Workgroup: NFSv4
Updates: 8881, 7530 (if approved)
Published: 13 October 2021
Intended Status: Standards Track
Expires: 16 April 2022
Authors: D. Noveck, Ed.
NetApp

Security for the NFSv4 Protocols

Abstract

This document describes the core security features of the NFSv4 family of protocols, applying to all minor versions. The discussion includes the use of security features provided by the RPC transport.

This preliminary version of the document, is intended, in large part, to result in working group discussion regarding existing NFSv4 security issues and to provide a framework for addressing these issues and obtaining working group consensus regarding necessary changes.

When a successor document is eventually published as an RFC, it will supersede the description of security appearing in existing minor version specification documents such as RFC 7530 and RFC 8881.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 16 April 2022.

Table of Contents

1. Overview

This document is intended to form the basis for a new description of NFSv4 security applying to all NFSv4 minor versions. The motivation for this new document and the need for major improvements in NFSv4 security are explained in Section 1.1.

Because this document anticipates making major changes in material covered in previous standards-track RFCs, extensive working group discussion will be necessary to make sure that there is a working group consensus to make the changes being proposed. These changes include the major improvements mentioned above and changes necessary to suitably describe features currently in an unsatisfactory state as described in Section 3.4

1.1. Document Motivation

A new treatment of security is necessary because:

- Previous treatments paid insufficient attention to security issues regarding data in flight.

- The presentation of AUTH_SYS as an "'**OPTIONAL**' means of authentication" obscured the significant security problems that come with its use.

- The security considerations sections of existing minor version specifications contain no threat analyses and focus on particular security issues in a way that obscures, rather than clarifying, the security issues that need to be addressed.

- The availability of RPC-with-TLS (described in [12]) provides facilities that NFSv4 clients and servers will need to use to provide security for data in flight and mitigate the lack of authentication when AUTH_SYS is used.

1.2. Document Annotation

The first version of this preliminary document contained many notes with headers in brackets, requesting comments regarding confusing or otherwise dubious passages in existing documents and noting other choices that need to made. Comments about and working group discussion of these issues will be important in arriving at an adequate RFC candidate. In this version, those specific items have been removed and are replaced by the sorts of items described below which show the troublesome existing text, explain the issues with it, and and provide a proposed replacement.

In order to make further progress on these difficult issues, including many whose resolution will probably involve compatibility issues with existing implementations, the author has tried his best to resolve these issues, even though there is no assurance that the resolution adopted by consensus will match the author's current best efforts. To provide a possible resolution that might be the basis of discussion while not foreclosing other possibilities, proposed changes are organized into a series of consensus items, which are listed in Appendix B.

For such pending issues, the following annotations will be used:

*A paragraph headed "[Author Aside]:", provides the author's comments about possible changes and will probably not appear in an eventual RFC.

This paragraph can specify that certain changes within the current section are to be implicitly considered as part of a specific consensus item.

The paragraph can indicate that all unannotated material in the current section is to be considered either the previous treatment or the proposed replacement text for a specific consensus item.

*A paragraph headed "[Consensus Needed (Item #NNx)]:", provides the author's preferred treatment of the matter and should only appear in the eventual RFC if working group consensus on the matter is obtained allowing the necessary changes to be made permanent, without being conditional on a future consensus.

The item id, represented above by "NNx" consists of a number identifying the specific consensus item and letter which is unique to appearance of that consensus item in a particular section. In cases in which a pending item is cited with no part of the discussion appearing in the current section, an item id of the form "#NN" is used.

*A paragraph headed "[Previous Treatment]:", indicates text that is provided for context but which the author believes, should not appear in the eventual RFC, because it is expected to be superseded by a corresponding consensus item

The corresponding consensus item is often easily inferred, but can be specified explicitly, as it is for items associated with the consensus item itself.

Each of the annotations above can be modified by addition of the phrase, "Including List" to indicate that it applies to a following bulleted list as well as the current paragraph.

2. Requirements Language

2.1. Keyword Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as specified in BCP 14 [1] [5] when, and only when, they appear in all capitals, as shown here.

2.2. Special Considerations

Because this document needs to revise previous treatments of its subject, it will need to cite previous treatments of issues that now need to be dealt with in a different way. This will take the form of quotations from documents whose treatment of the subject is being obsoleted, most often direct but sometimes indirect as well.

Paragraphs headed "[Previous Treatment] or otherwise annotated as having that status, as described in Section 1, can be considered quotations in this context.

Such treatments in quotations will involve use of these BCP14-defined terms in two noteworthy ways:

* The term may have been used inappropriately (i.e not in accord with [1]), as has been the case for the "**RECOMMENDED**" attributes, which are in fact **OPTIONAL**.

  In such cases, the surrounding text will make clear that the quoted text does not have a normative effect.

  Some specific issues relating to this case are described below Section 7.1.

* The term may been used in accord with [1], although the resulting normative statement is now felt to be inappropriate.

  In such cases, the surrounding text will need to make clear that the text quoted is no longer to be considered normative, often by providing new text that conflicts with the quoted, previously normative, text.

  An important instance of this situation is the description of AUTH_SYS as an "**OPTIONAL**" means of authentication. For detailed discussion of this case, see Sections 13 and 17.1.5

3. Introduction to this Update

There are a number of noteworthy aspects to the updated approach to NFSv4 security presented in this document:

* There is a major rework of the security framework to take advantage of work done in RPC-with-TLS, as described in Section 1.1.

  NFSv4 security is still built on RPC, as had been done previously. However, it is now able to take advantage of security-related transport properties. For more information about this transformation, see Section 3.1.

For an overview of changes made so far as part of this rework, see Appendix A.1.

* This document deals with all minor versions together, although there is a need for exceptions to deal with, for example, pNFS security.

  For more detail about how minor version differences will be addressed, see Sections 3.2 and 3.3.

* There is a new Security Considerations section including a threat analysis.

* There has been extensive work to clarify the multiple types of authorization within NFSv4 and deal more completely with the co-ordination of ACL-based and mode-based file access authorization.

3.1. Transport-based Security Features

There are a number of security-related facilities that can be provided at the transport layer eliminating the need to provide such support to as part of RPC proper.

These will initially be provided by RPC-with-TLS but similar facilities might be provided by new versions of existing transports or new RPC transports.

* The transport might provide encryption of requests and replies, eliminating the need for privacy and integrity services to be negotiated later and applied on a per-request basis.

  While clients might choose to establish connections with such encryption, servers can establish policies allowing access to certain pieces of the namespace using such transports, or limiting access to those providing privacy, allowing the use of either transport-based encryption or privacy services provided by RPCSEC_GSS.

* The transport might provide mutual authentication of the client and server peers as part of the establishment of the connection This authentication is distinct from the the mutual authentication of the client user and server peer, implemented within the GSSSEC_RPC framework.

  This form of authentication is of particular importance when when the server allows the use of the flavors AUTH_SYS and AUTH_NONE, which have no provision for the authentication of the user requesting the operation.

While clients might choose, on their own,to establish connections with such peer authentication, servers can establish policies a limiting access to certain pieces of the namespace without such peer authentication or only allowing it when using RPCSEC_GSS.

To enable server policies to be effectively communicated to clients, the security negotiation framework now allows connection characteristics to be specified using pseudo-flavors returned as part of the response to SECINFO and SECINFO_NONAME. See Section 15 for details.

3.2. Handling of Multiple Minor Versions

In some cases, there are differences between minor versions in that there are security-related features, not present in all minor versions.

To deal with this issue, this document will focus on a few major areas listed below which are common to all minor versions.

*File access authorization (discussed in Section 7) is the same in all minor versions together with the identification/ authentication infrastructure supporting it (discussed in Section 13) provided by RPC and applying to all of NFS.

An exception is made regarding labelled NFS, an optional feature within NFSv4.2, described in [10]. This is discussed as a version-specific feature in this document in Section 10

*Features to secure data in-flight, all provided by RPC, together with the negotiation infrastructure to support them are common to all NFSv4 minor versions, are discussed in Section 15

However, the use of SECINFO_NONAME, together with changes needed for transport level encryption, paralleling those proposed here for SECINFO, is treated as a version-specific feature and, while mentioned here, will be fully documented in new NFSv4.1 specification documents.

*The protection of state data from unauthorized modification is discussed in Section 11) is the same in all minor versions together with the identification/ authentication infrastructure supporting it (discussed in Section 13) provided by secure transports such as RPC-over-TLS.

It should be noted that state protection based on RPCSEC_GSS is treated as a version-specific feature and will continue to be described by [8] or its successors. Also, it needs to be noted that the use of state protection was not discussed in [6].

3.3. Handling of Minor-version-specific features

There are a number of areas in which security features differ among minor versions, as discussed below. In some cases, a new feature requires specific security support while in others one version will have a new feature related to enhancing the security infrastructure.

How such features are dealt with in this document depends on the specific feature.

* In addition to SECINFO, whose enhanced description appears in this document, NFSv4.1 added a new SECINFO_NONAME operation, useful for pNFS file as well as having some non-pNFS uses.

  While the enhanced description of SECINFO mentions SECINFO_NONAME, this is handled as one of a number of cases in which the description has to indicate that different actions need to be taken for different minor versions.

  The definitive description of SECINFO_NONAME, now appearing in RFC8881 needs to be modified to match the description of SECINFO appearing in this document. It is expected that this will be done as part of the rfc5661bis process.

  The security implications of the security negotiation facilities as a whole will be addressed in the security considerations section of this document.

* The pNFS optional feature added in NFSv4.1 has its own security needs which parallel closely those of non-pNFS access but are distinct, especially when the storage access protocol used are not RPC protocols. As a result, these needs and the means to satisfy them are not discussed in this document.

  The definitive description of pNFS security will remain in RFC8881 and its successors (i.e. the rfc5661bis document suite). However, because pNFS security relies heavily on the infrastructure discussed here, it is anticipated that the new treatment of pNFS security will deal with many matters by referencing the overall NFS security document.

  The security considerations section of rfc5661bis will deal with pNFS security issues.

* In addition to the state protection facilities described in this document, NFS has another set of such facilities that are only implemented in NFSv4.1.

While this document will discuss the security implications of protection against state modification, it will not discuss the details of the NFSv4.1-specific features to accomplish it.

*The additional NFSv4.1 acl attributes, sacl and dacl, are discussed in this document, together with the ACL inheritance features they enable.

As a result, the responsibility for the definitive description of these attributes will move to overall NFS security document, with the fact that they are not available in NFSv4.0 duly noted. While these attributes will continue to be mentioned in NFSv4.1 specification documents, the detailed description appearing in RFC8881 will be removed in successor documents.

*Both NFSv4.0 and NFSv4.1 specifications discussed the coordination of the values the mode and ACL-related attributes. While the treatment in RFC8881 is more detailed, the differences in the approaches are quite minor.

[Consensus Item #25a]: This document will provide a unified treatment of these issues, which will note any differences of treatment that apply to NFSv4.0. Changes applying to NFSv4.2 will also be noted.

As a result, this document will override the treatment within RFC7530 and RFC8881. This material will be removed in the rfc5661bis document suite and replaced by a reference to the treatment in the NFSv4 security RFC.

*The protocol extension defined in [13], now part of NFSv4.2, is also related to the issue of co-ordination of acl and mode attributes and will be discussed in that context.

Nevertheless, the description in [13] will remain definitive.

*The NFSv4.1 attribute set-mode-masked attribute is mentioned together with the other attributes implementing the POSIX authorization model.

Because this attribute. while related to security, does not substantively modify the security properties of the protocol, the full description of this attribute, will continue to be the province of the NFSv4.1 specification proper.

*There is a brief description of the v4.2 Labelled NFS feature in Section 10. Part of that description discusses the limitations in the description of that feature within [10].

Because of some limitations in the description, it is not possible to provide an appropriate security considerations section for that feature in this document.

As a result, the responsibility for providing an appropriate Security Considerations section remains, unrealized for now, with the NFSv4.2 specification document and its possible successors.

3.4. Features Needing Extensive Clarification

For a number of authorization-related features, the existing descriptions are inadequate for various reasons:

* In the description of the the use of the mode attribute in implementing the POSIX-based authorization model, critical pieces of the semantics are not mentioned, while, ironically, the corresponding semantics for ACL-based authorization are discussed.

  This includes the authorization of file deletion and of modification of the mode, owner and owner-group attributes. For ACL-based authorization, there is a an attempt to provide the description.

  The situation for authorization of RENAME is similar, although, in this case, the corresponding semantics for the ACL case are also absent.

* The description of authorization for ACLs is more complete but it needs further work, because the previous specifications make extensive efforts, in my view misguided, to allow an enormous range of server behaviors, making it hard for a client to know what the effect of many actions, and the corresponding security-related consequences, might be.

  Troublesome in this connection are the discussion of ACE mask bits which essentially treats every mask bit, as its own OPTIONAL feature, the use of "**SHOULD**" and "**SHOULD NOT**" in situations which it is unclear what valid reasons to ignore the recommendation might be, and cases in which it is is simply stated that some servers do some particular thing, leaving the unfortunate implication that clients need to be prepared for a vast range of server behaviors.

  This approach essentially treated ACLs in a manner appropriate to an experimental feature.

* Similar issues apply to descriptions related to the need to co-ordinate the values of the mode attribute and the ACL-related attributes.

Although the need for such coordination is recognized. There are multiple modes of mapping an ACL to a corresponding mode together with multiple sources of uncertainty about the reverse mapping.

In addition, certain of the mapping algorithms have flaws in that their behavior under unusual circumstances give results that appear erroneous.

Dealing with these issues is not straightforward, because the appropriate resolution will depend on:

* The actual existence of server implementations with non-preferred semantics.

  In some cases in which "**SHOULD**" was used, there may not have been any actual severs choosing to ignore the recommendation, eliminating the possibility of compatibility issues when changing the "**SHOULD**" to a formulation that restricts the server's choices.

* The difficulty of modifying server implementations to eliminate or narrow the effect of non-standard semantics.

  One aspect of that difficulty might be client or application expectations based on existing server implementations, even if the existing specifications give the client no assurance that that server's behavior is mandated by the standard.

* Whether the existing flaw in some existing recommended actions to be performed by the server is sufficiently troublesome to justify changing the specification at this point.

This sort of information will be used in deciding whether to:

* Narrow the scope of allowable server behavior to those actually used by existing severs.

* Limiting the negative effects of unmotivated **SHOULD**s by limiting valid reasons to ignore the recommendation to the difficulty of changing existing implementations.

  This would give significant guidance to future implementations, while forcing clients to live with the uncertainty about existing servers

* Tie a more restricted set of semantics to nominally unrelated OPTIONAL features such as implementation of dacl and sacl.

  This would provide a way to allow the development of newer servers to proceed in a way that

* Provide means that clients to use to determine, experimentally, what semantics are provided by the server.

  Would need to be supported by a requirement/assurance that a server behave uniformly, at least within the scope of a single filesystem.

* Allow the provision of other ways for the client to know the semantics choices made by the server.

Despite the difficulty of addressing these issues, if the protocol is to be secure and ACLs are to be widely available, these problems must be addressed. While there has not been significant effort to provide client-side ACL APIs and there might not be for a while, we cannot have a situation if which the security specification makes that development essentially impossible.

3.5. Process Going Forward

Because of the scope of this document, and the fact that it is necessary to modify previous treatments of the subject previously published as Proposed Standards, it is necessary that the process of determining whether there is Working Group Consensus to submit it for publication be more structured than that used for the antecedent documents.

In order to facilitate this process, the necessary changes which need to be made, beyond those clearly editorial in nature, are listed in Appendix B. As working group review and discussion of this document and its successors proceeds, there will be occasion to discuss each of these changes, identified by the annotations described in Section 1.2.

Based on working group discussions, successive document versions will do one of the following for some set of consensus items:

* Deciding that the replacement text is now part of a new working group consensus.

  When this happens, future drafts of the document will be modified to remove the previous treatment, treat the proposed text as adopted, and remove Author Asides or replace them by new text explaining why a new treatment of the matter has been adopted or pointing the reader to an explanation in Appendix A.

  At this point, the consensus item will be removed from Appendix B and an explanation for the change will be added to Appendix A.

* Deciding that the general approach to the issue, if not necessarily the specific current text has reached the point of "general acceptance" as defined in Appendix B

  In this case, to facilitate discussion of remaining issues, the text of the document proper will remain as it is.

  At this point, the consensus item will be marked within the table in Appendix B as having reached general acceptance, indicating the need to prioritize discussion in the next document cycle, aimed at arriving at final text to address the issue.

  In addition, an explanation for the change will be added to Appendix A.

* Deciding that modification of the existing text is necessary to facilitate eventual consensus, based on the working group's input.

  In this case, there will be changes to the document proper in the next draft revision. In some cases, because of the need for a coherent description, text outside the consensus item may be affected.

  The table in Appendix B will be updated to reflect the new item status while Appendix A is not expected to change.

* Deciding that the item is best dropped in the next draft.

  In this case, the changes to the document proper will be the inverse of those when a change is accepted by consensus. The previous treatment will be restored as the current text while the proposed new text will vanish from the document at the next draft revision. The Author Aside will be the basis for an explanation of the consequences of not dealing with the issue.

  At this point, the consensus item will be removed from Appendix B.

The changes that the working group will need to reach consensus on, either to accept (as-is or with significant modifications) or reject can be divided into three groups.

* A large set of changes, all addressing issues mentioned in Section 1.1, were already present in the initial I-D so that there has been the opportunity for working group discussion of them, although that discussion has been quite limited so far.

  As a result, a small set of these changes is marked, in Appendix B, as having reached general acceptance.

That subset of these changes changes, together with the organizational changes to support them are described in Appendix A.1.

*Another large set of changes were made in draft -02. These mostly concern the issues mentioned in Section 3.4 None of these changes is yet considered to have reached general acceptance.

The organizational changes to support these changes are described in Appendix A.2.

*There remain a set of potential changes for which a need is expected but for which no text is yet available.

Such changes have associated Author Asides and are listed in Appendix B.

The text for these changes is expected to be made available in future document revisions and they will be processed then, in the same way as other changes will be processed now.

If and when such changes reach general acceptance, they will be explained in the appropriate subsection of Appendix A.

4. Introduction to NFSv4 Security

Because the basic approach to security issues is so similar for all minor versions, this document applies to all NFSv4 minor versions. The details of the transition to an NFSv4-wide document are discussed in Sections 3.2 and 3.3.

NFSv4 security is built on facilities provided by the RPC layer, including various authentication flavors and underlying transports.

[Consensus Needed, Including List (Item #1a)}: Support for multiple authentication flavors can be provided. Not all of these actually provide authentication, as discussed in Section 13.

*Support for RPCSEC_GSS is **REQUIRED**, although use of other flavors is provided for.

This flavor provides for mutual authentication of the principal making the request and the server performing it.

This flavor allows the client to request the provision of encryption-based services to provide privacy or integrity for specific requests. Although such services are often provided by the underlying RPC transport, this support is useful, when the transport services are not supported or are otherwise unavailable.

*AUTH_SYS, provides identification of the principal making the request but **SHOULD NOT** be used unless the client peer sending the request can be authenticated and there is protection against the modification of the request in flight.

Both of the above require transport-level support.

*AUTH_NONE does not provide identification of the principal making the request so only should be used for requests for which there is no such principal or for which it would irrelevant.

The restrictions mentioned above for AUTH_SYS apply to AUTH_NONE as well.

[Consensus Needed, Including List (Item #1a)}: There are important services that can be provided by the RPC transport when RPC-with-TLS is available, or when other transports provide similar services

*The transport can provide data security to all requests on the. This is to be preferred to data security provided by the authentication flavor because it provides protection to the request headers, because it applies to requests using all authentication flavors, and because it is more likely to be offloadable.

*The transport can authenticate the server to the client peer. This is desirable since that authentication applies even when AUTH_SYS or AUTH_NONE is used.

*The transport can authenticate the client-peer to the server at the time the connection is set up. This is essential to allow AUTH_SYS to be used with a modicum of security, based on the server's level of trust with regard to the client peer.

[Consensus Needed (Item #2a)}: Because important security-related services depend on the transport used, rather the authentication, the process of security negotiation, described in Section 15 provides for the negotiation of an appropriate transport at connection time if the server's policy limits the range of transports being used and also when use of a particular transport/ flavor combination causes NFS4ERR_WRONGSEC to be returned,

[Consensus Needed (Item #1a)}: The authentication provided by RPC, is used to provide the basis of authorization, which is discussed in general in Section 6. This includes file access authorization, discussed in Sections 7 through 9 and state modification authorization, discussed in Section 11

File access is controlled by the server support for and client use of certain recommended attributes, as described in Section 7.1.

Multiple file access model are provided for and the considerations discussed in Section 8 apply to all of them.

- The mode attribute provides a POSIX-based authorization model, as described in Section 7.3

- The ACL-related attributes acl, sacl, and dacl (the last two introduced in NFSv4.1) support a finer grained authorization model and provide additional securiy-related services. The structure of ACLs is described in Section 5.

  The ACL-based authorization model is described in Section 7.4

  The additional security-related services are described in Section 12. These also rely on the authentication provided by RPC.

- Because there are two different approaches to file-access authorization, servers might implement both, in which case the associated attributes need to be coordinated as described in Section 9.

- NFSv4.2 provides an file access authorization model oriented toward Mandatory Access Control. It is described in Section 10. For reasons described there, its security properties are hard to analyze in detail and this document will not consider it as part of the NFSv4 threat analysis.

Authorization of locking state modification is discussed in Section 11. This form of authorization relies on the authentication of the client peer as opposed to file access authorization, which relies on authentication of the client principal.

5. Structure of Access Control Lists

Access Control Lists consisting of multiple Access Control Elements, while originally designed to support a more flexible authorization model, have multiple uses within NFSv4, with the use of each element depending on its type, as defined in Section 5.2

- They may be used to provide a more flexible authorization model as described in Section 7.4. This involves use of Access Control Entries of the ALLOW and DENY types.

- They may be used to provide the security-related services described in Section 12. This involves use of Access Control Entries of the AUDIT and ALARM types.

Subsections of this section define the structure of ACLs and discuss ACL-related matters that apply to multiple types of ACL use, including the definitions of the acl and aclsupport attributes.

