Network Working Group
Internet Draft
Document: draft-ietf-nfsv4-acl-mapping-03.txt

Marius Aamodt Eriksen
J. Bruce Fields

February 2005

Mapping Between NFSv4 and Posix Draft ACLs

Status of this Memo

Abstract

NFS version 4 [rfc3530] (NFSv4) specifies a flavor of Access Control Lists (ACLs) resembling Windows NT ACLs. A number of operating systems use a different flavor of ACL based on a withdrawn POSIX draft. NFSv4 clients and servers on such operating systems may wish to map

between NFSv4 ACLs and their native ACLs. To this end, we describe a mapping from POSIX draft ACLs to a subset of NFSv4 ACLs.

Table of Contents

1. Introduction

Access Control Lists (ACLs) are used to specify fine-grained access rights to file system objects. An ACL is a list of Access Control Entries (ACEs), each specifying an entity (such as a user) and some level of access for that entity.

In the following sections we describe two ACL models: NFSv4 ACLs, and ACLs based on a withdrawn POSIX draft. We will refer to the latter as "POSIX ACLs". Since NFSv4 ACLs are more fine-grained than POSIX ACLs, it is not possible in general to map an arbitrary NFSv4 ACL to a POSIX ACL with the same semantics. However, it is possible to map any POSIX ACL to a NFSv4 ACL with nearly identical semantics. We will describe such a mapping, and discuss its use in NFSv4 clients and servers.

2. NFSv4 ACLs

An NFSv4 ACL is an ordered sequence of ACEs, each having an entity, a type, some flags, and an access mask.

The entity may be the name of a user or group, or may be one of a small set of special entities. Among the special entities are "OWNER@" (the current owner of the file), "GROUP@" (the group associated with the file), and "EVERYONE@".

The type may be ALLOW or DENY. (AUDIT or ALARM are also allowed, but they are not relevant to our discussion).

The access mask has 14 separate bits, including bits to control read, write, execute, append, ACL modification, file owner modification, etc.; consult [rfc3530] for the full list.

Of the flags, four are relevant here. The ACE4_IDENTIFIER_GROUP flag is used to indicate that the entity name is the name of a group. The other three concern inheritance: ACE4_DIRECTORY_INHERIT_ACE indicates that the ACE should be added to new subdirectories of the directory; ACE4_FILE_INHERIT_ACE does the same for new files; and ACE4_INHERIT_ONLY indicates that the ACE should be ignored when determining access to the directory itself.

The NFSv4 ACL permission-checking algorithm is straightforward. Assume a a requester asks for access, as specified by a single bit in

the access bitmask. We allow the access if the first ACE in the ACL that matches the requester and that has that bit set is an ALLOW ACE, and we deny the access if the first such ACE is a DENY ACE. If no matching ACE has the bit in question set, behaviour is undefined. If an access mask consisting of more than one bit is requested, it succeeds if and only if each bit in the mask is allowed.

We refer the reader to [rfc3530] for further details.

3. POSIX ACLs

A number of operating systems implement ACLs based on the withdrawn POSIX 1003.1e/1003.2c Draft Standard 17 [posixacl]. We will refer to such ACLs as "POSIX ACLs".

POSIX ACLs use access masks with only the traditional "read", "write", and "execute" bits. Each ACE in a POSIX ACL is one of five types: ACL_USER_OBJ, ACL_USER, ACL_GROUP_OBJ, ACL_GROUP, ACL_MASK, and ACL_OTHER. Each ACL_USER ACE has a uid associated with it, and each ACL_GROUP ACE has a gid associated with it. Every POSIX ACL must have exactly one ACL_USER_OBJ, ACL_GROUP, and ACL_OTHER ACE, and at most one ACL_MASK ACE. The ACL_MASK ACE is required if the ACL has any ACL_USER or ACL_GROUP ACEs. There may not be two ACL_USER ACEs with the same uid, and there may not be two ACL_GROUP ACEs with the same gid.

Given a POSIX ACL and a requester asking for access, permission is determined as follows:

1) If the requester is the file owner, then allow or deny access depending on whether the ACL_USER_OBJ ACE allows or denies it. Otherwise,

2) if the requester's uid matches the uid of one of the ACL_USER ACEs, then allow or deny access depending on whether the ACL_USER_OBJ ACE allows or denies it. Otherwise,

3) Consider the set of all ACL_GROUP ACEs whose gid the requester is a member of. Add to that set the ACL_GROUP_OBJ ACE, if the requester is also a member of the file's group. Allow access if any ACE in the resulting set allows access. If the set of matching ACEs is nonempty, and none allow access, then deny access. Otherwise, if the set of matching ACEs is empty,

4) if the requester's access mask is allowed by the ACL_OTHER ACE, then grant access. Otherwise, deny access.

The above description omits one detail: in steps (2) and (3), the requested bits must be granted both by the matching ACE and by the ACL_MASK ACE. The ACL_MASK ACE thus limits the maximum permissions which may be granted by any ACL_USER or ACL_GROUP ACE, or by the ACL_GROUP_OBJ ACE.

