Permissions Mapping in the Isilon OneFS File System

NTFS ACLs, NFSv4 ACLs, and POSIX Mode Bits

Steven Danneman and Zack Kirsch

Agenda

- What is OneFS?
- POSIX, NTFS and NFSv4 Permission Overview
- Isilon's Permission Implementation
  - Setting
  - Retrieval
  - Enforcement
- Advanced Permission Implementation
  - Special Identities
  - Inheritance
  - Canonical Order

Isilon OneFS Cluster

- NAS file server
- Scalable
  - Add more storage in 5 mins
- Reliable
  - 8x mirror / +4 parity
  - Striped across nodes
- Single volume file system (5.2 PB)
- 3 to 144 nodes
- Fully symmetric peers
  - No metadata servers
- Commodity hardware
  - CPU, Mem, Disks (12 to 36)

Isilon OneFS File System

- Concurrent access to all files with all protocols
  - CIFS/SMB
  - NFSv3
  - SSH
  - HTTP/FTP
- Coming Soon
  - NFSv4
  - SMB2

Permission Basics

Unix Permissions

- Mode bits
  - rwxrwxrwx
  - Read / Write / Execute
  - Owner / Group / Other
- POSIX ACLs
  - Give rwx permission to other users & groups
  - Closer to NTFS ACLs, but less expressive
  - Replaced in OneFS by NTFS ACLs

NTFS Access Control List

- Approximately 15 rights vs 3 rwx rights.
- Security Descriptor (SD)
  - Owner, Group
  - Discretionary ACL (ACL)
    - List of Access Control Entries (ACE)
  - System ACL
- ACE
  - User / Group Identifier (UID/GID in OneFS)
  - Allow & Deny
  - List of rights
  - Inheritance

Comparison

- POSIX modes are a complete subset of NTFS rights
  - Minus the top 3 bits
    - SetUID, SetGID, Sticky
- Order of enforcement is different
  - POSIX
    1. Determine identity
    2. Check 1 of 3 possible lists
  - NTFS
    1. Determine identity
    2. Check 1 list

Permission Modification

- POSIX semantics:
  - chmod: Only owner/root
  - chown: Only root
  - chgrp: Only owner/root, only to groups they are part of
- NTFS semantics:
  - chmod: Needs WRITE_DAC; owner can always change permissions
  - chown: Needs WRITE_OWNER; cannot give away a file
  - chgrp: Needs WRITE_OWNER; can change to any group
- OneFS: Global Policy dictates behavior regardless of protocol

NFSv4 Access Control List

- Small Differences
  - Uses principals instead of IDs, e.g. “user@domain”
  - uid/gid allowed for backwards compatibility
- New Rights
  - ACE4_WRITE_RETENTION / ACE4_WRITE_RETENTION_HOLD
  - Mappable to ACE4_WRITE_ATTRIBUTES
- Mostly Identical to NTFS ACL

Isilon Implementation

Design Goals

- Store one authoritative set of permissions per file
  - Preference NTFS ACL over mode bits
- Enforce identical permissions for all protocols
- Provide view of alternate permission type:
  - NFS is returned approximated mode bits
  - SMB is returned a SYNTHETIC ACL
- Provide configuration through global permission policy
- Extend standard Unix tools for all permission management
  - ls, chmod, chown, chgrp

SMB File Creation

- Store ACL
  1) SD sent with create : Store provided ACL
  2) Inheritable ACL exists on parent : Store Inherited ACL
  3) No Inheritable ACL exists : Store Default ACL
- Store approximated mode bits
  - Give NFS clients a view of the permissions
  - Stored mode bits are not used for enforcement
  - Permissive enough to trick client access evaluation

NFS File Creation

- No inheritable ACL exists
  - Store mode bits only
- Inheritable ACL exists on parent
  - Apply inheritable ACL only
  - Store approximated mode bits

Permissions Setting

- chmod w/ ACL (SMB or local)
  - Store ACL
  - Store approximated mode bits
- chmod w/ mode bits (NFS or local)
  - No ACL exists
    - Store mode bits
  - ACL exists
    - Merge mode bits with ACL
      - Add/modify ACEs for three identities: owner, group, everyone
      - Leave other identities unchanged
      - Add deny ACEs for bits that are not present
      - Inheritance hierarchy remains

Permissions Retrieval

- SMB
  - If ACL, ACL is returned
  - If mode bits, return SYNTHETIC ACL
    - Not stored on disk, translated on demand
- NFS
  - Always show stored mode bits

Basic Permission Enforcement

- Goal: Enforce the same access on all files, from all protocols.

