NFSv4 Working Group
Internet-Draft
Intended status: draft
Expires: September 7, 2011
Updates: 5663

S. Faibish
EMC Corporation
J. Glasgow
Google
D. Black
EMC Corporation

March 7, 2011

pNFS block disk protection
draft-faibish-nfsv4-pnfs-block-disk-protection-00

Status of this Memo

Copyright Notice

Abstract

Parallel NFS (pNFS) extends Network File Sharing version 4 (NFSv4) to allow client to directly access file data on the storage used by the NFSv4 server. This ability to bypass the server for data access can increase both performance and parallelism, but requires additional client functionality for data access, some of which is dependent of the class of storage used. The published protocol for block layout file systems describes how clients can recognize the volumes used for pNFS storage, but only after communicating with the server. This document proposes a mechanism by which clients can recognize block storage devices used by pNFS file systems, without having to communicate with the server, and therefore allows the clients to limit access to the devices at client boot time.

Table of Contents

1. Introduction.....3
   1.1. Non-pNFS clients use case.....5
   1.2. pNFS client use case.....5
2. Conventions used in this document.....5
3. Extending block/volume signature.....5
   3.1. Modifications to 2.3.5 RFC5663.....7
4. Problem Statement.....7
   4.1. Non-pNFS clients.....7
   4.2. Solution.....7
   4.3. pNFS clients.....8
5. Changes to GETDEVICEINFO and or LAYOUTCOMMIT (RFC5661).....8
6. Security Considerations.....8
7. IANA Considerations.....9
8. Conclusions.....9
9. References.....9
   9.1. Normative References.....9
   9.2. Informative References.....9
Authors Addresses.....10

1. Introduction

Figure 1 shows the overall architecture of a Parallel NFS (pNFS) system:

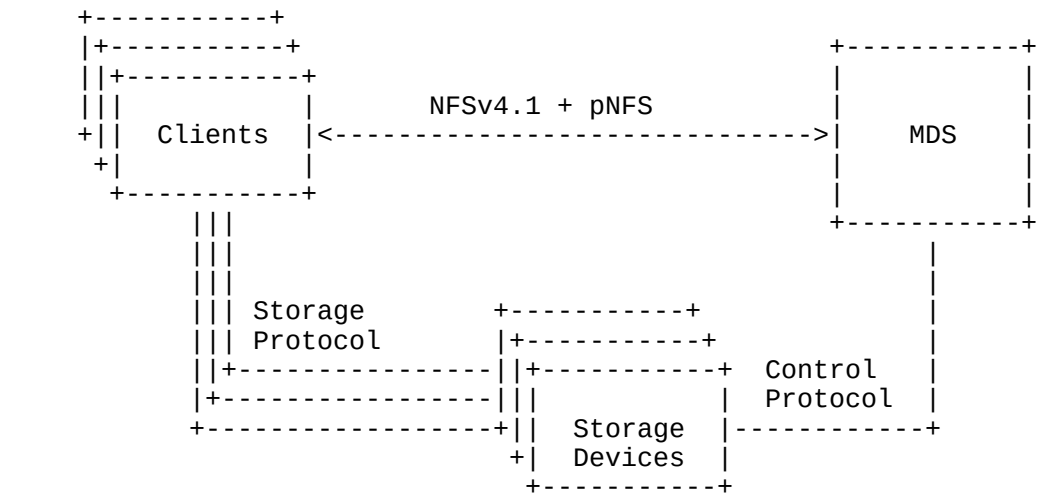

Figure 1 pNFS Architecture

In this document, "storage device" is used as a general term for a data server and/or storage server for the file, block or object pNFS layouts.

In the current pNFS block protocol [RFC5663] the client has to contact the MDS in order to get the signature offset of the devices used by the pNFS block client.

If an operating system wants to be able to identify a device/LUN being used to store pNFS data during the boot sequence, before the mount is issued, without having to contact the metadata server it is not possible because the protocol does not define a fixed location for a signature or for an identifier of a device as being used by pNFS. The MDS will send the device ID after mount time. In order for the OS to mount it needs to have the network configured and the IP address of the pNFS server and this always happens after the discovery of SCSI devices.

location of the signature is also not defined in the protocol but we can make it a configuration parameter in the client OS that will define the signature location for each pNFS block server, vendor neutral. The preferred solution would be to enhance the block protocol to include a signature unique identifying pNFS. It needs to be included in the protocol as both clients and server need to know this information.

In more details the pNFS FS writes a signature to the LUN, but to find out the offset of the signature the client first needs to talk with the MDS to be told the offset that the signature is located at using GETDEVICEINFO. But this is too late and can only be done after the boot ends. An OS that wants to hide LUNs from users that contain pNFS data even if that host doesn’t mount the pNFS block volume cannot do this.

The problem is that there is a window of time before an OS utility/daemon can be started and allow to protect/prevent to write to the pNFS devices and the time when kernel application with higher priority than the above daemon can overwrite the pNFS devices. As a result of this it may be possible that some kernel application will write over the pNFS FS devices/LUNs. This is only a pNFS block issue and will have no secondary effect on OS kernel operation.

