Implementing an Advanced Access Control Security Model on Linux

Aneesh Kumar K.V
<aneesh.kumar@linux.vnet.ibm.com>
Andreas Grünbacher
<agruen@suse.de>
Greg Banks
<gnb@fmeh.org>

Linux Symposium
July 2010, Ottawa

Overview

- **Introduction**
- Permission Models
- RichACLs
- Experiences
- Future Work
- End Matter

Authors

- Andreas Grünbacher, SUSE Labs
  - Theoretician and driving force since 2006
- Aneesh Kumar, K.V., IBM India
  - Working on RichACLs since 2009
- Greg Banks
  - Part of a team that worked on predecessor to RichACLs at SGI 2008-2009
  - These are not my employer's opinions

Problem Statement

- Most enterprises have heterogeneous environments
  - Windows clients entrenched
  - ActiveDirectory widely used
  - Filesharing widely used
- Linux would gain by fitting more easily into these environments as a fileserver.
- But! ...

Permission Model Mismatch

- UNIX-like systems support POSIX
  - Traditional model (POSIX.1)
  - Most also support POSIX Access Control Lists (POSIX.1e)
- Windows has a completely different model
- Two ugly choices:
  - Do mapping in the NAS server when a client asks to get or set an ACL
  - Do all permission checks in the NAS server

Mapping Permission Models

- Some features of the Windows model are not expressible in the POSIX models
- Some *almost* are
- Mapping is difficult, not obvious, and sometimes lossy
- Messy user-visible artifacts
  - Permission leakage (security issue)
  - Hysteresis

Goal!

Make Linux a better fileserver OS by storing and enforcing Windows-like ACLs *natively*.

Benefits

- Eliminate messy mapping artifacts
- Simplify filesharing server, client software
- Avoid security holes by enforcing permissions consistently for
  - Apps on CIFS clients
  - Apps on NFS clients
  - Apps running locally
- Fix issues in Linux NFSv4

Overall Approach (1)

- We call our approach **RichACLs**
- Start with NFSv4 ACLs
  - Standardised by IETF: vendor neutral
  - Easy to map to Windows ACLs
  - Widely adopted by modern fs:
    - JFS2, GPFS2, ZFS

Overall Approach (2)

- Tweak design for
  - POSIX compliance
  - Local storage focus
  - Practicality of Linux implementation
- Define mappings
  - to/from NFSv4 ACLs
  - from POSIX ACLs

Overall Approach (3)

- Provide kernel & userspace infrastructure to get, set & store ACLs
- Implement ACL get/set mapping in CIFS & NFS client & server
- Use xattrs to store ACLs
- Implement enforcement in local filesystems
  - Today: ext4
  - Previously: ext3, xfs

Overview

- Introduction
- **Permission Models**
- RichACLs
- Experiences
- Future Work
- End Matter

Permission Models

- Four existing permission models:
  - Traditional UNIX model
  - POSIX.1e ACLs
  - Windows ACLs
  - NFSv4 ACLs

Traditional UNIX Model (1)

- Standardised in POSIX.1
- Users, groups identified by integer IDs
  - Today typically 32b
  - Scoped to the host
    - Mapping issues
  - Separate namespaces for uids, gids
- Each file has
  - Owning uid
  - Owning gid

Traditional UNIX Model (2)

- Each process has
  - Effective uid
  - Effective gid
  - Supplementary gids
- Processes are classified into classes by comparing their uids/gids with the file's
  - *User:* uids match, or
  - *Group*: gids match, or
  - *Other*: otherwise

Traditional UNIX Model (3)

- Three permission bits
  - r: read a file, list a directory
  - w: write a file, create an object in a directory
  - x: execute a file, traverse a directory
- Each of 3 classes has a fixed 3b permission bit mask = 9b of the file mode, stored on the object
  - Yes I'm ignoring many dusty corners

POSIX.1e ACLs: Why?

