# Mapping Windows ACLs into POSIX ACLs


## By Jeremy Allison


**Development Team**

**email: jra@samba.org**

**email: jeremy@valinux.com**

# Why attempt to map Windows to POSIX ACLs in Samba ?

- Windows administrators are used to simple ACL controls.
- The Samba mapping of UNIX user/group/world triple is not considered enough granularity for Windows permissions.
- Competing SMB implementations implement Windows ACLs.
  - It becomes a 'checkbox' feature, no matter how used.
- Fits with Samba philosophy of allowing OS to control acces, less user-space security policy.
  - UNIX and client view of permissions are consistant.

# POSIX ACLs - the non-standard standard.

- Not an official POSIX standard.
- Draft standard 1003.1e revision 17 is the API Samba standardized on.
- Differences in vendor implementations of this API mean Samba needs a interface layer to map to underlying OS.
  - Linux - the UNIX defragmentation tool - uses 1003.1e draft 17 as the API - so identity mapping used on Linux.
- Implementation of ACL support in Samba has increased pressure on ACL standardization.

# POSIX ACLs

- Are extension of UNIX u/g/w permissions.
- Designed for simplicity. Allow additional users and groups to have access specified to a file or directory.
- Do not extend UNIX permission model with extra modes of access ( rwx only).
- Two extra features added, inheritance (for directories) and masks.
  - Inheritance applied to files and directories alike.
  - Masks override group and additional permissions.

# Examining a POSIX file ACL

- Sample POSIX file ACL :

```
# file: testfile          <-- file name
# owner: jeremy           <-- file owner
# group: users            <-- POSIX group owner
user::rwx                 <-- perms. for file owner (standard 'user')
user:tpot:r-x             <-- perms. for extra user 'tpot'
group::r--                <-- perms. group owner (standard 'group')
group:pcguest:r--         <-- perms. for extra group 'pcguest'
mask:rwx                  <-- mask 'ANDed' with groups
other:---                 <-- perms. for any other user (standard 'world')
```

# Examining a POSIX directory ACL.

- Sample POSIX directory ACL :

```
# file: testdir/                <-- File name
# owner: jeremy                 <-- File owner
# group: jeremy                 <-- POSIX group owner
user::rwx                       <-- perms. for directory owner (standard 'user')
group::rwx                      <-- perms. for group owner (standard 'group')
mask:rwx                        <-- mask applied (ANDed) to group perms.
other:r-x                       <-- perms. for all other access (standard 'other')
default:user::rwx               <-- Inherited owner perms.
default:user:tpot:rwx           <-- Inherited extra perms for user tpot
default:group::r-x              <-- Inherited group perms.
default:mask:rwx                <-- Inherited default mask
default:other:---               <-- Inherited other perms.
```

# POSIX ACL rules

- There are some special rules applied.
  - As all POSIX creation calls specify a default mode_t (created permissions) argument, then the most restictive set of inherited and requested permissions is used on creation of a filesystem object.
  - When the chmod call changes group permissions, then the change is applied to the mask if the object has an ACL.
  - This ensures users using non ACL-aware tools don't grant more access than they intended to users or groups with existing ACL entries.

# POSIX ACL evaluation

- A POSIX process has an associated effective user id (euid), effective primary group id (egid), and a list of additional groups (gid's).
- When checking the requested access (rwx) against an object with a POSIX ACL, the order of evaluation is as follows :
  - uid matches are made first (starting with the owner uid). If any uid entries in the ACL match, this entry is used for access.
  - Search for any matching gid entries, if the requested access is granted for any gid associated with the process then allow access.
  - If no other entry matches, use the "other" entry for access.

# "Overdesigned, underused and added to NFSv4" - Win32 ACLs

- Win32 ACLs are (IMHO) a mes.
  - Beautifully designed from a computing science point of view, they are so complex to use that almost NO Windows administrator understands them.
  - In addition, so few Win32 programmers understand them that in practice most applications also ignore ACLs.
  - Order dependent, moving the entries within an ACL can completely change the access decisions granted by that ACL.
- Win32 ACLs (like most things in Win32) are a moving target. Many changes introduced in Windows 2000.

# Win32 ACL details

- Win32 ACLs contain an owner and group owner SID, plus an order dependent list of entries.
  - Each entry contains a SID to which the entry applies, a set of flags and 32 bits of actual permissions.
  - The flags can specify that the permission bits either GRANT or DENY the given action.
  - Flags can also specify if an entry is inherited by objects within a container, and what type of object can inherit the entry.
- They have richer semantics than POSIX ACLs.
  - ACL Inheritance changes made in Windows 2000 make this problem even worse.