Matters that relate to only a single one of these use classes, including the definition of the NFSv4.1-specific attributes dacl and sacl, are discussed in subsections of Sections 7.4 or 12.

5.1. Access Control Entries

The attributes acl, sacl (v4.1 only) and dacl (v4.1 only) each contain an array of Access Control Entries (ACEs) that are associated with the file system object. The client can set and get these attributes attribute, the server is responsible for using the ACL-related attributes to perform access control. The client can use the OPEN or ACCESS operations to check access without modifying or explicitly reading data or metadata.

The NFS ACE structure is defined as follows:

```
typedef uint32_t        acetype4;

typedef uint32_t aceflag4;

typedef uint32_t        acemask4;

struct nfsace4 {
        acetype4        type;
        aceflag4        flag;
        acemask4        access_mask;
        utf8str_mixed   who;
};
```

5.2. ACE Type

The constants used for the type field (acetype4) are as follows:

```
const ACE4_ACCESS_ALLOWED_ACE_TYPE      = 0x00000000;
const ACE4_ACCESS_DENIED_ACE_TYPE       = 0x00000001;
const ACE4_SYSTEM_AUDIT_ACE_TYPE        = 0x00000002;
const ACE4_SYSTEM_ALARM_ACE_TYPE        = 0x00000003;
```

All four are permitted in the acl attribute. For NFSv4.1 and beyond, only the ALLOWED and DENIED types may be used in the dacl attribute, and only the AUDIT and ALARM types.x used in the sacl attribute.

Value	Abbreviation	Description
ACE4_ACCESS_ALLOWED_ACE_TYPE	ALLOW	Explicitly grants the ability to perform the action specified in

Value	Abbreviation	Description
		acemask4 to the file or directory.
ACE4_ACCESS_DENIED_ACE_TYPE	DENY	Explicitly denies the ability to perform the action specified in acemask4 to the file or directory.
ACE4_SYSTEM_AUDIT_ACE_TYPE	AUDIT	Log (in a system-dependent way) any attempt to perform, for the file or directory, any of the actions specified in acemask4.
ACE4_SYSTEM_ALARM_ACE_TYPE	ALARM	Generate an alarm (in a system-dependent way) any attempt to perform, for the file or directory, any of the actions specified in acemask4.

Table 1

The "Abbreviation" column denotes how the types will be referred to throughout the rest of this document.

5.3. ACE Access Mask

The bitmask constants used for the access mask field of the ACE are as follows:

```
const ACE4_READ_DATA            = 0x00000001;
const ACE4_LIST_DIRECTORY       = 0x00000001;
const ACE4_WRITE_DATA           = 0x00000002;
const ACE4_ADD_FILE             = 0x00000002;
const ACE4_APPEND_DATA          = 0x00000004;
const ACE4_ADD_SUBDIRECTORY     = 0x00000004;
const ACE4_READ_NAMED_ATTRS     = 0x00000008;
const ACE4_WRITE_NAMED_ATTRS    = 0x00000010;
const ACE4_EXECUTE              = 0x00000020;
const ACE4_DELETE_CHILD         = 0x00000040;
const ACE4_READ_ATTRIBUTES      = 0x00000080;
const ACE4_WRITE_ATTRIBUTES     = 0x00000100;
const ACE4_WRITE_RETENTION      = 0x00000200;
const ACE4_WRITE_RETENTION_HOLD = 0x00000400;

const ACE4_DELETE               = 0x00010000;
const ACE4_READ_ACL             = 0x00020000;
const ACE4_WRITE_ACL            = 0x00040000;
const ACE4_WRITE_OWNER          = 0x00080000;
const ACE4_SYNCHRONIZE          = 0x00100000;
```

Note that some masks have coincident values, for example, ACE4_READ_DATA and ACE4_LIST_DIRECTORY. The mask entries ACE4_LIST_DIRECTORY, ACE4_ADD_FILE, and ACE4_ADD_SUBDIRECTORY are intended to be used with directory objects, while ACE4_READ_DATA, ACE4_WRITE_DATA, and ACE4_APPEND_DATA are intended to be used with non-directory objects.

5.4. Uses of Mask Bits

[Author Aside]: Because this section has been moved to be part of a general description of ACEs, not limited to authorization, the descriptions no longer refer to permissions but rather to actions. This is best considered a purely editorial change, but, to allow for possible disagreement on the matter, it will be considered, here and in Appendix B, as consensus item #3a.

[Author Aside]: In a large number of places, **SHOULD** is used inappropriately, since there appear to be no valid reasons to allow a server to ignore what might well be a requirement. Such changes are not noted individually below. However, they will be considered, here and in Appendix B, as consensus item #4a.

[Author Aside}: In a significant number of cases the ACCESS operation is not listed as a operation affected by the mask bit. These additions are not noted individually below. However, they will be considered, here and in Appendix B, as consensus item #5a.

[Author Aside, Including List]: In a number of cases, there are additional changes which go beyond editorial or arguably do so. These will be marked as their own consensus items usually with an accompanying author aside but without necessarily citing the previous treatment. These include:

- *Revisions were necessary to clarify the relationship between READ_DATA and EXECUTE. These are part of consensus item #7a.

- *Revisions were necessary to clarify the relationship between WRITE_DATA and APPEND_DATA. These are part of consensus item #8a.

- *Clarification of the handling of RENAME by ADD_SUBDIRECTORY. This is part of consensus item #9a.

- *Revisions in handling of the masks WRITE_RETENTION and WRITE_RETENTION_HOLD. These are parts of consensus items #10a.

[Author Aside]: Because of the need to address sticky-bit issues as part of of the ACE mask descriptions, it is appropriate to introduce the subject here.

[Consensus Item (Item #6a)]: Despite the fact that ACLs and mode bits are separate means of authentication, it has been necessary, even if only for the purpose of providing compatibility with earlier implementations, to introduce the issue here, since reference to this mode bit are necessary to resolve issues regard directory entry deletion, as is done in Section 5.6.

[Consensus Item, Including List (Item #6a): The full description of the role of the sticky-bit appears in Section 7.3.1. In evaluating and understanding the relationship between the handling of this bit when ACLs are used and when they are not, the following points need to be kept in mind:

- *This is troublesome in that it combines data normally assigned to two different authorization models and breaks the overall architectural arrangement in which the mask bits represent the mode bits but provide a finer granularity of control.

- *It might have been possible to conform to the existing architectural model if a new mask bit were created to represent to directory sticky bit. It is probably too late to so now, even though it would be allowed as an NFSv4.2 extension.

- *The new treatment in Section 5.6 is more restrictive than the previous one appearing in Section 5.6.1. This raises potential compatibility issues since the new treatment, while designed to address the same issues was designed to match existing Unix handling of this bit.

*This handling initially addresses REMOVE and does not address directory sticky bit semantics with regard to RENAME. Whether it should do so is still uncertain.

*The handling of this mode bit was not documented in previous specifications. However, there is a preliminary attempt to do so in Section 7.3.1. The reason for doing so, is that given the Unix orientation of the mode attribute, it is likely that servers currently implement this, even though there is no NFSv4 documentation of this semantics

This treatment needs to be checked for compatibility issues and also to establish a model that we might adapt to the ACL case.

*In the long term, it would make more sense to allow the client rather than the server to have the primary role in determining the semantics for things like this. That does not seem possible right now but it is worth considering.

ACE4_READ_DATA

Operation(s) affected:

READ

OPEN

ACCESS

Discussion:

The action of reading the data to the data of the file.

[Previous Treatment (Item #7a)]: Servers **SHOULD** allow a user the ability to read the data of the file when only the ACE4_EXECUTE access mask bit is allowed.

[Author Aside]: The treatment needs to be clarified to make it appropriate to all ACE types.

[Consensus Needed (Item #7a)]: When used to handle READ or OPEN operations, the handling **MUST** be identical whether this bit, ACE4_EXECUTE, or both are present, as the server has no way of determining whether a file is being read for execution are not. The only occasion for different handling is in construction of a corresponding mode or in responding to the ACCESS operation.

ACE4_LIST_DIRECTORY

Operation(s) affected:

READDIR

Discussion:
The action of listing the contents of a directory.

ACE4_WRITE_DATA

Operation(s) affected:
WRITE

OPEN

ACCESS

SETATTR of size

Discussion:
[Author Aside]: Needs to be revised to deal with issues related to the interaction of WRITE_DATA and APPEND_DATA.

[Consensus Needed (Item #8a)]: The action of modifying existing data bytes within a file's current data.

[Consensus Needed (Item #8a)]: As there is no way for the server to decide, in processing an OPEN or ACCESS request, whether subsequent WRITEs will extend the file or not, the server will, in processing an OPEN treat masks containing only WRITE_DATA, only APPEND_DATA, or both identically.

[Consensus Needed (Item #8a)]: In processing a WRITE request, the server is presumed to have the to determine whether the current request extends the file and whether it modifies bytes already in the file.

[Consensus Needed (Item #8a)]: ACE4_WRITE_DATA is required to process a WRITE which spans pre-existing byte in the file, whether the file is extended or not.

ACE4_ADD_FILE

Operation(s) affected:
CREATE

LINK

OPEN

RENAME

Discussion:
The action of adding a new file in a directory. The CREATE operation is affected when nfs_ftype4 is NF4LNK, NF4BLK,

NF4CHR, NF4SOCK, or NF4FIFO. (NF4DIR is not included because it is covered by ACE4_ADD_SUBDIRECTORY.) OPEN is affected when used to create a regular file. LINK and RENAME are always affected.

ACE4_APPEND_DATA

Operation(s) affected:

WRITE

ACCESS

OPEN

SETATTR of size

Discussion:

[Author Aside]: Also needs to be revised to deal with issues related to the interaction of WRITE_DATA and APPEND_DATA.

The action of modifying a file's data, but only starting at EOF. This allows for the specification of append-only files, by allowing ACE4_APPEND_DATA and denying ACE4_WRITE_DATA to the same user or group.

[Consensus Needed (Item #8a)]: As there is no way for the server to decide, in processing an OPEN or ACCESS request, whether subsequent WRITEs will extend the file or not, the server will treat masks containing only WRITE_DATA, only APPEND_DATA or both, identically.

[Consensus Needed (Item #8a)]: If the server is processing a WRITE request and the area to be written extends beyond the existing EOF of the file then the state of APPEND_DATA mask bit is consulted to determine whether the operation is permitted or whether alarm or audit activities are to be performed. If a file has an ACL allowing only APPEND_DATA (and not WRITE_DATA) and a WRITE request is made at an offset below EOF, the server **MUST** return NFS4ERR_ACCESS.

[Consensus Needed (Item #8a)]: If the server is processing a WRITE request and the area to be written does not extend beyond the existing EOF of the file then the state of APPEND_DATA mask bit does not need to be consulted to determine whether the operation is permitted or whether alarm or audit activities are to be performed. In this case, only the WRITE_DATA mask bit needs to be checked to determine whether the WRITE is authorized.

ACE4_ADD_SUBDIRECTORY

Operation(s) affected:

CREATE

RENAME

Discussion:

[Author Aside]: The RENAME cases need to be limited to the renaming of directories, rather than saying, "The RENAME operating is always affected."

[Consensus Needed (Item #9a)]: The action of creating a subdirectory in a directory. The CREATE operation is affected when nfs_ftype4 is NF4DIR. The RENAME operation is always affected when directories are renamed and the target directory ACL contains the mask ACE4_ADD_SUBDIRECTORY.

ACE4_READ_NAMED_ATTRS

Operation(s) affected:

OPENATTR

Discussion:

The action of reading the named attributes of a file or of looking up the named attribute directory. OPENATTR is affected when it is not used to create a named attribute directory. This is when 1) createdir is TRUE, but a named attribute directory already exists, or 2) createdir is FALSE.

ACE4_WRITE_NAMED_ATTRS

Operation(s) affected:

OPENATTR

Discussion:

The action of writing the named attributes of a file or creating a named attribute directory. OPENATTR is affected when it is used to create a named attribute directory. This is when createdir is TRUE and no named attribute directory exists. The ability to check whether or not a named attribute directory exists depends on the ability to look it up; therefore, users also need the ACE4_READ_NAMED_ATTRS permission in order to create a named attribute directory.

ACE4_EXECUTE

Operation(s) affected:

READ

OPEN

ACCESS

REMOVE

RENAME

LINK

CREATE

Discussion:
The action of reading a file in order to execute it.

Servers **MUST** allow a user the ability to read the data of the file when only the ACE4_EXECUTE access mask bit is allowed. This is because there is no way to execute a file without reading the contents. Though a server may treat ACE4_EXECUTE and ACE4_READ_DATA bits identically when deciding to permit a READ or OPEN operation, it **MUST** still allow the two bits to be set independently in ACLs, and distinguish between them when replying to ACCESS operations. In particular, servers **MUST NOT** silently turn on one of the two bits when the other is set, as that would make it impossible for the client to correctly enforce the distinction between read and execute permissions.

As an example, following a SETATTR of the following ACL:

nfsuser:ACE4_EXECUTE:ALLOW

A subsequent GETATTR of ACL for that file will return:

nfsuser:ACE4_EXECUTE:ALLOW

and **MUST NOT** return:

nfsuser:ACE4_EXECUTE/ACE4_READ_DATA:ALLOW

ACE4_EXECUTE

Operation(s) affected:
LOOKUP

Discussion:
The action of traversing/searching a directory.

ACE4_DELETE_CHILD

Operation(s) affected:
REMOVE

RENAME

Discussion:

The action of deleting a file or directory within a directory. See Section 5.6 for information on now ACE4_DELETE and ACE4_DELETE_CHILD are to interact.

ACE4_READ_ATTRIBUTES

Operation(s) affected:

GETATTR of file system object attributes

VERIFY

NVERIFY

READDIR

Discussion:

The action of reading basic attributes (non-ACLs) of a file. On a UNIX system, such basic attributes can be thought of as the stat-level attributes. Allowing this access mask bit would mean that the entity can execute "ls -l" and stat. If a READDIR operation requests attributes, this mask must be allowed for the READDIR to succeed.

ACE4_WRITE_ATTRIBUTES

Operation(s) affected:

SETATTR of time_access_set, time_backup, time_create, time_modify_set, mimetype, hidden, system.

Discussion:

The action of changing the times associated with a file or directory to an arbitrary value. Also permission to change the mimetype, hidden, and system attributes. A user having ACE4_WRITE_DATA or ACE4_WRITE_ATTRIBUTES will be allowed to set the times associated with a file to the current server time.

ACE4_WRITE_RETENTION

Operation(s) affected:

SETATTR of retention_set, retentevt_set.

Discussion:

The action of modifying the durations for event and non-event-based retention. Also includes enabling event and non-event-based retention.

[Author Aside]: The use of "**MAY**" here ignores the potential for harm which unexpected modification of the associated attributes might cause for security/compliance.

[Previous Treatment]: A server **MAY** behave such that setting ACE4_WRITE_ATTRIBUTES allows ACE4_WRITE_RETENTION.

[Consensus Needed (Items #10a, #11a)]: Options for coarser-grained treatment involving this mask bit are discussed in Section 5.5

ACE4_WRITE_RETENTION_HOLD

Operation(s) affected:
SETATTR of retention_hold.

Discussion:
The action of modifying the administration retention holds.

[Previous Treatment]: A server **MAY** map ACE4_WRITE_ATTRIBUTES to ACE_WRITE_RETENTION_HOLD.

[Author Aside]: The use of "**MAY**" here ignores the potential for harm which unexpected modification of the associated attributes might cause for security/compliance.

[Consensus Needed (Items #10a, #11a)]: Options for coarser-grained treatment of this mask bit are discussed in Section 5.5

ACE4_DELETE

Operation(s) affected:
REMOVE

Discussion:
The action of deleting the associated file or directory. See Section 5.6 for information on how ACE4_DELETE and ACE4_DELETE_CHILD are to interact.

ACE4_READ_ACL

Operation(s) affected:
GETATTR of acl, dacl, or sacl

NVERIFY

VERIFY

Discussion:
The action of reading the ACL.

ACE4_WRITE_ACL

Operation(s) affected:
SETATTR of acl and mode

Discussion:
The action of modifying the acl or mode attributes.

ACE4_WRITE_OWNER

Operation(s) affected:
SETATTR of owner and owner_group

Discussion:
The action of modifying the owner or owner_group attributes. On UNIX systems, this done by executing chown() and chgrp().

ACE4_SYNCHRONIZE

Operation(s) affected:
NONE

Discussion:
Permission to use the file object as a synchronization primitive for interprocess communication. This permission is not enforced or interpreted by the NFSv4.1 server on behalf of the client.

Typically, the ACE4_SYNCHRONIZE permission is only meaningful on local file systems, i.e., file systems not accessed via NFSv4.1. The reason that the permission bit exists is that some operating environments, such as Windows, use ACE4_SYNCHRONIZE.

For example, if a client copies a file that has ACE4_SYNCHRONIZE set from a local file system to an NFSv4.1 server, and then later copies the file from the NFSv4.1 server to a local file system, it is likely that if ACE4_SYNCHRONIZE was set in the original file, the client will want it set in the second copy. The first copy will not have the permission set unless the NFSv4.1 server has the means to set the ACE4_SYNCHRONIZE bit. The second copy will not have the permission set unless the NFSv4.1 server has the means to retrieve the ACE4_SYNCHRONIZE bit.

5.5. Requirements and Recommendations Regarding Mask Granularity

This is new section which replaces material formerly in the previous section, cited here as "Previous Treatment. The new material, constituting the remainder of the section is proposed to replace it. All such unannotated material is to be considered, as part of consensus item #11b.

[Previous Treatment (Item #11b)]: Server implementations need not provide the granularity of control that is implied by this list of masks. For example, POSIX-based systems might not distinguish ACE4_APPEND_DATA (the ability to append to a file) from ACE4_WRITE_DATA (the ability to modify existing contents); both masks would be tied to a single "write" permission bit. When such a server returns attributes to the client that contain such masks, it would show ACE4_APPEND_DATA and ACE4_WRITE_DATA if and only if the the write permission bit is enabled.

[Previous Treatment (Item #11b)]: If a server receives a SETATTR request that it cannot accurately implement, it should err in the direction of more restricted access, except in the previously discussed cases of execute and read. For example, suppose a server cannot distinguish overwriting data from appending new data, as described in the previous paragraph. If a client submits an ALLOW ACE where ACE4_APPEND_DATA is set but ACE4_WRITE_DATA is not (or vice versa), the server should either turn off ACE4_APPEND_DATA or reject the request with NFS4ERR_ATTRNOTSUPP.

[Author Aside]: Giving servers a general freedom to to not support the masks defined in this section, creates an unacceptable level of potential interoperability problems. With regard to the specific example given, it is hard to imagine a server incapable of distinguishing a write to an offset within existing file and one beyond it. This applies whether the server in question is implemented within a POSIX-based system or not. It is true that a server that used the unmodified POSIX interface to interact with the file system, rather than a purpose-built VFS, would face this difficulty, but it not clear that that fact justifies the client compatibility issues that accommodating this behavior in the protocol would generate. A further difficulty with the previous treatment is that it at variance with the approach to other cases in which ACEs are stored with the understanding that implementations of other protocols might be responsible for enforcement.

[Author Aside]: A replacement should clearly be based on the idea that these mask bits were included in NFSv4 for a reason, and that exceptions need to be justified, and take interoperability issues into account. The treatment below attempts to do that.

All implementations of the acl, dacl, and sacl attributes **SHOULD** follow the definitions provided above in Section 5.4, which allow finer-grained control of the actions allowed to specific users than is provided by the mode attribute. Valid reasons to bypass this guidance include the need for compatibility with clients expecting a coarser-grained implementation.

The specific cases in which servers may validly provide coarser-grained implementations are discussed below.

Servers not providing the mask granularity described in Section 5.4 **MUST NOT** treat masks other than described in that section except as listed below.

- Servers that do not distinguish between WRITE_DATA and APPEND_DATA need to make it clear to clients that support for append-only files is not present. To do that, requests to set ACLs where the handling for these two masks are different for any specified user or group are to be rejected with NFS4ERR_ATTRNOTSUPP.

- [Consensus Needed (Items #10b, #11b)]: Servers that combine either of the masks WRITE_RETENTION or WRITE_RETENTION_HOLD with WRITE_ATTRIBUTES need to make it clear to clients that the finer-grained treatment normally expected is not available. To do that, requests to set ACLs in which the two combined masks are explicitly assigned different permission states (i.e. one is ALLOWED while the other is DENIED) for any specific user or group are to be rejected with NFS4ERR_ATTRNOTSUPP.

The above are in line with the requirement that attempts to set ACLs that the server cannot enforce, it needs to be clear that there are cases in which such ACLs need to be set with the expectation that enforcement will be done by the local file system or by another file access protocol. In particular,

- In handling the mask bit SYNCHRONIZE, the server is not responsible for enforcement and so can accept ACLs it has no way of enforcing.

- When mask bits refers to an OPTIONAL feature that the server does not support such as named attributes or retention attributes, the server is allowed to accept ACLs containing mask bits associated with the unimplemented feature, even though there is no way these cold be enforced. The expectation is that the files might be accessed by other protocols having such support or might be copied, together with associated ACLs to severs capable of enforcing them.

5.6. Handling of Deletion

[Author Aside]: This section, exclusive of subsections contains a proposal for the revision of the ACL-based handling of requests to delete directory entries. All unannotated material within it is to be considered part of consensus item #12a.

[Author Aside]: The associated previous treatment is to be found in Section 5.6.1

This section describes the handling requests of that involve deletion of a directory entry. It needs to be noted that:

- Modification or transfer of a directory, as happens in RENAME is not covered.

- The deletion of the file's data is dealt with separately as this, like a truncation to length zero, requires ACE4_WRITE_DATA.

In general, the recognition of such an operation for authorization/auditing/alarm depends on either of two bits mask bits being set: ACE4_MASK_DELETE on the file being deleted and ACE4_MASK_DELETE_CHILD on the directory from which the entry is being deleted.

In the case of authorization, the above applies even when one of the bits is allowed and the other is explicitly denied.

