Workgroup: NFSv4
Updates: 8881, 7530 (if approved)
Published: 29 July 2023
Intended Status: Standards Track
Expires: 30 January 2024
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 RPC on a per-connection basis.

The current 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 the resulting 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 30 January 2024.

Copyright Notice

Copyright (c) 2023 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

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 described in a way that is inappropriate in Stanrds Track document, as described in Section 3.4

The need to make major changes in the security approach for three Proposed Standards ([RFC7530] for NFSv4.0, [RFC8881] for NFSv4.1, and [RFC7862] for NFSv4.2) raises troubling issues. These changes are necessary as explained in Section 1.1. These troubling issues often concern compatibility and compliance issues as described in Section 1.3.

1.1. Document Motivation

A new treatment of security is necessary because:

- Previous treatments paid insufficient attention to security issues regarding data in flight, assuming that security could reasonably be provided on an optional basis, to secure some portion of the server namespace.

- 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 [RFC9289]) provides facilities that NFSv4 clients and servers will need to use to provide security for data in flight and mitigate the lack of user authentication when AUTH_SYS is used.

- The handling of ACLs is not described in the detail normally necessary to make it possible to implement servers. Because of the broad license granted by previous specifications to server implementations to choose how to behave, clients are forced to

accept a broad range of server behaviors, with no way of reliably determining server behavior actually implemented.

A considerable part of the difficulty that came with these choices is alleviated as described in Section 8.1 which provided a way for the clients to be unaware of the structure of ACLs, even though there is a discussion of the structure of these objects, which is not to be relied upon.

The remaining issue caused by this approach, of providing a way for clients to determine the authorization-related effects of setting. ACLs or other security-related attributes, cannot be addressed in this way. The obvious alternative, of respecifying server behavioral requirements, is unlikely to be doable in a reasonable time given the existence of server implementations with a wide range of behavioral choices. The most likely resolution includes being clear, in this document, on the wide range of behavioral choices currently allowed, and providing ways, as described in Section 16.2, for clients to be aware of these choices, or to allow the negotiatiation of mutually acceptable approaches. See Appendix A.3 for further discussion.

1.2. Document Annotation

In order to make progress on difficult issues which will require that changes be made in the existing handling of security 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 will 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, need 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 or the phase "Entire Bulleted Item" to indicate it applies to all paragraphs within a specific bulleted item.

1.3. Compatibility and Compliance Issues

Changes that need to be adopted in this document will need to eventually change the behavior of clients and servers that were written to conform to earlier protocol specifications. There are two important classes of such changes discussed in Sections 1.3.1 and 1.3.2 below.

As [RFC2119] was originally conceived, compliance and compatibility issues were tightly bound together so that a change to compliance specifications would inevitably give rise to compatibility issues. However, over time, behaviors have come to be denigrated by use of the terms "**SHOULD NOT**" and "**MUST NOT**" to warn implementors of a harm deriving from from insecure operation rather than peer incompatibility.

When making changes in compliance requirements/recommendations we need to deal with the possibility that, in changing the specification, we might cause a previously compliant implementation to become non-compliant. Some implementers take the view that the compliance status of their implementations is of less importance than other considerations such as compatibility with local file

system semantics. Others feel it is important to maintain the compliance status of existing implementations. In any case, the working group has been reluctant, when making such necessary changes, to make previously compliant implementations non-compliant, and will try not to do in cases in which it is known or can reasonably be expected that such implementations exist.

Although there is no way to be sure about the non-existence of such implementations, the working group has made judgments about what implementations are likely to exist. For example, when internationalization for NFSv4.0 was changed in the transition to [RFC7530], many previously non-compliant server implementations became officially compliant and there was a potential for conflict with implementations compliant with RFC3530 [RFC3530], if any such implementations existed.

In that particular case, it was decided that no such RFC3530-compliant server implementations existed and there was no need to accommodate servers whose internationalization was written to conform to [RFC3530] since no such servers existed and there were no client implementations expecting such behavior. As a result, no actual implementations became non-compliant as a result of this necessary shift.

In the corresponding cases dealt with in this document, the situation is more difficult since it may be harder to determine the actual behavior of all existing implementations, since the authors might no longer be actively involved with implementation issues. However, it will be necessary to warn implementors of the negative consequences of certain behaviors without going so far as to declare these practices non-compliant. For the most part, this document will use the terms "**SHOULD**" and "**SHOULD NOT**" to draw attention to the problems with troublesome behaviors that previous specifications mentioned with no indication of the problems with them. In such contexts, it is made clear that the need to maintain existing patterns of interoperation is a valid reason to bypass the normative term. The intention is to continue to allow previously acceptable implementations to be considered compliant while not placing the troublesome behaviors on the same levels as other alternatives as would happen if we used the terms "**MAY**" or "**OPTIONAL**" or continued to use some of the unfortunate practices discussed in Section 1.3.2.

This approach allows implementations to accept input from peers written in accord with previous specification while not obligating them to do so. The intent is to allow a transition to newer, better

behaviors over time as client and server policies evolve. However, the specifics of this anticipated transition will vary:

* In the case of the issues dealt with in Section 1.3.1, the focus is on making implementers aware of the security problems with practices previously considered acceptable.

  It is specified that these practices **SHOULD NOT** be used by clients or allowed by servers in order to draw attention to the security problems with them. At the same time, the discussion of valid reasons to by pass these recommendations allows them to continue to be used while the infrastructure is developed to make their replacements easy to use.

  Servers are encouraged to adopt policies foreclosing client use but not obligated to do so.

* In the case of the issues dealt with in Section 1.3.2, the focus is different. In these cases, while we anticipate making changes in compliance specifications, there is no need to address a specific set of troublesome practices. Instead, the problem to be addressed is the vast range of allowable server behaviors previously defined as allowed, although not necessarily explicitly. This server-centered approach has made compatibility a hit-or-miss matter, requiring serious consideration of the question of what specific instances of multiple server behaviors need to be allowed and how clients can find out the choices that servers have made and possibly affect them, also making appropriate provision, as with other optional features, about how to adapt to the server's behavioral choices or to decide that they do not meet its needs.

  It appears that this approach was motivated, at least in large part, by the desire to fully support much of Windows ACL semantics while accommodating Unix servers incapable of providing much of that functionality. As a result, the working group will need to provide a way for the server to explicitly opt out of providing Windows functionality that it cannot provide and that Linux clients are not prepared to use.

  In addition, the working group will have to restrict, or at least better organize, sever behavioral choices related to the handling of ACLs.

  Some approaches to providing the client information about server choices or allowing negotiation of mutually satisfactory handling of these issues are discussed in Section 16.2.

1.3.1. Dealing with Recognized Mistakes

As an example, we consider the handling of AUTH_SYS (presumably in the clear, without client peer authentication), described in previous specifications as "**OPTIONAL**". While the authors might have only been indicating that servers could choose not to support it, and that clients had to be prepared for it not to be supported by the server, the likely import of this designation, for clients, was to indicate to implementers that they could choose to use AUTH_SYS and that the authors of the spec were, by not recommending otherwise, as we might now feel they should have done, indicating there there were no issues whereby using AUTH_SYS in this form had the capacity to cause harm and that clients using AUTH_SYS in this way were to be considered specification- compliant.

As the protocol was implemented and further developed in subsequent minor versions, the specifications, which had Security Considerations sections that did not contain threat analyses, had no place to indicate to users and implementers of NFSv4 the security problems that come with AUTH_SYS use and it continued to be used heavily while encryption was only available to clients using RPCSEC_GSS and left as a choice that was not frequently used, despite the security issues that this raised.

We are now at a point at which we have to recognize that a mistake was made in this regard and have to be clear about the security issues present in many common implementations of the protocol. As we seek to do this, it is important to understand the compliance effects of doing so. This document, when adopted, will supersede previous specifications which took a different approach. Although it might, given the security issues with AUTH_SYS, make sense to say that it "**MUST NOT**" be used in that way, the working group is very reluctant to retroactively declare previously compliant behavior non-compliant, even in this case where there is good reason to do so. A more likely approach is to say that clients "**SHOULD NOT**" do this while making it clear that the difficulty of changing existing implementations and potential compatibility with existing peers are a valid reason to bypass the recommendation. Servers are, as before, allowed to support AUTH_SYS but "**SHOULD**" only do so when using additional security facilities that make this safe. The effect would be to create a clear set of recommendations to new implementations while providing for continued use of previously compliant implementations to continue as needed,

This approach gives rise to compatibility issues, but leaves them to implementors and users to resolve, while making clear the security issues with the old approach.

1.3.2. Dealing with Pervasive Uncertainty

Addressing the issues described in Section 3.4 raises similar issues. In this case as well, we will also need to make changes in implementation behavior going forward and try to do so without declaring existing behavior non-compliant. However, we cannot, as we did in the example above, identify a specific set of bad choices, and try to come up with replacements, that reflect our new understanding of security issues.

In this case, it is necessary, as has been done in other cases in which NFSv4 tried to accommodate the needs of both UNIX and Windows, to decide what part of the non-UNIX semantics is required and which part is an optional extension, which unix-oriented clients would not use and unix-oriented servers might not support. When this sort of issue is not given the attention it needs, problems can result, although the nature and severity of the problems depend on the specifics of feature.

In the case of byte-range locks, severs were given a choice as to implement byte-range locks in an advisory or mandatory fashion. Although servers could choose to do either, there was no way for a client to determine which of these two incompatible semantic models the server implemented. As a result, unix-based clients and applications assumed the advisory model and could not interoperate successfully with servers implementing the mandatory model. Applications requiring mandatory semantics could only interoperate with a small set of servers which chose to support the mandatory model that has very few users. The normal way of dealing with situations like this is to make the server's behavioral choice available to the client as an attribute, as is provded for in [RFC8178]

In the case of ACLs, we have a difficult situation to resolve. Instead of having a set of individual mistakes which can now be recognized as such, we have a situation in which the existing specifications have created an unacceptable interoperability situation in relation to ACL implementations. Existing specs have not paid proper attention to the need to make decisions in the face of disagreements regarding proper server behavior and have in various ways avoided the need to compromise and reach a reasonable consensus but instead have made it the job of the specifications to consider valid any remotely similar server implementations as valid, leaving clients little that could do accept a wide range of server behavior as valid.

The working group did not provide any definition of a restricted level of ACL support that Unix servers could provide and that would address the needs of Unix clients. Instead the, only definition of

ACL support was one that met the needs of Windows clients while providing a large set of facilities that did not fit within ACL needs of Unix systems.

As it became clear that considerable pieces of the ACL functionality were only of value to Windows clients and were difficult to provide on Unix servers, the working group declined to draw that line but instead haphazardly opened the spec to a wide variety of possible server behaviors by the following methods:

* Essentially made each ACL mask bit its own optional feature with each server free to choose to implement that bit or not. Unfortunately, unlike most cases in which an optional feature is provided for, there was no means for the client to find out whether the server provided support a for particular mask bit or, if it did not, how it dealt with authorizations normally controlled by that mask bit.

* Similarly, made all ACL flags essentially optional.

  This was done, as in the case above, without using the terms "**OPTIONAL**" or "**MAY**" which might have suggested the need to provide a means for clients to find out about the choices that servers were allowed to make.

* Used "**SHOULD**" in situations in which it was not clear why the term was used and possible to determine what might be considered valid reasons to bypass the recommendation.

  Since clients have no means to find out whether these unspecified reasons apply, they find themselves in a position similar the one that would exist if "**MAY**" were used.

  Again a range of server behavior was allowed without any analysis of the question of whether allowing such variability was necessary, its effect on client interoperability, or how the client might find out about the server's choices.

* While building many aspects of the interactions of the mode attribute and ACLs around a mapping from ACLs to modes, the existing specifications allowed at least two different such mappings to be used, creating a difficult interoperability situation for clients.

  The preferred mapping is introduced using "**SHOULD**", again being unclear about possible reasons to bypass the recommendation, if that is what it is. It is stated that the use of "**SHOULD**" is "intentional", leaving one to wonder how to deal with the presumably unintentional uses of "**SHOULD**".

Although, it is claimed that the intention was "to avoid invalidating existing implementations that compute the mode according to the withdrawn POSIX ACL draft (1003.1e draft 17)", it is unclear how this choice relates to the working group's decision to base NFSv4 ACL on Windows ACLs rather than on the withdrawn POSIX draft ACLs. It is possible that there were different opinions on the proper mapping and that, instead of resolving the issue, the working group avoided the need to resolve this disagreement, by simply allowing both methods, ignoring the effect on interoperability.

The effect of "invalidating " such implementations would be to make it clear that they are implementations of a different sort of ACL, rather then of the NFSv4 ACL model, adopted by the working group. Instead, the specification allowed for many possible hybrids of both models.