Each file may have a single POSIX ACL associated with it, used to determine access to that file. Directories, however, may have two ACLs: one, the "access ACL", used to determine access to the directory, and one, the "default ACL", used only as the ACL to be inherited by newly created objects in the directory.

4. Mapping POSIX ACLs to NFSv4 ACLs

We now describe an algorithm which maps any POSIX ACL to an NFSv4 ACL with the same semantics.

First, translate the uid's and gid's on the ACL_USER and ACL_GROUP ACEs into NFSv4 names. This is an implementation-dependent process. It might be done, for example, by consulting a directory service or a password file. Also, the special ACL_USER_OBJ, ACL_GROUP_OBJ, and ACL_OTHER ACEs must be translated to NFSv4 ACEs with the special entities "OWNER@", "GROUP@", and "EVERYONE@", respectively.

Next, map each POSIX ACE (excepting any mask ACE) in the given POSIX ACL to an NFSv4 ALLOW ACE with an entity determined as above, and with a bitmask determined from the permission bits on the POSIX ACE as follows:

1) If the read bit is set in the POSIX ACE, then set ACE4_READ_DATA.

2) If the write bit is set in the POSIX ACE, then set ACE4_WRITE_DATA and ACE4_APPEND_DATA. If the object carrying the ACL is a directory, set ACE4_DELETE_CHILD as well.

3) If the execute bit is set in the POSIX ACE, then set ACE4_EXECUTE.

4) Set ACE4_READ_ACL, ACE4_READ_ATTRIBUTES, and ACE4_SYNCHRONIZE unconditionally.

5) If the ACE is for the special "OWNER@" entity, set ACE4_WRITE_ACL and ACE4_WRITE_ATTRIBUTES.

6) Clear all other bits in the NFSv4 bitmask.

In addition, we set the GROUP flag in each ACE which corresponds to a named group (but not in the GROUP@ ACE, or any of the other special entity ACEs). At this point, we've replaced the POSIX ACL by an NFSv4 ACL with the same number of ACEs (ignoring any mask ACE). To emulate the POSIX ACL permission-checking algorithm, we need to modify the ACL further, as follows:

1) Order the ACL so that the OWNER@ ACE is the first ACE of the ACL, followed by any user ACEs, followed by the GROUP@ ACE, followed by any group ACEs, and ending finally with the EVERYONE@ ACE.

2) The POSIX algorithm stops as soon as the requester matches an ACL_USER_OBJ, ACL_OTHER, or ACL_USER ACE. To emulate this behaviour, add a single DENY ACE after each ALLOW ACE for OWNER@, EVERYONE@, or any named user. The DENY ACE should have the same entity and flags as the corresponding ALLOW ACE. The bitmask on the DENY ACE should be the bitwise NOT of the bitmask on the ALLOW ACE, except that the ACE4_WRITE_OWNER and ACE4_DELETE bits should be cleared, and the ACE4_DELETE_CHILD bit should be cleared on non-directories. (Also, in the xdr-encoded ACL that is transmitted, all bits not defined in the protocol should be cleared.)

3) Unlike the other ACEs in step 2, all of the ACL_GROUP_OBJ and ACL_GROUP ACEs are consulted by the POSIX algorithm before determining permissions. However, if the requester matches any one of them, then it must deny any permissions they do not allow. To emulate this behaviour, instead of adding a single DENY after each corresponding GROUP@ or named group ACE, we insert a list of DENY ACEs at the end of the list of GROUP@ and named group ACEs. Each DENY ACE is determined from its corresponding ALLOW ACE exactly as in step 2, and should occur in the inserted list in the same position as the corresponding ALLOW ACE occurs in the list of ALLOW ACEs.

4) Finally, we enforce the POSIX mask ACE by prepending each ALLOW ACE for a named user, GROUP@, or named group, with a single DENY ACE whose entity and flags are the same as those for the corresponding ALLOW ACE, but whose bitmask is the inverse of the bitmask determined from the mask ACE, with the inverse calculated as described in step 2.

As an example, take a POSIX ACL with two named users (u1 and u2) and two named groups (g1 and g2), in addition to the required ACL_USER_OBJ, ACL_GROUP_OBJ, ACL_OTHER, and ACL_MASK ACEs.

Such an ACL will map to an NFSv4 ACL of the form

```
ALLOW OWNER@
DENY  OWNER@
DENY  u1 (mask)
ALLOW u1
DENY  u1
DENY  u2 (mask)
ALLOW u2
DENY  u2
DENY  GROUP@ (mask)
ALLOW GROUP@
DENY  g1 (mask)
ALLOW g1
DENY  g2 (mask)
ALLOW g2
DENY  GROUP@
DENY  g1
DENY  g2
ALLOW EVERYONE@
DENY  EVERYONE@
```

where the ACEs marked with (mask) are those whose bitmask are determined from the ACL_MASK ACE as described in step 4 above.

In general, a POSIX ACL with m named users and n named groups will map to an NFSv4 ACL with (3*(m + n) + 7) ACLs, unless m and n are both zero, in which case the result will have either 6 or 7 ACLs, depending on whether the original ACL had an ACL_MASK ACE.