[Consensus Items, Including List (#6b, #12a): When neither of the mask bits is set, the result is normally negative. That is, permission is denied and no audit or alarm event is recognized. However, in the case of authorization, the server **MAY** make permission dependent on the setting of MODE4_SVTX if the mode attribute is supported, as follows:

- If that bit not set, allow the removal if and only if ACE4_ADD_FILE is permitted.

- If that bit is set, allow the removal if and only if ACE4_ADD_FILE is permitted and the requester is the owner of the target.

5.6.1. Previous Handling of Deletion

[Author Aside]: This section contains the former content of Section 5.6. All unannotated paragraphs within it are to be considered the Previous Treatment associated with consensus item #12b.

[Author Aside, Including List]: Listed below are some of the reasons that I have tried to replace the existing treatment rather than address the specific issues mentioned here and in later asides.

- The fact that there is no clear message about what servers are to do and about whether behavior clients might rely rely on. This derives in turn from the use of SHOULD in contexts in which it is clearly not appropriate, combined with non-normative reports of what some systems do, and the statement that the approach suggested is a way of providing "something like traditional UNIX-like semantics".

- The complexity of the approach without any indication that there is a need for such complexity makes me doubtful that anything was actually implemented, especially since the text is so wishy-washy about the need for server implementation. The probability that it would be implemented so widely that clients might depend on it is even more remote.

- The fact that how audit and alarm issues are to be dealt with is not addressed at all.

- The fact that this treatment combines ACL data with mode bit information in a confused way without any consideration of the fact that the mode attribute is OPTIONAL.

Two access mask bits govern the ability to delete a directory entry: ACE4_DELETE on the object itself (the "target") and ACE4_DELETE_CHILD on the containing directory (the "parent").

Many systems also take the "sticky bit" (MODE4_SVTX) on a directory to allow unlink only to a user that owns either the target or the parent; on some such systems the decision also depends on whether the target is writable.

Servers **SHOULD** allow unlink if either ACE4_DELETE is permitted on the target, or ACE4_DELETE_CHILD is permitted on the parent. (Note that this is true even if the parent or target explicitly denies one of these permissions.)

If the ACLs in question neither explicitly ALLOW nor DENY either of the above, and if MODE4_SVTX is not set on the parent, then the server **SHOULD** allow the removal if and only if ACE4_ADD_FILE is permitted. In the case where MODE4_SVTX is set, the server may also require the remover to own either the parent or the target, or may require the target to be writable.

This allows servers to support something close to traditional UNIX-like semantics, with ACE4_ADD_FILE taking the place of the write bit.

5.7. ACE flag bits

The bitmask constants used for the flag field are as follows:

```
const ACE4_FILE_INHERIT_ACE             = 0x00000001;
const ACE4_DIRECTORY_INHERIT_ACE        = 0x00000002;
const ACE4_NO_PROPAGATE_INHERIT_ACE     = 0x00000004;
const ACE4_INHERIT_ONLY_ACE             = 0x00000008;
const ACE4_SUCCESSFUL_ACCESS_ACE_FLAG   = 0x00000010;
const ACE4_FAILED_ACCESS_ACE_FLAG       = 0x00000020;
const ACE4_IDENTIFIER_GROUP             = 0x00000040;
const ACE4_INHERITED_ACE                = 0x00000080;
```

[Author Aside]: Although there are multiple distinct issues that might need to be changed, in the interest of simplifying the review, all such issues within this section will be considered part of Consensus Item #13a with a single revised treatment addressing all the issues noted.

[Previous Treatment]: A server need not support any of these flags.

[Author Aside]: It is hard to understand why such broad license is granted to the server, leaving the client to deal, without an explicit non-support indication, with 256 possible combinations of supported and unsupported flags. If there were specific issues with some flags that makes it reasonable for a server not to support them, then these need to be specifically noted. Also problematic is the use of the term "need not", suggesting that the server does not need any justification for choosing these flags, defined by the protocol. At least it needs to be said that the server **SHOULD** support the defined ACE flags. After all they were included in the protocol for a reason.

[Previous Treatment]: If the server supports flags that are similar to, but not exactly the same as, these flags, the implementation may define a mapping between the protocol-defined flags and the implementation-defined flags.

[Author Aside]: The above dealing how an implementation might store the bits it support, while valid, is out-of-scope and should be deleted.

[Previous Treatment]: For example, suppose a client tries to set an ACE with ACE4_FILE_INHERIT_ACE set but not ACE4_DIRECTORY_INHERIT_ACE. If the server does not support any form of ACL inheritance, the server should reject the request with NFS4ERR_ATTRNOTSUPP. If the server supports a single "inherit ACE" flag that applies to both files and directories, the server may

reject the request (i.e., requiring the client to set both the file and directory inheritance flags). The server may also accept the request and silently turn on the ACE4_DIRECTORY_INHERIT_ACE flag.

]Author Aside]: What is the possible for justification for accepting a request asking you do something and then, without notice to the client do, something else. I believe there is none.

Consensus Needed (Item #13a)]: Servers **SHOULD** support the flag bits defined above as described in Section 5.8. When a server which does not support all the flags bits receives a request to set an ACL containing an ACE with an unsupported flag bit set the server **MUST** reject the request with NFS4ERR_ATTRNOTSUPP.

Consensus Needed (Item #13a)]: The case of servers which do not provide support for particular flag combinations is to be treated similarly. If a server supports a single "inherit ACE" flag that applies to both files and directories, receives a request set an ACL with ACE ACE4_FILE_INHERIT_ACE set but ACE4_DIRECTORY_INHERIT_ACE not set, it **MUST** reject the request with NFS4ERR_ATTRNOTSUPP.

5.8. Details Regarding ACE Flag Bits

ACE4_FILE_INHERIT_ACE
: Any non-directory file in any sub-directory will get this ACE inherited.

ACE4_DIRECTORY_INHERIT_ACE
: Can be placed on a directory and indicates that this ACE should be added to each new directory created.

  If this flag is set in an ACE in an ACL attribute to be set on a non-directory file system object, the operation attempting to set the ACL **SHOULD** fail with NFS4ERR_ATTRNOTSUPP.

ACE4_NO_PROPAGATE_INHERIT_ACE
: Can be placed on a directory. This flag tells the server that inheritance of this ACE should stop at newly created child directories.

ACE4_INHERIT_ONLY_ACE
: Can be placed on a directory but does not apply to the directory; ALLOW and DENY ACEs with this bit set do not affect access to the directory, and AUDIT and ALARM ACEs with this bit set do not trigger log or alarm events. Such ACEs only take effect once they are applied (with this bit cleared) to newly created files and directories as specified by the ACE4_FILE_INHERIT_ACE and ACE4_DIRECTORY_INHERIT_ACE flags.

If this flag is present on an ACE, but neither ACE4_DIRECTORY_INHERIT_ACE nor ACE4_FILE_INHERIT_ACE is present, then an operation attempting to set such an attribute **SHOULD** fail with NFS4ERR_ATTRNOTSUPP.

ACE4_SUCCESSFUL_ACCESS_ACE_FLAG and ACE4_FAILED_ACCESS_ACE_FLAG

The ACE4_SUCCESSFUL_ACCESS_ACE_FLAG (SUCCESS) and ACE4_FAILED_ACCESS_ACE_FLAG (FAILED) flag bits may be set only on ACE4_SYSTEM_AUDIT_ACE_TYPE (AUDIT) and ACE4_SYSTEM_ALARM_ACE_TYPE (ALARM) ACE types. If during the processing of the file's ACL, the server encounters an AUDIT or ALARM ACE that matches the principal attempting the OPEN, the server notes that fact, and the presence, if any, of the SUCCESS and FAILED flags encountered in the AUDIT or ALARM ACE. Once the server completes the ACL processing, it then notes if the operation succeeded or failed. If the operation succeeded, and if the SUCCESS flag was set for a matching AUDIT or ALARM ACE, then the appropriate AUDIT or ALARM event occurs. If the operation failed, and if the FAILED flag was set for the matching AUDIT or ALARM ACE, then the appropriate AUDIT or ALARM event occurs. Either or both of the SUCCESS or FAILED can be set, but if neither is set, the AUDIT or ALARM ACE is not useful.

The previously described processing applies to ACCESS operations even when they return NFS4_OK. For the purposes of AUDIT and ALARM, we consider an ACCESS operation to be a "failure" if it fails to return a bit that was requested and supported.

ACE4_IDENTIFIER_GROUP

Indicates that the "who" refers to a GROUP as defined under UNIX or a GROUP ACCOUNT as defined under Windows. Clients and servers **MUST** ignore the ACE4_IDENTIFIER_GROUP flag on ACEs with a who value equal to one of the special identifiers outlined in Section 5.9.

ACE4_INHERITED_ACE

Indicates that this ACE is inherited from a parent directory. A server that supports automatic inheritance will place this flag on any ACEs inherited from the parent directory when creating a new object. Client applications will use this to perform automatic inheritance. Clients and servers **MUST** clear this bit in the acl attribute; it may only be used in the dacl and sacl attributes.

5.9. ACE Who

The "who" field of an ACE is an identifier that specifies the principal or principals to whom the ACE applies. It may refer to a

user or a group, with the flag bit ACE4_IDENTIFIER_GROUP specifying which.

There are several special identifiers that need to be understood universally, rather than in the context of a particular DNS domain. Some of these identifiers cannot be understood when an NFS client accesses the server, but have meaning when a local process accesses the file. The ability to display and modify these permissions is permitted over NFS, even if none of the access methods on the server understands the identifiers.

Who	Description
OWNER	The owner of the file.
GROUP	The group associated with the file.
EVERYONE	The world, including the owner and owning group.
INTERACTIVE	Accessed from an interactive terminal.
NETWORK	Accessed via the network.
DIALUP	Accessed as a dialup user to the server.
BATCH	Accessed from a batch job.
ANONYMOUS	Accessed without any authentication.
AUTHENTICATED	Any authenticated user (opposite of ANONYMOUS).
SERVICE	Access from a system service.

Table 2

To avoid conflict, these special identifiers are distinguished by an appended "@" and should appear in the form "xxxx@" (with no domain name after the "@"), for example, ANONYMOUS@.

The ACE4_IDENTIFIER_GROUP flag **MUST** be ignored on entries with these special identifiers. When encoding entries with these special identifiers, the ACE4_IDENTIFIER_GROUP flag **SHOULD** be set to zero.

[working group input needed]: I don't understand what might be valid reasons to ignore this. Would "**MUST**" be appropriate here?

It is important to note that "EVERYONE@" is not equivalent to the UNIX "other" entity. This is because, by definition, UNIX "other" does not include the owner or owning group of a file. "EVERYONE@" means literally everyone, including the owner or owning group.

[working group input needed]: Some of these require that changes be made as discussed below:

* For "INTERACTIVE", "NETWORK", "DIALUP", "BATCH", and "SERVICE" it needs to be specified that server's ability to make these distinctions is limited, making their use where security is an issue quite problematic.

*For "ANONYMOUS", clearly requests using AUTH_NONE fit but what else?

Request by nobody and by users root-squashed to nobody are probably OK, although you could argue about the case of a user "nobody" authenticated by RPCSEC_GSS.

ON a more contentious note, I would argue that users "authenticated" using AUTH_SYS, in the clear, without client-peer authentication fit here, but we need to get to consensus on this point.

*Issues regarding "AUTHENTICATED" will be the mirror image of those for "ANONYMOUS".

5.10. Automatic Inheritance Features

The acl attribute consists only of an array of ACEs, but the sacl (Section 12.1) and dacl (Section 7.4.2) attributes also include an additional flag field.

```
struct nfsacl41 {
        aclflag4        na41_flag;
        nfsace4         na41_aces<>;
};
```

The flag field applies to the entire sacl or dacl; three flag values are defined:

```
const ACL4_AUTO_INHERIT         = 0x00000001;
const ACL4_PROTECTED            = 0x00000002;
const ACL4_DEFAULTED            = 0x00000004;
```

and all other bits must be cleared. The ACE4_INHERITED_ACE flag may be set in the ACEs of the sacl or dacl (whereas it must always be cleared in the acl).

Together these features allow a server to support automatic inheritance, which we now explain in more detail.

Inheritable ACEs are normally inherited by child objects only at the time that the child objects are created; later modifications to inheritable ACEs do not result in modifications to inherited ACEs on descendants.

However, the dacl and sacl provide an **OPTIONAL** mechanism that allows a client application to propagate changes to inheritable ACEs to an entire directory hierarchy.

A server that supports this feature performs inheritance at object creation time in the normal way, and **SHOULD** set the ACE4_INHERITED_ACE flag on any inherited ACEs as they are added to the new object.

A client application such as an ACL editor may then propagate changes to inheritable ACEs on a directory by recursively traversing that directory's descendants and modifying each ACL encountered to remove any ACEs with the ACE4_INHERITED_ACE flag and to replace them by the new inheritable ACEs (also with the ACE4_INHERITED_ACE flag set). It uses the existing ACE inheritance flags in the obvious way to decide which ACEs to propagate. (Note that it may encounter further inheritable ACEs when descending the directory hierarchy and that those will also need to be taken into account when propagating inheritable ACEs to further descendants.)

The reach of this propagation may be limited in two ways: first, automatic inheritance is not performed from any directory ACL that has the ACL4_AUTO_INHERIT flag cleared; and second, automatic inheritance stops wherever an ACL with the ACL4_PROTECTED flag is set, preventing modification of that ACL and also (if the ACL is set on a directory) of the ACL on any of the object's descendants.

This propagation is performed independently for the sacl and the dacl attributes; thus, the ACL4_AUTO_INHERIT and ACL4_PROTECTED flags may be independently set for the sacl and the dacl, and propagation of one type of acl may continue down a hierarchy even where propagation of the other acl has stopped.

New objects should be created with a dacl and a sacl that both have the ACL4_PROTECTED flag cleared and the ACL4_AUTO_INHERIT flag set to the same value as that on, respectively, the sacl or dacl of the parent object.

Both the dacl and sacl attributes are Recommended, and a server may support one without supporting the other.

A server that supports both the old acl attribute and one or both of the new dacl or sacl attributes must do so in such a way as to keep all three attributes consistent with each other. Thus, the ACEs reported in the acl attribute should be the union of the ACEs reported in the dacl and sacl attributes, except that the ACE4_INHERITED_ACE flag must be cleared from the ACEs in the acl. And of course a client that queries only the acl will be unable to determine the values of the sacl or dacl flag fields.

When a client performs a SETATTR for the acl attribute, the server **SHOULD** set the ACL4_PROTECTED flag to true on both the sacl and the dacl. By using the acl attribute, as opposed to the dacl or sacl attributes, the client signals that it may not understand automatic inheritance, and thus cannot be trusted to set an ACL for which automatic inheritance would make sense.

When a client application queries an ACL, modifies it, and sets it again, it should leave any ACEs marked with ACE4_INHERITED_ACE unchanged, in their original order, at the end of the ACL. If the application is unable to do this, it should set the ACL4_PROTECTED flag. This behavior is not enforced by servers, but violations of this rule may lead to unexpected results when applications perform automatic inheritance.

If a server also supports the mode attribute, it **SHOULD** set the mode in such a way that leaves inherited ACEs unchanged, in their original order, at the end of the ACL. If it is unable to do so, it **SHOULD** set the ACL4_PROTECTED flag on the file's dacl.

Finally, in the case where the request that creates a new file or directory does not also set permissions for that file or directory, and there are also no ACEs to inherit from the parent's directory, then the server's choice of ACL for the new object is implementation-dependent. In this case, the server **SHOULD** set the ACL4_DEFAULTED flag on the ACL it chooses for the new object. An application performing automatic inheritance takes the ACL4_DEFAULTED flag as a sign that the ACL should be completely replaced by one generated using the automatic inheritance rules.

5.11. Attribute 13: aclsupport

A server need not support all of the above ACE types. This attribute indicates which ACE types are supported for the current file system. The bit mask constants used to represent the above definitions within the aclsupport attribute are as follows:

```
const ACL4_SUPPORT_ALLOW_ACL    = 0x00000001;
const ACL4_SUPPORT_DENY_ACL     = 0x00000002;
const ACL4_SUPPORT_AUDIT_ACL    = 0x00000004;
const ACL4_SUPPORT_ALARM_ACL    = 0x00000008;
```

[Author Aside]: Even though support aclsupport is OPTIONAL, there has been no mention of the possibility of it not being supported.

[Consensus Needed (Item #14a)]: If this attribute is not supported for a server, the client is entitled to assume that if the acl attribute is supported, support for ALLOW and DENY ACEs is present.

Thus, if such a server supports the the sacl attribute, clients are not likely to use it if aclsupport is not supported by the server.

[Previous Treatment]: Servers that support either the ALLOW or DENY ACE type **SHOULD** support both ALLOW and DENY ACE types.

[Author Aside]: It needs to be made clearer what the harm is that is to be prevented by this. Further if such harm exists, it is not clear what are the valid reasons not do this?

[Consensus Needed (Item #15a)]: There is little point in implementing a server which supports either ALLOW or DENY ACE types without supporting both. For reasons explained in Section 7.1 the ACL_based authorization cannot be used if only a single ACE type is available.

Clients should not attempt to set an ACE unless the server claims support for that ACE type. If the server receives a request to set an ACE that it cannot store, it **MUST** reject the request with NFS4ERR_ATTRNOTSUPP.

[Previous Treatment (Item #12c)]: If the server receives a request to set an ACE that it can store but cannot enforce, the server **SHOULD** reject the request with NFS4ERR_ATTRNOTSUPP.

[Author Aside]: Beyond the issues with the use of **SHOULD**, it is better to centralize this material and be clearer about the whole issue of ACL enforcement.

[Consensus Needed (Item #12c)]: The case of ACEs that cannot be enforced is similar, with the details of enforcement discussed in Section 5.5.

Support for any of the ACL attributes is OPTIONAL, although Recommended. However, a server (NFSv4.1 and above) that supports either of the new ACL attributes (dacl or sacl) **MUST** allow use of the new ACL attributes to access all of the ACE types that it supports. In other words, if a server which supports sacl or dacl supports ALLOW or DENY ACEs, then it **MUST** support the dacl attribute, and if it supports AUDIT or ALARM ACEs, then it **MUST** support the sacl attribute.

5.12. Attribute 12: acl

The acl attribute, as opposed to the sacl and dacl attributes, consists only of an ACE array and does not support automatic inheritance.

The acl attribute is recommended and there is no requirement that a server support it. However, when the dacl attribute is supported, it

is a good idea to provide support for the acl attribute as well, in order to accommodate clients that have not been upgraded to use the dacl attribute.

[Author Aside]: Although it has generally been assumed that changes to sacl and dacl attributes are to be visible in the acl and vice versa, NFSv4.1 specification do not appear to document this fact.

[Consensus Item, Including List (Item #16a)]: For NFSv4.1 servers that support Both the acl attribute and one or more of the sacl and dacl attributes, changes to the ACE's need to be immediately reflected in the other supported attributes:

*The result of reading the dacl attribute **MUST** consist of a set of ACEs that are exactly the same as the ACEs ALLOW and DENY ACEs within the the acl attribute, in the same order.

*The result of reading the sacl attribute **MUST** consist of a set of ACEs that are exactly the same as the ACEs AUDIT and ALARM ACEs within the the acl attribute, in the same order.

*The result of reading the acl attribute **MUST** consist of a set of ACEs that are exactly the same as the union of ACEs within the sacl and dacl attributes. Two ACEs that both appear in one of the sacl or dacl attributes must appear in the same order in the acl attribute.

6. Authorization in General

There are three distinct methods of checking whether NFSv4 requests are authorized:

*The most important method of authorization is used to effect user-based file access control, as described in Section 7.

This requires the identification of the user making the request. Because of the central role of such access control in providing NFSv4 security, server implementations **SHOULD NOT** use such identifications when they are not authenticated. In this context, valid reasons to do otherwise are limited to the compatibility and maturity issues discussed in Section 17.1.4

*NFSv4.2, via the labelled NFS feature, provides an additional potential requirement for request authorization.

For reasons made clear in Section 10, there is no realistic possibility of the server using the data defined by existing specifications of this feature to effect request authorization. While it is possible for clients to provide this authorization, the lack of detailed specifications makes it impossible to

determine the nature of the identification used and whether it can appropriately be described as "authentication".

*Since undesired changes to server-maintained locking state (and, for NFSv4.1, session state) can result in denial of service attacks (see Section 17.6), server implementations **SHOULD** take steps to prevent unauthorized state changes. This can be done by implementing the state authorization restrictions discussed in Section 11

7. User-based File Access Authorization

7.1. Attributes for User-based File Access Authorization

NFSv4.1 provides for multiple authentication models, controlled by the support for particular recommended attributes implemented by the server, as discussed below:

*Consensus Needed (Item #18a)]: The attributes owner, owning_group, and mode enable use of a POSIX-based authorization model, as described in Section 7.3. When all of these attributes are supported, this authorization model can be implemented.

Consensus Needed (Item #18a)]:When none of these attributes or only a proper subset of them are supported, this authorization model is unavailable.

*[Consensus Needed (Item #17a)]: The acl attribute (or the attribute dacl in NFSv4.1) can provide an ACL-based authorization model as described in Section 7.4 as long as support for ALLOW and DENY ACEs is provided.

[Consensus Needed (Items #17a, #18a)]: When some of these ACE types are not supported or the owner or owning_group attribute is not supported, this authorization model is unavailable, since there are some modes that cannot be represented as corresponding ACL, when using only a single ACE type. See Section 9.2 for details.

7.2. Handling of Multiple Parallel File Access Authorization Models

ACLs and modes represent two well-established models for specifying user-based file access permissions. NFSv4 provides support for either or both depending on the attributes supported by the server and, in cases in which both ACLs and the mode attribute are supported, the actual attributes set for a particular object.

*[Consensus Needed (item #18b)]: When the attributes mode, owner, owner group are all supported, the posix-based authorization model, described in Section 7.3 can be used.

* [Consensus Needed (Items #17b, #18b)]: When the acl (or dacl) attribute is supported together with both of the ACE types ALLOW and DENY, the acl based authorization model, described in Section 7.4 can be used as long as the attributes owner and owner_group are also supported.