A case can be made that clients, who often support only ACL-related APIs based on the Unix ACL model are entitled to matching ACL support, but the hybrid approach adopted makes no provision for client choice and does not even allow the client to find out the characteristics of the particular hybrid implementation, chosen by the server.

*The updating of ACLs in response to change in the mode attribute is another area in which previous specifications have chosen, for reasons which remain unclear, to allow a wide range of server behavior.

Only a single part of the necessary ACL change is clearly specified as compliant with the spec.

On the other hand, use of this approach is discouraged using non-normative terms

The combination of all the above has created a difficult interoperability situation. While it is often noted that clients are working in this environment, without complaining about the situation, we have to understand the reason why this might be so:

*Unix clients have good reason to stay away from ACLs that use features, such as extend-only ACLs, or ACLs whose set of users with permission to modify ACLs is different from the single owning user. This leaves them no occasion to discover that the handling of these is not clearly specified.

Some clients have ACL-related APIs (e.g. those appropriate to Unix ACLs) while for clients that have APIs oriented to NFSv4 ACLs, any incompatibilities will be perceived by the applications, leading to a situation in which problems cannot be

brought to the attention on the working group in any context in which a reasonably prompt resolution can be expected.

* By discouraging direct client access to the ACLs, the need for clear specification of ACL semantics were reduced.

  On the other hand, clients were given no corresponding way to avoid clear specification of ACL semantics when deciding on the ACL to effect a given pattern of authorization for a file.

* No interoperability testing.

* Low expectations in the ACL support area, which might be a consequence of choices made and not made decades ago

Given the importance of security, this unfortunate situation cannot be allowed to persist indefinitely even if it might not be possible to fully resolve without the sort of protocol extensions that cannot be done in a document such as this. However, we can make a set of changes such as the following in this document to describe clearly designate optional server behavior and leave it to later documents to provide the necessary extensions defining ways for the client to find out about or affect server choices.

* Eliminate the suggestions that the fact that servers behave in a particular way necessarily implies that clients have to accept this behavior

* Use "**MAY**" rather than "**SHOULD**" in cases in which "**SHOULD**" is used without any clear indications of valid reasons to ignore a recommendation.

* Eliminate, to the degree possible, situation in which, multiple behaviors are allowed, with no clear understanding why allowing such freedom is needed.

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 [RFC2119] [RFC8174] 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 [RFC2119]), 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 [RFC2119], 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 [RFC9289], 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 facilities provided on a per-connection basis. 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. this work is discussed in Section 3.4

3.1. Per-connection Security Features

There are a number of security-related facilities that can be provided on a per-connection basis, eliminating the need to provide such support on a per-request basis, based on the RPC auth-flavor used.

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

* The transport or a layer above it 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 that provide such encryption, servers can establish policies allowing access to certain pieces of the namespace using such security facilities, or limiting access to those providing privacy, allowing the use of either per-connection encryption or privacy services provided by RPCSEC_GSS.

* The transport or a layer above it 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 RPCSEC_GSS framework.

  This form of authentication is of particular importance when the server allows the use of the auth-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 without 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 [RFC7862]. 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 connection-based 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 by security services such as those provided by RPC-with-TLS.

It needs to be noted that state protection based on RPCSEC_GSS is treated as a version-specific feature and will continue to be described by [RFC8881] or its successors. Also, it needs to be noted that the use of state protection was not discussed in [RFC7530].

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 OPTIONAL pNFS 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 eventual rfc5661bis document). 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 [RFC8257], 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 [RFC8257] 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 [RFC7862].

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 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 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 on a firmer basis, without requiring changes

on older servers that do not support these SMB-oriented attributes.

* 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 file system.

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

Despite the difficulty of addressing these issues, if the protocol is to be secure and ACLs are to be widely available, these problems have to 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 in 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 issues that need to be addressed are described in Appendix A.2 while the possible approaches that might be taken to resolve these issues are described in Appendix A.3.

*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 auth-flavors and other security-related services provided by RPC.

Support for multiple auth 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 auth-flavors is provided for.

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

This auth-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, on a per-connectio basis, by RPC, this support is useful, when such 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 specific RPC support such as that provided by RPC-with-TLS [RFC9289].

- AUTH_NONE does not provide identification of the principal making the request so would only 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.

There are important services that can be provided by RPC, when RPC-with-TLS or similar transport-level facilities are available.

- Such services can provide data security to all requests on the connection. This is to be preferred to data security provided by the RPC auth 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.

- These services 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 client-peer can be authenticated 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.

Because important security-related services depend on the security services, rather than the auth flavor, the process of security negotiation, described in Section 15, has been extended to provide for the negotiation of appropriate connection characteristics at connection time if the server's policy limits the range of transports being used and also when use of a particular auth flavor on a connection with inappropriate security characteristics causes NFS4ERR_WRONGSEC to be returned.

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.

4.1. NFSv4 Security Terminology

In this section, we will define the security-related terminology used in this document. This is particularly important for NFSv4 because many of the terms terms related to security in previous specification may be hard to understand because their meanings have changed or have been used inconsistently, resulting in confusion.

The following terms are listed in alphabetical order:

- "Access Control" denotes any control implemented by a server peer to limit or regulate file system access to file system objects. It includes but is not limited to authorization decisions. Access control features can be divided into those which are "Discretionary" or "Mandatory" as described below.

* "ACL" or "Access Control List" denotes a structure used, like the mode (see below), to defines the privileges that individual users have with respect to a given file. These structures provide more options than modes with regard to the association of privileges with specific users or group and often provide a finer-graned privilege structure as well. This specification will have need to refer to two types of ACLs.

  The ACLs present in the acl, sacl, and dacl attributes are called "NFSv4 ACLs". This ACL format, was modeled on the the semantics of the SMB ACL format which provide a privilege model substantially finer-grained than that provided by POSIX modes.

  [Consensus needed (Item #56a)]: Another ACL type derives from an attempt to define, within POSIX, a UNIX-oriented approach to ACLs which was published as a draft (POSIX 1003.1e draft 17), but subsequently withdrawn. Despite the withdrawal of this draft and the working group's decision to adopt a native NFsv4 ACL format based on SMB ACLs, this document will have to discuss these ACLs, which we will term "UNIX ACLs" because many server file systems do not support the finer-grained privilege model needed by the the NFSv4 ACL model and because many clients are built on systems whose only ACL-related API is based on the UNIX ACL model.

* "authentication" refers to a reliable determination that one making a request is in fact who he purports to be. Often this involves cryptographic means of demonstrating identity.

  This is to be distinguished from "identification" which simply provides a specified identity without any evidence to verify that the identification is accurate.

  In the past, these terms have been confused, most likely because of confusion engendered by th use of the term "authentication flavor" including flavors for which only identification is provided or which do not provide even identification.

* "authorization" refers to the process of determining whether a request is authorized, depending on the resources (e.g. files) to be accessed, the identity of the entity on whose behalf the request was issued, and the particular action to be performed.

  Depending on the type of request, the entity whose identity is referenced can be a user, a peer, or a combination of both.

  Authorization is distinct from authentication. However, performing authorization based on identities which have not been authenticated makes secure operation impossible since use of unauthenticated identities allows acceptance of requests that are

not properly authorized if the sender has the ability, as it typically does, to pretend to be an authorized user/peer.

* "client" refers to the entity responsible for setting up a connection. In most cases the client and the requester reside on the same node but this not always the case for NFSv4 because of the possibility of callback requests in which the server makes some request of the client.

* "confidentiality" refers to the assurance provided, typically through encryption, that the contents of requests and responses are not inadvertently disclosed to unauthorized parties.

* "Discretionary Access Control" denotes forms of access control, that rely on a user, such as the owner, specifying the privileges that various users are to have.

* "Mandatory Access Control" denotes forms of access control that reflect choices made by the server peer and based on its policy and that are typically based on the identity of the client peer rather than the specific user making a request. While such access control is discussed in this document, it is important to note that many forms of mandatory access control are discussed by other NFsv4 documents and that there are forms that are not standardized.

* [Consensus Needed, Entire Bulleted Item (Item #21a)]: "Mode" designates a set of twelve flag bits used by POSIX-based systems to control access to the file with which it is associated. In NFSv4, there are represented by an **OPTIONAL** attribute, which, in practical terms, is always supported by servers and expected by clients.

  The three high-order flags are generally accessed only by the client while low-order bits are divided into three three-bit fields, which give, in order of decreasing numeric value, the privileges to be associated with, the owner of the file, other users in the group owning the file, and users not in the above two categories.

  In most cases, the privileges associated with each successive group are no greater than those for the previous group. Modes whose privileges are of this form are referred to as "forward-slope modes" because the privilege level proceeds downward as successive groups of users are specified. Cases in which the contrary possibility is realized are referred to as "reverse-slope modes".

*"peer" refer to the entity which is charged with requesting or performing a specified request as opposed to the entity on whose behalf the request is requested or performed, the principal;

*"principal" refers to the specific entity (e.g. user) on whose behalf a request is being made.

*"privacy", has in the past been used to refer, to what is now referred to as "confidentiality".

over time, this usage has changed so that the word most often refers to applicability of data to a single individual and person's right to prevent its unauthorized disclosure

As a result, many references to "privacy" in previous are no longer appropriate and really refer to confidentiality.

The NFSv4 protocol has no way to determine whether particular data items raise privacy concerns (In the new sense). NFSv4 provides confidentiality whatever type of data is being accessed so that private data is kept private.

*"integrity" refers to the assurance that data in a request has not been modified in the process of transmission. Such an assurance is generally provided b means of a cryptographic hash of the requests or response.

*"requester" is the entity making a request, whether that entity is on the client-side, as it most often is (forward-direction request) or the server side, in th case of callback (reverse-direction requests)

*"responder" is the entity performing a request, whether that entity is on the server side, as it most often is (forward-direction request) or the client side, in the case of callbacks (reverse-direction requests.

*"server" refers to the entity to which the client connects. In most cases the client and the responder reside on the same node but this not always the case for NFSv4 because of the possibility of callback requests in which the server makes some request of the client.

4.2. NFSv4 Security Scope Limitations

This document describes the security features of the NFSv4 protocol and is unable to address security threats that are inherently outside the control of the protocol implementors. Such matters as out of this document's scope.

As a way of clarifying the threats that this document, and the threat analysis in Section 17.4 can and cannot deal with, we list below the potential threats discussed Section 3.1 of [nist-209] and review how, if at all, it is discussed in the current document. In cases in which the threat is dealt with in this document, distinctions are to be made between cases in which the issues have been dealt with directly or have been delegated to a lower layer on which the protocol is built and whether the issue has been addressed by the changes to NFSv4 security made by this document.

* Regarding the possibility of "Credential Theft or Compromise", this is not a matter that the NFSv4 protocols concern themselves with or can address directly, despite its importance for security. Depending on the auth flavor chosen, either the client (for AUTH_SYS) or a third-party (for RPCSEC_GSS), usually Kerberos, will be responsible for credential verification.

  Since experience has shown that credential compromise (e.g. through "phishing" attacks) is a common occurrence, this problem cannot be ignored, even though NFSv4's reliance on RPC facilities for authentication might be thought to make it out-of-scope as it would be RPC if had an effective solution to the issue. However, the urgency of the situation this issue is such that will be discussed in Section 17.4.2, even though no definitive solutions to this issue are likely before this document is completed and published.

  Regardless of such issues, the likelihood of such compromise has had a role in decisions made regarding the acceptance and use of "superuser" credentials. The possibility of such compromise is also relevant to implementation of means to synchronize credentials when they are managed by the client, as described in Section 17.4.6.1

* Regarding the possibility of "Cracking Encryption", prevention of this is responsibility of the NFSv4 protocols but it is one which has been delegated to RPC, so that its discussion in Security Considerations will rely on RPCSEC_GSS and RPC-with-TLS impelentations to manage the selectio and replacement of kets for encryption so as to limit the possibility of such unwanted encryption key discovery.

* Regarding the possibility of "Infection of Malware and Ransomware", NFSv4 has no direct role in preventing such infection, but does have an important role in limiting its consequences, by limiting the the ability of Malware to access or modify data, through the file access authorization model supported by NFSv4 to limit access to authorized users. Of course, malware will be able to execute on behalf of the user

mistakenly invoking it but the authorization model will server to limit the potential damage.

The possibility of vertical privilege escalation is of concern as regard the possible elevation to "superuser" privileges. For this reason, this document recommends that any such escalation not be effective on the server, even if it happens on local clients for which NFSv4 has no role.

Execution of a ransomeware-based attack requires the attacker to have the ability to read existing data and replacing it with an encrypted version together with the ability to temporarily hide the encryption from ongoing operations by intercepting requests to read encrypted data and substitute the unencrypted data.