- The main problem with the traditional model is lack of flexibility
  - Ad-hoc teams of users require sysadmin to create a formal UNIX group
- Enter POSIX ACLs (Access Control Lists)
- Based on the **unratified** POSIX.1e draft standard
- Extends the POSIX model

POSIX.1e ACLs: Structure

- Objects have an ACL
- ACL is a variable length (>=3) list of ACEs (Access Control Entries)
- Each ACE has 3b permission mask & tag
  - *user::* 1, identical to POSIX User class
  - *group::* 1, process matches file gid
  - *user:uid:* N, process matches *uid*
  - *group:gid:* N, process matches *gid*
  - *other::* 1, identical to POSIX Other class

POSIX.1e ACLs: chmod

- In POSIX, the *chmod* syscall supplies a new mode.
- POSIX requires this new mode be an upper limit on granted permissions
  - Even in the presence of ACLs
- Trivial for *User*, *Other* classes: write new mask to the corresponding ACE
- But the *Group* class is now a mix of potentially many ACEs

POSIX.1e ACLs: mask::

- To make *chmod* behave properly in this case, POSIX defines the *mask*:: tag
  - Any non-trivial ACL has a *mask::* ACE
  - An upper bound on permissions which can be granted by *group::*,*user:foo:*,*group:foo:*
- So *chmod* is easy: just write new masks to the *user::*, *mask::*, *other::* ACEs

POSIX.1e ACLs: Inheriting

- Directories have a 2nd ACL, the Default ACL.
  - Identical structure to the normal ”Access” ACL
  - Not used in testing access
- A newly created object gets its initial ACL from it's parent's Default ACL.

Windows ACLs

- Introduced by M$ with NTFS in 1993
- Applies to all sorts of OS objects, not just fs
- Controlled objects have DACL & SACL
- Users, groups identified by Security Identifiers (SIDs)
  - Variable length binary identifiers
  - Includes hierarchical scoping
  - Users and groups in the same namespace

Windows ACLs: Permission Bits

- Rich set of permission bits

ReadData / ListFolder	WriteData / CreateFiles
AppendData / CreateFolders	DeleteSubfoldersAndFiles
TracerseFolder / ExecuteFile	ReadAttributes
WriteAttributes	Synchronize
Delete	ReadPermissions
ChangePermissions	TakeOwnership
ReadExtendedAttributes	WriteExtendedAttributes

Windows ACLs: ACEs

- Three ACE types
  - AccessDenied (DENY)
  - AccessAllowed (ALLOW)
  - SystemAudit (only in SACLs)
- DENY means that ACE order matters
- Each ACE has a SID
  - Some special pre-defined SIDs

Windows ACLs: Special SIDs

- No equivalent to POSIX Other class
  - Special SID *Everyone*
  - Includes owner, all SIDs mentioned in ACEs
- No equivalent to the POSIX User class
  - Special SID *FileCreator*
  - Can only be used in inherit-only ACEs on directories
  - Expanded to a real SID at file create time, unchanged when file owner changed

Windows ACLs: Inheriting

- Each ACE has flags which control how it is inherited from the parent at create
  - OBJECT_INHERIT – the ACE is copied to new files
  - CONTAINER_INHERIT – the ACE is copied to new directories
  - INHERIT_ONLY – the ACE is not used normally and is **only** inherited
- Different approach to POSIX default ACLs

Windows ACLs: Automatic Inheritance

- Automatic Inheritance
  - Actually semi-manual tree propagation
  - Think *chmod -R* with extra complexity
  - ACL and ACE flags to control this

NFSv4 ACLs (1)

- Introduced with NFSv4
  - Optional for server to implement
- Based strongly on the Windows model
  - With some modifications for POSIX
- Same 14 permission bits as Windows
  - Plus 2 new ones in NFSv4.1
- Similar ACE types to Windows: ALLOW, DENY, AUDIT and also ALARM

NFSv4 ACLs (2)

- Same permission algorithm as Windows
- Users, groups identified by **principals**, strings with a domain part
  - e.g. *fred@example.com*
  - ACE flag to distinguish groups
- Special principals:
  - OWNER@ - same as POSIX *user::*
  - GROUP@ - same as POSIX *group::*
  - EVERYONE@ - same as Windows Everyone

NFSv4 ACLs (3)

- Same Automatic Inheritance as Windows
  - Same model with ACE & ACL flags
  - Client is expected to do the treewalk