[Consensus Needed (item #18b)]: While formally recommended (essentially **OPTIONAL**) attributes, it appears that the owner and owner_group attributes need to be available to support any file access authorization model. As a result, this document will not discuss the possibility of servers that do not support both of these attributes and clients have no need to support such servers.

When both authorization models can be used, there are difficulties that can arise because the ACL-based model provides finer-grained access control than the POSIX model. The ways of dealing with these difficulties appear later in this section while more detail on the appropriate handling of this situation, which might depend on the minor version used, appears in Section 9.

The following describe NFSv4's handling in supporting multiple authorization models for file access.

* If a server supports the mode attribute, it should provide the appropriate POSIX semantics if no ACL-based attributes have ever been assigned to object. These semantics include the restriction of the ability to modify the mode, owner and owner-group to the current owner of the file.

* If a server supports ACL attributes, it should provide ACL semantics as described in this document for all objects for which the ACL attributes have actually been set. This includes the ACL-based restrictions on the authorization to modify the mode, owner and owner_group attributes.

* On servers that support the mode attribute, if ACL attributes have never been set on an object, via inheritance or explicitly, the behavior should be the behavior mandated by POSIX, including the those that restrict the setting of authorization-related attributes.

* On servers that support the mode attribute, if the ACL attributes have been previously set on an object, either explicitly or via inheritance:

  - [Previous Treatment]: Setting only the mode attribute should effectively control the traditional UNIX-like permissions of read, write, and execute on owner, owner_group, and other.

[Author Aside]: It isn't really clear what the above paragraph means, especially as it governs the handling of aces designating specific users and groups which are not the owner and have no overlap with the owning group

{Consensus Needed (Item #19a)]: Setting only the mode attribute, should result in the access of the file being controlled just it would be if the existing acl did not exist, with file access decisions as to read made in accordance with the mode set. The ALLOW and DENY aces in the ACL should reflect the modified security although there is no need to modify AUDIT and ALARM aces or mask bits not affected by the mode bits, such as SYNCHRONIZE.

[Author Aside]: the above may need to modified to reflect the resolution of Consensus Item #??.

-[Previous Treatment]: Setting only the mode attribute should provide reasonable security. For example, setting a mode of 000 should be enough to ensure that future OPEN operations for OPEN4_SHARE_ACCESS_READ or OPEN4_SHARE_ACCESS_WRITE by any principal fail, regardless of a previously existing or inherited ACL.

[Author Aside]: We need to get rid of or provide some some replacement for the subjective first sentence. While the specific example give is unexceptionable, it raises questions in other cases as to what would constitutes "reasonable semantics". While the resolution of such questions would be subject to dispute, the author believes that consensus item #19a deals with the matter adequately. As a result he proposes, that the that this bullet be removed and the second-level list collapsed to single paragraph.

*Although RFCs 7530 and 8881 present different descriptions of the specific semantic requirements relating to the interaction of mode and ACL attributes, the difference are quite small, with the most important ones deriving from the absence of the set_mode_masked attribute. The unified treatment in Section 9 will indicate where version-specific differences exist.

7.3. Posix Authorization Model

7.3.1. Attribute 33: mode

The NFSv4.1 mode attribute is based on the UNIX mode bits. The following bits are defined:

```
const MODE4_SUID = 0x800;  /* set user id on execution */
const MODE4_SGID = 0x400;  /* set group id on execution */
const MODE4_SVTX = 0x200;  /* save text even after use */
const MODE4_RUSR = 0x100;  /* read permission: owner */
const MODE4_WUSR = 0x080;  /* write permission: owner */
const MODE4_XUSR = 0x040;  /* execute permission: owner */
const MODE4_RGRP = 0x020;  /* read permission: group */
const MODE4_WGRP = 0x010;  /* write permission: group */
const MODE4_XGRP = 0x008;  /* execute permission: group */
const MODE4_ROTH = 0x004;  /* read permission: other */
const MODE4_WOTH = 0x002;  /* write permission: other */
const MODE4_XOTH = 0x001;  /* execute permission: other */
```

Bits MODE4_RUSR, MODE4_WUSR, and MODE4_XUSR apply to the principal identified by the owner attribute. Bits MODE4_RGRP, MODE4_WGRP, and MODE4_XGRP apply to principals belonging to the group identified in the owner_group attribute but who are not identified by the owner attribute. Bits MODE4_ROTH, MODE4_WOTH, and MODE4_XOTH apply to any principal that does not match that in the owner attribute and does not belong to a group matching that of the owner_group attribute. These nine bits are used in providing authorization information.

[Previous Treatment]: The bits MODE4_SUID, MODE4_SGID, and MODE4_SVTX do not provide authorization information and do not affect server behavior. Instead, they are acted on by the client just as they would be for corresponding mode bits obtained from local file systems.

[Consensus needed (Item #6c)]: For objects which are not directories, the bits MODE4_SUID, MODE4_SGID, and MODE4_SVTX do not provide authorization information and do not affect server behavior. Instead, they are acted on by the client just as they would be for corresponding mode bits obtained from local file systems.

[Consensus needed (Item #6c)]: For directories, the bits MODE4_SUID and MODE4_SGID, do not provide authorization information and do not affect server behavior. Instead, they are acted on by the client just as they would be for corresponding mode bits obtained from local file systems. The mode bit MODE_SVTX does have an authorization-related role as described later in this section

[Consensus Needed, Including List (Item #6c]): When handling RENAME and REMOVE operations the check for authorization depends on the setting of MODE_SVTX for the directory.

- When MODE_SVTX is not set on the directory, authorization requires write permission on both the file being renamed and the source directory.

- When MODE_SVTX is not on the directory, authorization requires, in addition that the requesting principal be the owner of the file to be named or removed.

[Consensus needed (Item #6c)]: It should be noted that this approach is similar to ACL-based approach documented in Section 5.6. However there are some semantics differences whose motivation remains unclear and the specification does not mention RENAME, as it should.

[Author Aside]: Bringing the above into more alignment with the ACL-based semantics is certainly desirable but the necessary work has not been done yet. For tracking purposes, that realignment will be considered Consensus Item #20.

Bits within a mode other than those specified above are not defined by this protocol. A server **MUST NOT** return bits other than those defined above in a GETATTR or READDIR operation, and it **MUST** return NFS4ERR_INVAL if bits other than those defined above are set in a SETATTR, CREATE, OPEN, VERIFY, or NVERIFY operation.

[Consensus Needed (Item #21a)]: In the typical case, the nine low-order bits are set such that each successive set of three bits is a subset, not necessarily proper, of the previous three bits. Such modes are described as forward-slope nodes because the privilege level goes downward as you proceed forward. There are, however, cases in which there is an increase of privilege going from owner to group or from group to owner. Such modes are considered reverse-slope modes. As will be seen in Sections 9.3 and 9.6, many straightforward ways of dealing with mode that work well with forward-slope modes need adjustment to properly deal with reverse-slope modes.

7.3.2. NFSv4.1 Attribute 74: mode_set_masked

The mode_set_masked attribute is a write-only attribute that allows individual bits in the mode attribute to be set or reset, without changing others. It allows, for example, the bits MODE4_SUID, MODE4_SGID, and MODE4_SVTX to be modified while leaving unmodified any of the nine low-order mode bits devoted to permissions.

When minor versions other than NFSv4.0 are used, instances of use of the set_mode_masked attribute such that none of the nine low-order

bits are subject to modification, then neither the acl nor the dacl attribute should be automatically modified as discussed in Sections 9.6 and 9.8.

The mode_set_masked attribute consists of two words, each in the form of a mode4. The first consists of the value to be applied to the current mode value and the second is a mask. Only bits set to one in the mask word are changed (set or reset) in the file's mode. All other bits in the mode remain unchanged. Bits in the first word that correspond to bits that are zero in the mask are ignored, except that undefined bits are checked for validity and can result in NFS4ERR_INVAL as described below.

The mode_set_masked attribute is only valid in a SETATTR operation. If it is used in a CREATE or OPEN operation, the server **MUST** return NFS4ERR_INVAL.

Bits not defined as valid in the mode attribute are not valid in either word of the mode_set_masked attribute. The server **MUST** return NFS4ERR_INVAL if any such bits are set to one in a SETATTR. If the mode and mode_set_masked attributes are both specified in the same SETATTR, the server **MUST** also return NFS4ERR_INVAL.

7.4. ACL-based Authorization Model

7.4.1. Processing Access Control Entries

To determine if a request succeeds, the server processes each nfsace4 entry of type ALLOW or DENY in turn as ordered in the array. Only ACEs that have a "who" that matches the requester are considered. An ACE is considered to match a given requester if at least one of the following is true:

*The "who' designates a specific user which is the user making the request.

*The "who" specifies "OWNER@" and the user making the request is the owner of the file.

*The "who" designates a specific group and the user making the request is a member of that group.

*The "who" specifies "GROUP@" and the user making the request is a member of the group owning the file.

*The "who" specifies "EVERYONE@".

*The "who" specifies "INTERACTIVE@", "NETWORK@", "DIALUP@", "BATCH@", or "SERVICE@" and the requester, in the judgment of the

server, feels that designation appropriately describes the requester.

*The "who" specifies "ANONYMOUS@" or "AUTHENTICATED@" and the requestor's authentication status matches the who, using the definitions in Section 5.9

Each ACE is processed until all of the bits of the requester's access have been ALLOWED. Once a bit (see below) has been ALLOWED by an ACCESS_ALLOWED_ACE, it is no longer considered in the processing of later ACEs. If an ACCESS_DENIED_ACE is encountered where the requester's access still has unALLOWED bits in common with the "access_mask" of the ACE, the request is denied. When the ACL is fully processed, if there are bits in the requester's mask that have not been ALLOWED or DENIED, access is denied.

Unlike the ALLOW and DENY ACE types, the ALARM and AUDIT ACE types do not affect a requester's access, and instead are for triggering events as a result of a requester's access attempt. AUDIT and ALARM ACEs are processed only after processing ALLOW and DENY ACEs if any exist. This is necessary since the handling of AUDIT and ALARM ACEs are affected by whether the access attempt is successful.

[Previous Treatment]: The NFSv4.1 ACL model is quite rich. Some server platforms may provide access-control functionality that goes beyond the UNIX-style mode attribute, but that is not as rich as the NFS ACL model. So that users can take advantage of this more limited functionality, the server may support the acl attributes by mapping between its ACL model and the NFSv4.1 ACL model. Servers must ensure that the ACL they actually store or enforce is at least as strict as the NFSv4 ACL that was set. It is tempting to accomplish this by rejecting any ACL that falls outside the small set that can be represented accurately. However, such an approach can render ACLs unusable without special client-side knowledge of the server's mapping, which defeats the purpose of having a common NFSv4 ACL protocol. Therefore, servers should accept every ACL that they can without compromising security. To help accomplish this, servers may make a special exception, in the case of unsupported permission bits, to the rule that bits not ALLOWED or DENIED by an ACL must be denied. For example, a UNIX-style server might choose to silently allow read attribute permissions even though an ACL does not explicitly allow those permissions. (An ACL that explicitly denies permission to read attributes should still be rejected.)

[Author Aside]: While the NFSv4.1 provides that many might not need or use, it is the one that the working group adopted by the working group, and I have to assume that alternatives, such as the withdrawn POSIX ACL proposal were considered but not adopted. The phrase "unsupported permission bits" with no definition of the bit whose

support might be dispensed with, implies that the server is free to support whatever subset of these bits it chooses. As a result, clients would not be able to rely on a functioning server implementation of this OPTIONAL attribute. If there are specific compatibility issues that make it necessary to allow non-support of specific mask bits, then these need to be limited and the client needs guidance about determining the set of unsupported mask bits.

[Previous Treatment]: The situation is complicated by the fact that a server may have multiple modules that enforce ACLs. For example, the enforcement for NFSv4.1 access may be different from, but not weaker than, the enforcement for local access, and both may be different from the enforcement for access through other protocols such as SMB (Server Message Block). So it may be useful for a server to accept an ACL even if not all of its modules are able to support it.

[Author Aside]: The following paragraph does not provide helpful guidance and takes no account of the need of the the client to be able to rely on the server implementing protocol-specifying semantics and giving notice in those cases in which it is unable to so

[Previous Treatment]: The guiding principle with regard to NFSv4 access is that the server must not accept ACLs that appear to make access to the file more restrictive than it really is.

7.4.2. V4.1 Attribute 58: dacl

The dacl attribute is like the acl attribute, but dacl allows only ALLOW and DENY ACEs. The dacl attribute supports automatic inheritance (see Section 5.10).

8. Common Considerations for Both File access Models

[Author Aside, Including List]: This subsections within this section are derived from Section 6.3 of 8881, entitled "Common Methods. However, its content is different because it has been rewritten to deal with issues common to both file access models, which now appears to have not been the original intention. Nevertheless, the following changes have been made:

- The section "Server Considerations" has been revised to deal with both the mode and acl attributes, since the points being made apply, in almost all cases, to both attributes.

- The section "Client Considerations" has been heavily revised, since what had been there did not make any sense to me.

*The section "Computing a Mode Attribute from an ACL" has been moved to Section 9.3 since it deals with the co-ordination of the posix and acl authorization models.

8.1. Server Considerations

The server uses the mode attribute or the acl attribute applying the algorithm described in Section 7.4.1 to determine whether an ACL allows access to an object.

[Author Aside, Including List]: The list previously in this section (now described as "Previous Treatment" combines two related issues in a way which obscures the very different security-related consequences of two distinct issues:

*In some cases an operation will be authorized but is not allowed for reasons unrelated to authorization.

  This has no negative effect on security.

*The converse case does have troubling effects on security which are mentioned in this section and discussed in more detail in Section 17

[Author Aside, Including List]: The items in that list have been dealt with as follows:

*The first and sixth items fit under the first (i.e. less troublesome) of these issues. They have have been transferred into an appropriate replacement list.

*The third item is to be deleted since it does not manifest either of these issues. In fact, it refers to the semantics already described in Section 5.4. is already described in ...

*The second, fourth and fifth items need to be addressed in a new list dealing with the potentially troublesome issues arising from occasions in which the access semantics previously described are relaxed, for various reasons.

  Included are cases in which previous specifications explicitly allowed this by using the term "**MAY**" and others in which the existence of servers manifesting such behavior was reported, with the implication that clients need to prepared for such behavior.

[Previous Treatment, Including List (Item #22a)]: However, these attributes might not be the sole determiner of access. For example:

- In the case of a file system exported as read-only, the server will deny write access even though an object's file access attributes would grant it.

- Server implementations **MAY** grant ACE4_WRITE_ACL and ACE4_READ_ACL permissions to prevent a situation from arising in which there is no valid way to ever modify the ACL.

- All servers will allow a user the ability to read the data of the file when only the execute permission is granted (e.g., if the ACL denies the user the ACE4_READ_DATA access and allows the user ACE4_EXECUTE, the server will allow the user to read the data of the file).

- Many servers implement owner-override semantics in which the owner of the object is allowed to override accesses that are denied by the ACL. This may be helpful, for example, to allow users continued access to open files on which the permissions have changed.

- Many servers provide for the existence of a "superuser" that has privileges beyond an ordinary user. The superuser may be able to read or write data or metadata in ways that would not be permitted by the ACL or mode attributes.

- A retention attribute might also block access otherwise allowed by ACLs (see Section 5.13 of [8]).

[Consensus Needed, Including List (Item #22a)]: It should be noted that, even when an operation is authorized, it may be denied for reasons unrelated to authorization. For example:

- In the case of a file system exported as read-only, the server will deny write access even though an object's file access attributes would authorize it.

- A retention attribute might also block access otherwise allowed by ACLs (see Section 5.13 of [8]).

[Consensus Needed, (Item #22a)]: There are also cases in which the converse issue arises, so that an operation which is not authorized as specified by the mode and ACL attributes is, nevertheless, executed as if it were authorized. Because previous NFSv4 specifications have cited the cases listed below without reference to the security problems that they create, it is necessary to discuss them here to provide clarification of the security implications of following this guidance, which is now superseded.

These cases are listed below and discussed in more detail in Section 17.1.3.

[Consensus Needed, Including List (Item #22a)]: In the following list, the treatment used in RFC8881 is quoted, while the corresponding text in RFC7530 is essentially identical.

*RFC8881 contains the following, which is now superseded:

> Server implementations **MAY** grant ACE4_WRITE_ACL and ACE4_READ_ACL permissions to prevent a situation from arising in which there is no valid way to ever modify the ACL.

While, as a practical matter, there do need to be provisions to deal with this issue, the "**MAY**" above is too broad,in that it describes the motivation without any limits providing appropriate restriction on the step that might be taken to deal with the issue. See Section 17.1.3 for the updated treatment of this issue.

*RFC8881 contains the following, which is now superseded:

> Many servers implement owner-override semantics in which the owner of the object is allowed to override accesses that are denied by the ACL. This may be helpful, for example, to allow users continued access to open files on which the permissions have changed.

Regardless of the truth of the first sentence above, either when it was written or today, it needs to be stressed that the fact that a server manifests a particular behavior does not imply that it is valid according to the protocol specification. In this case, the supposed "owner-override semantics" clearly are not valid, since they contradict the specification of both the mode-based and ACL-based approaches to file access authorization.

With regard to the second sentence of the quotation above, it is not clear whether it is helpful or hurtful to allow continued access to open files which have become inaccessible due to changes in security and it is not clear that the working group will make a decision on the matter in this document, despite the obvious security implications. In any case, the resolution is unlikely to depend on whether the owner is involved.

*RFC8881 contains the following, which is now superseded:

> Many servers have the notion of a "superuser" that has privileges beyond an ordinary user. The superuser may be able to read or write data or metadata in ways that would not be permitted by the ACL or mode attributes.

While many (or almost all) systems in which NFSv4 servers are embedded, have provisions for such privileged access to be provided, it does not follow that NFSv4 servers, as such, need to have provision for such access.

Providing such access as part of the NFSv4 protocols, would necessitate a major revision of the semantics of ACL including such troublesome matters as the proper handling of AUDIT and ALARM ACEs in the face of such privileged access.

Because of the effect such unrestricted access might have in facilitating and perpetuating attacks, Section 17.1.3 will explain that the treat analysis in Section 17, will not cover servers which choose to allow such access.

8.2. Client Considerations

[Previous Treatment]: Clients **SHOULD NOT** do their own access checks based on their interpretation of the ACL, but rather use the OPEN and ACCESS operations to do access checks. This allows the client to act on the results of having the server determine whether or not access should be granted based on its interpretation of the ACL.

[Author Aside]: With regard to the use of "**SHOULD NOT**" in the paragraph above, it is not clear what might be valid reasons to bypass this recommendation. Perhaps "**MUST NOT**" or "should not" would be more appropriate.

[Consensus Needed (Item #23a)]: Clients are expected to do their own access checks based on their interpretation of the ACL, but instead use the OPEN and ACCESS operations to do access checks. This allows the client to act on the results of having the server determine whether or not access should be granted based on its interpretation of the ACL.

[Previous Treatment]: Clients must be aware of situations in which an object's ACL will define a certain access even though the server will not enforce it. In general, but especially in these situations, the client needs to do its part in the enforcement of access as defined by the ACL.

[Author Aside]: Despite what is said later, the only such case I know of is the use of READ and EXECUTE where the client, but not the server, has any means of distinguishing these. I don't know of any others. If there were, how could ACCESS or OPEN be used to verify access?

[Consensus Needed (Item #23a)]; Clients need to be aware of situations in which an object's ACL will define a certain access even though the server is not in position to enforce it because the

server does not have the relevant information, such as knowing whether a READ is for the purpose of executing a file. Because of such situations, the client needs to do be prepared to do its part in the enforcement of access as defined by the ACL.

To do this, the client will send the appropriate ACCESS operation prior to servicing the request of the user or application in order to determine whether the user or application should be granted the access requested.

[Previous Treatment (Item #24a)]: For examples in which the ACL may define accesses that the server doesn't enforce, see Section 8.1.

[Author Aside]: The sentence above is clearly wrong since that section is about enforcement the server does do. The expectation is that it will be deleted as part of Consensus Item #24a.

9. Combining Authorization Models

9.1. Background for Combined Authorization Model

Both RFCs 7530 and 5661 contain material relating to the need, when both mode and ACL attributes are supported, to make sure that the values are appropriately co-ordinated. Despite the fact that these discussions are different, they are compatible, and differ in only a small number of areas.

[Author Aside]: From this point on, all unannotated paragraphs in this section are to be assumed part of Consensus Item #25b

As a result, in this document, we will have a single treatment of this issue, in Sections 9.2 through 9.11. In addition, an NFSv4.2-based extension related to attribute co-ordination will be described in Section 9.12.

The current NFSv4.0 and NFSv4.1 descriptions of this share one unfortunate characteristic in that they both are written to give server implementations a broad latitude in implementation choices while neglecting entirely the need for the clients and users to have a reliable description of what servers are to do in this area.

As a result, one of the goals of this new combined treatment will be to limit this excessive freedom, while taking proper account of the possibility of compatibility issues that a more tightly drawn specification might give rise to.

[Author Aside, Including List]: The various ways in which these kinds of issues have been dealt with are listed below:

* In some cases, the term "**MAY**" is used in contexts where it is inappropriate, since the allowed variation has the potential to cause harm in that it leaves the client unsure exactly what security-related action will be performed by the server.

* There are also cases in which no RFC2119-defined keywords are used but it is stated that certain server implementations do a particular, leaving the impression that that action is to be allowed, just as if "**MAY**" had been used.

* There is a case in which the term "**SHOULD**" is clearly used intentionally, without it being clear what the valid reasons to ignore the guidance might be.

* There are many case in which the term "**SHOULD**" is used without any clear indication why it was used. In this situation it is possible that the "**SHOULD**" was essentially treated as a "**MAY**" but also possible that servers chose to follow the recommendation.

In order to deal with the many uses of the terms "**SHOULD**" and "**SHOULD NOT**" in Section 9 and included subsections, which have no clear motivation, it is to be assumed that the valid reasons to act contrary to the recommendation given are the difficulty of changing implementations based on previous analogous guidance, which may have given the impression that the server was free to ignore the guidance for any reason the implementer chose. This allows the possibility of more individualized treatment of these instances once compatibility issues have been adequately discussed.