*Regarding the possibility of "Backdoors and Unpatched Vulnerabilities", it needs to be noted that the NFSv4 protocols do not specify any backdoors even though it is possible that might choose to provide such backdoors. Since it is not practical to specifically prohibit the existence of such backdoors nor would they be enforceable if written, this document will not attempt to do so. Instead, Section 17.2.3 will note the possibility of such backdoors and recommend against any such implementation, and include implementations containing backdoors in the category of insecure use that will not be dealt with in Section 17.4.

Although it is expected that vulnerabilities will be due to incorrect implementations and thus outside the scope of this document, the possibility of a protocol design errors cannot be excluded. In dealing with such eventualities, it is likely that complete remediation would require co-ordinated changes on the client and server

*Regarding the possibility of "Privilege Escalation", NFSv4 has dealt with the possibility of vertical escalation by not allowing a client-local escalation to superuser privileges to be effective on the server.

With regard to horizontal "escalation", NFSv4 provides for the use of various means RPC authentication of principals but relies on the client operating system to make sure that one user principal cannot masquerade as another.

*Regarding the possibility of "Human Error and Deliberate Misconfiguration", the approach taken is to limit the need for the server to make complicated decisions regarding the security requirements of each section of its namespace, with many opportunities for misconfiguration, if the chosen security

requirements are insufficiently restrictive. This is in contrast to previous specifications which made such configuration the centerpiece of the security approach.

Although it is possible to create configurations where certain data, generally publicly accessible, are to be made available without encryption, this is expected to be a rarely used option with the possibility of in-transit modification kept in mind before adopting such use.

*Regarding the possibility of "Physical Theft of Storage Media", this a matter which, while of concern to those deploying NFSv4 server, will be considered out-of-scope since there is nothing that the protocol could do to deal with this threat.

*Regarding the possibility of "Network Eavesdropping", when the protocol implementation follows the recommendations in this document, the protocol's use of RPC facilities is designed, through the consistent use of encryption to make it difficult for an attacker to have access to the data being transmitted, to modify it, or inject requests into an existing data stream.

The possibility of an attacker with access to the network creating a new connection is best considered as a case of the attacker pretending to be a client and is addressed in Section 17.4.3.

*Regarding the possibility of "Insecure Images, Software and Firmware", while attention to such matters is important for those deploying NFSv4, it is important to note that these are matters outside the control the NFSv4, which has to assume that the infrastructure it is built is working properly. As a result, this document will not deal with the possibility of such threats.

5. Structure of NFSv4 Access Control Lists

NFSv4 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

*ACLs 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 NFSv4 ACLs and discuss ACL-related matters that apply to multiple uses of NFSv4 ACLs, 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 while 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 are 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

Value	Abbreviation	Description
		action specified in 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 #3.

[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 NFSv4 ACLs and mode bits are separate means of authorization, 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 NFSv4 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 the directory sticky bit. It is probably too late to so now, even though it would be allowed, from the protocol point of view, 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 will 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 case of NFSv4 ACLs.

*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 ability 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 NFSv4 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 operation 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 NFSv4 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 NFSv4 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 NFSv4 ACL:

```
nfsuser:ACE4_EXECUTE:ALLOW
```

A subsequent GETATTR of acl attribute 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 need s to be 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 NFSv4 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 clearly needs to 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 NFSv4 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 NFSv4 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 NFSv4 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 NFSv4 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 NFSv4 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 servers 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 supports, while valid, is out-of-scope and need to 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 NFSv4 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 NFSv4 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 is to be added to each new sub-directory created.

If this flag is set in an ACE in an NFSv4 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 is to 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 NFSv4 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.

[Author Aside, including list]: so far, so good, but the following problems need to be addressed:

*Lack of clarity about which special identifiers can be understood by NFSv4.

*Confusion of "authentication" and "identification".

[Previous treatment (Item #50a)]: 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.

[Consensus Needed (Item #50a)]: These identifiers, except for OWNER@, GROUP@, EVERONE@, cannot be reliably understood when an NFS client accesses the server, but might have meaning when a local process accesses the file or when protocols other than NFSv4 are used As a result, when ACEs containing these who values are encountered, the server is free to make its own judgment as to whether any particular request will be treated as matching.

[Consensus Needed (Item #50a)]: The ability to display and modify these permissions is provide for by NFSv4, even though they are not useful when processing NFSv4 requests,

Who	Description
OWNER	The owner of the file.
GROUP	The group associated with the file.
EVERYONE	[Previous treatment (Item #50a)]: The world, including the owner and owning group. [Consensus Needed (Item #50a)]: All requesters, including the owner, members of the owning group, and requests for which no user information is available.
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	[Consensus Needed (Item #50a)]: Accessed without any authentication of the user principal.
AUTHENTICATED	[Consensus Needed (Item #50a)]: Any authenticated user (opposite of ANONYMOUS).
SERVICE	Accessed from a system service.

Table 2

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

{Previous treatment (Item #51a)]: 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.

[Author Aside]: I don't understand what might be valid reasons to ignore this or how a server would respond in the case the that it was ignored.

[Consensus Needed (Item #51a)]: 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 **MUST** be set to zero.

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.

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 are to be cleared. The ACE4_INHERITED_ACE flag can be set in the ACEs of the sacl or dacl (whereas it always needs to 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 NFSv4 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 are to 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 will be the union of the ACEs reported in the dacl and sacl attributes, except that the ACE4_INHERITED_ACE flag will 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 NFSv4 ACL, modifies it, and sets it again, it needs to leave any ACEs marked with ACE4_INHERITED_ACE unchanged, in their original order, at the end of the NFSv4 ACL. If the application is unable to do this, it needs to 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 is to 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 are not to 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 will 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 methods of authorization is used to effect user-based file access control, as described in Section 7. These methods are often termed "Discretionary access control" because they rely on attributes set by particular users, to control acceptable file access.

  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. The labelled NFS provides "Mandatory access control" not under the control of individual users.

  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 17.4.7), server implementations **SHOULD** take steps to prevent unauthorized state changes. This can be done by implementing the state authorization restrictions discussed in Section 11. Because these restrictions apply on a per-peer basis rather than being affected by the identity of the user making the request, it is better to consider them as part of "Mandatory access control".

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 a corresponding NFSv4 ACL, when using only a single ACE type. See Section 9.2 for details.

7.2. Handling of Multiple Parallel File Access Authorization Models

NFSv4 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 NFSv4 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 needs to 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 needs to provide NFSv4 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 is to be the behavior mandated by POSIX, including tthose provisions 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, will 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 will 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 that the second-level list be collapsed to a single paragraph.

*Although [RFC7530] and [RFC8881] 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 needs to be noted that this approach is similar to the ACL-based approach documented in Section 5.6. However there are some semantic differences whose motivation remains unclear and the specification does not mention RENAME, as it needs to.

[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 #21b)]: As will be seen in Sections 9.3 and 9.7, many straightforward ways of dealing with mode that work well with forward-slope modes need adjustment to properly deal with reverse-slope modes, as defined in Section 4.1

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 needs to be automatically modified as discussed in Sections 9.7 and 9.9.

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. Handling of ACCESS and OPEN Operations

The primary means by which client-side users find out whether particular operations are authorized is to attempt those operations and have them executed successfully or rejected with an error. However, clients can use the ACCESS operation to determine in advance whether operations are permitted done without actually attempting those operations.

The use of the ACCESS operation is of particular importance when ACLs exist. This is due in part to the complexity of the ACLs but also derives from cases in which the client is incapable of determining whether a specific ACE applies to a particular request, as when some of the special who-field values are used in an ACE.

An additional difficulty precluding the client performing its own interpretation of an ACL concerns the existing specifications' lack of clear requirements for server support, making it difficult for the client to determine how a server might choose to behave in some cases. Even if this lack of specificity were remedied by clarifying the specification servers would still exist that reflected the previously valid scope of acceptable server behavior. As a result, it is impossible for the client to predict determine the choices a server might make without using ACCESS.

Similarly, the OPEN operation can determine acces rights in advance of actual access. There are two differences from ACCESS that clients need to be aware of, since they might require use of ACCESS in addition to OPEN or make the results of OPEN incompatible with the results of attempting IO operations.

* The permission model of OPEN is coarser-grained than that provided for by ACCESS or the ACE masks defined as part of the definition of NFSv4 ACLs.

  When reading files, the fact that OPEN treats requests to open a file to read it and to execute it together means that the client will need to do an ACCESS operation to determine whether particular reads are to be allowed.

  When writing files, the fact OPEN that does not distinguish between WRITEs which extend the file and those that modify bytes already written means that the client will need to get the permission information using ACCESS or be prepared to have some set of WRITES on a file open for Write rejected as unauthorized.

* [Consensus needed, Entire Bulleted Item (Item #23a)]: Just as with ACCESS, the granting of permission does not foreclose subsequent permission changes, there are good reasons for servers to provide ways of allowing IO allowed at OPEN time to continue even in the face of permission changes that would normally be expected to make the IO operation invalid. These reasons derive from the fact that local operations work this way and that it is is often necessary for requests over NFSv4 to behave just as they would if done locally. As a result, it is desirable for servers to provide this expected behavior in some way since application programs have come to depend on it.

One way of handling this situation is described in Section 8.2. It deals with the most common of these situations in which the owner of the is both writing a file and seeking to make it subsequently unwritable, and while it does not deal with all such situations, it has proven satisfactory for many NFS protocols over a long time period

Another way of dealing with this situation involves the server explicitly using the stateid to reference a particular open made by a user, and avoiding reverification of access for READs and WRITEs made by that user since that verification was made at OPEN time. To do this safely, the server needs to have authenticated principals and client peers and, in order to prevent man-in-middle attacks, it s necessary for all connections on which stateids are sent to provide encryption.

This approach allows clients, for many purposes, to operate without reliable knowledge of ACl-based authorization semantic. However, as noted in Section 1.1, there are other aspects of client-server interaction which require more client knowledge for effective interoperability, as described, for example, in Section 16.2.

8.2. 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 can 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.

* 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 be prepared for such behavior.

[Previous Treatment, Including List (Items #22a, #41a, #52a)]: 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 perform accesses that are denied by the ACL or mode bits 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 [RFC8881]).

[Consensus Needed, Including List (Item #22a)]: It needs to 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 [RFC8881]).

[Author aside, including List, (Item #22a)]: Unlike other cases in which previous specs have granted permission to the server to expand allowed behavior, in the case of "owner-override semantics", the need for supercession does not arise from security issues but instead derives from:

- The unacceptable implication in the superseded text that the fact that a server chooses to do something, means that the client needs to accept that behavior.

- The lack of any definition of the term "owner-override semantics. Subsequent investigation has led to the conclusion that the semantics being referred to derive from the following material in [RFC1813]:

  > Another problem arises due to the usually stateful open operation. Most operating systems check permission at open time, and then check that the file is open on each read and write request. With stateless servers, the server cannot detect that the file is open and must do permission checking on each read and write call. UNIX client semantics of access permission checking on open can be provided with the ACCESS procedure call in this revision, which allows a client to explicitly check access permissions without resorting to trying the operation. On a local file system, a user can open a file and then change the permissions so that no one is allowed to touch it, but will still be able to write to the file because it is open. On a remote file system, by contrast, the write would fail. To get around this problem, the server's permission checking algorithm should allow the owner of a file to access it regardless of the permission setting. This is needed in a practical NFS version 3 protocol server implementation, but it does depart from correct local file system semantics. This should not affect the return result of access permissions as returned by the ACCESS

  As a result, it appears that NFSv4 servers have implemented semantics different from that provided for by POSIX that were

defined in order to deal with lack of an open operation in NFsv3 in the context of NFSv4, which does have an open operation.

* The need to make it clear that an alternate approach to the issue of permission change after open is to allow the OPEN's permission check to be dispositive as long as the appropriate security infrastructure is in place to allow client stateids to be trusted.

[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 might now be superseded. These cases are listed below and discussed in more detail in Section 17.1.3.

[Consensus Needed, Including List (Item #22a, #41a, #52a)]: 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 steps 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.

The principal problem with the above statement is the lack of a clear definition of the term "owner-override semantics". Also, it needs to be made clear 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, investigation has led to the conclusion that the semantics being referred to derive from the discussion of NFSv3 semantics appearing Section 4.4 of [RFC1813] and that this handling has been implemented in a number of NFSv4 servers, despite the fact that NFSv4, unlike NFSv3, does have an open operation.

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.

Since this divergence from POSIX semantics is unlikely to result in security issues, we can clarify the above by saying that a server **MAY** diverge from POSIX semantics by always allowing READ or WRITE to be done by the file owner but that this divergence **MUST NOT** affect the handling of OPEN and ACCESS. It also needs to be noted that servers could address the issue of preventing permission change affecting that handling of READs and WRITEs of open files as described in Section 8.1.

*[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 the new approach to this issue, while Section 17.4.1, will explain how such access is addressed in the threat analysis.

8.3. 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 is to 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 "are not advised to" would be more appropriate.