- SMB access on file with ACL
  - Scan through ACL, until desired rights are allowed or denied
- NFS access on file with mode bits
  - Simple comparison against owner, group or other

- Algorithm:
  1. Convert desired rights / access mask to file's permission type
  2. Basic permission enforcement

Advanced Enforcement I

- SMB access on file with mode bits
  - Convert desired rights to Unix permissions
    - *List Folder* -> Unix READ
    - *Create Files* or *Create Folders* or *Delete Subfolders/Files* -> Unix WRITE
    - *Traverse Folder* -> Unix EXECUTE
  - *Change Permissions*, *Take Ownership* and *Delete* do not map
    - ACL Policy: rwx = Full Control

Advanced Enforcement II

- NFS/Local access on file with ACL
  - Convert desired access mask to ACL rights
    - Unix READ -> *List Folder*
    - Unix WRITE -> *Create Files* AND *Create Folders* AND *Delete Subfolders/Files*
    - Unix EXECUTE -> *Traverse Folder*
  - NFS Server uses Windows rights
    - E.g. Asks for *Create Files* access instead of WRITE access
  - NFS Access Request needs approximation
    - Unix WRITE -> *Create Files* OR *Create Folders* OR *Delete Subfolders/Files*

Advanced Implementation

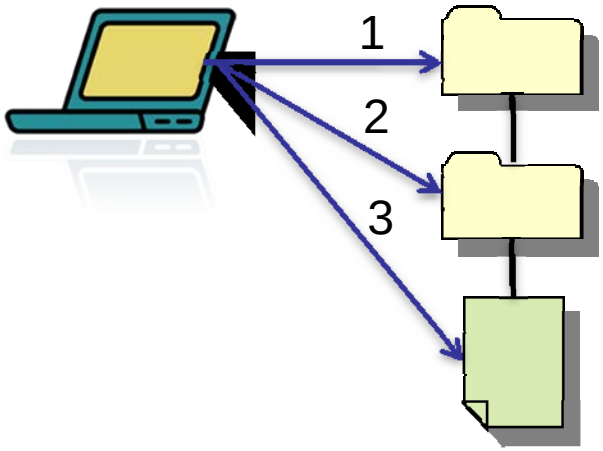
SMB Special Identities

- Changed UID/GID to struct identity
  - Type / ID
- Everyone
- Null
  - Used only for owner or group
- Group owner
  - Used only for owner attribute
- CREATOR OWNER / CREATOR GROUP
  - Inherit_only ACE on directory

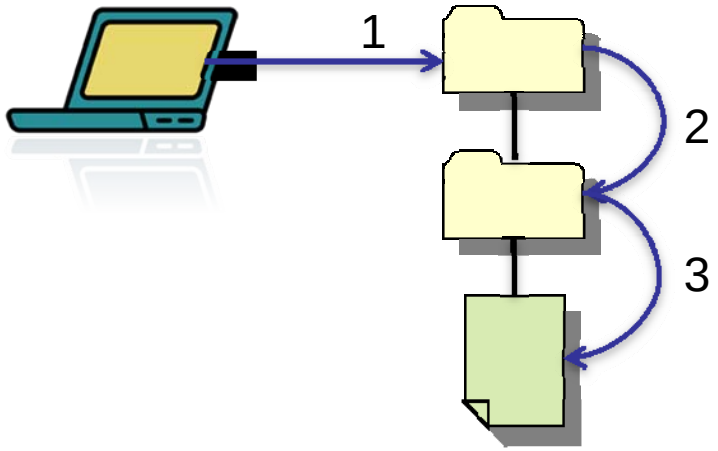
Inheritance

- Auto Inheritance vs. Dynamic inheritance
  - Auto - provide client with info to propagate ACLs
  - Dynamic - file system handles ACL propagation
    - Necessary for local inheritance propagation

Auto Inheritance



Dynamic Inheritance



Canonical Order

- Canonical order:
  - Explicit Deny
  - Explicit Allow
  - Inherited Deny
  - Inherited Allow
  - Enforced by Windows GUI
    - Moves deny ACEs up to the top
  - Windows API allows setting ACEs in any order

- Problem: Out of order ACLs are necessary to represent POSIX ACLs
  - r-- : Allow read, deny write, deny execute

Canonical Order - Example I

- Mode 754 with deny ACEs
- Simplified output:

```
# chmod 754 file.txt
# ls -le file.txt
-rwxr-xr-- 1 test-user  test-group  0 Sep  1 02:04 file.txt
 SYNTHETIC ACL
 0: user:test-user allow full_control
 1: group:test-group allow read, execute
 2: group:test-group deny write
 3: everyone allow read
 4: everyone deny write, execute
```

Canonical Order - Example 2

- After adding "execute" rights for Everyone via Windows GUI:
- Mode changed from 754 to 555, instead of 755

```
# ls -le file.txt
-r-xr-xr-x 1 test-user  test-group  0 Sep  1 02:04 file.txt
 0: group:test-group deny write
 1: everyone deny write
 2: user:test-user allow full_control
 3: group:test-group allow read, execute
 4: everyone allow read, execute
```

Canonical Order - Example 3

- Mode 754 without deny ACEs
- Simplified output:

```
# chmod 754 file.txt
# ls -le file.txt
-rwxr-xr-- 1 test-user  test-group  0 Sep  1 02:04 file.txt
 SYNTHETIC ACL
 0: user:test-user allow full_control
 1: group:test-group allow generic_read, generic_execute
 2: everyone allow generic_read
```

- Configurable ACL policies for dealing with deny ACEs

Configurable Permission Policies

- Disallow ACL creation
- Disallow chmod from NFS

- Chown: Modify the owner/group permissions?

- Owning group on file creation
  - BSD -> parent folder's owning group
  - Windows/Linux -> user's primary GID

Lessons Learned

- Mixed permissions are challenging, but possible
- Some decisions must be left up to policy
- Best practice is to choose a default and document
- Call to Arms: ACL Interop Spec

Questions?

- Zack Kirsch
  - zack.kirsch@isilon.com
- Steven Danneman
  - steven.danneman@isilon.com