Permission Model Summary

Feature	Traditional UNIX	POSIX.1e ACLs	Windows ACLs	NFSv4 ACLs
Standardising Body	POSIX	Almost POSIX	M$	IETF
Permission Bits	3	3	14	16
User & Group IDs	uid, gid	uid, gid	SID	principal
Number of ACEs	3	3 - N	0 - N	0 - N
DENY ACE?	-	No	Yes	Yes
ACE Order Significant?	-	No	Yes	Yes
Source of Initial ACL at Create	Syscall param	Parent's Default ACL	Parent's Inheritable ACEs	Parent's Inheritable ACEs

Overview

- Introduction
- Permission Models
- **RichACLs**
- Experiences
- Future Work
- End Matter

Design: IDs

- Identify users and groups with UNIX numerical IDs
  - Mapping to Windows SIDs or NFSv4 principals is left to userspace
  - Leverages existing NFSv4 id mapping infrastructure
  - Optimal for local apps

Design: ACE Types

- Support ALLOW, DENY types
  - Need the flexibility of DENY for interop
  - Despite the headache
- AUDIT, ALARM not implemented
  - Accepted and stored

Design: ACE Count & Order

- ACE order is significant
  - Because of DENY
  - Unlike POSIX ACLs
- No minimum number of ACEs
- Maximum number of ACES controlled by binary format storage limit
  - Windows: 64 KiB
  - Linux xattr: 64 KiB
  - Linux NFSv4: < 4 KiB

Design: Permission Bits (1)

- Same 14 permission bits as NFSv4
  - Plus 2 from NFSv4.1

READ_DATA	Read data from a file
WRITE_DATA	Write data to a file
APPEND_DATA	Write to a file in O_APPEND mode
LIST_DIRECTORY	Read the contents of a directory
ADD_FILE	Create a file object in a directory
ADD_SUBDIRECTORY	Create a directory in a directory
DELETE_CHILD	Delete a file or subdirectory from a directory
EXECUTE	Execute a file Traverse a directory

Design: Permission Bits (2)

DELETE	Delete the file itself, without DELETE_CHILD on the parent
READ_ATTRIBUTES	Read the stat() information for an object Always allowed
WRITE_ATTRIBUTES	Set the atime/mtime on an object
READ_ACL	Read the ACL of an object Always allowed
WRITE_ACL	Set the ACL and POSIX mode of an object
WRITE_OWNER	Take ownership of an object. Set owning group of an object to one of our gids
SYNCHRONIZE, READ_NAMED_ATTRS, WRITE_NAMED_ATTRS, WRITE_RETENTION, WRITE_RETENTION_HOLD	Stored for compatibility but not interpreted

Design: Classes

- Process classification rules similar to POSIX ACLs
  - *User* – process' uid matches file's uid
  - *Group* –
    - process' gids match file's, or
    - process' uid matches any ACE, or
    - process' gids matches any ACE
  - *Other* – otherwise

Design: File Masks

- To make *chmod* work, we define file masks
  - Analagous to POSIX.1e *mask::*
  - 3 × 16b masks stored with ACL
  - One for each class: User, Group, Other
  - Each mask is an upper bound on granted permissions for its class
  - So *chmod* just writes the file masks
- More complexity
  - But no *chmod* hysteresis

Specific Changes: Basics

- Modify VFS to allow fs to do a permission check for create/destroy of an object
- Define a standard binary ACL encoding
  - Machine independent, XDR-like
  - **Not** same as NFSv4 wire format
- Choose a standard xattr name *system.richacl*.
  - Will be used client side, in userspace, and server side

Specific Changes: Kernel

- Provide an in-kernel library for representing & manipulating RichACLs
- Use the kernel library to add ext4 support
  - Stores, retrieves, enforces
  - New superblock option *richacl*
- Use the kernel library to add NFSv4 client & server support
  - Stores, retrieves using fs xattrs
  - Maps to/from NFSv4 wire format

Specific Changes: Userspace

- Provide a userspace library for representing & manipulating RichACLs
  - Nearly same as in-kernel library
  - No permission check
- Use the userspace library to provide a utility *richacl* to set/get ACLs on objects.
  - Not *setrichacl* like the paper says