[Consensus Needed (Item #23b)]: Clients are not 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, where this is possible. This allows the client to act on the results of having the server determine whether or not access is to be granted based on its interpretation of the ACL, rather then the client's which might be different. For a full discussion of limitations on the use of ACCESS and appropriate client approaches to deal these limitations, see Section 8.1.

[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 #23b)]; 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 is to 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 8.2.

[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 [RFC7530] and [RFC8881] 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 relating to the existence absence of the set-mode-masked attribute.

Such co-ordination is necessary is necessary since it is expected that servers providing both sets of attributes will encounter users who have no or very limited knowledge of one and need to work effectively when other users changes that attribute. As a result, these attributes cannot each be applied independently since that would create an untenable situation in which some users who have the right to control file access would find themselves unable to do so.

[Author Aside]: From this point on, all paragraphs in this section, not other annotated are to be considered part of Consensus Item #25b. The description in this section of changes to be made reflects the author's proposal to address this issue and related issues. It might have to be adjusted based on working group decisions.

As a result, in this document, we will have a single treatment of this issue, in Sections 9.2 through 9.12. In addition, an NFSv4.2-based extension related to attribute co-ordination will be described in Section 9.13.

The current NFSv4.0 and NFSv4.1 descriptions of this co-ordination share an unfortunate characteristic in that they are both written to give server implementations a broad latitude in implementation choices while neglecting entirely the need for 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 the uncertainty that the previous approach created for clients, while still taking proper account of the possibility of compatibility issues that a more tightly drawn specification might give rise to.

The various ways in which these kinds of issues have been dealt with are listed below together with a description of the needed changes proposed to address each issue.

*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.

The new treatment will limit use use of **MAY** to cases in which it is truly necessary, in order to give clients proper notice of cases in which server behavior cannot be determined and limit the work necessary to deal with a large array of possible behaviors.

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

  If the flexibility is necessary, **MAY** will be used. In other cases, **SHOULD** will be used with the understanding that maintaining compatibility with clients that have adapted to a particular approach to this issue is a valid reason to bypass the recommendation. However, in no case will it be implied, as it is in the current specifications, that the server **MAY** do whatever it chooses, with the client having no option but to adapt to that choice.

* There was a case, in Section 9.2, in which the term "**SHOULD**" was explicitly used intentionally, without it being made clear what the valid reasons to ignore the guidance might be, although there was a reference to servers built to support the now-withdrawn draft definition of POSIX ACLs, which are referred to in this document as "UNIX ACLs", ass described in Section 4.1. A discussion of the issues for support of for these ACLs appears in Section 9.5.

  [Author Aside]: Despite the statement, now cited in Section 9.2, that this was to accommodate implementations "POSIX" ACLs, it now appears that this was not complete. I've been given to understand that this was the result of two groups disagreeing on the appropriate mapping from ACLs, and specifying both, using the "intentional" "**SHOULD**" essentially as a **MAY**, with the text now in Section 9.2 discouraging such use as potentially confusing, not intended to be taken seriously. Since the above information might not be appropriate in a standards-track RFC, we intend to retain this as an Author Aside which the working group might consider as it discusses how to navigate our way out of this situation.

  The new approach will use the term "**RECOMMENDED**" without use of the confusing term "intentional". The valid reasons to bypass the recommendation will be clearly explained as will be the consequences of choosing to do other than what is recommended.

* There are many case in which the terms "**SHOULD**" and "**SHOULD NOT**" are used without any clear indication why they were 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 these terms 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 needs to 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) are to be determined entirely by the value of the ACL, so that use of the mode is never required for anything other than setting and interrogating the three high-order bits. See Sections 9.7 through 9.9 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]: one 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 9.7 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 one 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, Including List (Items #26a, #28a)]: A server that supports both mode and ACL attributes needs 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 requires:

*When the mode is changed, in most cases, the ACL attributes will need to be modified as described in Section 9.7.

*When the ACL is changed, the corresponding mode is determined and used to set the nine low-oder bits of the mode attribute.

This is relatively straightforward in the case of forward-slope modes, but the case of reverse-slope modes needs to be addressed as well. It is RECOMMENDED that the procedure presented in

Section 9.3 be used or another one that provides the same results.

The valid reasons to bypass this recommendation together with a alternate procedures to be used are discussed in Section 9.4.

{Consensus Needed (Item #26a)]: How other ACEs are dealt with when setting mode is described in Section 9.7. This includes ACEs with other who values, all AUDIT and ALARM ACEs, and all ACES that affect ACL inheritance.

[Previous Treatment (Item #27a)]: In this section, much depends on the method in specified Section 9.3. Many requirements refer to this section. It needs to 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.

[Consensus (Item #27a)]: In performing the co-ordinarion discussed in this section, the method used to compute the mode from the ACL has an important role. While the approach specified in Section 9.3 is **RECOMMENDED**, it needs to be noted that the alternate approaches discussed in Section 9.4 are valid in some cases. As discussed in that section, an important reason for allowing multiple ways of doing this is to accommodate server 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. While, this means that a client, having no way of determining the method the server uses may face interoperability difficulties in moving between servers which approach this matter differently, these problems need to be accepted for the time being. A more complete discussion of handling of the UNIX ACLs is to be found in Section 9.5.

[Consensus Needed, Including List (Items #27a, #28a)]: All valid methods of computing the mode from an ACL use the following procedure to derive a set of mode bits from a set of three ACL masks, with the only difference being in how the set of ACL masks is constructed. The calculated mask for for each set of bits in mode are derived from the ACL mask for owner, group, other are derived 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.

9.3. 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, one of the valid reasons to bypass the recommendation includes 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. Other reasons to bypass the recommendation are discussed in Section 9.4.

[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 #28b)]: 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 #28b)]: 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 #28b)]: 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 #28b)]: Once these ACL masks are constructed, the mode bits for, user, group, and other can be obtained as described in Section 9.2 above.

9.4. Alternatives in Computing Mode Bits

[Author Aside]: 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.

[Consensus Needed, Including List (Item #27c)]: In cases in which the mode is not computed as described in Section 9.3, one of the following analogous procedures or their equivalents, **MUST** be used.

* First, for each of the special identifiers OWNER@ and EVERYONE@, evaluate the ACL in order, considering only ALLOW and DENY ACEs for the identifier EVERYONE@ and for the identifier under consideration.

  For the special identifier GROUP@, ALLOW and DENY ACEs for GROUP@ and EVERYONE@ are to be considered, together with ALLOW ACEs for named users and groups.

  This represents the approach previously recommended to compute mode in previous specification, as modified to reflect the UNIX ACL practice of reflecting permissions for named users and groups. It does not deal properly with reverse-slope modes.

* Compute a set of ACL mask according to the procedure in Section 9.3 and then, for the mask associated with GROUP@, or in the masks for all ALLOW ACEs for named users and groups.

  This represents the approach currently recommended to compute mode in Section 9.3 as modified to reflect the UNIX ACL practice of reflecting permissions for named users and groups.

[Consensus Needed, Including List (Item #27c)]: In both cases, The results of the evaluation will be a set of NFSv4 ACL masks which can be converted to the set on nine low-order mode bits using the procedure appearing in Section 9.2 above.

[Consensus Needed, Including List (Item #27c)]: When the recommendation to use Section 9.3 is bypassed, it needs to be

understood, that the modes derived will differ from the expected values and might cause interoperability issues. This is particularly the case when clients have no way to determine that the server's behavior is other than standard.

9.5. Handling of UNIX ACLs

[Author Aside]: All paragraphs in this section are consider part of Consensus Item #56b.

Although the working group did not adopt the acls in the withdrawn POSIX draft, their continued existence in UNIX contexts has created protocol difficulties that need to be resolved. In many cases these ACLS and their associated semantics are the basis for ACL support in UNIX client APIs and in UNIX file systems supported by NFSv4

Although the semantic range of UNIX ACLs is a subset of that for NFSv4 ACLs, expecting clients to perform that mapping on their own has not worked well, leading to the following issues which will, at some point, need to be addressed:

* There is a considerable uncertainty about the proper mapping from ACLs to modes.

* The corresponding mapping from modes to ACLs is dealt with different ways by different sections of the spec.

* These individual uncertainties are compounded since it is difficult, in this environment, to ensure that these independently chosen mappings are inverses of one another, as they are intended to be.

Some possible approaches to these issues are discussed in Section 16.2,

9.6. 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.

9.7. Setting Mode and not ACL (overall)

9.7.1. Setting Mode and not ACL (vestigial)

[Author Aside]: All unannotated paragraphs are to be 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 9.11).

9.7.2. Setting Mode and not ACL (Discussion)

[Author Aside]: All unannotated paragraphs are to be 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 9.7.1, 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 9.11).

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 9.7.1 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 9.7.1, 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.

9.7.3. 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 needs to 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

[RFC7530] and [RFC8881], 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 [RFC7530] [RFC8881]. 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.

9.8. 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.

9.9. 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 9.8.

9.10. 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.

9.11. 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 need to 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 9.7, 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 9.8.

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 9.9.

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 need to 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.

9.12. 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 is to 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.

9.13. 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 [RFC8257], 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 are 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 auth flavor. However, RPC can, when suitably configured, provide this authentication on a per-connection basis.

NFSv4.1 defines a number of ways to provide appropriate authorization facilities. These will not be discussed in detail here but the following points need to 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 access a specified file by specified sets of principals.

- ALARM ACEs provide a means to draw special attention to attempts to access 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 "Auth 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 inappropriate 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 Auth Flavors

Different auth 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 other RPC Security Services

Depending on the connection type used, RPC may provide additional means of authentication. In contrast with the case of RPC auth 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 are authenticated.

* For many types of connections such as those created TCP without RPC-with-TLS and RPC-over-RDMA version 1, there is no provision for peer authentication.

  As a result use of AUTH_SYS using such connections is inherently problematic.

* Some connection types provide for the possibility of mutual peer authentication. These currently include only those established by RPC-with-TLS. However, given the value of peer authentication, there is reason to believe further means of providing such services will be defined.

14. Security of Data in Flight

14.1. Data Security Provided by Services Associated with Auth Flavors

The only auth flavor providing these facilities is RPCSEC_GSS. When this auth flavor is used, data security can be negotiated between client and server as described in Section 15. However, when data security is provided for the connection level, as described in Section 14.2, the negotiation of privacy and integrity support, provided by the auth flavor, is unnecessary,

Other auth flavors, such as AUTH_SYS and AUTH_NONE have no such data security facilities. When these auth flavors are used, the only data security is provided on a per-connection basis.

14.2. Data Security Provided for a Connection by RPC

RPC, in many case, provide data security for all transactions performed on a connection, eliminating the need for that security to be provided or negotiated by the selection of particular auth flavors, mechanisms, or auth-flavor-associated services.

15. Security Negotiation

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

Because of the availability of security-related services associated with the transport layer, the security negotiation process needs to be enhanced so that the server can indicate the connection-level services needed. Without the addition of pseudo-flavors defining connection-level services needed, use of SECINFO would be limited to selection of a particular auth flavor with connection characteristics selected by a process of trial and error.

When the SECINFO response does not indicate the connection chacteristics needed, either because it does not contain any of the pseudoflavors described below or because the only such pseudo-flavor is PFL_ANY, the client only knows the available auth flavors together with the auth-flavor-associated services and needs to use trial and error to determines the server requirments for connection-provided services, as described below:

*For each SECINFO entry the client attempts to make a connection with whatever connection characteristics that its (i.e. the client's) security policy requires. When that connection is successfully established, it can be used to access the current portion of the server, as indicated by SECINFO or SECINFO_NONAME.

  In many cases, this will be a connection without rpc-with-tls, while in other cases the client will require rpc-with-tls to be assured of encryption or of entryption with mutual peer authentication.

*If that connection cannot be established, because the server does no provide support for the requested characteristics, then the client moves on to the next auth-flavor entry. If there are no further entry, this portion ofthe server namespace cannot be accessed.

*If the connection cannot be established because it is rejected by the server because of its security policy, the client takes note and proceeds to request addutional connection-level services on the next connection request.

  For example, if the client attempts an RPC connection of the server's security policy does not allow that, the connection rejection will indicate to the client that use of rpc-with-tls is required. If the server's policy allows non-tls connection but restricts the auth-flavors that may use such connections, the requests made with those flavors may be rejected and the slient will also try to establish an rpc-with-tls connection.

  Similarly, when an rpc-tls-connection is attempted without mutual peer authentication, its rejection will resut in the client attempting to establish a connection with mutual peer authentication.

  When the client and server are unabble to estanlish a mutually accptable connection to use with a specific auth-flavor, the client moves on to the next auth-flavor in the SECINFO response.

In order for the server to better communicate its security policies to the client and limit the cases in which connections are established only to be rejected due ti inapproprriate connection

characteristics, the followin pseudo-flavor can be used in the response to SECINFO and SECINFO_NONAME requests.

