Network Working Group Marius Aamodt Eriksen
Internet Draft J. Bruce Fields
Document: draft-ietf-nfsv4-acl-mapping-02.txt October 2004

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 consists of a number of Access Control Entries (ACEs), each specifying some level of access for an entity. The entity may be a user, a group, or a special entity (such as "everyone"). The level of access is described using an access mask, which is a bitmask with each bit corresponding to a type of access (such as "read" or "append").

In the following sections we describe two ACL models: NFSv4 ACLs, and ACLs based on a withdrawn POSIX draft, which we will refer to as "POSIX ACLs". Since NFSv4 ACLs are much finer-grained than POSIX ACLs, it is not possible in general to map an arbitrary NFSv4 ACL to a POSIX ACL with the same semantics. It does, however, turn out to be possible to map any POSIX ACL to a NFSv4 ACL that has nearly iden- tical semantics. We will describe such a mapping, and discuss how it might be used in NFSv4 client and server implementations.

2. NFSv4 ACLs

An NFSv4 ACL is an ordered sequence of ACEs, each having an entity, a type, and an access mask. The entity may be the name of a user or group, or may also 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 access mask includes bits for access types that are more fine- grained than the traditional "read", "write", and "execute" permis- sions used in UNIX mode bits.

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

The NFSv4 ACL permission-checking algorithm is straightforward. Given an ACL and a requestor asking for a set of permissions speci- fied by an access mask:

1) Walk through the list of ACEs from the ACL in order.

2) Ignore any ACE for with an entity not matching requestor.

3) Process all ACEs until all the bits in the requested access mask have been ALLOWed by an ALLOW ace with that bit set. Once a particular bit has been ALLOWed by an ACE, it is no longer considered in further processing.

4) If a bit in the requested access mask is DENYed (while that bit is still under consideration), the request is denied.

5) If all bits have been ALLOWed, the access is granted. Otherwise behavior is undefined.

There are also a number of flags that can be applied to an NFSv4 ACE. Three flags that we will need to use in the following discussion apply to ACEs in a directory ACL. They are: ACE4_DIRECTORY_INHERIT_ACE, which indicates that the ACE should be added to new subdirectories of the directory; ACE4_FILE_INHERIT_ACE, which does the same for new files; and ACE4_INHERIT_ONLY, which indicates that the ACE should be ignored when determining access to the directory itself.

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

3. POSIX ACLs

A number of operating systems, including Linux and FreeBSD, 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 requestor asking for access, permission is determined as follows:

1) If the requestor 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 requestor's uid matches the uid of one of the ACL_USER ACE's, 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 ACE's whose gid the requestor is a member of. Add to that set the ACL_GROUP_OBJ ACE, if the requestor is also a member of that group. Allow access if one of the ACE's in the resulting set allows access. If the set of matching ACEs is nonempty, and none allow access, then deny access. Otherwise, if none of these ACEs match,

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

Steps (2) and (3) have an additional criteria; in addition to checking whether the requested access mask is allowed by the access mask in the ACE, the requested bits also have to be in the access mask of the special ACE with the ACL_MASK entity. This allows file owners to specify a maximum level of access allowed by any other user or group that has any access to the file system object.

In addition to a regular POSIX ACL, a directory in the file system may also have associated with it a default ACL. This default ACL does not affect permissions to the directory itself. Instead, it governs the ACL a file system object will be assigned initially when it is created as a child of the particular directory.

4. Mapping Posix ACLs to NFSv4 ACLs

Given the differences between POSIX and NFSv4 ACLs, any conversion between the two is difficult. However, POSIX ACLs are a subset of NFSv4 ACLs, and any POSIX ACL can be emulated with an NFSv4 ACL using the following mapping.

First, the uid's and gid's on the ACL_USER and ACL_GROUP ACEs must be translated into NFSv4 names--a system-dependent process, which, on UNIX for example, may be done by lookups to /etc/passwd. 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.