# Win32 ACL evaluation

- NT Kernel processes have a token containing a user SID, and a list of group SIDs.
  - These are exactly analogous to a POSIX user context.
- Windows ACL editor insists on DENY ACLs preceeding ALLOW.
- Evaluation walks the list given the requested access, terminating if a DENY match is found, then masking off ALLOW bits granted.
  - If the remaining access has had all bits masked out, access is granted, else denied.

# The easy part - returning POSIX ACLs and Win32 ACLs

- Easier as Win32 ACLs are (almost) a superset of POSIX ACLs.
- Problem was finding a mapping for "empty" POSIX entries.
  - Empty POSIX entries mean "no access".
  - "No access" Win32 entries block following permit entries.
  - Worst case is "rw-rw----". Cannot map other:--- into "Everyone: No Access" as this would override any other access to the file.
  - Use "Take Ownership" (O) flag as meaning "no access".

# Mapping POSIX ACLs to Win32 ACLs

- All ACL entries returned are "ALLOW" entries.
  - Even POSIX user:--- entries are mapped into a "Take Ownership" "ALLOW" entry.
  - This is to preserve the entry in the list, so when this entry is returned to the Samba server, we know to keep the user:--- entry (meaning deny).
  - If this entry was not returned, then on read, no modify, write from the NT ACL editor, the POSIX user:--- entry would be removed (maybe then allowing access that should not have been granted).
  - Group entries are not returned as "DENY" as the order of evaluation would change.
  - The POSIX mask is applied before returning entries.

# A digression - mapping SIDs to uids/gids

- When returning SIDs to an NT client, some mapping between UNIX uid/gids must be done.
- All entries on the UNIX filesystem must have valid uid/gids (by definition).
  - When returning these are either mapped to SIDs local to the server box, to (via winbind) to Domain SIDs.
- When mapping Win32 ACL SIDs to POSIX uid/gid's smbd tries to contact winbindd, and then does a best effort mapping.
  - If no mapping can be done, the ACL set call fails.

# The hard problem - mapping Win32 ACLs to POSIX

- This is very complex code (as it states in the Samba code).
- "Best effort" is the best we can do, given that we are losing information.
- Firstly we convert the incoming Win32 ACL into an internal canonical format (convert SID's to uids/gids, map generic bits to specific bits etc.).
- Next we cope with NT's tendency to canonicalize ACLs that were given to it, and then send back a different ACL....

# Mapping Win32 ACLs to POSIX ACLs (continued).

- We refuse any ACLs not in canonical form (ie. all DENY entries must precede ALLOW entries).
- We then merge ACL entries containing duplicate SIDs (ie. ensure each SID can appear only once in the list).
  - This is done by OR'ing ALLOW permissions, and masking off DENY permissions.
- Then we need to walk the list three times.
- The first pas, we look for a DEN  Y Everyone entry. If found, we truncate the list at this point.

# Mapping Win32 ACLs to POSIX ACLs (continued).

- Second pass - look for user DENY entries. If found, look for ALLOW group entries for which the user is a member of that group, turn the DENY entry into an ALLOW entry with the ALLOW bits from the group entry, masked with the DENY bits from the group entry.
- Third pass - look for group DENY entries. For each one, then look for user ALLOW entries where the user is a member of that group, and mask out the DENY bits.
  - If there exists an ALLOW Everyone entry, convert the DENY group entry to an allow, with the allow bits masked by the deny bits.

# Converting Win32 ACLs to POSIX ACLs (continued).

- What we're creating with all this code, is a snapshot of ALLOW entries, based on the current group memberships on the UNIX server.
  - If group memberships change, the ACLs will not update automatically to reflect this.
- Finally, we ensure that the resultant POSIX ACL is well formed, according to the POSIX spec (ie. must have u/g/w).
- The POSIX ACL mask is always set to rwx, as no mask information is given in the Win32 ACL.
  - When doing chmod() inside Samba, we have to reset the ACL mask, as a standard chmod() call changes it.

# Mapping Win32 ACLs to POSIX ACLs (continued).

- Changes in ownership or group ownership are noted, and the standard UNIX chown() call is made.
  - This will fail if the user is not root of course.
- The result of all this pain, is a mapping that is not exact, but a "reasonable" approximation of what the user wanted.
- Given the light use of ACLs on Win32, this seems to be popular within Samba.
- Tweaks and changes are still being made to the algorithms.¨

# Futures

- Code has been developed to give full Win32 (based on current Windows 2000 ACLs) semantics to Samba.
- This will be kept in a sparse tdb database, external to the UNIX filesystem.
- This unfortunately will decouple the UNIX and Win32 ACLs, but might be useful to Samba NAS appliance vendors.
- Performance implications are not yet known.

# Questions ?

- Samba code is at : www.samba.org
- Samba build farm at : build.samba.org
- Technical mailing list :
  - samba-technical@samba.org