- *PFL_CRYPT indicates that encryption, such as thar provided by rpc-with-tls is to be required on any connection to support access to the current portion of the server namespace.

- *PFL_CRYPTMPA indicates that encryption such as that provided by rpc-with-tls is to be required together with mutual peer authentication on any connection to support access to the current portion of the server namespace.

- *PFL_ANY indicates that, for subsequent auth-flavors, there are no connection-based restritios to be assumed by the client, allowing connections that do not use encryption to be used as well as those that do.

When any of these flavors are incuded in a SECINFO or SECINFO_NONAME response, the procEdure described above is modified as follows:

- *When scanning the resonse, if pseudo-flavor entries are encountered, the client takes note of them, keeping track of the last such entry encountered.

- *When processing a flavor entry in the response, the most recent pseudo-flavor is used in establishing the connection to be used, to avoid using connection that the server;s scurity policy will not allow to be used. The initial connection characteristics used will be a rhose that meets the requiremnts specified by the pseudo-flavor and is consistent with the client's security policy.

As an example of how these pseudoflavors can be used, consider a situation in which it is the server's security to only accept encrypted requests, whether the encryption is provide by RPCSEC_GSS or by rpc-with-tls. A response consisting of the following entries will specify this behavior so the client can connect appropriately.

- *PFL_CRYPT to indicate that the following auth-flavor entry, in this case AUTH_SYS, is to be imited to use on connection that provide encryption of requests and responses.

- *AUTH_SYS indicates that use of the auth-flavor AUTH_SYS is acceptable to access the current potiono of the server bamespace. Because of the preceeding PFL_CRYPT, the client knows that this is only accepted on encrypted connections.

  If the server supports rpc-with-tls and allows its use together with AUTH_SYS, this connection is used. Otherwise processing proceeds to use subsequent entries.

*PFL_ANY to indicate that the subsequent auth-flavor entry, in this case RPCSEC_GSS, is not limited to any particular connection type.

*RPC_SECGSS indicating the requirement for encryption provided by rpcsec_gss.

15.1. Dealing with Multiple Connections

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

Because effective security will require both an appropriate auth flavor (and possibly services provide by the auth flavor) together with appropriate connection characteristics, it is often necessary that clients and server be aware of connection characteristics:

*When multiple connections with different security-related characteristics, are used to access a server, the clients needs to ensure that each request is issued on an appropriate connection.

*Similarly, in such situations, the server needs to be aware of the security-related characteristics for the connection on which each request is received, in order to enforce its security policy.

Depending on how the client and server implementations are structured, implementations may have to be changed to accomplish the above.

In the case of NFSv4.1 and above, the protocol requires that requests associated with a given session only be issued on connections bound to that session and accepted by the server only when that binding is present. This makes it likely that clients or servers will be able to correctly associate requests with the appropriate connections although additional work might be necessary to enable them to determine, for any given connection, its security characteristics.

In the case of NFSv4.0, no such binding is present in the protocol so that, depending on existing implementations' layering, channel binding functionality might have to be added.

16. Future Security Needs

This section deals with weaknesses in the security of the existing protocol which might be dealt with in future minor version or in

extensions to existing minor versions. These weaknesses are divided into two groups:

- Potential improvements to NFSv4 security are discussed in Section 16.1.

- Potential extensions to NFSv4 to make the client aware of server behavioral choices and possibly negotiate mutually acceptable choices are discussed in Section 16.2.

16.1. Desirable Additional Security Facilities

[Author Aside]: All unannotated paragraphs in this section are to be 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 RDMA-based transport that provides for encryption, which might be provided in number of ways.

16.2. Extensions to deal with Authorization-related Server Behavioral Differences

[Author Aside]: All unannotated paragraphs in this section are to be considered part of Consensus Item #56c.

This section discusses possible new attributes or related extensions to help deal with the issues described in Section 3.4. As discussed in Section 1.3.2, there are a large set optional behaviors allowed to the server with no means for the client to determine the option chosen by the server. While it is possible that this set of options will be cut back if it is found that no servers use them, there will need to be attributes defined for each case in which the eventual RFC continues to allow muiltple server behaviors.

It is possible that a low-level approach would be used if the working group is unable to organize the existing behavioral options into optional features meaningful to the client. This approach would provide a large set of related fs-scoped attributes such as the following:

* A new attribute would contain an element for each optional ACE mask bit. The element would indicate whether the mask bit is supported, and, if it is not, how the authorizations normally controlled by that mask bit are determined (e.g. file ownership, a different mask bit).

  In cases such as ACE4_APPEND_DATA, where it might not be clear exactly which WRITEs fit under the category of appending operations, there need to be an indication of how this decision is to be made, given uncertainty regarding WRITEs beyond EOF and whether there are WRITEs understood as both appending and modifying existing data.

  Although, according to current specifications, all such mask bits are optional, but it is likely that the three masks bits, corresponding to the three masks bits in the mode (read, write, search/execute) can be made mandatory.

* A new attribute could contain a mask of ACE flags supported.

* In each case in which "**SHOULD**" is used in current specifications with an interoperability effect similar to the one implied by use of "**MAY**", there needs to be one bit within the attribute to allow the client to determine when the formally disfavored behavior is provided by the server.

  This would be the case even if the working group were to decide that the granting of this flexibility had been a mistake. As long as the server is allowed the variant behavior, there needs to be a way to provide notice to the client, even if is the case that the only valid reasons to bypass the recommendation is to allow time to adjust clients and severs to the necessary change.

* Given that it is allowed for servers to choose different algorithms to arrive at a mode corresponding to an ACL, there needs to be an attribute to enable the client to find out which algorithm is used.

* The changes that need to be made to ACLs when the mode is set are a further source of server behavioral variation, that an fs-scoped attribute could address While existing specifications indicate that is acceptable to simply drop the ACL and make subsequent authorization decisions solely on the basis of the mode, there are many statements (non-normative) that various

aspects of this are undesirable, suggesting that various sorts of alternative behavior are allowed.

One issue concerns the handling of ACEs for named users and groups and for special who values. It is certainly allowed, as stated above, to simply

Regarding AUDIT and ALARM ACEs, the following, appearing in Section 9.7.1 seems to disallow simply dropping the ACL.

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

Even if the working group could clarify this issue, the current specifications could have given rise to existing conflicting implementations requiring one bit (or one bit for each ACE type) indicate whether such ACEs are maintained or dropped.

For mask bits outside the three mode bits (read, write, share/ execute), such ACEs can be deleted or not (Dropping the ACL would get rid of them, while Section 9.7.1 says such ALLOW ACEs "**MAY** be deleted. The corresponding case of DENY ACEs is not discussed. In any case, these behavioral variants need to be addressed

There are also a number of possible variant behaviors related to ACL inheritance. Existing specifications indicate it is "desirable" to leave inherit-only ACEs in place but leaves this as a server choice. The parallel case of converting inheritable ACEs to inherit-only ACEs is not mentioned but this is probably an oversight. In any case, at least two attribute bits are required to specify the handling of inheritance in the event of a SETATTR of mode.

This set of attribute values would provide the client an indication of where the server might avail itself of the flexibility currently provided for in the specification of authorization semantics. A principal use would be to allow clients to determine whether it possible to construct an ACL to authorize any particular set of activities according to the NFSv4 ACL model, and if it is, how that ACL is to be constructed. This could provide a necessary complement to the use of ACCESS, which avoids the need for the client to interpret ACLs to determine authorization, which the looseness of the current specifications make impossible.

It is possible that the working group might choose, to organize a set of more structured attribute choices reflect the underlying conflict between the NFSv4 ACL model (windows-oriented) and the subset more attuned to the Unix ACL Model. If there are servers who provide extensions to the Unix model that are compatible with the

NFSv4 model, they can be defined as optional extensions to the Unix model.

This reorganization, which the working group might be able to do in the corresponding RFC would provide interoperability for clients using and servers providing support for Unix ACLs. It would also provide interoperability for clients using the full power of NFSv4 ACLs and servers providing the corresponding server-side implementation. Interoperability between these two classes would be hit-or-miss, as it is now.

As most servers capable of supporting the full NFSv4 ACL model, could provide the subset necessary to support clients that expect Unix ACL support, the working group will need to provide means of allowing the support level to be negotiated. Two possibilities are:

*Defining a new attribute unix_acl whose definition is a Unix-oriented subset of sacl.

*Allowing the choice of Unix or Nfsv4 ACLs as part of the EXCHANGE_ID (adding new flags) or using a new op devoted to security related negotiation.

It turns out that multiple elements are probably required. While providing two separate attributes for each of the two ACL semantic models provides for client choice, allowing two read-write attributes results in considerable additional complexity. Allowing negotiated restrictions on the values to be set using a single writable ACL attribute accomplishes the main goal but there are problems in that unix-oriented clients will encounter non-unix ACLs stored by others. For the most part, clients will not access these values, relying on ACCESS. However, there is a need to use such non-Unix ACLs when copying files. As result, we might have the following set of extensions:

*Providing for client-negotiated semantic restrictions on values accepted by and returned by the sacl and dacl attributes. To enable unix-oriented clients their normal ACL semantic model, sacl could be restricted to the equivalent of Unix ACLs, with dacl only accepting and returning empty sets of ACE's.

*Providing, in addition, new full_sacl and full_dacl attributes, not subject to the restrictions above. The server, in setting such attributes would not be bound to respect them full (e.g. no append-only files) but would make these unsupported features available for use by servers that do not support them, in the fashion that is done now in the case of ACE4_SYNCHRONIZE.

There are also a set of behavioral differences that derive from modifications to authorization semantics. These special semantics,

which now apply to use of both ACLs and modes, were originally necessary in NFSv2 and NFSv3 to deal with the need to emulate local file semantics using stateless protocols. These semantics were carried over into NFSv4 as allowed behaviors even though they were no longer necessary and can interfere with correct authorization semantics when non-writeable files need to be supported.

An attribute could be provided to allow the client to determine whether these alternate semantics are enforced by the server, and, if so, for which operations. Given that some clients might have security reasons to be wary of such non-POSIX authorization semantics, it would be helpful to allow adherence to full POSIX authorization semantics to be negotiated using EXCHANGE_ID or other ops devoted to security-related negotiation.

17. Security Considerations

17.1. Changes in Security Considerations

Beyond the needed inclusion of a threat analysis as Section 17.4 and the fact that all minor versions are dealt with together, the Security Considerations in this section differ substantially from those in [RFC7530] and [RFC8881]. These differences derive from a number of substantive changes in the approach to NFSv4 security presented in [RFC7530] and [RFC8881] and that appearing in this document.

These changes were made in order to improve the security of the NFSv4 protocols because it had been concluded that the previous treatment of these matters was in error, leading to a situation in which NFSv4's security goals were not met. As a result, this document supersedes the treatment of security in earlier documents, now viewed as incorrect. 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 provided by RPC on a per-connection basis can be used to deal with security issues, as described in Section 17.1.2. These include the use encryption on a per-connection basis, and the use of peer mutual authentication, to mitigate the security problems that come with the use of AUTH_SYS.

* The handling of identities with superuser privileges is no longer part of NFSv4 semantics, even though many platforms on which NFSv4 servers are implemented continue to depend, for local operation, on the existence of such identities.

NFSv4 servers **SHOULD NOT** provide for such unrestricted access since doing so would provide a means by which an escalation-of-privilege on a client could be used to compromise a server to which it was connected, affecting all clients of that server.

In connection with the use of "**SHOULD NOT**" above, and similar uses elsewhere, it is to be understood that valid reasons to do other than recommended are limited to the difficulty of promptly changing existing server implementations and the need to accommodate clients that have become dependent upon the existing handling. Further, those maintaining or using such implementations need to be aware of the security consequences of such use as well as the fact that clients who become aware of this characteristic may not be inclined to store their data on such a system.

*The appropriate handling of ACL-based authorization and necessary interactions between ACLs and modes is now specified in this standards-track document rather it being assumed that the behavior of server implementations needs to be accepted and deferred to.

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 and placed the burden of providing for secure use on those deploying such systems with very limited guidance as to how such secure use could be provided.

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 considerations 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. Connection-oriented Security Facilities

Such RPC facilities as RPC-with-TLS provide important ways of providing better security for all the NFSv4 minor versions.

In particular:

* The presence of encryption by default deals with security issues regarding data-in-flight, whether RPCSEC_GSS or AUTH_SYS is used for client principal identification.

* Peer authentication provided by the server eliminates the possibility of a server-impersonation attack, even when AUTH_SYS or AUTH_NONE is used to issue requests

* When mutual authentication is part of connection establishment, there is a possibility, where an appropriate trust relationship exists, of treating the uids and gids presented in AUTH_SYS requests, as effectively authenticated, based on the authentication of the client peer.