The ACE access mask is translated as follows. The read bit of the POSIX access mask is translated to the logical OR of the

ACE4_READ_DATA and ACE4_READ_NAMED_ATTRS NFSv4 access mask fields. The write bit of the POSIX access mask is translated to the logical OR of the ACE4_WRITE_DATA, ACE4_WRITE_NAMED_ATTRS and ACE4_APPEND_DATA NFSv4 access mask fields. The execute bit of the POSIX access mask is translated into the ACE4_EXECUTE and ACE4_READ_DATA NFSv4 acess mask fields. Note that NFSv4 defines ACE4_READ_DATA, ACE4_WRITE_DATA, and ACE4_APPEND_DATA to be equal to ACE4_LIST_DIRECTORY, ACE4_ADD_FILE, and ACE4_ADD_SUBDIRECTORY, respectively, so this translation makes sense for directories as well. However, on directories the ACE4_DELETE_CHILD field must be included in the translation of the POSIX write bit.

In addition to the above, the OWNER entity must always be given ACE4_WRITE_ACL and ACE4_WRITE_ATTRIBUTES, and all entities must be given ACE4_READ_ACL, ACE4_READ_ATTRIBUTES, and ACE4_SYNCHRONIZE. The ACE4_DELETE bit should be neither allowed nor denied by any ACE.

The ACE flag field also has a simple translation. If the file system object is a directory, and the POSIX ACE belongs to a default ACL, the ACE4_INHERIT_ONLY_ACE, ACE4_DIRECTORY_INHERIT, and ACE4_FILE_INHERIT flags are set in the NFSv4 ACE. If the entity in the POSIX ACE refers to a group, the "ACE4_IDENTIFIER_GROUP" flag is set in the NFSv4 ACE.

The POSIX ACL_USER_OBJ ACE is also always given the permission bits "ACE4_READ_ACL" and "ACE4_WRITE_ACL."

Completing the mapping reduces to being able to emulate an ACL_MASK and compensate for some differences in the permission-checking algorithms of the two ACL implementations.

The difference in permission-checking algorithms is handled as follows:

Every user ACE in the POSIX ACL maps into 2 NFSv4 ACEs; one ALLOW ACE which is translated as specified by the above scheme, then a complementing DENY ACE which is also translated as specified by the above scheme, with the exception that the access mask is inverted. Note that the ACL_USER_OBJ ACE is placed first in this list.

Every group ACE in the POSIX ACL produces a similar pair, but instead of being in sequence, all of the ALLOW ACEs are all in sequence, followed by all the DENY ACEs. The ACL_GROUP_OBJ ACE is placed first in both lists.

Lastly, the POSIX ACL_OTHER ACE is translated into a pair of ACEs as

in the user ACE case.

With this done, the NFSv4 permission-checking algorithm applied to the resulting NFSv4 ACL will produce the same result as the POSIX permission-checking algorithm did on the original POSIX ACL.

To handle the special POSIX entity ACL_MASK, we slightly modify the above translation:

With the exception of the "OWNER" and "EVERYONE" ACEs, another ACE is prepended to the ACE. The prepended ACE is a DENY ACE with the same entity as the following ALLOW ACE, but with a permission mask set to the complement of the POSIX ACL_MASK.

This method allows us to preserve the real permission bits of each ACE should the ACL_MASK be changed.

5. Using the Mapping in NFSv4 Implementations

Note that the algorithm described in the previous section not only provides a way to map any POSIX ACL to be mapped to an NFSv4 ACL with similar semantics, but also provides the reverse mapping 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 happens to be in precisely the format of an ACL produced by this mapping (as would happen if, for example, the client was performing the same translation), then the server can map it to a POSIX ACL to store on the filesystem. But for any other NFSv4 ACL, the server should return an error to avoid any chance of inaccurately representing the client's intention.

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."

It may therefore be possible for a server to accept a wider range of NFSv4 ACLs, as long as it can ensure that in every case the resulting POSIX ACL denies at least all access that the original NFSv4 ACL denied. The results of such a mapping may, however, be somewhat unexpected, and it is preferable simply to refuse all NFSv4 ACLs that do not map accurately, and provide clients with software to help generate POSIX-mappable NFSv4 ACLs if necessary.

Similarly, a client that uses NFSv4 ACLS to implement user interfaces that only deal in POSIX ACLs may handle user requests to set ACLs easily enough, but should return errors when the user requests ACLs that, on consulting the server, turn out to not be mappable to POSIX ACLs.

6. Security Considerations

Any automatic mapping from one ACL model to another must provide guarantees that the mapping preserves semantics, or risk misleading users about the permissions set on filesystem objects. For this reason, we recommend performing such mapping only when it can be done accurately, and returning errors in all other cases.

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:

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