL mappings. These functionalities ensure that permissions set in one environment are correctly translated and enforced in another, maintaining a unified security posture.

For example, administrative staff on Windows systems may need access to patient records for billing and scheduling purposes. They rely on detailed ACLs to manage who can view or modify specific pieces of information. Simultaneously, medical personnel using Linux-based devices require access to these records to update patient information, enter medical notes, and review treatment plans. The POSIX permissions they use must align with the ACLs set by the administrative staff to prevent any discrepancies that could lead to unauthorized access.

The XPP framework ensures that when permissions are set by administrative staff using ACLs, these are accurately mapped to the corresponding POSIX permissions for medical personnel. This mapping process includes ID equivalence checks to ensure that user identities are consistently recognized across different systems, preventing access issues that could arise from mismatched permissions.

For instance, if an administrator sets an ACL to grant read/write access to a medical team, Qumulo translates this into the appropriate POSIX mode bits so that the same level of access is granted when medical personnel access the files via Linux-based devices. This seamless translation ensures that the security policies are uniformly applied, no matter which protocol or operating system is used to access the data.

Qumulo's XPP framework ensures that the healthcare provider can confidently manage access to medical records across their diverse IT environment, ensuring that all staff members have the necessary access to perform their duties while maintaining the highest standards of data protection. This approach not only safeguards patient information but also supports the provider's compliance with HIPAA and other regulatory requirements.

Additional Examples

Example #1

An Administrator needs to ensure that NTFS ACLs are inherited across an entire directory tree, while simultaneously maintaining the correct 700 permissions for a POSIX user's .ssh directory within this directory.

XPP uses the concept of "Generation Skipping" for inherited ACLs. This allows a user to perform a POSIX SETATTR command while ensuring that ACL inheritance continues for child objects of the recently modified directory.

```
/Home (Inherited ACL)
/User_Directory (POSIX 700 mode + Future Inheritance in QACL)
    /New_Directory (Inherited ACL)
```

This approach allows individual user modifications via POSIX tools without disrupting ACL inheritance across large directories, satisfying both administrative and user requirements.

Example #2

POSIX users inherently have the right to read the attributes of any object within a container they can access, while NTFS permissions can control this right.

Qumulo ensures that POSIX users retain this implicit right, preventing the display of "????" when performing an "ls -l" on a directory over NFS.

Example #3

Typically, POSIX object owners can always change the permission mode bits of their objects.

Qumulo XPP allows Administrators to apply NTFS "Owner Rights" management to POSIX users, preventing NFS file owners from using chmod on their own files.

This feature, greatly valued by Administrators, is unique to Qumulo and not possible in most other file systems. When Owner Rights are managed via NTFS ACLs, Qumulo provides a "False Positive" to POSIX tools attempting a SETATTR command on objects. This ensures that POSIX tools do not fail unexpectedly, which is crucial for tools like **rsync**, **cp**, and **vi**.

Example #4

XPP supports the use of POSIX-only tools, such as SetGID, in SMB operations.

SMB clients will respect SetGID and assign the correct Group Owner to any new objects in the SetGID-managed directory. A default set of permissions, similar to the POSIX umask setting, can be specified for each SMB Share. These default permissions are used only in the absence of any Inheritable ACLs.

The default mode can be set via the Qumulo Web UI or the **qq smb_mod_share** command.

Conclusion

Managing cross-protocol permissions and authentication in today's multi-protocol environments is a complex but essential task. Organizations often struggle with the inherent challenges and inconsistencies that arise from using different permission models across various systems and protocols. With the XPP framework, Qumulo provides a robust and unified solution that ensures seamless, secure data access across multiple platforms such as Windows, Linux, and macOS.

Qumulo's XPP framework addresses the critical need for consistent and accurate permission enforcement in mixed-protocol environments. By translating and mapping permissions between POSIX and ACLs, XPP ensures that access controls are uniformly applied, preventing unauthorized access and maintaining data integrity. This approach is crucial for industries that handle sensitive data, such as healthcare, media production, and higher education, where regulatory compliance and data protection are paramount.

One of the standout features of Qumulo's XPP is its ability to perform ID equivalence checks and Qumulo Access Control List (QACL) mappings. These features ensure that permissions set in one environment are correctly translated and enforced in another, maintaining a unified security posture. For instance, administrative staff on Windows systems can set detailed ACLs for managing patient records, while medical personnel on Linux-based devices can access the same records with appropriately translated POSIX permissions. This seamless translation process ensures that all security policies are consistently enforced, regardless of the operating system or protocol used.

Furthermore, Qumulo's XPP framework includes advanced mechanisms for handling complex permission inheritance and propagation. Changes in permissions are consistently and accurately reflected across all protocols, ensuring that updates in directory services or modifications in user roles are dynamically managed. This real-time permissions management capability is essential for maintaining operational efficiency and security in dynamic environments.

Contributors

This article is maintained by Qumulo. It was originally written by the following contributors.

Principal authors:

James Walkenhorst | Senior Technical Marketing Manager

Joe Costa | Solutions Architect at Qumulo

Berat G. Ulualan | Solutions Architect at Qumulo

Related resources

Managing Multi-Protocol File Data Access Workflows with Cross-Protocol Permissions

Cross-Protocol Permissions (XPP) in Common Scenarios

Authentication in Qumulo Core