17.1.3. Necessary Security Changes

[Consensus Needed (Items #36a, #37a)]: For a variety of reasons, there are many cases in which a change to the security approach has been adopted but for which provisions have been made in order to give implementers time to adapt to the new approach. In such cases the words "**SHOULD**", "**SHOULD NOT**", and "**RECOMMENDED**" are used to introduce the new approach while use of the previous approach is

allowed on a temporary basis, 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 RPC on a per-connection basis, allowance has been made for implementations for which these new facilities are not available and for which, based on previous standards-track guidance, AUTH_SYS was 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 this document now clearly defines the recommended behavior in dealing with these issues, allowance has been made to provide time 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 connection-oriented security facilities such as RPC-with-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 connection-oriented support from RPC.

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 client 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 networks 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 corporate 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.

In corporate networks, as opposed to the Internet, there is good reason to be less concerned about denial-of-service attacks, since there is no tangible benefit to attackers inside the organization, and the anonymity that makes such attacks attractive to outside attackers will not be present.

The above 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.

17.2.3. Insecure Environments

As noted in Section 17.2.2, NFSv4 is often used in environments of much smaller scope than the internet, with the assumption often being made, that the prevention of NFSv4 access from outside the organization makes the attention to security recommended by this document unnecessary, the possibility of insider attacks being explicitly or implicitly disregarded.

As a result, there will be implementations that do not conform to these recommendations, many of which because the implementations were based on the RFCs [RFC3530], [RFC7530], [RFC5661], or [RFC8881]. In addition to these cases in which the disregard of the recommendations is considered valid because implementors relied on existing normative guidance, there will be other cases in which implementors choose to ignore these recommendations,

Despite the original focus of [RFC2119] on interoperability, many such implementations will interoperate, albeit without effective security, whether the reasons that the recommendations are not adhered to are considered valid or not.

When such insecure use is mentioned in this Security Considerations section it will only be in explaining the need for the recommendations, by explaining the likely consequences of not following them. The threat analysis, in Section 17.4 and included subsections, will not consider such insecure use and will concern itself with situation in which these recommendations are followed.

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 #41b.

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 on a per-connection basis, 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.

When these recommendations are not followed, data, including data for which disclosure is a severe [problem is exposed to unwanted disclosure or modification in flight. Depending on the server to be aware of the need for confidentiality or integrity, as expected by previous specifications, has not proved workable, making encryption by default as provided uniformly by RPC (e.g. through RPC-with-TLS) necessary.

17.3.2. Recommendations Regarding Client Peer Authentication

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #41c.

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 (i.e. AUTH_NONE Or AUTH_SYS) are allowed to be executed.

When these recommendations are not followed, it is possible for connections to be established between servers and client peers that have not been authenticated with the following consequences:

* The server will be in the position of executing requests where the identity used in the authorization of operations is not authenticated, including cases in which the identification has been fabricated by an attacker.

* When no identification of a specific user is needed or present (i.e AUTH_NO is used) there is no way of verifying that the request was issued by the appropriate client peer.

When the recommendations are followed, use of AUTH_SYS can be valid means of user authentication, so long as due attention is paid to the discussion in Section 17.4.6.1. Despite this fact, the description of AUTH_SYS as an "**OPTIONAL** means of authentication"is

no longer appropriate since choosing to use it requires heightened attention to security as discussed later in this document.

17.3.3. Recommendations Regarding Superuser Semantics

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #52b.

It is **RECOMMENDED** that servers adhere to the ACL semantics defined in this document and avoid granting to any remote user, however authenticated, unrestricted access capable of authorizing access where the file/directory ACL would deny it.

Servers are free to conform to this recommendation either by implementing authorization semantics without provisions for superusers or by mapping authenticated users that would have superuser privileges to users with with more limited privileges (e.g. "nobody").

It needs to b e understood that the second of these choices is preferable when there are NFsv4-accessible files owned by a special users (e.g. root) whose compromise might be taken advantage of by attackers to enable permanent unauthorized access to a server.

17.3.4. Issues Regarding Valid Reasons to Bypass Recommendations

[Author Aside]: All unannotated paragraphs within this section are considered part of Consensus Item #41d.

Clearly, the maturity and compatibility issues mentioned in Section 17.1.4 are valid reasons to bypass the proposed recommendations requiring pervasive use of encryption, 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 conection-based encryption 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 recommendations requiring pervasive use of connection-based encryption, it needs to be understood that the connection-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. Threat Analysis

17.4.1. Threat Analysis Scope

Because of the changes that have been made in NFSv4 security, it needs to be made clear that the primary goal of this threat analysis is to explore the threats that would be faced by implementations that follow the recommendations in this document.

When the possibility is raised of implementations that do not conform to these recommendations, the intention is to explain why these recommendations were made rather that to expand the the scope of the threat analysis to include implementations that bypass/ignore the recommendations.

The typical audience for threat analyses is client and server implementers, to enable implementations to be developed that are resistant to possible threats. While much of the material in Section 17.4 is of that form, it also contains material that relates to threats whose success depends primarily on the ways in which the implementation is deployed, such as the threats discussed in Sections 17.4.2, 17.4.4 and 17.4.3. While it is not anticipated that those deploying implementations will be aware of the detail of this threat analysis, it is expected that implementors could use this material to properly set expectations and provide guidance helpful to making deployments secure.

17.4.2. Threats based on Credential Compromise

In the past, it had been assumed that a user-selected password could serve as a credential, the knowledge of which was adequate to authenticate users and provide a basis for authorization.

That assumption is no longer valid for a number of reasons:

* The inability or unwillingness of users to remember multiple passwords has meant that the single password they will remember controls access to large set of resources, increasing the value of this knowledge to attackers and the effort that will be expended to obtain it.

  In addition, the common use of a single password for applying to all of a user's data has resulted in a situation in which the client is aware of user passwords (since they are used for client

login) that apply to data on many servers. As will be seen later, this has the effect of changing the considerations appropriate to comparing the security of AUTH_SYS and RPCSEC_GSS.

*CPU developments have made exhaustive search possible for larger classes of passwords.

*The success of "phishing" attacks taking advantage of user gullibility provides an additional path to credential compromise which need to be addressed in the near-term by those deploying NFSv4, and will eventually need work in the security infrastructure on which NFSv4 is built.

In the near term, there are a number of steps, listed below that those deploying NFSv4 servers can take to mitigate these weaknesses. These steps are outside the scope of the NFSv4 protocols and implementors only role with regard to them is to make it clear that these weaknesses exist and generally require mitigation.

*Limitations on password choice to eliminate weak passwords.

*Requirements to change passwords periodically.

*User education about "phishing" attacks including ways to report them and effective ways of replacing a compromised password.

From a longer-term perspective, it appears that password-based credentials need to be either replaced or supplemented by some form of multi-factor authentication. Since NFSv4's approach to security relies on RPC, that work would most probably be done within the RPC layer, limiting the work that implementations and the NFSv4 protocols would have to do to adapt to these changes once they are available. While the precise form of these changes is not predictable, the following points need to be kept in mind.

*[Verification Needed (Item #53a)]: For those using RPCSEC_GSS authentication of principals, it appears that RPCSEC_GSS interface is flexible enough that the addition of a second credential element, in the form of a one-time code could be added.

[Elaboration/Verification Needed (Item #53a)]: Enhancement of Kerberos is one possibility to provide multi-factor authentication. However, work on this is not far enough along to enable deployment to be discussed now.

If this approach were taken, rogue servers would still have access to user passwords but their value would be reduced since the second credential element would have a very limited lifetime.

* For those using AUTH_SYS to identify principals, the client operating system's authentication of user at login would need to be enhanced to use multi-factor authentication.

  If this were done, the client would retain responsibility for credential verification with the server needing to trust the client, as discussed in Section 17.4.6.1.

  Although there is need for protocol standardization to enable this approach to be commonly used, it is not likely to be widely used until some operating system adopts it for user login.

* One important variant of AUTH_SYS use concerns clients used by a single user, when, as recommended, client-peer authentication is in effect For such clients, it is possible for the authentication of that specific client peer to effectively become the second factor, in a multi-factor authentication scheme.

  Despite the fact that the the RPC-with-TLS specification [RFC9289]) does not allow TLS to used for user authentication, this arrangement in which the user identity is inferred from the peer authentication, could be used to negate the effects of credential compromise since an attacker would need both the user password, and the physical client to gain access.

17.4.3. Threats Based on Rouge Clients

When client peers are not authenticated, it is possible to a node on the network to pretend to be a client. In the past, in which servers only checked the from-IP address for correctness, address spoofing would allow unauthenticated request to be executed, allowing confidential data to be read or modified.

Now that such use of AUTH_SYS is recommended against, this cannot happen. The recommended practice is to always authenticate client peers making this sort of imposture easily detectable by the server.

Despite this protection, it is possible that an attacker, through a client vulnerability unrelated to NFSv4, or the installation of malware, could effectively control the client peer and act as imposter client would, effectively undercutting the authentication of the client. This possibility makes it necessary, as discussed in Section 17.4.6.1 that those deploying NFSv4 clients using AUTH_SYS takes steps to limit the set of user identifications accepted by a server and to limit the ability of rogue code running on the server to present itself as a client entitled to use AUTH_SYS.

17.4.4. Threats Based on Rouge Servers

When server peers are not authenticated, it is possible for a node on the network to act as if it were an NFSv4 server, with the ability to save data sent to it and use it or pass it to other, rather than saving it in the file system, as it needs to do..

When current recommendations are adhered to, this is be prevented as follows:

- When RPCSEC_GSS is used, the mutual authentication of the server and client principal provides assurance the server is not an imposter.

- When AUTH_SYS or AUTH_NONE is used, the mutual authentication of client and server peers provides assurance the server is not an imposter.

Despite this protection, it is possible that an attacker, through a operating system vulnerability unrelated to NFSv4, or the installation of malware, could effectively control the server peer and act as an imposter server would, effectively undercutting the authentication of the server.

The above possibility makes it necessary, that those deploying NFSv4 servers take the following steps, particularly in cases in cases in which the server has access to user credentials, including, but not limited to, cases in which AUTH_SYS is supported

When an NFSv4 is implemented as part of a general-purpose operating system, as it often is, steps need to be taken to limit the ability of attackers to take advantage of operating system vulnerabilities that might allow the attacker to obtain privileged access and subvert the servers operation, turning it, effectively, into a rogue server.

Such steps include controls on the software installed on the machine acting as the server, and limitation of the network access to potentially dangerous sites.

17.4.5. Data Security Threats

When file data is transferred in the clear, it is exposed to unwanted exposure. As a result, this document recommends that encryption always be used to transfer NFSv4 requests and responses.

That encryption, whether done on encrypted connections, or on a per-request basis, using RPCSEC_GSS security services, provides the

necessary confidentiality. In addition, it contributes to security in other ways as well:

- The ability of an attacker to plan and execute attacks is enhanced by the monitoring of client-server traffic, even if none of the data intercepted is actually confidentiality.

  An attacker can deduce which users are allowed to read or write a specific file by examining the results of OPEN and ACCESS operations allowing later attacks to impersonate users with the appropriate access.

- All the methods of encryption used with NFS4 provide a checksum, to enable the detection of unwanted modifications to data being read or written.

17.4.6. Authentication-based threats

17.4.6.1. Attacks based on the use of AUTH_SYS

Servers, when they allow access using AUTH_SYS, to a specific client machines using AUTH_SYS are responsible for ensuring that the principal identifications presented to the server can be relied upon.

The existence of client-peer authentication as recommended in Section 17.1.5 means that imposter servers can be detected and not allowed to use AUTH_SYS. However there are an additional number of issues that need to be addressed to adequately protect against use of AUTH_SYS enabling attacks:

- The server accepting requests using AUTH_SYS needs to determine that the authenticated client-peer can be trusted to properly authenticate the principals that it identifies in requests.

  The specific standards for trustworthiness are up to the server but they need to take account of the controls in place to prevent malware from pretending to be a client and thus taking advantage of the fact that the request is from the expected client machine.

  This server **MUST NOT** accept AUTH_SYS requests from unknown clients or from unauthenticated clients.

- [Elaboration Needed (Item #54a)]: The client verification procedure needs to take steps to prevent code on a compromised client to presenting itself as the successor to a legitimate client, taking advantage of the fact that the machine is the same.

* Given the inherent vulnerabilities of client operating systems, it is desirable, to limit the set of users whose identification will be accepted. The elimination of particular users such as "root' is one long-standing approach to the issue but it probably isn't sufficient in most environments. More helpful would be the ability to exclude multiple sensitive users or group of users or to limit the user identifications accepted to a user group or a single user.

Another important that issue that arises when AUTH_SYS is used concerns the storage of credentials on the clients. While it is theoretically possible for these not to be of use elsewhere, the reluctance/inability of users to remember multiple passwords means that these credentials will be used by many clients and will need to be updated as users are added or deleted or when passwords are changed. The propagation of these credentials and their storage on clients could be the basis for attacks if appropriate step are not taken to secure this data.