On directories with default ACLs, we translate the default ACL as above, but set the ACE4_INHERIT_ONLY_ACE, ACE4_DIRECTORY_INHERIT_ACE, and ACE4_FILE_INHERIT_ACE flags on every ACE in the resulting ACL. On directories with both default and access ACLs, we translate the two ACLs and then concatenate them. The order of the concatenation is unimportant.

There is one extremely minor inaccuracy in this mapping: if a requester that is a member of more than one group listed in the ACL requests multiple bits simultaneously, the POSIX algorithm requires all of the bits to be granted simultaneously by one of the group ACEs. Thus a POSIX ACL such as

```
ACL_USER_OBJ: ---
ACL_GROUP_OBJ: ---
g1: r--
g2: -w-
ACL_MASK: rw-
ACL_OTHER: ---
```

will prevent a user that is a member of groups g1 and g2 from opening a file for both read and write, even though read and write would be individually permitted.

The NFSv4 ACL permission-checking algorithm has the property that it permits a group of bits whenever it would permit each bit individually, so it is impossible to mimic this behaviour with an NFSv4 ACL.

5. Using the Mapping in NFSv4 Implementations

Examination of the algorithm described in the previous section shows that no information is lost; the original POSIX ACL can be reconstructed from the mapped NFSv4 ACL. Thus we also have a way to map NFSv4 ACLs to POSIX ACLs in the case where the NFSv4 ACL is precisely in the format of an ACL produced by the algorithm above.

The algorithm can therefore be used to implement a subset of the NFSv4 ACL model. This may be useful to NFSv4 clients and servers with preexisting system interfaces that support POSIX ACLs and that cannot be modified to support NFSv4 ACLs.

A server, for example, that wishes to export via NFSv4 a filesystem that supports only POSIX ACLs, may use this mapping to answer client requests for existing ACLs by translating POSIX ACLs on its filesystem to NFSv4 ACLs to send to the client. However, when a client attempts to set an ACL, the server faces a problem. If the given ACL is not in precisely the format of an ACL produced by this mapping, then the server may be required to return an error to avoid inaccurately representing the client's intention. The correct error to return in this case is NFS4ERR_ATTRNOTSUPP.

In the case where a client sets an ACL that leaves certain bits neither allowed nor denied, the server may choose to allow or deny those bits as necessary to make mapping possible. In some situations it may also be possible for a server to map the ACL if it adds a DENY ACE or denies a few additional bits. The language of [rfc3530] allows a server some flexibility in handling ACLs that it cannot enforce completely accurately, as long as it adheres to "the guiding principle... that the server must not accept ACLs that appear to make [a file] more secure than it really is."

Given the choice, as long as the "guiding principle" is not violated, servers should opt to be forgiving. The complexity of the POSIX<->NFSv4 mapping makes difficult the task of generating ACLs

that will satisfy a server using the mapping. By making the mapping more forgiving, the server can simplify that task, improving interoperability.

Servers that implement the full NFSv4 protocol should also handle carefully ACLs that leave bits neither allowed nor denied. It is better to fall back on some reasonable default rather than to always allow or always deny. A client that, for example, sets ACE4_WRITE_DATA but leaves unspecified ACE4_APPEND_DATA probably does so because its system interfaces are incapable of independently representing ACE4_APPEND_DATA, not because it intends to deny ACE4_APPEND_DATA. By leaving the bit unspecified, the client leaves the server the opportunity to provide the reasonable default of setting it to match ACE4_WRITE_DATA.

Similar issues exist when a client uses NFSv4 ACLs to implement user interfaces that only deal in POSIX ACLs. When the client translates ACLs received from the server to POSIX ACLs, some flexibility may help interopability, but the client must take care not to represent any ACLs as stricter than they really are. Clients that provide access to the full set of NFSv4 ACLs may also wish to provide users with utilities to generate and interpret POSIX-mapped NFSv4 ACLs, to aid users working with servers using the POSIX mapping.

6. Security Considerations

Any automatic mapping from one ACL model to another must provide guarantees as to how the mapping affects the meaning of ACLs, or risk misleading users about the permissions set on filesystem objects. For this reason, caution is recommended when implementing this mapping. It is better to return errors than to break any such guarantees.

Note also that this ACL mapping requires mapping between NFSv4 usernames and local id's. When the mapping of id's depends on remote services, the method used for the mapping must be at least as secure as the method used to set or get ACLs.

7. Bibliography

[rfc3530]
Shepler, S. et. al., "NFS version 4 Protocol", April 2003.

http://www.ietf.org/rfc/rfc3530.txt

[posixacl]
IEEE, "IEEE Draft P1003.1e", October 1997 (last draft).

http://wt.xpilot.org/publications/posix.1e/download.html

8. Author's Address

Address comments related to this memorandum to:

nfsv4@umich.edu marius@umich.edu bfields@umich.edu

Marius Aamodt Eriksen
J. Bruce Fields
University of Michigan / CITI
535 West William
Ann Arbor, Michigan

E-mail: marius@umich.edu
E-mail: bfields@umich.edu

9. Copyright