[Author Aside]: In each subsection in which the the interpretation of these term in the previous paragraph applies there will be an explicit reference to Consensus Item #25, to draw attention to this change, even in the absence of modified text.

9.2. Needed Attribute Coordination

On servers that support both the mode and the acl or dacl attributes, the server must keep the two consistent with each other. The value of the mode attribute (with the exception of the high-order bits reserved for client use as described in Section 7.3.1) must be determined entirely by the value of the ACL, so that use of the mode is never required for anything other than setting the three high-order bits. See Sections 9.6 through 9.8 for detailed discussion.

[Previous Treatment (Item #25c)]: When a mode attribute is set on an object, the ACL attributes may need to be modified in order to not

conflict with the new mode. In such cases, it is desirable that the ACL keep as much information as possible. This includes information about inheritance, AUDIT and ALARM ACEs, and permissions granted and denied that do not conflict with the new mode.

[Author Aside]: the things that this formulation leaves uncertain, is whether, if the ACL specifies permission for a named user group or group, it "conflicts" with the node. Ordinarily, one might think it does not, unless the specified user is the owner of the file or a member of the owning group, or the specified group is the owning group. However, while some parts of the existing treatment seem to agree with this, other parts, while unclear, seem to suggest otherwise, while the treatment in Section 9.6 is directly in conflict.

[Previous Treatment (Item #26a)]: The server that supports both mode and ACL must take care to synchronize the MODE4_*USR, MODE4_*GRP, and MODE4_*OTH bits with the ACEs that have respective who fields of "OWNER@", "GROUP@", and "EVERYONE@".

[Author Aside]: This sentence ignores named owners and group, giving the impressions that there is no need to change them.

[Previous Treatment (Item #26a)]: This way, the client can see if semantically equivalent access permissions exist whether the client asks for the owner, owner_group, and mode attributes or for just the ACL.

[Author Aside, Including List:] The above sentence, while hard to interpret for a number a reasons, is worth looking at in detail because it might suggest an approach different from the in the previous sentence from the initial paragraph for The Previous Treatment of Item #26a.

*The introductory phrase "this way" adds confusion because it suggests that there are other valid ways of doing this, while not giving any hint about what these might be.

*It is hard to understand the intention of "client can see if semantically equivalent access permissions" especially as the client is told elsewhere that he is not to interpret the ACL himself.

*If this sentence is to have any effect at all it, it would be to suggest that the result be the same "whether the client asks for the owner, owner_group, and mode attributes or for just the ACL."

If these are to be semantically equivalent it would be necessary to delete ACEs for named users, which requires a different approach form the first sentence of the original paragraph.

{Consensus Needed (Items #26a, #28a)]: A server that supports both mode and ACL need to take care to synchronize the MODE4_*USR, MODE4_*GRP, and MODE4_*OTH bits with the ACEs that have respective who fields of "OWNER@", "GROUP@", and "EVERYONE@". This is relatively straightforward in the case of forward-slope modes, but the case of reverse-slope modes need to be addressed as described in Sections 9.3 and 9.6.

{Consensus Needed (Item #26a)]: How other ACEs are dealt with when setting mode is described in Section 9.6. This includes ACEs with other who value, all AUDIT and ALARM ACEs, and all ACES that affect ACL inheritance.

[Author Aside, Including List]: The author believes that the material now associated with Item #27, including the following paragraph and Section 9.4 are best deleted. This is because of reasons specified in that section and the following reasons listed below:

*Having multiple methods to map from ACL to mode undercuts the whole purpose of Section 9, which is to co-ordinate these attributes so that clients who use each of the attributes can use that without interfering with those that reference others. That cannot be accomplished if there were multiple valid ways that servers might choose, without providing any means by which the client might determine which mapping was being used.

*The withdrawn POSIX draft ACLs would almost certainly have been considered in connection with an NFSv4. In any case, they were not adopted and the current ACL model adopted. Given that fact there is no sense in burdening the new feature with the substantial burden of supporting the one that was rejected by the working group

*It is very unlikely that such implementations still exist, given that that it is over twenty years since the decision was made to adopt the more extensive NFSv4 ACL model and over ten years since RFC5661 was published. Even assuming this was justified as a transition measure, the time for any such transition mechanisms is long past.

*Despite the statement in the next section that this alternate model is "discouraged", its continued appearance as an alternate way of computing mode, on the same level as the one appropriate to the NFSv4 acl model encourages this use compared to a situation in which no alternate method of computing mode was mentioned.

[Previous Treatment (Item #27a)]: In this section, much depends on the method in specified Section 9.3. Many requirements refer to this section. It should be noted that the methods have behaviors specified with "**SHOULD**" and that alternate approaches are discussed in Section 9.4. This is intentional, to avoid invalidating existing implementations that compute the mode according to the withdrawn POSIX ACL draft (1003.1e draft 17), rather than by actual permissions on owner, group, and other.

[Author Aside]: Given the mixture of RFC2219 terms, I think all of them in Section 9 need review. Further, given the effort that has gone into Section 9, to accommodate these implementations of a draft that was withdrawn decades ago. The idea of trying to make mode and acl match is undercut when there are different valid ways of computing the mode. There shouldn't be. To specify one way to do this is necessary to accomplish the goal here and to do so would not "invalidate" anything. Rather, it would establish, correctly, that such implementations are not implementations of the NFSv4 ACL model, but of the withdrawn POSIX ACL draft.

9.3. Computing a Mode Attribute from an ACL

[Previous Treatment (Item #27b)]: The following method can be used to calculate the MODE4_R*, MODE4_W*, and MODE4_X* bits of a mode attribute, based upon an ACL.

[Author Aside]: "can be used" says essentially "do whatever you choose" and would make Section 9 essentially pointless. Would prefer "is to be used" or "**MUST**", with "**SHOULD**" available if valid reasons to do otherwise can be found.

[Consensus Needed (Items #27b, #28b)}: The following method (or another one providing exactly the same results) **SHOULD** be used to calculate the MODE4_R*, MODE4_W*, and MODE4_X* bits of a mode attribute, based upon an ACL. In this case valid reasons to bypass the recommendation are limited to implementor reliance on previous specifications which ignored the cases of the owner having less access than the owning group or the owning group having less access than others. Further, in implementing or the maintaining an implementation previously believed to be valid, the implementor needs to be aware that this will result invalid values in some uncommon cases.

[Author Aside, Including List]: The algorithm specified below, now considered the Previous Treatment associated with Item #24a, has an important flaw in does not deal with the (admittedly uncommon) case in which the owner_group has less access than the owner or others

have less access than the owner-group. In essence, this algorithm ignores the following facts:

  *That GROUP@ includes the owning user while group bits in the mode do not affect the owning user.

  *That EVERYONE includes the owning group while other bits in the mode do not affect users within the owning group.

[Previous Treatment (Item #28a)]: First, for each of the special identifiers OWNER@, GROUP@, and EVERYONE@, evaluate the ACL in order, considering only ALLOW and DENY ACEs for the identifier EVERYONE@ and for the identifier under consideration. The result of the evaluation will be an NFSv4 ACL mask showing exactly which bits are permitted to that identifier.

[Previous Treatment (Item #28a)]: Then translate the calculated mask for OWNER@, GROUP@, and EVERYONE@ into mode bits for, respectively, the user, group, and other, as follows:

[Consensus Needed, including List(Item #28a)]: First, for each of the sets of mode bits (i.e., user, group other, evaluate the ACL in order, with a specific evaluation procedure depending on the specific set of mode bits being determined. For each set there will be one or more special identifiers considered in a positive sense so that ALLOW and DENY ACE's are considered in arriving at the mode bit. In addition, for some sets of bits, there will be one or more special identifiers to be considered only in a negative sense, so that only DENY ACE's are considered in arriving at the mode it. The users to be considered are as follows:

  *For the owner bits, "OWNER@" and "EVERYONE@" are to be considered, both in a positive sense.

  *For the group bits, "GROUP@" and "EVERYONE@" are to be considered, both in a positive sense, while "OWNER@" is to be considered in a negative sense.

  *For the other bit, "EVERYONE@" is to be considered in a positive sense, while "OWNER@" and "GROUP@" are to be considered in a negative sense.

[Consensus Needed (Item #28a)]: Then translate the calculated mask for for each set of bit into the corresponding mode bits for, user, group, and other, as follows:

  1. Set the read bit (MODE4_RUSR, MODE4_RGRP, or MODE4_ROTH) if and only if ACE4_READ_DATA is set in the corresponding mask.

2. Set the write bit (MODE4_WUSR, MODE4_WGRP, or MODE4_WOTH) if and only if ACE4_WRITE_DATA and ACE4_APPEND_DATA are both set in the corresponding mask.

3. Set the execute bit (MODE4_XUSR, MODE4_XGRP, or MODE4_XOTH), if and only if ACE4_EXECUTE is set in the corresponding mask.

9.4. Alternatives in Computing Mode Bits

[Author Aside]: For reasons explained below, the author believes this section needs to deleted, as part of Consensus Item #27c. In order to enable this deletion or its replacement by an alternate formulation if the working group so decides, all unannotated paragraphs within this section are to be considered the Previous Treatment corresponding to Consensus Item #27c.

Some server implementations also add bits permitted to named users and groups to the group bits (MODE4_RGRP, MODE4_WGRP, and MODE4_XGRP).

Implementations are discouraged from doing this, because it has been found to cause confusion for users who see members of a file's group denied access that the mode bits appear to allow. (The presence of DENY ACEs may also lead to such behavior, but DENY ACEs are expected to be more rarely used.)

[Author Aside]: The text does not seem to really discourage this practice and makes no reference to the need to standardize behavior so the clients know what to expect or any other reason for providing standardization of server behavior.

The same user confusion seen when fetching the mode also results if setting the mode does not effectively control permissions for the owner, group, and other users; this motivates some of the requirements that follow.

[Author Aside]: The part before the semicolon appears to be relevant to Consensus Item #23 but does not point us to a clear conclusion. The statement certainly suggests that the 512-ACL approach is more desirable but the absence of a more direct statement to that effect suggest that this is a server implementer choice.

[Author Aside]: The part after the semicolon is hard to interpret in that it is not clear what "this" refers to or which which requirements are referred to by "some of the requirements that follow". The author would appreciate hearing from anyone who has insight about what might have been intended here.

9.5. Setting Multiple ACL Attributes

In the case where a server supports the sacl or dacl attribute, in addition to the acl attribute, the server **MUST** fail a request to set the acl attribute simultaneously with a dacl or sacl attribute. The error to be given is NFS4ERR_ATTRNOTSUPP.

9.6. Setting Mode and not ACL (overall)

9.6.1. Setting Mode and not ACL (vestigial)

[Author Aside]: All unannotated paragraphs as considered the Previous treatment of Consensus Item #30a.

[Previous Treatment (Item #?a)]: When any of the nine low-order mode bits are subject to change, either because the mode attribute was set or because the mode_set_masked attribute was set and the mask included one or more bits from the nine low-order mode bits, and no ACL attribute is explicitly set, the acl and dacl attributes must be modified in accordance with the updated value of those bits. This must happen even if the value of the low-order bits is the same after the mode is set as before.

Note that any AUDIT or ALARM ACEs (hence any ACEs in the sacl attribute) are unaffected by changes to the mode.

In cases in which the permissions bits are subject to change, the acl and dacl attributes **MUST** be modified such that the mode computed via the method in Section 9.3 yields the low-order nine bits (MODE4_R*, MODE4_W*, MODE4_X*) of the mode attribute as modified by the attribute change. The ACL attributes **SHOULD** also be modified such that:

1. If MODE4_RGRP is not set, entities explicitly listed in the ACL other than OWNER@ and EVERYONE@ **SHOULD NOT** be granted ACE4_READ_DATA.

2. If MODE4_WGRP is not set, entities explicitly listed in the ACL other than OWNER@ and EVERYONE@ **SHOULD NOT** be granted ACE4_WRITE_DATA or ACE4_APPEND_DATA.

3. If MODE4_XGRP is not set, entities explicitly listed in the ACL other than OWNER@ and EVERYONE@ **SHOULD NOT** be granted ACE4_EXECUTE.

Access mask bits other than those listed above, appearing in ALLOW ACEs, **MAY** also be disabled.

Note that ACEs with the flag ACE4_INHERIT_ONLY_ACE set do not affect the permissions of the ACL itself, nor do ACEs of the type AUDIT and

ALARM. As such, it is desirable to leave these ACEs unmodified when modifying the ACL attributes.

Also note that the requirement may be met by discarding the acl and dacl, in favor of an ACL that represents the mode and only the mode. This is permitted, but it is preferable for a server to preserve as much of the ACL as possible without violating the above requirements. Discarding the ACL makes it effectively impossible for a file created with a mode attribute to inherit an ACL (see Section 9.10).

9.6.2. Setting Mode and not ACL (Discussion)

[Author Aside]: All unannotated paragraphs as considered Author Asides relating to Consensus Item #30b.

Existing documents are unclear about the changes to be made to an existing ACL when the nine low-order bits of the mode attribute are subject to modification using SETATTR.

A new treatment needs to apply to all minor versions. It will be necessary to specify that, for all minor versions, setting of the mode attribute, subjects the low-order nine bits to modification.

One important source of this lack of clarity is the following paragraph from Section 9.6.1, which we refer to later as the trivial-implementation-remark".

> Also note that the requirement may be met by discarding the acl and dacl, in favor of an ACL that represents the mode and only the mode. This is permitted, but it is preferable for a server to preserve as much of the ACL as possible without violating the above requirements. Discarding the ACL makes it effectively impossible for a file created with a mode attribute to inherit an ACL (see Section 9.10).

The only "requirement" which might be met by the procedure mentioned above is the text quoted below.

> In cases in which the permissions bits are subject to change, the acl and dacl attributes **MUST** be modified such that the mode computed via the method in Section 9.3 yields the low-order nine bits (MODE4_R*, MODE4_W*, MODE4_X*) of the mode attribute as modified by the attribute change.

While it is true that this requirement could be met by the specified treatment, this fact does not, in itself, affect the numerous recommendations that appear between the above requirement and the trivial-implementation-remark.

It may well be that there are are implementations that have treated the trivial-implementation-remark as essentially allowing them to essentially ignore all of those recommendations, resulting in a situation in which were treated as if it were a trivial-implementation-ok indication. How that issue will be dealt with in a replacement for Section 9.6.1 will be affected by the working group's examination of compatibility issues.

The following specific issues need to be addressed:

- *Handling of inheritance.

  Beyond the possible issues that arise from the trivial-implementation-ok interpretation, the treatment in Section 9.6.1, by pointing specifically to existing INHERIT_ONLY ACEs obscures the corresponding need to convert ACE's that specify both inheritance and access permissions to be converted to INHERIT_ONLY ACEs.

- *Reverse-slope modes

- *Named users and groups.

- *The exact bounds of what within the ACL is covered by the low-order bits of the mode.

It appears that for many of the issues, there are many possible readings of the existing specs, leading to the possibility of multiple inconsistent server behaviors. Furthermore, there are cases in which none of the possible behaviors described in existing specifications meets the needs.

As a result of these issues, the existing specifications do not provide a reliable basis for client-side implementations of the ACL feature which a Proposed Standard is normally expected to provide.

9.6.3. Setting Mode and not ACL (Proposed)

[Author Aside]: This proposed section is part of Consensus Item #30c and all unannotated paragraphs within it are to be considered part of that Item. Since the proposed text includes support for reverse-slope modes, treats all minor versions together and assumes decisions about handling of ACEs for named users and groups, the relevance of consensus items #26, #28, and #29 should be noted.

[Author Aside]: As with all such Consensus Items, it is expected that the eventual text in a published RFC might be substantially different based on working group discussion of client and server needs and possible compatibility issues. In this particular case, that divergence can be expected to be larger, because the author was

forced to guess about compatibility issues and because earlier material, on which it is based left such a wide range of matters to the discretion of server implementers. It is the author's hope that, as the working group discusses matters, sufficient attention is placed on the need for client-side implementations to have reliable information about expected server-side actions.

This section describes how ACLs are to be updated in response to actual or potential changes in the mode attribute, when the attributes needed by both of the file access authorization models are supported. It supersedes the discussions of the subject in RFCs 7530 and 8881, each of which appeared in Section 6.4.1.1 of the corresponding document.

It is necessary to approach the matter differently than in the past because:

- Organizational changes are necessary to address all minor versions together.
- Those previous discussions are often internally inconsistent leaving it unclear what specification-mandated actions were being specified..
- In many cases, servers were granted an extraordinary degree of freedom to choose the action to take, either explicitly or via an apparently unmotivated use of SHOULD, leaving it unclear what might be considered "valid" reasons to ignore the recommendation.
- There appears to have been no concern for the problems that clients and applications might encounter dealing ACLs in such an uncertain environment.
- Cases involving reverse-slope modes were not adequately addressed.
- The security-related effects of SVTX were not addressed.

While that sort of approach might have been workable at one time, it made it difficult to devise client-side ACL implementations, even if there had been any interest in doing so. In order to enable this situation to eventually be rectified, we will define the preferred implementation here, but in order to provide temporary compatibility with existing implementations based on reasonable interpretations of RFCs 7530 and 8881. To enable such compatibility the term "**SHOULD**" will be used, with the understanding that valid reasons to bypass the recommendation, are limited to implementers' previous reliance on these earlier specifications and the difficulty of changing them now.

When the recommendation is bypassed in this way, it is necessary to understand, that, until the divergence is rectified, or the client is given a way to determine the detail of the server's non-standard behavior, client-side implementations may find it difficult to implement a client-side implementation that correctly interoperates with the existing server.

When mode bits involved in determining file access authorization are subject to modification, the server **MUST**, when ACL-related attributes have been set, modify the associated ACEs so as not to conflict with the new value of the mode attribute.

The occasions to which this requirement applies, vary with the attribute being set and the type of object being dealt with:

- For all minor versions, any change to the mode attribute, triggers this requirement

- When the set_mode_masked attribute is being set on an object which is not a directory, whether this requirement is triggered depends on whether any of the nine low-order bits of the mode is included in the mask.

- When the set_mode_masked attribute is being set on a directory, whether this requirement is triggered depends on whether any of the nine low-order bits of the mode or the SVTX bit is included in the mask of bit whose values are to be set.

When the requirement is triggered, ACEs need to be updated to be consistent with the new mode attribute. In the case of AUDIT and ALARM ACEs, which are outside of file access authorization, no change is to be made.

For ALLOW and DENY ACEs, changes are necessary to avoid conflicts with the mode in a number of areas:

- The handling of ACEs that have consequences relating to ACL inheritance.

- The handling of ACEs with a who-value of OWNER@, GROUP@, or EVERYONE@ need to be adapted to the new mode.

- ACEs whose who-value is a named user or group, are to be retained or not based on the mode being set as described below.

- ACEs whose who-value is one of the other special values defined in Section 5.9 are to be left unmodified.

In order to deal with inheritance issues, the following **SHOULD** be done:

- ACEs that specify inheritance-only need to be retained, regardless of the value of who specified, since inheritance issues are outside of the semantic range of the mode attribute.

- ACEs that specify inheritance, in addition to allowing or denying authorization for the current object need to be converted into inheritance-only ACEs. This needs to occur irrespective of the value of who appearing in the ACE.

For NFSv4 servers that support the dacl attribute, at least the first of the above **MUST** be done.

Other ACEs are to be treated are classified based on the ACE's who-value:

- ACEs whose who-value is OWNER@, GROUP@, or EVERYONE@ are referred to as mode-directed ACEs and are subject to extensive modification.

- ACEs whose who-value is a named user or group are either left alone or subject to extensive modification, as described below.

- ACEs whose who-value is one of the other special values defined in Section 5.9 are left as they are.

Mode-directed ACEs need to be modified so that they reflect the mode being set.

In effecting this modification, the server will need to distinguish mask bits deriving from mode attributes from those that have no such connection. The former can be categorized as follows:

- For non-directory objects, the mask bits ACE4_READ_DATA (from the read bit in the mode), ACE4_EXECUTE (from the execute bit in the mode), and ACE4_WRITE_DATA together with ACE4_APPEND_DATA (from the write bit in the mode) are all derived from the set of three mode bits appropriate to the current who-value.

- For directories, analogous mask bits are included: ACE4_LIST_DIRECTORY (from the read in the mode), ACE4_EXECUTE (from the execute bit in the mode), and ACE4_ADD_FILE together with ACE4_ADD_SUBDIRECTORY and ACE4_DELETE_CHILD> (from the write bit in the mode) are all included based on the set of three mode bits appropriate to the current who-value.

  When the SVTX bit is set, ACE4_DELETE_CHILD is set, regardless of the values of the low-order nine bit of the mode.

* When named attributes are supported for the object whose mode is subject to change, ACE4_READ_NAMED_ATTRIBUTES is set based on the read bit and ACE4_WRITE_NAMED_ATTRIBUTES is set based on the write bit based on the set of three mode bits appropriate to the current who-value.

* In the case of OWNER@, ACE4_WRITE_ACL, ACE4_WRITE_ATTRIBUTES ACE4_WRITE_ACL, ACE4_WRITE_OWNER are all set.

The union of these groups of mode bit are referred to as the mode-relevant mask bits.

[Author Aside]: Except for the case of ACE4_SYNCHRONIZE, the handling of mask bits which are not mode-relevant is yet to be clarified. For tracking purposes, the handling of mask bits ACE4_READ_ATTRIBUTES, ACE4_WRITE_RETENTION, ACE4_WRITE_RETENTION_HOLD, ACE4_READ_ACL will be dealt with as Consensus Item #31.

If the mode is of forward-slope, then each set of three bits is translated into a corresponding set of mode bits. Then, for each ALLOW ACE with one of these who values, all mask bits in this class are deleted and the computed mode bits for that who-value substituted. For DENY ACEs, all mask bits in this class are reset, and, if none remain, the ACE **MAY** be deleted.

In the case of reverse-slope modes, the following **SHOULD** be done:

* For mode-directed ACEs all mode-relevant mask bits are reset, and, if none remain, the ACE **MAY** be deleted.

* Then, proceeding from owner to others, ALLOW ACEs are generated based on the computed mode-relevant mask bits.