While it is helpful to store a cryptographic hash of the password rather than the password itself, this does not dispose of the issue, since possession of the hash would greatly simplify an exhaustive search for the password, since the attacker could limit login attempts to guessed password whose hash value matched the value obtained from the files on the client.

Although it is true that making clients responsible for authentication of user identities undercuts much of the original motivation for making RPCSEC_GSS **REQUIRED** to implement, it needs to be understood that the situation today is different from that when this decision was made.

* It has been recommended that servers not allow unauthenticated clients to issue requests using AUTH_SYS.

* The identification of a request as issued by the user with uid zero, no longer provides access without file access authorization.

* Given that users are unaware of where their files are located and it is desirable that they are able to remain unaware of this, it is natural that they use the same password to authenticate themselves for local resource use as for use of files located on NFSv4 servers.

Support for AUTH_SYS in NFSv4 was included for a number of reasons which still hold true today, despite the fact that the original mistake, to make no reference to the security consequences of doing so, is now being corrected. Such provision is necessary for the

following reasons, that go beyond the need to temporarily accommodate implementations following the older specifications, for a number of reasons:

- When considered, as NFS was to intended to be, as consistent with local access as possible, AUTH_SYS was the natural way of providing authentication, just as it had been done for local files.

  While use of AUTH_SYS exposes user passwords to the client operating system, the fact that user are unable or unwilling to use different passwords for different files in a multi-server namespace means this issue will be present even when AUTH_SYS is not used.

- [Elaboration Needed (Item #55a)]: In many important environments including cloud environments, important implementation constraints has made use of Kerberos impractical.

  [Verification Needed (Item #55a)]: In such environments, client credentials are maintained by the cloud customer while the cloud provider manages network access.

17.4.6.2. Attacks on Name/Userid Mapping Facilities

NFSv4 provides for the identification users and groups in two ways (i.e. by means of strings of the form name@domain or strings containing numeric uid/gid values) while file systems used on NFSv4 servers typically use 32-bit uids and gids.

As a result, NFSv4 server implementations are required to have some means of translating between the name@domain form and the numeric form used internally. While the specifics of this translation are not specified as part of the NFSv4 protocols, is required for server implementations to work, and, if it not done securely and attackers have the ability to interfere with this translation, it gives them the ability to interfere with authorization as follows:

- When authentication occurs using user names, as occurs when RPCSEC_GSS, a mistranslation might allow the numeric value used in authorization to allow access to a file the authenticated user would not be allowed to access.

- When any authentication occurs on the client and the uid is presented to the server using AUTH_SYS a mistranslation to the string form could result in confusion and uncertainty about the users allowed to access the file.

17.4.7. Disruption and Denial-of-Service Attacks

17.4.7.1. Attacks Based on the Disruption of Client-Server Shared State

When data is known to both the client and server, a rogue client can interfere with the correct interaction between client and server, by modifying that shared data, including locking state and session information.

For this reason, it is recommended that client-peer authentication be in effect, because, it it were not, a different client could could easily modify data that the current client depend on, disrupting ones interaction with the server.

It is still possible, if one's client is somehow compromised, as described in Section 17.4.3, for various forms of mischief to occur:

- Locks required for effective mutual exclusion can be released, causing application failures.

- Mandatory share locks can be obtained preventing those with valid access from opening file that they are supposed to have access to.

- Session slot sequence numbers may be rendered invalid if requests are issued on existing sessions. As a result, the client that issued a request would receive unexpected sequence errors.

17.4.7.2. Attacks Based on Forcing the Misuse of Server Resources

It is is also possible for attacks to be mounted, in the absence of the ability to obtain or modify confidential data, with the sole goal of the attack being to make spurious requests, with no expectation that the request will be authorized but with the goal of causing resources that would otherwise be used to service valid requests to be unavailable due to the burden of dealing with numerous invalid requests.

The design of the NFSv4 protocols requires that clients establishing new connections make initial requests which establishes a shared context referred to by subsequent requests which might request substantive actions (e.g. client and session ids). This structure helps mitigate the effect of such denial-of-service attacks as described below.

- The server can limit the resources devoted to connections not yet fully identified without unduly restricted connections which have identified themselves.

*The recommendation that client peers authenticate themselves, allows unknown clients to be dispensed with at an early stage negating their ability to make requests which could require file system action to obtain information needed to make authorization decisions (e.g. ACLs or other authorization-related) file attributes.

18. IANA Considerations

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

Because of the shift from implementing security-related services only in connection with RPCSEC_GSS to one in which connection-oriented security has a prominent role, a number if new values need to be assigned.

These include new authstat values to guide selection of a connection types 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 #32d.

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 connection type used. 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 #32e.

The new pseudo-flavors described in this section are to be made available to allow their return as part of the response to the SECINFO and SECINFO_NONAME operations. How these operations are to

used to negotiate the use of appropriate auth flavors and associated security-relevant connection characteristics is discussed in Section 15.

The following pseodo-flavors are to be defined:

* PFL_CRYPT is returned to indicate that subsequent secinfo entries are to considered limited to use on connections for which transport-level encryption is provided.

* PFL_CRYPTMPA is returned to indicate that subsequent secinfo entries are to be considered limited to use on connections on which mutual peer authentication has been provided at connection setup.

* PFL_ANY is returned to indicate that subsequent secinfo entries are not to be considered limited to any particular type of connection.

19. References

19.1. Normative References

[RFC2119] 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>.

[RFC8174] 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>.

[RFC7530] 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>.

[RFC8178] Noveck, D., "Rules for NFSv4 Extensions and Minor Versions", RFC 8178, DOI 10.17487/RFC8178, July 2017, <https://www.rfc-editor.org/info/rfc8178>.

[RFC8881] 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>.

[RFC7862] 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>.

[RFC9289]
Myklebust, T. and C. Lever, Ed., "Towards Remote Procedure Call Encryption by Default", RFC 9289, DOI 10.17487/RFC9289, September 2022, <https://www.rfc-editor.org/info/rfc9289>.

[nist-209] NIST, "SP 800-209 Security Guidelines for Storage Infrastructure".

19.2. Informative References

[RFC8257] 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>.

[RFC3010] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "NFS version 4 Protocol", RFC 3010, DOI 10.17487/RFC3010, December 2000, <https://www.rfc-editor.org/info/rfc3010>.

[RFC3530] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, DOI 10.17487/RFC3530, April 2003, <https://www.rfc-editor.org/info/rfc3530>.

[RFC5661] Shepler, S., Ed., Eisler, M., Ed., and D. Noveck, Ed., "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661, DOI 10.17487/RFC5661, January 2010, <https://www.rfc-editor.org/info/rfc5661>.

[RFC1813] Callaghan, B., Pawlowski, B., and P. Staubach, "NFS Version 3 Protocol Specification", RFC 1813, DOI 10.17487/RFC1813, June 1995, <https://www.rfc-editor.org/info/rfc1813>.

[I-D.ietf-nfsv4-internationalization]
Noveck, D., "Internationalization for the NFSv4 Protocols", Work in Progress, Internet-Draft, draft-ietf-nfsv4-internationalization-06, 20 May 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-nfsv4-internationalization-06>.

Appendix A. Changes to be Made

These appendices summarize the significant changes between the treatment of security in previous minor version specification

documents (i.e. RFCs 3630, 7530, 5661 and 8881) and the new treatment that is intended to apply to NFSv4 as a whole.

This is expected to be helpful to implementers familiar with previous specifications but also has an important role in guiding the search for working group consensus as to these changes and in identifying potential compatibility issues.

A.1. Fundamental Security 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 RPC on a per-connection basis rather than as a service ancillary to a specific auth flavor.

This motivated a major reorganization of the treatment of security together with a needed emphasis on the security of data in flight, described in detail in Section 14. In addition, the security negotiation framework for NFSv4 has been enhanced to support the combined negotiation of authentication-related services and connection characteristics, as described in Section 15.

Despite these major changes, there are not expected to be troublesome compatibility issues between peers supporting provision of security services on a per-connection basis and those without such support. Clients who are adapted to the new security facilities can use them when the server supports them, but will sometimes choose to use the older facilities when interacting with older servers. Servers who support the new facilities have the option of rejecting client connections from older clients but are encouraged to adopt policies rejecting such connections. Over time, it is expected that use of the older facilities will become less common as the newer facilities become more commonly implemented and the importance of security becomes more generally recognized

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 auth flavors and clearly distinguishes those flavors providing authentication from those providing identification only or neither identification nor authentication. For details, see Section 13.

As part of the necessary shift from a narrative focused on justifying security choices already made to evaluating the adequacy

of existing security facilities, there has been a need to discuss NFSv4 security gaps, which might be addressed later. See Section 16.1 for a discussion of possible security-related extensions.

There are twelve consensus items that are involved in effecting this class of changes. Although it is generally understood that the working group needs to take advantage of RPC-with-TLS, change how use of AUTH_SYS is presented, and include a credible threat analysis, there needs to be further discussion of these items as working group members are likely to disagree about the relative importance of:

- *accommodating implementations based on previous specifications.

- *providing a description of the expected path toward secure implementations

- *maintaining and improving implementation performance

For each of the following items, there may need to be extensive discussion to arrive at a consensus text.

- *Consensus Item #32 concerns extension of the process of SECINFO negotiation to accommodate rpc-with-tls, in addition to the current approach which directs only the choice of a suitable auth flavor.

  In the current document, pseudoflavors are added to allow the specification of connection characteristics, lthough it would be possible to avoid use and select connection characteristics by a process of trial and error.

- *Consensus Item #52 concerns the handlinng of superuser semantics within NFSv4.

  There is a long tradition, within NFS, of treating superusr semantic, involving the elimination of file authorization checks, as part of protocol semantics. The security problems that this creates have been dealt with via "root-squashing", outside the protocol specification. Given that a threat analysis would essentially make such suppression of superuser semantics unavoidable, the current document takes such semantics outside of the NFSv4 protocol.

- *Consensus Item #36 concerns how to deal with the troublesome legacy of use of AUTH_SYS without client authentication and the common transmission of requests and responses without encryption.

The specification now makes it clear that this has the potential to cause harm, while needing to deal ith the fact that it is a long time before this could be forbidden or recommmended against without allowances being made for current implementations. The ways that this issue needs to be dealt with require further discussion.

*Consensus Item #38 concerns temporary accommodations to deal with previous implementations which are unsafe.

This is a consequence of existing implemenations based on current RFCs. The wording will require extensive discussion.

*Consensus Item #39 defines the new safer approach to AUTH_SYS.

There needs to be discussion about how this safety should be characterized, given that you are trusting the client OS.

*Consensus Item #40 concerns the scope for the threat analysis in the Security Considerations section.

Will need to reach a consensus on this as there appears no way we can make allowances in which physical isolation makes a significant difference, since this ssentially ignores insider threats.

*Consensus Item #41 discusses the 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.

Will have to address concerns about performance effects.

*Consensus Item #47 concerns dubious paragraph regarding AUTH_NONE in SECINFO response which should be deleted if there are no compatibility issues that make that impossible.

Will have to ascertain whether this is in fact supported by any server or relied on by any client.

*Consensus Item #35 discusses possible work on future security needs.

Need to discuss whether there is too much or too little forward-looking material.

*Consensus Item #53 concerns the possible use of multi-factor authentication.

*Consensus Item #54 discusses prevention of code on a compromised client from hijacking the client machine's peer authentication.

This relates to getting clarity about the safety of the client peer authentication available in rpc-with-tls.

*Consensus Item #55 discusses issues related to potential use of Kerberos in cloud environments.

We will have to be sure we have a strog case here, since ther may be a feeling within the IESG that AUTH_SYS is per se unacceprable.

A.2. Need for Clarifying 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 that process, many issues involving unclear, inconsistent, or otherwise inappropriate text were uncovered and now need to be dealt with.

One important reason that such extensive changes are now necessary derives from a disagreement among working group partucipants as to the purpose of ACL support with one group of participants requiring new functionallity matching that of Windows ACLs with another major group uninterested in such features and unwilling to devote the effort involved in providing server-side support for them.

Within the NFSv4 architecture, such situations are dealt by defining one or more optional features, allowing different servers to provide different levels of support, with the client able to determine whether a selected server has the desired level of support.

Unfortunately, for reasons that remain unclear, this approach was not followed in writing RFCs 3530, 5661, 7530 or 8881. Instead, the full definition of the ACL was defined as an OPTIONAL attribute with no clear definition of useful support subsets or way of testing the support level. Essentially each ACE mask bit was made its own optional feature. To further complicate things the keyword "**SHOULD** was used raising the possibility that the support might be other than described. This made it impossible for clients to determine the level of ACL suppport provided, or to elect to use a server based on the level of support provided.