Neat Results

- Same tools can be used to examine & modify ACLs on client and server
  - Fixes current Linux NFSv4 debacle
- ACLs stored & enforced in one common place for all protocols and clients: the backend filesystem

Overview

- Introduction
- Permission Models
- RichACLs
- **Experiences**
- Future Work
- End Matter

Standards

- NFSv4 ACLs are a maze of twisty little standards, all alike
  - Initial versions based on wishful thinking
  - Not what you expect from the IETF
- Windows ACLs are well documented
  - Sometimes accurately
- POSIX.1e standard is well documented
  - But never ratified, so no pressure to implement it right

NFS Named Attributes

- NFSv4 defines Named Attributes and permission bits {READ,WRITE}_NAMED_ATTRS
- NFS Named Attributes are **not** POSIX xattrs; they're more like Windows ADS
- But Windows ADS uses ReadData not ReadExtendedAttributes permission
- Conclusion: NFSv4 utterly misdesigned.
  - These permissions can't be implemented

Always-granted Permissions

- Some NFSv4 permission bits are not deniable in POSIX
  - e.g. ACE4_READ_ACL
- We chose always to grant these
  - Regardless of the ACL settings
  - To limit impact on the Linux code
  - To make common client actions easier
- This breaches a section of the latest NFSv4 standard

Ugly Corners of POSIX

- Sticky bit
- Capabilities
  - e.g. CAP_DAC_OVERRIDE
- We chose to preserve these
  - To ease migrating a Linux system to RichACLs
  - e.g. /tmp and /home on same fs
- Lots of tricky little test cases

Migration (1)

- We chose to support gradual online migration
  - To ease migrating a Linux system to RichACLs
- Two phase process
  - Enable RichACLs on the fs with *tunefs*
  - Gradually convert files & directories from POSIX ACLs to RichACLs with *richacl*

Migration (2)

- Between phases, kernel *getxattr* synthesises a temporary RichACL to match the existing POSIX ACL
  - Works with a read-only fs
  - Can ”try before you buy”
  - Allows userspace to tweak mappings
- Kernel *setxattr* stores the new RichACL and deletes any POSIX ACL
  - So 2nd phase is just *setxattr(getxattr(f))*

Migration: Drawbacks

- No support for migrating back
- No way to tell when whole fs converted
  - Pointlessly, we always need to check for existence of a POSIX ACL
- Could write back the RichACL as soon as we generate it

Overview

- Introduction
- Permission Models
- RichACLs
- Experiences
- **Future Work**
- End Matter

Future Work (1)

- ACL set/get in Samba server
  - based on existing SGI patch
- ACL set/get in smbfs client
- *ls* & *find* convenience support
- ACL editor GUIs for GNOME/KDE
  - Need to work out how to make this easy for users
- Linux NFSv4 size limit issues

Future Work (2)

- ~~Fully support Automatic Inheritance~~
- Merge upstream
- ID mapping issues
  - Make idmapper less NFSv4 specific
  - Mapped IDs are stored on disk so must be **stable**, which is a problem
  - Maybe something more radical like storing SIDs and/or NFSv4 principals in-kernel?!

Overview

- Introduction
- Permission Models
- RichACLs
- Experiences
- Future Work
- **End Matter**

Current Status

- Patches posted Feb, again July
  - Some feedback
- Have kernel.org git trees now
- Work proceeding
- Want to be upstream
- Testers/reviewers welcome!

Resources

- Kernel git repo

  http://git.kernel.org/?p=linux/kernel/git/kvaneesh/linux-richacl.git;a=summary

- Userspace git repo

  http://git.kernel.org/?p=fs/acl/kvaneesh/acl.git;a=summary

- Patch for tune2fs

  http://kernel.org/pub/linux/kernel/people/kvaneesh/richaclv1/e2fsprogs/

- IRC irc.freenode.net #richacls

Conclusions

- We need a different permission model to support Windows interoperability
- RichACL shows this is possible
- The basic pieces are in place
- But more work remains before end users can use it

Questions?

- Stunned silence?

Greg Banks <gnb@fmeh.org>