  At each stage, if the mode-relevant mask bits for the current stage includes mask bits not set for the previous stage, then a DENY ACE needs to be added before the new ALLOW ACE. That ACE will have a who-value based on the previous stage and a mask consisting of the bit included in the current stage that were not included in the previous stage.

In cases in which the above recommendation is not followed, the server **MUST** follow a procedure which arrives at an ACL which behaves identically for all cases involving forward-slope mode values.

When dealing with ACEs whose who-value is a named user or group, they **SHOULD** be processed as follows:

* DENY ACEs are left as they are.

*ALLOW ACES are subject to filtering to effect mode changes that deny access to any principal other than the owner.

To determine the set of mode bits to which this filtering applies, the mode bits for group are combined with those for others, to get a set of three mode bits to determine which of the mode privileges (read, write, execute) are denied to all principals other than the owner, i.e. the set of bits not present in either the bits for group or the bits for others.

Those three bits are converted to the corresponding set of mask bits, according to the rules above.

All such mask bits are reset in the ACE, and, if none remain, the ACE **MAY** be deleted.

In cases in which the above recommendation is not followed, the server **MUST** follow a procedure which arrives at an ACL which behaves identically for all cases involving forward-slope mode values. This would be accomplished if the mask bits were reset based on the group bits alone, as had been recommended in earlier specifications.

9.7. Setting ACL and Not Mode

[Author Aside]: The handling of **SHOULD** in this section is considered as part of Consensus Item #25d.

When setting the acl or dacl and not setting the mode or mode_set_masked attributes, the permission bits of the mode need to be derived from the ACL. In this case, the ACL attribute **SHOULD** be set as given. The nine low-order bits of the mode attribute (MODE4_R*, MODE4_W*, MODE4_X*) **MUST** be modified to match the result of the method in Section 9.3. The three high-order bits of the mode (MODE4_SUID, MODE4_SGID, MODE4_SVTX) **SHOULD** remain unchanged.

9.8. Setting Both ACL and Mode

When setting both the mode (includes use of either the mode attribute or the mode_set_masked attribute) and the acl or dacl attributes in the same operation, the attributes **MUST** be applied in the following order order: mode (or mode_set_masked), then ACL. The mode-related attribute is set as given, then the ACL attribute is set as given, possibly changing the final mode, as described above in Section 9.7.

9.9. Retrieving the Mode and/or ACL Attributes

[Author Aside]: The handling of **SHOULD** in this section is considered as part of Consensus Item #25e.

Some server implementations may provide for the existence of "objects without ACLs", meaning that all permissions are granted and denied according to the mode attribute and that no ACL attribute is stored for that object. If an ACL attribute is requested of such a server, the server **SHOULD** return an ACL that does not conflict with the mode; that is to say, the ACL returned **SHOULD** represent the nine low-order bits of the mode attribute (MODE4_R*, MODE4_W*, MODE4_X*) as described in Section 9.3.

For other server implementations, the ACL attribute is always present for every object. Such servers **SHOULD** store at least the three high-order bits of the mode attribute (MODE4_SUID, MODE4_SGID, MODE4_SVTX). The server **SHOULD** return a mode attribute if one is requested, and the low-order nine bits of the mode (MODE4_R*, MODE4_W*, MODE4_X*) **MUST** match the result of applying the method in Section 9.3 to the ACL attribute.

9.10. Creating New Objects

[Author Aside]: The handling of **SHOULD** in this section is considered as part of Consensus Item #25f.

If a server supports any ACL attributes, it may use the ACL attributes on the parent directory to compute an initial ACL attribute for a newly created object. This will be referred to as the inherited ACL within this section. The act of adding one or more ACEs to the inherited ACL that are based upon ACEs in the parent directory's ACL will be referred to as inheriting an ACE within this section.

Implementors should base the behavior of CREATE and OPEN depending on the presence or absence of the mode and ACL attributes by following the directions below:

1. If just the mode is given in the call:

   In this case, inheritance **SHOULD** take place, but the mode **MUST** be applied to the inherited ACL as described in Section 9.6, thereby modifying the ACL.

2. If just the ACL is given in the call:

   In this case, inheritance **SHOULD NOT** take place, and the ACL as defined in the CREATE or OPEN will be set without modification, and the mode modified as in Section 9.7.

3. If both mode and ACL are given in the call:

   In this case, inheritance **SHOULD NOT** take place, and both attributes will be set as described in Section 9.8.

4. If neither mode nor ACL is given in the call:

   In the case where an object is being created without any initial attributes at all, e.g., an OPEN operation with an opentype4 of OPEN4_CREATE and a createmode4 of EXCLUSIVE4, inheritance **SHOULD NOT** take place (note that EXCLUSIVE4_1 is a better choice of createmode4, since it does permit initial attributes). Instead, the server **SHOULD** set permissions to deny all access to the newly created object. It is expected that the appropriate client will set the desired attributes in a subsequent SETATTR operation, and the server **SHOULD** allow that operation to succeed, regardless of what permissions the object is created with. For example, an empty ACL denies all permissions, but the server should allow the owner's SETATTR to succeed even though WRITE_ACL is implicitly denied.

   In other cases, inheritance **SHOULD** take place, and no modifications to the ACL will happen. The mode attribute, if supported, **MUST** be as computed in Section 9.3, with the MODE4_SUID, MODE4_SGID, and MODE4_SVTX bits clear. If no inheritable ACEs exist on the parent directory, the rules for creating acl, dacl, or sacl attributes are implementation defined. If either the dacl or sacl attribute is supported, then the ACL4_DEFAULTED flag **SHOULD** be set on the newly created attributes.

9.11. Use of Inherited ACL When Creating Objects

[Author Aside]: The handling of **SHOULD** in this section is considered as part of Consensus Item #25g.

If the object being created is not a directory, the inherited ACL **SHOULD NOT** inherit ACEs from the parent directory ACL unless the ACE4_FILE_INHERIT_ACE flag is set.

If the object being created is a directory, the inherited ACL should inherit all inheritable ACEs from the parent directory, that is, those that have the ACE4_FILE_INHERIT_ACE or ACE4_DIRECTORY_INHERIT_ACE flag set. If the inheritable ACE has ACE4_FILE_INHERIT_ACE set but ACE4_DIRECTORY_INHERIT_ACE is clear, the inherited ACE on the newly created directory **MUST** have the ACE4_INHERIT_ONLY_ACE flag set to prevent the directory from being affected by ACEs meant for non-directories.

When a new directory is created, the server **MAY** split any inherited ACE that is both inheritable and effective (in other words, that has neither ACE4_INHERIT_ONLY_ACE nor ACE4_NO_PROPAGATE_INHERIT_ACE set), into two ACEs, one with no inheritance flags and one with ACE4_INHERIT_ONLY_ACE set. (In the case of a dacl or sacl attribute,

both of those ACEs **SHOULD** also have the ACE4_INHERITED_ACE flag set.) This makes it simpler to modify the effective permissions on the directory without modifying the ACE that is to be inherited to the new directory's children.

9.12. Combined Authorization Models for NFSv4.2

The NFSv4 server implementation requirements described in the subsections above apply to NFSv4.2 as well and NFSv4.2 clients can assume that the server follows them.

NFSv4.2 contains an **OPTIONAL** extension, defined in [13], which is intended to reduce the interference of modes, restricted by the umask mechanism, with the acl inheritance mechanism. The extension allows the client to specify the umask separately from the mask attribute.

10. Labelled NFS Authorization Model

The labelled NFS feature of NFSv4.2 is designed to support Mandatory Access control.

The attribute sec_label enables an authorization model focused on Mandatory Access Control and is described in Section 10.

Not much can be said about this feature because the specification, in the interest of flexibility, has left important features undefined in order to allow future extension. As a result, we have something that is a framework to allow Mandatory Access Control rather than one to provide it. In particular,

- *The sec_label attribute, which provides the objects label has no existing specification.
- *There is no specification of the of the format of the subject label or way to authenticate them.
- *As a result, all authorization takes place on the client, and the server simply accepts the client's determination.

This arrangements shares important similarities with AUTH_SYS. As such it makes sense:

- *To require/recommend that an encrypted connection be used.
- *To require/recommend that client and server peers mutually authenticate as part of connection establishment.
- *That work be devoted to providing a replacement without the above issues.

11. State Modification Authorization

Modification of locking and session state data should not be done by a client other than the one that created the lock. For this form of authorization, the server needs to identify and authenticate client peers rather than client users.

Such authentication is not directly provided by any RPC authentication flavor. However, RPC-based transports, when suitably configured, can provide this authentication.

NFSv4.1 defines a number of ways to provide appropriate authorization facilities. These will not be discussed in detail here but the following points should be noted:

*NFSv4.1 defines the MACHCRED mechanism which uses the RPCSEC_GSS infrastructure to provide authentication of the clients peer. However, this is of no value when AUTH_SYS is being used.

*NFSv4.1 also defines the SSV mechanism which uses the RPCSEC_GSS infrastructure to enable it to be reliably determined whether two different client connections are connected to the same client. It is unclear whether the word "authentication" is appropriate in this case. As with MACHCRED, this is of no value when AUTH_SYS is being used.

*Because of the lack of support for AUTH_SYS and for NFSv4.0, it is quite desirable for clients to use and for servers to require the use of client-peer authentication as part of connection establishment.

When unauthenticated clients are allowed, their state is exposed to unwanted modification as part of disruption or denial-of-service attacks. As a result, the potential burdens of such attacks are felt principally by clients who choose not to provide such authentication.

12. Other Uses of Access Control Lists

Whether the acl or sacl attributes are used, AUDIT and ALARM ACEs provide security-related facilities separate from the the file access authorization provide by ALLOw and DENY ACEs

*AUDIT ACEs provide a means to audit attempts to acess specified file by specified sets of principals.

*ALARM ACEs provide a means to draw special attention to attempts to acess specified files by specified sets of principals.

12.1. V4.1 Attribute 59: sacl

The sacl attribute is like the acl attribute, but sacl allows only AUDIT and ALARM ACEs. The sacl attribute supports automatic inheritance (see Section 5.10).

13. Identification and Authentication

Various objects and subjects need to be identified for a protocol to function. For it to be secure, many of these need to be authenticated so that incorrect identification is not the basis for attacks.

13.1. Identification vs. Authentication

It is necessary to be clear about this distinction which has been obscured in the past, by the use of the term "RPC Authentication Flavor" in connection with situation in which identification without authentication occurred or in which there was neither identification nor authentication involved. As a result, we will use the term "RPC Flavors" instead

13.2. Items to be Identified

Some identifier are not security-relevant and can used be used without authentication, given that, in the authorization decision, the object acted upon needs only to be properly identified

*File names are of this type.

Unlike the case for some other protocols, confusion of names that result from internationalization issues, while an annoyance, are not relevant to security. If the confusion between LATIN CAPITAL LETTER O and CYRILLIC CAPITAL LETTER O, results in the wrong file being accessed, the mechanisms described in Section 7 prevent in appropriate access being granted.

Despite the above, it is desirable if file names together with similar are not transferred in the clear as the information exposed may give attackers useful information helpful in planning and executing attacks.

*The case of file handles is similar.

Identifiers that refer to state shared between client and server can be the basis of disruption attacks since clients and server necessarily assume that neither side will change the state corpus without appropriate notice.

While these identifiers do not need to be authenticated, they are associated with higher-level entities for which change of the state represented by those entities is subject to peer authentication.

*Unexpected closure of stateids or changes in state sequence values can disrupt client access as no clients have provision to deal with this source of interference.

While encryption may make it more difficult to execute such attacks attackers can often guess stateid's since server generally not randomize them.

*Similarly, modification to NFSv4.1 session state information can result in confusion if an attacker changes the slot sequence by assuring spurious requests. Even if the request is rejected, the slot sequence is changed and clients may a difficult time getting back in sync with the server.

While encryption may make it more difficult to execute such attacks attackers can often guess slot id's and obtain sessinid's since server generally do not randomize them.

*

it is necessary that modification of the higher-levell entities be restricted to the client that created them.

*For NFSv4.0, the relevant entity is the clientid.

*for NFSv4.1, the relevant entity is the sessionid.

Identifiers describing the issuer of the request, whether in numeric or string form always require authentication.

13.3. Authentication Provided by specific RPC Flavors

Different flavors differ quite considerably, as discussed below;

*When AUTH_NONE is used, the user making the request is neither authenticated nor identified to the server.

Also, the server is not authenticated to the client and has no way to determine whether the server it is communicating with is an imposter.

*When AUTH_SYS is used, the user making is the request identified but there no authentication of that identification.

As in the previous case, the server is not authenticated to the client and has no way to determine whether the server it is communicating with is an imposter.

*When RPCSEC_GSS is used, the user making the request is authenticated as is the server peer responding.

13.4. Authentication Provided by the RPC Transport

Different transports differ quite considerably, as discussed below. In contrast to the case of RPC flavors, any authentication happens once, at connection establishment, rather than on each RPC request. As a result, it is the client and server peers, rather than individual users that is authenticated.

*For most transports, such as TCP and RPC-over-RDMA version 1, there is no provision for peer authentication.

As a result use of AUTH_SYS together with such transports is inherently problematic.

*Some transports provide for the possibility of mutual peer authentication.

14. Security of Data in Flight

14.1. Data Security Provided by the Flavor-associated Services

The only flavor providing these facilities is RPCSEC_GSS. When this flavor is used, data security can be negotiated between client and server as described in Section 15.2. However, when data security is provided at the transport level, as described in Section 14.2, the negotiation of privacy and integrity support is unnecessary,

Other flavors, such as AUTH_SYS and AUTH_NONE have no such data security facilities. When these flavor are used, the only data security is provided by the transport.

14.2. Data Security Provided by the RPC Transport

Some transports provide data security for all transactions performed on them, eliminating the need for that security to be provided or negotiated by the selection of particular flavors, mechanism, or services.

15. Security Negotiation

[Author Aside]: All unannotated paragraphs in this section are considered to b part of Consensus Item #32a

As previously in NFSv4, we use the term "negotiation" to characterize the process of the server providing a set of options and the client selecting one.

The use of SECINFO, possibly with SECINFO_NONAME, remains the primary means by which the security parameters are determined. The addition of transports to flavors in providing security has resulted in the following changes:

*Transport-related security choices are typically decided at connection-establishment so there needs to be provision for negotiation at this point.

*Despite the above, because the choices of flavor and transport affect one another, SECINFO has been extended by the addition of pseudo-flavors, while retaining the existing XDR, to allow negotiation of transport choices and accompanying connection establishment options, in addition to selection of flavors and accompanying services. This allows server policies for such matters to be different for different portions of the namespace.

15.1. Flavors and Pseudo-flavors

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #32b

The flavor field of the secinfo4 items returned by SECINFO and SECINO_NONAME have always allowed pseudo flavors to be included. However, previous treatments of these operations have not provided information about how responses containing such pseudo-flavors are to be interpreted.

Those pseudo-flavors now provide a means of extending the negotiation process so it is capable of providing for the negotiation of the use particular RPC transports and security-related options for the connections established using those transports.

The flavors AUTH_NONE, AUTH_SYS and RPCSEC_GSS continue to indicate the acceptability of the corresponding method of user authentication, user identification, or user non-identification, when used with a particular RPC transport.

The flavor AUTH_TLS, which is not used as part of issuing requests is not included in this list and is treated as a connection-type-specifying pseudo-flavor.

secinfo4s for the flavor RPCSEC_GSS contains additional information describing the specific security algorithm to be used and the

ancillary services to be provided (e.g. integrity, privacy) when these services are not provided by the transport.

Such flavors are referred to as "identification-specifying flavors"

The classification below organizes the flavors and pseudo-flavors used in security negotiation while Section 15.4 describes how the set of secinfos in a response can be used by the client to select acceptable combinations of security flavor, security mechanism, security services, security-related transports, and security-related connection characteristics.

*The pseudo-flavors designating a particular transport type such as XPT_TCP or XPT_RDMA.

These pseudo-flavors are referred to as "transport- specifying flavors".

*The pseudo-flavors designating restrictions on acceptable connection characteristics include XPCH_ENCRYPT, XPCH_PEERAUTH, and XPCH_SECURE.

Such pseudo-flavors are referred to as "transport-restriction pseudo-flavors".

*The pseudo-flavors denoting sets of allowable connection types. While many connection types are designated by a combination of a flavor designating a transport with on designating a set of connection characteristics, there are pseudo-flavors, called "conection-type pseido-flavror that designate a a set of connection types directly.

These include the flavor AUTH_TLS which is equivalent to XP_TCP combined with XPCH_ENCRYPT, and the pseudo-flavor XP_TCP_SECURE equivalent to XP_TCP combined with XPCH_SECURE.

*The special pseudo-flavors, XPBREAK, XPCLEAR and XPCURRENT

15.2. Negotiation of Security Flavors and Mechanisms

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #32c

For the current connection, this proceeds as it has previously, when security-relevant transports were not available. Flavor entries, including those including mechanism information are listed in order of server preference and apply, by default, to the current connection, which is normally is favored by the server.

When other transport-identifying pseudo-flavors appear before the flavor entries, then the server is indicating that these transport types are also acceptable, with the server preference following the ordering of the entries. In this case, any flavor entries that follow a transport entry specify that those flavor are usable with the transport types or connection types denoted by that transport entry.

15.3. Negotiation of RPC Transports and Characteristics

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #32d

First we define some necessary terminology.

*A transport-specifying pseudo-flavor specifies one of a small set of RPC transport types such as TCP or RDMA. There are also pseudo-flavors that specify a set of transport types such as XPT_ALL.

*Connection characteristics are designations of security-relevant characteristics or sets of characteristics that connections might have.

There are pseudo-flavors associated with connection characteristics such as XPCH_CLPEERAUTH, denoting client-peer authentication and XPCH_ENCRYPT, denoting the presence of an encrypted channel. The pseudo-flavor XPCH_SECURE denotes the presence of peer mutual authentication together with the use of an encrypted channel.

*The combination of a transport type with a set of connection characteristics is considered a connection type. While many connection types are designated by a combination of a flavor designating a transport with on designating a set of connection characteristics, there are pseudo-flavors that designate a set of connection types directly.

For example, the flavor AUTH_TLS is equivalent to XP_TCP combined with XPCH_ENCRYPT and XPCH_CLPEERAUTH while the pseudo-flavor XP_TCP_SECURE equivalent to XP_TCP combined with CONCH_SECURE.

*A flavor specification designates a specific flavor, or, in the case of RPCSEC_GSS, a flavor combined with additional mechanism and service information.

*A flavor assignment denotes the association of a specific flavor specification with a connection type.

A secinfo response will designate a set of valid flavor assignments with an implied server ordering derived from the order that the entries appear in.

In interpreting the response array the client is to maintain sets of designated transport types, connection characteristics and connection types specified individually (i.e. without separately specifying transport types and connection characteristics). When a flavor specification is encountered, that flavor is considered valid when used with all currently active connection types, defined by the union of the individually specified connection types and the Cartesian product of the current transport types and current connection types.

The presumed ordering of these assignments is as follows:

*When one of the connection types was specified directly by a connection type, the position of that specification is compared to that of either the other individually-specified connection type or the earlier of the transport-type specification and the connection characteristics specification.

*In other cases, the position of the transport type specifications are considered first withe the position of the connection characteristics considered if necessary.

*If neither of the above resolve issue, the position of the flavor specification is considered.

*The type of the current connection is considered to be specified first, implicitly.

*There are provisions, described in Section 15.4 to modify this ordering, as may be necessary, for example, when the current connection, while acceptable is, of lower server preference.

15.4. Overall Interpretation of SECINFO Response Arrays

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #32e.

This section summarizes the processing necessary on the client to interpret the response to a SECINFO or SECINFO_NONAME request and determine, at the specified part of the server's namespace:

*The set of transport types acceptable to the server.

*The set of connection types acceptable to the server.

*For each acceptable connection type, the set of flavors acceptable to the server.

*For each acceptable connection type for which encryption is not provided and for which the flavor RPCSEC_GSS is accepted, a set of services to be required when using the flavor on connections of that type.

For each of the items for which the set of acceptable elments has more than one element, the server's preference order can be communicated to the client.

This section provides the same information as Sections 15 through 15.3 but the presentation is in the form of an algorithm.

The algorithm needs to maintain the following information as part of the context shared with the operations defined in Sections 15.4.2 and 15.4.3

*The ordered set of currently specified transport types.

Because of the need to retain ordering information, a mask cannot be used to represent this.

Because duplicates are not allowed, the size of this data can be limited, based on the number of valid transport types.

The initial value is the empty list.

*An array of sets of current transport restrictions.

Since there are three possible transport characteristics: encryption, client-peer authentication, and server-peer authentication, a given connection may have eight possible states and a set of allowed characteristics represented by an eight-bit mask of allowed combinations of chaacteristics.

The initial value is a single entry with all bits set, indicating no current restrictions.

*The ordered set of additional connection types, beyond the Cartesian product of the current sets of transport types and of connection restrictions.

Each entry consists of a transport type together with a connection characteristics mask.

The initial value is a single-entry list whose only entry consists of the transport type of the current connection combined

with a set of transport characteristics in effect for the current connection and no other possibilities.

*The pseudo-flavor most previously processed.

When this is not one of the special pseudo-flavors, the pseudo-flavor type is sufficient.

The initial value is transport-restriction pseudo-flavor type reflecting the fact that the state of the current connection is the initial basis for flavor specification.

*The output list showing, in order, the combinations of connection types combined with flavors, or, in the case of RPCSEC_GSS, of flavor triples.

15.4.1. Interpretation of SECINFO Response Arrays (Core)

[Author Aside]: This preliminary section, which is currently incomplete, is considered Consensus Item #49a.

[Author Aside]: There are problems with the indenting in this section. This may be due to to an xml2rfc bug or I may be using ul incorrectly, or both. Will try to fix this for the next draft.

Processing of the response proceeds through each secinfo4 in the response. For each such entry, the flavor value, which may be a pseudo-flavor, controls what is to be done.

The current entry is fetched.

The pseudo-flavor for that entry controls what is done next.

If the pseudo-flavor specifies a transport type or a set of transport types, the following is done:

If the previous pseudo-flavor was not a transport-specifying flavor,

Connection type expansion, as described in Section 15.4.2 is performed.