While the author believes that changes in this situation 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.

The primary source of these not-clearly specified elements is the disparate goals of many participants in the specification process:

- For many participants, the ACL definition needed to include a large part of Windows-oriented semantics, including elements clearly distinct from the Unix semantics supported by earlier versions of NFS.

- For other participants, such semantic extensions were of little interest.

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 the need for re-organization of subjects relating to authorization, 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.

The subjects related to authorization needed to be reorganized because of the following complexities:

- There are three kinds of authorization to deal with.

- The most important kind of authorization, file access authorization, is controlled by mode attributes and ACL-related attributes, creating multiple authorization models and the need to coordinate them.

- ACLs, besides their prominent role with regard to file access authorization, provide control of alarm and audit facilities.

As a result, these matters are dealt with in the eight top-level sections from 5 through 12.

- Section 5, a new top-level section describes the the structure of ACLs with their uses described later in sections 7 through 9 and in Section 12,

- In Section 6, there is a discussion of authorization in general. This includes a discussion of user-based file authentication, NFSv4's approach to mandatory access control, and a discussion of locking state authorization, each of which is described in later top-level sections.

  File access authorization is described primarily in Section 7. Because this authorization can be controlled by either the mode

attribute or various ACL-related attributes, there is an introduction to these authorization- determining attributes in Section 7.1.

Because file access authorization can be controlled by the mode attribute and by ACL-related attributes, there is an introduction to this subject in Section 7.2 followed by discussion of the two authorization models in Sections 7.3.1 and 7.4.

Issues related to the existence of two separate file access are dealt with in Sections 8, dealing with the similarities between these two, and 9 dealing with their co-existence and potential conflict.

*Sections 10 through 12 deal with, labelled NFS authorization, state modification authorization, and the uses of ACL outside authorization.

Beyond the reorganization described above, there are numerous matters to be dealt with, reflecting the previous inability to clearly specify two sets of authorization behaviors and their potential interactions.

*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 interoperability 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 existing RFCs and some 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.

A.3. Addressing the Need for Clarifying Changes

Unlike the issues discussed in Appendix A.1, for which the path to make the necessary changes is laid out in that same section, the changes discussed in Appendix A.2, are discussed here, because resolving them is a more involved matter.

The difficulty derives principally from the fact that there are existing clients and server with very different approaches to these matters, making it difficult or impossible to reach a consensus that will make some existing clients non-compliant.

In such a situation, we need to allow a set of possibitlities, as has been done for internationalization of file names in [RFC7530] for NFSv4.0 and in [I-D.ietf-nfsv4-internationalization], for all minor versions.

As in the case of the internationalization of file names, we will have to accommodate a range of possible server implementations, but the task will, in this case, involve a lot more work. To see why, we will first explore the situation for internationalization of file names and later see how the situation for these issues poses greater difficulties and how they might be addressed. Some of these dificulties match other internationalization issues, such as the handling of internationalized domain names and the proper treatment of case-insensitive handling of file names that dealt with incorrectly or not at all in [RFC7530].

*The treatment of internationalization of file names in [RFC7530] and [RFC5661] was so divorced from the needs of implementations that it was necessary to be ignore it.

  On the other hand, the treatment of internationalization of file names in the obsoleted document [RFC3010] was available as a base and had been implemented by many clients and servers.

*The vast majority of clients and servers implemented the same approach to internationalization, even though it was not limited to the use of UTF-8.

  Despite the general aversion to this approach within the IESG, it was accepted, most likely because we were describing NFSv4 internationalization on "as-implemented" basis.

*Overall, we wound up allowing a total of three sorts of server behaviors, two of which were UTF-8-aware.

  One UTF-8-aware option that had been implemented, simply considred canonically equivalent names as deignating the same file with changing the name to a different canonically equivalent name. Although this approach diverged from IESG expectations regarding normalization, it was accepted.

  Another UTF-8-aware option was allowed, despite the absence of any implementations. It allowed servers to normalize file names while forbidding them from rejecting unormalized names. This was allowed because it would cause no difficulties for clients and because forbidding it might have caused problems for IESG acceptance.

*We were able, where necessary, to give the client the ability to detrmine which form of internationalization supported provided by

the server for any particular file system, using an **OPTIONAL** attribute made available in NFSv4.1.

The situation with regard to the clarifying changes needed in this document is different for the reasons listed below.

* Rather than three valid choices for acceptable server behavior, with authorization we have a vast number.

  Given that there are nine ACE mask bits, each of whick might not be supported, there are at least 512 possible valid behaviors, even if none of the dubious instances of **SHOULD** creates additional valid behavioral patterns.

  Added to that are multiple (at least two) ways of mapping ACLs to modes and at leat three different behaviors in mdifying ACLs to reflect changes in mode.

  As a result, existing RFCs allow at least three thousand different server behaviors.

* There is little information available about the particular behaviors manifested by existing server-fs combinations.

  It may be difficult to get this informarion because the implementations were done by people no longer participating in the working or who were never involved in the working group

* There are no existing attributes that might be used to communicates server/fs choices to the client.

This creates a difficult situation. To resolve it will require substantial work as follows:

* Determining whether each of the dubious instances of "**SHOULD**" is in fact relied on by existing servers or might be needed by future servers.

* Determining which ACE mask bits are supported by all existing servers or not supported by any existing server to further limit the set of **OPTIONAL** features that clients need to be made aware of.

  There might also be cases where some set of mask bits are are always either supported or unsupported together.

* Determining the set of mappings from acls to modes that are actually and making each such **OPTIONAL** as opposed to the current situation in which an "intentional" "**SHOULD**" is used despite the mismatch between this use and [RFC2119].

* Making a similar determination of the actions actually applied to acl when the mode is changed.

Once we have the above information, it will be clear how we could create an appropriate extension as described in Section 16.2 to provide clients with an adequate description of expected server behavior. This document will be limited to providing client implementation guidance about dealing with the uncertinty in cases in which new attributes is not available. The situation will be similar to that described in Section 15 of [I-D.ietf-nfsv4-internationalization].

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 separate sets of code define for pending items and for those which are no longer pending.

The following codes are defined for pending consensus items:

* "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.

*"NE" denotes change based on an existing issue in the spec but for which the replacement text is incomplete and needs further elaboration.

*"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.

*"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.

*"EV" denote a potential change which is tentative or incomplete because further details need to be provide or because the author is unsure that he has a correct explanation of the issue. It is expected that replacement text will be available in a later draft.

The following codes are defined for consensus items which are no longer pending.

*"RT" designates a former item which has been retired, because it has been merged with another one or otherwise organized out of existence.

Such items no longer are referred to the document source although the item id is never reassigned. They are no longer counted among the set of total items.

*"CA" designates a former item for which consensus has been achieved in the judgment of the author, although not by any official process.

Items reaching this state are effected in the document source including the deletion of annotations and the elimination of obsoleted previous treatments.

Items in this state are still counted among the total of item but are no longer considered pending

*"CV" designates a former item for which consensus has been achieved and officially verified.

Because the author is a Working Group co-chair,it is probably best if he is not involved in this process and intends to leave it to the other co-chair, the Document Shepherd and the Area Director.

Items in this state are not counted among the item totals. They may be kept in the table but only to indicate that the item id is still reserved.

*"DR" designates a former item which has been dropped, because it appears that working group acceptance of it, even with some modification, is unlikely.

Such items no longer are referred to the document source although the item id is never reassigned. They are no longer counted among the set of total items.

When asterisk is appended to a state of "NM", "BC" 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 remain in this table as a non-pending item. It will be mentioned in Appendices A.1 or A.2 to summarize the changes that have been made.

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	CA	Assumed OK.	New approach distinguishing authetication and identification.
2	CA	Assumed OK.	Outline of new negotiation framework.
3	BE	#3a in S 5.4	

#	Type	...References...	Substance
			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 #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

#	Type	...References...	Substance
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 4.1 #21b in S 7.3.1	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 8.2	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 8.1 #23b in S 8.3	Revise discussion of client interpretation of ACLs and the use of ACCESS instead
24	BE	#24a in S 8.3	Delete bogus reference.
25	CI	#25a in S 3.3 #25b in S 9.1 #25d in S 9.8 #25e in S 9.10 #25f in S 9.11 #25g in S 9.12	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 9.7.3	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 possible existence of multiple methods of computing a mode from an acl that clients can depend on, and the proper elationship among these methods.
28	WI	#28a in S 9.2 #28b in S 9.3 #28 in S 9.7.3	Decide how to address flaws in mapping to/from reverse- slope modes.
29	BC	#29 in S 9.7.3	Address the coordination of mode and ACL-based attributes in unified way for all minor versions.
30	CI	#30a in S 9.7.1	

#	Type	...References...	Substance
		#30b in S 9.7.2 #30c in S 9.7.3	New proposed treatment of setting mode incorporating some consequences of anticipated decisions regarding other consensus items (#26, #28, #29)
31	WI	#31a in S 9.7.3	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 18 #32d in S 18.1 #32e in S 18.2	Expanded negotiation framework to accommodate multiple transport types and security services provided on a per-connection basis, i.e. encryption and peer authentication.
33	RJ	Material formerly here moved to #32.	Reorganization of description of SECINFO op to apply to all minor versions. (Dropped)
34	RJ	Superseded by simpler treatment.	Revision to NFSv4.0 SECINFO implementation section (Dropped.
35	NE	#35a in S 16.1	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 per-connection 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
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 8.2 #41b in S 17.3.1 #41c in S 17.3.2 #41d in S 17.3.4	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	RT	#42a in S 17.4.5	Former placeholders for threat analysis subsections have now been superseded by new proposed subsections.
43	RT	#43a in S 17.4.6.1	
44	RT		

#	Type	...References...	Substance
		#44a in S 17.4.6.2	
45	RT	#45a in S 17.4.7.1	
46	RT	#46a in S 17.4.7.2	
47	CI	gone fir now.	Dubious paragraph regarding AUTH_NONE is SECINFO response which should be deleted if there are no compatibility issues that make that impossible.
48	RJ	Superseded by simpler treatment.	Missing pieces of secinfo processing algorithm that didn't get done in -02.
49	RJ	Superseded by simpler treatment.	Secinfo processing algorithm that was expected to be finished in -04.
50	BC	#50a in S 5.9	Revise handling of "special" who values, making it clear for which ones "special" is a euphemism for "semantics-challenged".
51	BC	#51a in S 5.9	Clarify the handling of the group bit for the special who values.
52	BC	#52a in S 8.2 #52b in S 17.3.3	Eliminate superuser semantics as it had been, as valid by implication. Also, deal with the security consequences of its inclusion appropriately.
53	EV	#53a in S 17.4.2	Discussion of possible adaptation of RPCSEC_GSS/Kerberos to multi-factor authentication.
54	EV	#54a in S 17.4.6.1	Discussion of prevention of code on a compromised client from hijacking the client machine's peer authentication.
55	EV	#55a in S 17.4.6.1	Discussion of issues with potential use of Kerberos in cloud environments
56	WI	#56a in S 4.1 #56b in S 9.5 #56c in S 16.2	Discussion of issues related to the handling of allowed variants of the NFSv4 ACL model, including subsets based on the Unix ACL model.

Table 3

The following table summarizes the issues in each particular pending 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
BC(M)	2	32, 52
CI(M)	5	36, 38, 39, 40, 41
WI(M)	1	47
NE(M)	1	35
EV(M)	3	53, 54, 55
All(M)	12	As listed above.
NM(O)	1	16
BE(O)	4	3, 5, 7, 24
BC(O)	14	9, 10, 12, 13, 14, 15, 17, 18, 21, 22, 23, 29, 50, 51
CI(O)	10	4, 6, 8, 11, 19, 25, 26, 27, 30, 37
WI(O)	4	20, 28, 31, 56
All(O)	33	As described above
All	45	Grand total for above table.

Table 4

The following table summarizes the issues in each particular non-pending 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
CA(M)	2	1, 2
RT(M)	5	42, 43, 44, 45, 46
RJ(M)	4	33, 34, 48, 49
All(M)	11	As listed above.
All(O)	0	Nothing yet.
All	11	Grand total for above table.

Table 5

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 working on putting NFSv4 security on a firmer footing. We appreciate Nico's perceptive guidance.

The author wishes to thank Bruce Fields for his helpful comments regarding ACL support which had a major role in the evolution of this document.

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 much important guidance he provided that is reflected in this document, including his work to streamline the security negotiation process by the definition of new pseudo-flavors.

The author wishes to thank Rick Macklem for his help in resolving NFSv4 security issues. These include clarifying possible server policies regarding RPC-with-TLS, helping to clarify "owner-override semantics" and bringing to the Working Group's attention the possibility of deriving limited principal identification from client peer authentication while still staying within the boundaries of RPC-with-TLS.

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