If the flavor designates a single transport type,

The connection type is added to the current list, if it is not already present in the list.

Otherwise, each included transport type is added to the list in turn.

If the pseudo-flavor specifies a connection restriction, the following is done:

If the previous pseudo-flavor was not a transport-specifying flavor, is not a restriction-specifying flavor and is not XPBREAK,

Connection type expansion, as described in Section 15.4.2 is performed.

If the previous pseudo-favor was XPBREAK,

A new restriction entry, initialized with all bits one, is added to the list.

In any case, the newly-specified restrictions are anded with the last entry in the list.

If the pseudo-flavor specifies a set of connection types or is XPCURRENT, the following is done:

Inoformation specified by the current flavor is added to the list of additional connection types, if that same set of connection type is not already present.

If the pseudo-flavor is XPBREAK,

Nothing specific is done at this point.

If the pseudo-flavor is XPCLEAR,

The set of data maintained by algorithm, including the flavor output is reset to its initial state.

In addition the set of additional connection types is cleared to empty state, i.e. information about the current connection is removed.

If the pseudo-flavor is an authentication flavor,

Flavor expansion, as described in Section 15.4.3 is performed to combine the current flavor or flavor triple with each of the currently specified connection types.

Then the current pseudo-flavor is saved as the previous pseudo-flavor.

We then move on to the next entry.

15.4.2. Connection Type Transcription

[Author Aside]: This section will be provided in a later draft as part of Consensus Item #48a.

15.4.3. Flavor Transcription

[Author Aside]: This section will be provided in a later draft as part of Consensus Item #48b.

15.5. SECINFO

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #33a.

The description in the sub-sections below, while it adheres to the XDR appearing [6], [7], [8], [9] and [11]. will supersede the descriptions in [7] and [8].

This is necessary to adapt the security negotiation process to the presence of transport-level security services such as encryption and peer authentication.

Similar changes are necessary in the parallel SECINFO_NONAME operation introduced in NFSv4.1. These are expected to be done as part of the rfc5661bis effort.

15.5.1. SECINFO ARGUMENTS

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #33b.

```
struct SECINFO4args {
        /* CURRENT_FH: directory */
        component4      name;
};
```

Figure 1

15.5.2. SECINFO RESULTS

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #33c.

```
/*
 * From RFC 2203
 */
enum rpc_gss_svc_t {
        RPC_GSS_SVC_NONE        = 1,
        RPC_GSS_SVC_INTEGRITY   = 2,
        RPC_GSS_SVC_PRIVACY     = 3
};

struct rpcsec_gss_info {
        sec_oid4        oid;
        qop4            qop;
        rpc_gss_svc_t   service;
};

/* RPCSEC_GSS has a value of '6' - See RFC 2203 */
union secinfo4 switch (uint32_t flavor) {
 case RPCSEC_GSS:
         rpcsec_gss_info        flavor_info;
 default:
         void;
};

typedef secinfo4 SECINFO4resok<>;

union SECINFO4res switch (nfsstat4 status) {
 case NFS4_OK:
        /* CURRENTFH: consumed */
         SECINFO4resok resok4;
 default:
         void;
};
```

Figure 2

15.5.3. SECINFO DESCRIPTION

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #33d.

The SECINFO operation is used by the client determine the appropriate RPC authentication flavors, security mechanisms and encrypting transports to access a specific directory filehandle, file name pair. SECINFO should apply the same access approach used for LOOKUP when evaluating the name. In consequence, if the requester does not have the appropriate access to LOOKUP the name, then SECINFO will behave the same way and return NFS4ERR_ACCESS.

The result will contain an array that represents the security flavor, security mechanisms and transports available, with an order corresponding to the server's preferences, the most preferred being first in the array. The client is free to pick whatever security flavors, mechanisms and transports it both desires and supports, or to pick in the server's preference order the first one it supports. The array entries are represented by the secinfo4 structure. The field 'flavor' will contain one of the following sorts of values:

*a value of AUTH_NONE, AUTH_SYS (as defined in RFC 5531 [4]).

*AUTH_TLS as described in ...

*A pseudo-flavor defined in Section 18.2

*RPCSEC_GSS (as defined in RFC 2203 [2]).

*Any other security flavor or pseudo-flavor registered with IANA.

For the flavors other than RPCSEC_GSS, no additional security information is returned. For a return value of RPCSEC_GSS, a security triple is returned that contains the mechanism object identifier (OID, as defined in RFC 2743 [3]), the quality of protection (as defined in RFC 2743 [3]), and the service type (as defined in RFC 2203 [2]). It is possible for SECINFO to return multiple entries with flavor equal to RPCSEC_GSS with different security triple values.

On success, the current filehandle is consumed, so that, if the operation following SECINFO tries to use the current filehandle, that operation will fail with the status NFS4ERR_NOFILEHANDLE.

If the name has a length of zero, or if the name does not obey the UTF-8 definition in circumstances in which UTF-8 names are required, the error NFS4ERR_INVAL will be returned.

See Sections 15.2 through 15.4 for additional information on the use of SECINFO.

15.5.4. SECINFO IMPLEMENTATION (general)

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #33e.

The SECINFO operation is expected to be used by the NFS client when the error value of NFS4ERR_WRONGSEC is returned from another NFS operation. This signifies to the client that the server's security policy is different from what the client is currently using. At this point, the client is expected to obtain a list of possible security flavors and choose what best suits its policies.

15.5.5. SECINFO IMPLEMENTATION (for NFSv4.0)

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #34a.

The server's security policies will determine when a client request receives NFS4ERR_WRONGSEC. The operations that may receive this error are LINK, LOOKUP, LOOKUPP, OPEN, PUTFH, PUTPUBFH, PUTROOTFH, RENAME, RESTOREFH, and, indirectly, READDIR. LINK and RENAME will only receive this error if the security used for the operation is inappropriate for the saved filehandle. With the exception of READDIR, these operations represent the point at which the client can instantiate a filehandle into the current filehandle at the server. The filehandle is either provided by the client (PUTFH, PUTPUBFH, PUTROOTFH) or generated as a result of a name-to-filehandle translation (LOOKUP and OPEN). RESTOREFH is treated differently because the filehandle is a result of a previous SAVEFH. Even though the filehandle, for RESTOREFH, might have previously passed the server's inspection for a security match, the server will check it again on RESTOREFH to ensure that the security policy has not changed.

If the client is to resolve an error return of NFS4ERR_WRONGSEC, the following will occur:

* For LOOKUP and OPEN, the client will use SECINFO with the same current filehandle and name as provided in the original LOOKUP or OPEN to determine the acceptable combinations of transport types, transport restrictions, and flavor-based triple use to make requests directed at the specified portion of the server namespace.

* For LINK, PUTFH, RENAME, and RESTOREFH, the client will use SECINFO and provide the parent directory filehandle and the object name that corresponds to the filehandle originally provided by the PUTFH or RESTOREFH, or, for LINK and RENAME, the SAVEFH.

* For LOOKUPP, PUTROOTFH, and PUTPUBFH, the client will be unable to use the SECINFO operation since SECINFO requires a current filehandle and none exist for these three operations. Therefore, the client must iterate through the security triples expected to be available at the client for use by the current connection (i.e, because they are **REQUIRED** and attempt the PUTROOTFH or PUTPUBFH operation repeatedly, once for each possible triple. In the unfortunate event that none of the MANDATORY security triples are supported by the client and server, the client should try using others that are believed to be available. It is desirable to do so in a manner which provides encryption or at least

integrity support integrity. Often his will be possible if the connection is encrypted. In other cases. the client can try using AUTH_NONE, but because such forms lack integrity checks, there is an element of risk in doing so. However the risk can be made small if the server returns NFS4ERR_WRONGSEC when entering any subdirectory of the root or public filehandle.

The READDIR operation will not directly return the NFS4ERR_WRONGSEC error. However, if the READDIR request included a request for attributes, it is possible that the READDIR request's security triple does not match that of a directory entry. If this is the case and the client has requested the rdattr_error attribute, the server will return the NFS4ERR_WRONGSEC error in rdattr_error for the entry. This will allow SECINFO to be issued for that entry with the same current file handle as used for the READDIR and a name derived from the entry for which the error was noted.

[Author Aside]: The following paragraph seems dubious since it would best if the server tells the client to use, rather than leaving him to guess, and the server will know what he supports. Would like to delete this as part of Consensus Item #47a, unless compatibility issues make that impossible.

[Previous Treatment (Item #47a)]: Note that a server MAY use the AUTH_NONE flavor to signify that the client is allowed to attempt to use authentication flavors that are not explicitly listed in the SECINFO results. Instead of using a listed flavor, the client might then, for instance, opt to use an otherwise unlisted RPCSEC_GSS mechanism instead of AUTH_NONE. It may wish to do so in order to meet an application requirement for data integrity or privacy. In choosing to use an unlisted flavor, the client SHOULD always be prepared to handle a failure by falling back to using AUTH_NONE or another listed flavor. It cannot assume that identity mapping is supported and should be prepared for the fact that its identity is squashed.

15.5.6. SECINFO IMPLEMENTATION (for NFSv4.1 and v4.2)

[Author Aside]: All unannotated paragraphs in this section are considered to be part of Consensus Item #33f.

As mentioned, the server's security policies will determine when a client request receives NFS4ERR_WRONGSEC.

See Table 14 of [8] for a list of operations that can return NFS4ERR_WRONGSEC. in the case of v4.2, there might be extensions allowed to return NFS4ERR_WRONGSEC. In addition, when READDIR returns attributes, the rdattr_error (Section 5.8.1.12 of [8]) can contain NFS4ERR_WRONGSEC.

Note that CREATE and REMOVE MUST NOT return NFS4ERR_WRONGSEC. The rationale for CREATE is that unless the target name exists, it cannot have a separate security policy from the parent directory, and the security policy of the parent was checked when its filehandle was injected into the COMPOUND request's operations stream (for similar reasons, an OPEN operation that creates the target MUST NOT return NFS4ERR_WRONGSEC). If the target name exists, while it might have a separate security policy, that is irrelevant because CREATE MUST return NFS4ERR_EXIST. The rationale for REMOVE is that while that target might have a separate security policy, the target is going to be removed, and so the security policy of the parent trumps that of the object being removed. RENAME and LINK MAY return NFS4ERR_WRONGSEC, but the NFS4ERR_WRONGSEC error applies only to the saved filehandle (see Section 2.6.3.1.2 of [8]). Any NFS4ERR_WRONGSEC error on the current filehandle used by LINK and RENAME MUST be returned by the PUTFH, PUTPUBFH, PUTROOTFH, or RESTOREFH operation that injected the current filehandle.

With the exception of LINK and RENAME, the set of operations that can return NFS4ERR_WRONGSEC represents the point at which the client can inject a filehandle into the "current filehandle" at the server. The filehandle is either provided by the client (PUTFH, PUTPUBFH, PUTROOTFH), generated as a result of a name-to-filehandle translation (LOOKUP and OPEN), or generated from the saved filehandle via RESTOREFH. As Section 2.6.3.1.1.1 of [8] states, a put filehandle operation followed by SAVEFH MUST NOT return NFS4ERR_WRONGSEC. Thus, the RESTOREFH operation, under certain conditions (see Section 2.6.3.1.1 of [8]), is permitted to return NFS4ERR_WRONGSEC so that security policies can be honored.

The READDIR operation will not directly return the NFS4ERR_WRONGSEC error. However, if the READDIR request included a request for attributes, it is possible that the READDIR request's security triple did not match that of a directory entry. If this is the case and the client has requested the rdattr_error attribute, the server will return the NFS4ERR_WRONGSEC error in rdattr_error for the entry.

To resolve an error return of NFS4ERR_WRONGSEC, the client does the following:

*For LOOKUP and OPEN, the client will use SECINFO with the same current filehandle and name as provided in the original LOOKUP or OPEN to enumerate the available security triples.

*For the rdattr_error, the client will use SECINFO with the same current filehandle as provided in the original READDIR. The name passed to SECINFO will be that of the directory entry (as

returned from READDIR) that had the NFS4ERR_WRONGSEC error in the rdattr_error attribute.

*For PUTFH, PUTROOTFH, PUTPUBFH, RESTOREFH, LINK, and RENAME, the client will use SECINFO_NO_NAME { style = SECINFO_STYLE4_CURRENT_FH }. The client will prefix the SECINFO_NO_NAME operation with the appropriate PUTFH, PUTPUBFH, or PUTROOTFH operation that provides the filehandle originally provided by the PUTFH, PUTPUBFH, PUTROOTFH, or RESTOREFH operation.

NOTE: In NFSv4.0, the client was required to use SECINFO, and had to reconstruct the parent of the original filehandle and the component name of the original filehandle. The introduction in NFSv4.1 of SECINFO_NO_NAME obviates the need for reconstruction.

*For LOOKUPP, the client will use SECINFO_NO_NAME { style = SECINFO_STYLE4_PARENT } and provide the filehandle that equals the filehandle originally provided to LOOKUPP.

16. Future Security Needs

[Author Aside]: All unannotated paragraphs in this section are considered part of Consensus Item #35a.

[Author Aside]: This section is basically an outline for now, to be filled out later based on Working Group input, particularly from Chuck Lever who suggested this section and has ideas about many of the items in it.

*Security for data-at-rest, most probably based on facilities defined within SAN.

*Support for content signing.

*Revision/extension of labelled NFS to provide true interoperability and server-based authorization.

*Work to provide more security for RDMA-based transports. This would include the peer authentication infrastructure now being developed as part of RPC-over-RDMA version 2. In addition, there is a need for an RPC-based transport that provides for encryption, which might be provided in number of ways.

*Work, via extensions, to provide attributes describing server behavior to the client. This is likely to have an important role in resolving security issues connected with ACLs where there is both a new preferred approach together with legacy implementations built when the specifications wither offered no

preferred approach or treated that preference as easily dispensed with.

17. Security Considerations

17.1. Changes in Security Considerations

Beyond the needed inclusion of a threat analysis and the fact that all minor versions are dealt with together, there are a number of substantive changes in the approach to NFSv4 security presented in RFCs 7530 and 8881 and that appearing in this document.

This document will not seek to speculate how the previous treatment, now viewed as incorrect, came to be written, approved, and published. However, it will, for the benefit of those familiar with the previous treatment of these matters, draw attention to the important changes listed here.

- There is a vastly expanded range of threats being considered as described in Section 17.1.1

- New facilities available at the RPC transport level can be used to deal with security issues, as described in Section 17.1.2

17.1.1. Wider View of Threats

Although the absence of a threat analysis in previous treatments makes comparison most difficult, the security-related features described in previous specifications and the associated discussion in their security considerations sections makes it clear that earlier specifications took a quite narrow view of threats to be protected against.

One aspect of that narrow view that merits special attention is the handling of AUTH_SYS, at that time in the clear, with no client peer authentication.

With regard to specific threats, there is no mention in existing security consideration sections of:

- Denial-of-service attacks.

- Client-impersonation attacks.

- Server-impersonation attacks.

The handling of data security in-flight is even more troubling.

- Although there was considerable work in the protocol to allow use of encryption to be negotiated when using RPCSEC_GSS. The

existing security considerations do not mention the potential need for encryption at all.

It is not clear why this was omitted but it is a pattern that cannot repeated in this document.

* The case of negotiation of integrity services is similar and uses the same negotiation infrastructure.

  In this case, use of integrity is recommended but not to prevent the corruption of user data being read or written.

  The use of integrity services is recommended in connection with issuing SECINFO (and for NFSv4.1, SECINFO_NONAME). The presence of this recommendation in the associated security considerations sections has the unfortunate effect of suggesting that the protection of user data is of relatively low importance.

17.1.2. Transport-layer Security Facilities

Such transport-level RPC facilities as RPC-over-TLS provide important ways of proving better security for all the NFSv4 minor versions.

In particular:

* The presence of encryption by default will deal with security issue regarding data-in-flight, for both RPCSEC_GSS and AUTH_SYS.

* Peer authentication provided by the server eliminates the possibility of a server-impersonation attack, even when using AUTH_SYS.

* When mutual authentication is part of connection establishment, there is a possibility, where an appropriate trust relationship exists, of treating the userid's presented in AUTH_SYS, as effectively authenticated, based on the authentication of the client peer.

17.1.3. Approach to Implementation Semantic Divergences

[Consensus Needed (Items #36a, #37a)]: For a variety of reasons, there are many cases in which one approach to security is preferred where others are allowed, if only to give time for implementers to adapt to the preferred approach. In such cases the word "**SHOULD** is used to introduce the preferred while others are allowed to allow

compatibility by limiting the valid reasons to bypass the recommendation. Such instances fall into two classes:

* [Consensus Needed (Item #36a)]: In adapting to the availability of security services provided by the RPC transport, allowance has been made for implementations for which these new transport are not available and for which, based on previous standards-track guidance, AUTH_SYS us used, in the clear, without client-peer authentication.

* [Consensus Needed (Item #37a)]: In dealing with server implementations that support both ACLs and the mode attribute, previous specifications have allowed a wide range of possible server behavior in coordinating these attributes. While now clearly defining the recommended behavior in dealing with these issues, allowance has been made to give times for implementations to conform to the new recommendations.

[Consensus Needed (Items #36a, #37a)]: The threat analysis within this Security Considerations section will not deal with older servers for which allowance has been made but will explore the consequences of the recommendations made in this document.

17.1.4. Compatibility and Maturity Issues

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #38a.

Given the need to drastically change the NFSv4 security approach from that specified previously, it is necessary for us to be mindful of:

* The difficulty that might be faced in adapting to the newer guidance because the delays involved in designing, developing, and testing new transport- level security facilities such as RPC-over-TLS.

* The difficulty in discarding or substantially modifying previous existing deployments and practices, developed on the basis of previous normative guidance.

For these reasons, we will not use the term "**MUST NOT**" in some situations in which the use of that term might have been justified earlier. In such cases, previous guidance together with the passage of time may have created a situation in which the considerations mentioned above in this section may be valid reasons to defer, for a limited time, correction of the current situation making the term "**SHOULD NOT**" appropriate, since the difficulties cited would constitute a valid reason to not allow what had been recommended against.

17.1.5. Discussion of AUTH_SYS

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #39a.

An important change concerns the treatment of AUTH_SYS which is now divided into two distinct cases given the possible availability of support from the transport layer.

When such support is not available, AUTH_SYS **SHOULD NOT** be used, since it makes the following attacks quite easy to execute:

- The absence of authentication of the server to the client allow server impersonation in which an imposter server can obtain data to be written by the user and supply corrupted data to read requests.

- The absence of authentication of the client user to the server allow server impersonation in which an imposter client can issue requests and have them executed as a user designated by imposter client, vitiating the server's authorization policy.

  With no authentication of the client peer, common approaches, such as using the source IP address can be easily defeated, allowing unauthenticated execution of requests made by the pseudo clients

- The absence of any support to protect data-in-flight when AUTH_SYS is used result in further serious security weaknesses.

In connection with the use of the term "**SHOULD NOT**" above, it is understood that the "valid reasons" to use this form of access reflect the Compatibility and Maturity Issue discussed above in Section 17.1.4 and that it is expected that, over time, these will become less applicable.

17.2. Security Considerations Scope

17.2.1. Discussion of Potential Classification of Environments

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #40a.

This document will not consider different security policies for different sorts of environments. This is because,

- Doing so would add considerable complexity to this document.

*The additional complexity would undercut our main goal here, which is to discuss secure use on the internet, which remain an important NFSv4 goal.

*The ubiquity of internet access makes it hard to treat corporate network separately from the internet per se.

*While small networks might be sufficiently isolated to make it reasonable use NFSv4 without serious attention to security issues, the complexity of characterizing the necessary isolation makes it impractical to deal with such cases in this document.

17.2.2. Discussion of Environments

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #40b.

Although the security goal for Nfsv4 has been and remains "secure use on the internet", much use of NFSv4 occurs on more restricted IP networks with NFS access from outside the owning organization prevented by firewalls.

This security considerations section will not deal separately with such environments since the threats that need to be discussed are essentially the same, despite the assumption by many that the restricted network access would eliminate the possibility of attacks originating inside the network by attackers who have some legitimate Nfsv4 access within it.

In organizations of significant size, this sort of assumption of trusted access is usually not valid and this document will not deal with them explicitly. In any case, there is little point in doing so, since, if everyone can be trusted, there can be no attackers, rendering threat analysis superfluous.

This does not mean that NFSv4 use cannot, as a practical matter, be made secure through means outside the scope of this document including strict administrative controls on all software running within it, frequent polygraph tests, and threats of prosecution. However, this document is not prepared to discuss the details of such policies, their implementation, or legal issues associated with them and treats such matters as out-of-scope.

Nfsv4 can be used in very restrictive IP network environments where outside access is quite restricted and there is sufficient trust to allow, for example, every node to have the same root password. The case of a simple network only accessible by a single user is similar. In such networks, many thing that this document says "SHOULD NOT" be done are unexceptionable but the responsibility for making that determination is one for those creating such networks to

take on. This document will not deal further with NFSv4 use on such networks.

17.3. Major New Recommendations

17.3.1. Recommendations Regarding Security of Data in Flight

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #41a.

It is RECOMMENDED that requesters always issue requests with data security (i.e. with protection from disclosure or modification in flight) whether provided at the RPC request level or by the RPC transport, irrespective of the responder's requirements.

It is RECOMMENDED that implementers provide servers the ability to configure policies in which requests without data security will be rejected as having insufficient security.

it is RECOMMENDED that servers use such policies over either their entire local namespace or for all file systems except those clearly designed for the general dissemination of non-sensitive data.

17.3.2. Recommendations Regarding Client Peer Authentication

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #41b.

It is RECOMMENDED that clients provide authentication material whenever a connection is established with a server capable of using it to provide client peer authentication.

It is RECOMMENDED that implementers provide servers the ability to configure policies in which attempts to establish connections without client peer authentication will be rejected.

it is RECOMMENDED that servers adopt such policies whenever requests not using RPCSEC_GSS are allowed to be executed.

17.3.3. Issues Regarding Valid Reasons to Bypass Recommendations

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #41c.

Clearly, the maturity and compatibility issues mentioned in Section 17.1.4 are valid reasons to bypass the above recommendations, as long as these issues continue to exist.

[Author Aside]: The question the working group needs to address is whether other valid reasons exist.

[Author Aside]: In particular, some members of the group might feel that the performance cost of encrypted transports constitutes, in itself, a valid reason to ignore the above recommendations.

[Author Aside]: I cannot agree and feel that accepting that as a valid reason would undercut Nfsv4 security improvement, and probably would not be acceptable to the security directorate. However, I do want to work out an a generally acceptable compromise. I propose something along the following lines:

In dealing with these issues it needs to be understood that the transport-based encryption facilities are designed to be compatible with facilities to offload the work of encryption and decryption. When such facilities are not available, at a reasonable cost, to NFSv4 servers and clients anticipating heavy use of NFSv4, then the lack of such facilities can be considered a valid reason to bypass the above recommendations, as long as that situation continues.

17.4. Data Security Threats

Will be addressed in a later draft as part of Consensus Item #42a.

17.5. Authentication-based threats

17.5.1. Attacks based on the use of AUTH_SYS

Will be addressed in a later draft as part of Consensus Item #43a.

17.5.2. Attacks on Name/Userid Mapping Facilities

Will be addressed in a later draft as part of Consensus Item #44a.

17.6. Disruption and Denial-of-Service Attacks

17.6.1. Attacks Based on the Disruption of Client-Server Shared State

Will be addressed in a later draft as part of Consensus Item #45a.

17.6.2. Attacks Based on Forcing the Misuse of Server Resources

Will be addressed in a later draft as part of Consensus Item #46a.

18. IANA Considerations

[Author Aside]: All unannotated paragraphs in this section are to be considered part of Consensus Item #33f.

Because of the shift from implementing security-related services only in connection with RPCSEC_GSS to one in which transport-level

security has a prominent role, a number if new values need to be assigned.

These include new authstat values to guide selection of a Transports acceptable to both client and server, presented in Section 18.1 and new pseudo-flavors to be used in the process of security negotiation, presented in Section 18.2.

18.1. New Authstat Values

[Author Aside]: All unannotated paragraphs in this section are to be considered part of Consensus Item #33g.

The following new authstat values are necessary to enable a server to indicate that the server's policy does not allows requests to be made on the current connection because of security issues associated with the rpc transport. In the event they are received, the client needs to establish a new connection.

* The value XP_CRYPT indicates that the server will not support access using unencrypted connections while the current connection is not encrypted.

* The value XP_CPAUTH indicates that the server will not support access using connections for which the client peer has not authenticated itself as part of connection while the current connection has not been set up in that way.

18.2. New Authentication Pseudo-Flavors

[Author Aside]: All unannotated paragraphs in this section are to be considered part of Consensus Item #33h.

The new pseudo-flavors described in this section are to be made available to allow their return as part of the response to SECINFO operation described in Section 15.5 and for similar operations.

The following transport-specifying flavors are to be defined:

* XPT_TCP denotes use of a TCP transport to support to RPC. The use of TLS as provided by RPC-with-TLS is orthogonal to the transport type, as is the use of optional authentication features. Such facilities are treated as transport characteristics.

  When RDMA support is layered on TCP, that fact is not relevant to the transport type, which is still XPT_RDMA.

* XPT_RDMA denotes use of any version of RPC-over-RDMA to support RPC. Although Version 1 has no security-support, future version may have such facilities.

In any case, the specification of the presence or need for such facilities are handled as transport characteristics.

*XPT_ALL is currently equivalent to XPT_TCP followed by XPT_RDMA. When new transport types are made available for use wit NFSv4, it is intended that

The following transport-restricting flavors are to be defined:

*XPCH_ENCRYPT restrict connections to those providing encryption.

*XPCH_SVRAUTH restricts connections allowed to those that provide, at connection time authentication of the server peer.

*XPCH_CLAUTH restricts connections allowed to those that provide, at connection time authentication of the server peer.

*XPCH_PEERAUTH is equivalent to XPCH_SVRAUTH combined with XPCH_CLAUTH.

*XPCH_SECURE is equivalent to XPCH_ENCRYPT combined with XPCH_PEERAUTH.

The follow connection-specifying flavors are to be defined:

*AUTH_TLS is equivalent to XP_TCP combined with XPCH_ENCRYPT and XPCH_CLPEERAUTH

*XP_TCP_SECURE is equivalent to XP_TCP combined with XPCH_SECURE.

The following special flavors are to be defined:

*XPCLEAR reset the state of processing to an empty state. This is useful if the current connection type is not usable for the specified region of the namespace or if it is of lower server preference.

*XPBREAK forces the use of a new set transport restrictions, separate from previous ones and applying to the same set of transport types.

*XPCURRENT specifies that the type of the current connection is usable for access, with the preference derived from its location in the SECINFO response array.

19. References

19.1. Normative References

[1]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[2] Eisler, M., Chiu, A., and L. Ling, "RPCSEC_GSS Protocol Specification", RFC 2203, DOI 10.17487/RFC2203, September 1997, <https://www.rfc-editor.org/info/rfc2203>.

[3] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, DOI 10.17487/RFC2743, January 2000, <https://www.rfc-editor.org/info/rfc2743>.

[4] Thurlow, R., "RPC: Remote Procedure Call Protocol Specification Version 2", RFC 5531, DOI 10.17487/RFC5531, May 2009, <https://www.rfc-editor.org/info/rfc5531>.

[5] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[6] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 Protocol", RFC 7530, DOI 10.17487/RFC7530, March 2015, <https://www.rfc-editor.org/info/rfc7530>.

[7] Haynes, T., Ed. and D. Noveck, Ed., "Network File System (NFS) Version 4 External Data Representation Standard (XDR) Description", RFC 7531, DOI 10.17487/RFC7531, March 2015, <https://www.rfc-editor.org/info/rfc7531>.

[8] Noveck, D., Ed. and C. Lever, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 8881, DOI 10.17487/RFC8881, August 2020, <https://www.rfc-editor.org/info/rfc8881>.

[9] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 External Data Representation Standard (XDR) Description", RFC 5662, DOI 10.17487/RFC5662, January 2010, <https://www.rfc-editor.org/info/rfc5662>.

[10] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 Protocol", RFC 7862, DOI 10.17487/RFC7862, November 2016, <https://www.rfc-editor.org/info/rfc7862>.

[11] Haynes, T., "Network File System (NFS) Version 4 Minor Version 2 External Data Representation Standard (XDR)

Description", RFC 7863, DOI 10.17487/RFC7863, November 2016, <https://www.rfc-editor.org/info/rfc7863>.

[12] Myklebust, T. and C. Lever, "Towards Remote Procedure Call Encryption By Default", Work in Progress, Internet-Draft, draft-ietf-nfsv4-rpc-tls-11, 23 November 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-nfsv4-rpc-tls-11>.

19.2. Informative References

[13] Bensley, S., Thaler, D., Balasubramanian, P., Eggert, L., and G. Judd, "Data Center TCP (DCTCP): TCP Congestion Control for Data Centers", RFC 8257, DOI 10.17487/RFC8257, October 2017, <https://www.rfc-editor.org/info/rfc8257>.

Appendix A. Changes Made

This section summarizes the substantive changes between the treatment of security in previous minor version specification documents (i.e. RFCs 7530 and 8881) and the new treatment applying to NFSv4 as a whole.

This is expected to be helpful to implementers familiar with previous specifications but also has an important role in verifying the working group consensus for these changes and in guiding the search for potential compatibility issues.

A.1. Motivating Changes

A number of changes reflect the basic motivation for a new treatment of NFSv4 security. These include the ability to obtain privacy and integrity services from the RPC transport rather than as a service ancillary to a specific authentication flavor.

This motivated a major reorganization of the treatment of security together with a needed emphasis on the security of data in flight. In addition, the security negotiation framework for NFSv4 has been significantly enhanced to support the combined negotiation of authentication-related services and transport characteristics.

Despite these major changes there are not expected to be compatibility issues between peers supporting secure transport characteristics and those without such support.

Another such change was in the treatment of AUTH_SYS, previously described, inaccurately, as an "OPTIONAL means of authentication" with the unfortunate use of the RFC2119 keyword obscuring the negative consequences of the typical use of AUTH_SYS (in the clear;

without client-peer authentication) for security by enabling the execution of unauthenticated requests.

The new treatment avoids the inappropriate use of term "authentication" for all activities triggered by the use of RPC authentication flavors and clearly distinguishes those flavors providing authentication from those providing identification only or neither identification nor authentication.

A.2. Other Major Changes

The need to make the major changes discussed in Appendix A.1 has meant that much text dealing with security has needed to be significantly revised or rewritten. As a result of the process, may issues involving unclear, inconsistent, or otherwise inappropriate text were uncovered and needed to be dealt with.

While the author believes such changes are necessary, the fact that we are changing a document adopted by consensus requires the working group to be clear about the acceptability of the changes. This applies to both the troublesome issues discussed in Section 3.4 and to the other changes included below.

Because of concurrent re-organizations, the ordering of the list follows the text of the current version which may differ considerably from that in earlier versions of the I-D.

*In order to deal better with the fact that ACLs have multiple uses some significant structural changes have been made.

  Section 5, a new top-level section describes the the structure of ACLs,

*In Section 7.2, makes clear that owner and owner group are essentially **REQUIRED** attributes.

*Also in Section 7.2, there is added clarity in the discussion of support for multiple authorization approaches by eliminating use of the subjective term "reasonable semantics".

  In connection with this clarification, we have switched from describing the needed co-ordination between modes and acls as "goals" to describing them as "requirements" to give clients some basis for expecting interopherability in handling these issues.

  As a result of this shift to a prescriptive framework applying to all minor versions it becomes possible to treat all minor versions together. In earlier versions of this document, it had been assumed that NFSv4.0 was free to ignore the relevant

prescriptions first put forth in RFC 5661 and only needed to address the "goals" for this co-ordination.

*

Appendix B. Issues for which Consensus Needs to be Ascertained

The section helps to keep track of specific changes which the author has made or intends to make to deal with issues found in RFCs 7530 and 7881. The changes listed here exclude those which are clearly editorial but includes some that the author believes are editorial but for which the issues are sufficiently complicated that working group consensus on the issue is probably necessary.

These changes are presented in the table below, organized into a set of "Consensus Items" identified by the numeric code appearing in annotations in the proposed document text. For each such item, a type code is assigned with the following codes defined:

*"NM" denotes a change which is new material that is not purely editorial and thus requires Working Group consensus for eventual publication.

*"BE" denotes a change which the author believes is editorial but for which the change is sufficiently complex that the judgment is best confirmed by the Working Group.

*"BC" denotes a change which is a substantive change that the author believes is correct. This does not exclude the possibility of compatibility issues becoming an issue but is used to indicate that the author believes any such issues are unlikely to prevent its eventual acceptance.

*"CI" denotes a change for which the potential for compatibility issues is major concern with the expected result that working group discussion of change will focus on clarifying our knowledge of how existing clients and server deal with the issue and how they might be affected by the change or the change modified to accommodate them.

*"NS" denotes a change which represents the author's best effort to resolve a difficulty but for which the author is not yet confident that it will be adopted in its present form, principally because of the possibility of troublesome compatibility issues.

When the document is promoted to a working group document, there should be very few issues in this state and, for each such issue, a clear plan to address the possibility of any compatibility

problems, enabling resolution of the issue to be reasonably anticipated.

* "NE" denotes change based on an existing issue in the spec but for which the replacement text is incomplete and needs further elaboration.

  There will be no changes in this state when promotion to a working group document is requested.

* "WI" denotes a potential change based on an existing issue in the spec but for which replacement text is not yet available because further working group input is necessary before drafting. It is expected that replacement text will be available in a later draft once that discussion is done.

  There will be no changes in this state when promotion to a working group document is requested.

* "LD" denotes a potential change based on an existing issue in the spec but for which replacement text is not yet available due to the press of time. It is expected that replacement text will be available in a later draft.

  There will be no changes in this state when promotion to a working group document is requested.

When asterisk is appended to a state of "NM", "BE" or "BE" it that there has been adequate working group discussion leading one to reasonably expect it will be adopted, without major change, in a subsequent document revision.

Such general acceptance is not equivalent to a formal working group consensus and it not expected to result in major changes to the draft document,

On the other hand, once there is a working group consensus with regard to a particular issue, the document will be modified to remove associated annotations, with the previously conditional text appearing just as other document text does. The issue will be removed from this table although it will be mentioned in Appendices A.2 or A.1

It is is expected that these designations will change as discussion proceeds and new document versions are published. It is hoped that most such shifts will be upward in the above list or result in the deletion of a pending item, by reaching a consensus to accept or reject it. This would enable, once all items are dealt with, an eventual request for publication as an RFC, with this appendix having been deleted.

#	Type	...References...	Substance
1	NM*	#1a in S 4	Outline of new approach to authetication/ identification, replacing confusion about the matter in previous specifications.
2	NM*	#2a in S 4	Introduction to and outline of changes needed in negotiation framework to support provision of security by the RPC transport.
3	BE	#3a in S 5.4	Conversion of mask bit descriptions from being about "permissions" to being about the action permitted, denied, or specified as being audited or generating alarms.
4	CI	#4a in S 5.4	Elimination of uses of **SHOULD** believed inappropriate in Section 5.4.
5	BE	#5a in S 5.4	Explicit inclusion of ACCESS as an operation affected in the mask bit definitions.
6	CI	#6a in S 5.4 #6b in S 5.6 #6c in S 7.3.1	New/revised description of the role of the "sticky bit" for directories, both with respect to ACL handling and mode handling.
7	BE	#7a in S 5.4	Clarification of relationship between READ_DATA and EXECUTE.
8	CI	#8a in S 5.4	Revised discussion of relationship between WRITE_DATA and APPEND_DATA.
9	BC	#9a in S 5.4	Clarification of how ADD_DIRECTORY relates to RENAME.
10	BC	#10a in S 5.4 #10b in S 5.5	Revisions in handling of the masks WRITE_RETENTION and WRITE_RETENTION_HOLD.
11	CI	#11a in S 5.4 #11b in S 5.5 #11c in S 5.11	Explicit recommendation and requirements for mask granularity, replacing the previous treatment which gave the server license to ignore most of the previous section, placing clients in an unfortunate situation.
12	BC	#12a in S 5.6 #12b in S 5.6.1	Revised treatment of directory entry deletion.
13	BC	#13a in 5.7	Attempt to put some reasonable limits on possible non-support (or variations in the support provided) for the ACE flags. This is to replace a situation in which the client has no real way to deal with the freedom granted to server implementations.
14	BC	#14a in S 5.11	Explicit discussion of the case in which aclsupport is not supported.
15	BC	#15a in S 5.11	

#	Type	...References...	Substance
		#15b in S 7.1 #15c in S 7.2	Handling of the proper relationship between support for ALLOW and DENY ACEs.
16	NM	#16a in S 5.1	Discussion of coherence of acl, sacl, and dacl attributes.
17	BC	#17a in S 7.1 #17b in S 7.2	Relationship of support for ALLOW and DENY ACEs
18	BC	#18a in S 7.1 #18b in S 7.2	Need for support of owner, owner_group.
19	CI	#19a in S 7.2	Revised discussion of coordination of mode and the ACL-related attributes.
20	WI	#20 in S 7.3.1	More closely align ACL_based and mode-based semantics with regard to SVTX.
21	BC	#21a in S 7.3.1 #21b in S 9.3 #21c in S 9.6	Introduce the concept of reverse-slope modes and deal properly with them. The decision as to the proper handling is addressed as Consensus Item #28.
22	BC	#22a in S 8.1	Revise treatment of divergences between AC/mode authorization and server behavior, dividing the treatment between cases in which something authorized is still not allowed (OK), and those in which something not authorized is allowed (highly problematic from a security point of view).
23	BC	#23a in S 8.2	Revise discussion of client access to of ACLs.
24	BE	#24a in S 8.2	Delete bogus reference.
25	CI	#25a in S 3.3 #25b in S 9.1 #25d in S 9.7 #25e in S 9.9 #25f in S 9.10 #25g in S 9.11	Revised description of co-ordination of acl and mode attributes to apply to NFSv4 as a whole. While this includes many aspects of the shift to be more specific about the co-ordination requirements including addressing apparently unmotivated uses of the terms "**SHOULD**" and "**SHOULD NOT**", it excludes some arguably related matters dealt with as Consensus Items #26 and #27.
26	CI	#26a in S 9.2 #26 in S 9.6.3	Decide how ACEs with who values other than OWNER@, Group, or EVERYONE@ are be dealt with when setting mode.
27	CI	#27a in S 9.2 #27b in S 9.3 #27c in S 9.4	Concerns the possibility of establishing one way of computing a mode from an acl that clients can depend on, rather than two or an unbounded number.
28	WI	#28a in S 9.3 #28 in S 9.6.3	Decide how to address flaws in mapping to/from reverse- slope modes.
29	BC	#29 in S 9.6.3	

#	Type	...References...	Substance
			Address the coordination of mode and ACL-based attributes in unified way for all minor versions.
30	CI	#30a in S 9.6.1 #30b in S 9.6.2 #30c in S 9.6.3	New proposed treatment of setting mode incorporating some consequences of anticipated decisions regarding other consensus items (#26, #28, #29)
31	WI	#31a in S 9.6.3	Need to deal with mask bits ACE4_READ_ATTRIBUTES, ACE4_WRITE_RETENTION, ACE4_WRITE_RETENTION_HOLD, ACE4_READ_ACL to reflect the semantics of the mode attribute.
32	BC	#32a in S 15 #32b in S 15.1 #32c in S 15.2 #32d in S 15.3 #32e in S 15.4	Expanded negotiation framework to accomodate multiple transport types and services derivable from transport characteristics, i.e. encryption and peer authentication.
33	BE	#33a in S 15.5 #33b in S 15.5.1 #33c in S 15.5.2 #33d in S 15.5.3 #33e in S 15.5.4 #33f in S 18 #33g in S 18.1 #33h in S 18.2	Reorganization of description of SECINFO op to apply to all minor versions. Assumes basics of proposal for Item #32.
34	BC	#34a in S 15.5.6	Revision to NFSv4.0 SECINFO implementation section to be compatible with expanded approach to negotiation. Assumes basics of proposals for Items #32 and #33.
35	NE	#35a in S 16	Now has preliminary work on future security needs.
36	CI	#36a in S 17.1.3	Threat analysis only dealing with RECOMMENDED behavior regarding use of transport security facilities and handling of AUTH_SYS.
37	CI	#37a in S 17.1.3	Threat analysis only dealing with RECOMMENDED behavior with regard to acl support including ACL/mode coordination.
38	CI	#38a in S 17.1.4	Address the need to temporarily allow unsafe behavior mistakenly allowed by previous specifications

#	Type	...References...	Substance
39	CI	#39a in S 17.1.5	Define new approach to AUTH_SYS.
40	CI	#40a in S 17.2.1 #40a in S 17.2.2	Discussion of scope for security considerations and the corresponding threat analysis.
41	CI	#41a in S 17.3.1 #41b in S 17.3.2 #41c in S 17.3.3	Discuss major new security recommendations regarding protection of data in flight and client peer authentication. Also, covers the circumstances under which such recommendations can be bypassed.
42	LD	#42a in S 17.4	Placeholder for threat analysis section regarding security of data in flight.
43	LD	#43a in S 17.5.1	Placeholder for threat analysis section dealing with the use of AUTH_SYS.
44	LD	#44a in S 17.5.2	Placeholder for threat analysis section dealing with attacks on userid/name mapping.
45	LD	#45a in S 17.6.1	Placeholder for threat analysis section dealing with disruption attacks based on attacks on shared state.
46	LD	#46a in S 17.6.2	Placeholder for threat analysis section dealing with attacks on shared state design to cause misuse of resources.
47	CI	#47a in S 15.5.5	Dubious paragraph which should be deleted if there are no compatibility issues that make that impossible.
48	CI	#48a in S 15.4.2 #48b in S 15.4.3	Missing pieces of secinfo processing algorithm that didn't get done in -02.
49	NE	#49a in S 15.4.1	Main secinfo processing algorithm that needs to finished in -02.

Table 3

The following table summarizes the issues in each particular state, dividing them into those associated with the motivating changes for this new document and those that derive from other issues, that were uncovered later, once work on a new treatment of NFSv4 security had begun.

Type	Cnt	Issues
NM*(M)	2	1, 2
BE(M)	1	33
BC(M)	2	32, 34
CI(M)	7	36, 38, 39, 40, 41, 47, 48

Type	Cnt	Issues
NE(M)	2	35, 49
LD(M)	5	42, 43, 44, 45, 46
All(M)	19	As listed above.
NM(O)	1	16
BE(O)	4	3, 5, 7, 24
BC(O)	12	9, 10, 12, 13, 14, 15, 17, 18, 21, 22, 23, 29
CI(O)	11	4, 6, 8, 11, 19, 25, 26, 27, 28, 30, 37
WI(O)	2	20, 31
All(O)	30	As described above
All	49	Grand total for above table.

Table 4

Acknowledgments

The author wishes to thank Tom Haynes for his helpful suggestion to deal with security for all NFSv4 minor versions in the same document.

The author wishes to draw people's attention to Nico Williams' remark that NFSv4 security was not so bad, except that there was no provision for authentication of the client peer. This perceptive remark, which now seems like common sense, did not seem so when made, but it has served as a beacon for those putting NFSv4 security on a firmer footing. We appreciate this perceptive guidance.

The author wishes to acknowledge the important role of the authors of RPC-with-TLS, Chuck Lever and Trond Myklebust, in moving the NFS security agenda forward and thank them for all their efforts to improve NFS security.

The author wishes to thank Chuck Lever for his many helpful comments about nfsv4 security issues, his explanation of many unclear points, and and much important guidance he provided that is reflected in this document.

The author wishes to thank Rick Macklem for his role in clarifying possible server policies regarding RPC-over-TLS and bringing possible approaches to the attention of the working group.

Author's Address

David Noveck (editor)
NetApp
1601 Trapelo Road, Suite 16
Waltham, MA 02451
United States of America

Phone: +1-781-572-8038

Email: davenoveck@gmail.com