An OS will check the signature of pNFS should be able to identify the devices and prevent from writing to them. But this enhancement should allow such a protection mechanism to be implemented. This is outside the scope of this draft to detect and protect the devices if the specific pNFS signature is detected and do so without the need to contact the MDS "if there is pNFS specific component of the signature. Even in this case we can use generic identifiers or some opaque in the protocol can recognize pNFS devices even though pNFS block protocol doesn’t require any protection.

Typically, storage area network (SAN) disk arrays and SAN protocols provide access control mechanisms (e.g., Logical Unit Number (LUN) mapping and/or masking), which operate at the granularity of individual hosts, not individual blocks. For this reason, block-based protection must be provided by the client software.

Since block/volume storage systems are generally not capable of enforcing such file-based security, in environments where pNFS clients cannot be trusted to enforce such policies, pNFS block/volume storage layouts SHOULD NOT be used.

1.1. Non-pNFS clients use case

The non-pNFS clients will need to check the signature related to pNFS block devices and prevent from WRITE data to the devices that have the pNFS identifier. The non-pNFS client OS MAY decide to implement an application that will read the pNFS signature at a known location (LBA) on the block device and write protect the device. Or it MAY just detect the pNFS devices and prevent using them for other applications that use block devices. The current draft only make these use cases possible and non-pNFS clients MAY NOT use this feature.

1.2. pNFS client use case

pNFS clients MAY use this signature in cases when a pNFS MDS server sends layout extents to devices that were erroneously not signed using this pNFS identification by the server and included in valid layouts. The pNFS client MAY check the pNFS signature for devices that are included in valid layouts and report an error to the MDS in the LAYOUTCOMMIT or using a permission access mechanism similar to [PAC]. This use case is relevant in cases when virtual LUs are used as pNFS devices and they are added after the client issued a GETDEVICEINFO. Alternatively when the pNFS client detects the missing signature it may send a GETDEVICEINFO command to the MDS for the faulty device and use a permission error code to the MDS.

2. Conventions used in this document

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 described in RFC-2119 [RFC2119].

3. Extending block/volume signature

The pNFS block layout identifies storage volumes by content, for example providing the means to match (unique portions of) labels used by volume managers. Volume identification is performed by matching one or more opaque byte sequences to specific parts of the stored data. Any block pNFS system using this layout MUST support a means of content-based unique volume identification that can be employed via the data structure given here. The location of the pNFS signature will be defined in the remaining of the document.

The GUID used in the GPT should not be confused with the device signature described in section 2.2.1 of [RFC5663]. The GUID is the same for all pNFS volumes where as the signature composed of pnfs_block_sig_component4 components is unique to each pNFS volume. The location on disk of these identifiers are unrelated.

A pNFS signature is an array of up to "PNFS_BLOCK_MAX_DISK_SIG_COMP" (defined below) signature components. The client MUST NOT assume that all signature components are co-located within a single sector on a block device. The pNFS client block layout driver uses this volume identification to identify block devices used by pNFS file systems.

The pNFS device signature may require several LBAs in order to ensure uniqueness of the signature in case that other disk signatures exist on the same location on the block device and prevent confusions with other signatures.

All the new pNFS identifiers must not be used in extents covering data access and need to be read only accessible to both pNFS and non- pNFS clients. Additional, pNFS clients SHOULD NOT access/write to volumes that do not have the pNFS specific volume identifier if included in a valid layout.

This can be relevant in cases of virtual LUs when they are extended to include new volumes uninitialized by the server. After the GETDEVICEINFO is received by the client the client MAY read the pNFS identifier in order to validate the extents on that device and prevent from writing to devices that do not include pNFS identifier.

In this case pNFS clients will send an error to the server and inform it that the layout is invalid and or that there is a permission access error. And will fallback the I/O to the MDS. The error mechanism can be similar to the one used in the [pNFS-draft-permission-access] for lack of permission to access reinforced by the pNFS client.

pNFS client implementations MAY chose not to validate that the device included in the layouts received from the server is a pNFS device. The pNFS client can OPTIONALLY check the pNFS identifier and send an error message in the layoutcommit for any device with a missing pNFS identifier.

3.1. Modifications to 2.3.5 RFC5663

The above assumption that extents are permissions may be in contradiction to the case when a device in the layout has no pNFS identity without the knowledge of the pNFS server. This because the pNFS client is the first to detect the lack of identifier while the server is unaware of this issue and assumed that the device used in the layout and the respective extent has the right permissions. The server MAY check that the device used in the layout is a legitimate pNFS device at the time it respond to the GETDEVICEINFO call from the client.

Alternatively in the case when a device has no pNFS identifier the pNFS client SHOULD send an GETDEVICEINFO for that device to ensure that the device included in a valid layout can be accessed for IO. It may or may not report an error in the GETDEVICEINFO and/or in the LAYOUTCOMMIT at the end of the write I/O.

The check can be made for both read-only layouts but SHOULD be done in the case of read-write layouts.

The server is responsible to ensure that the pNFS id is written to the devices used by the pNFS FSID before allowing pNFS clients to mount the fsid.

4. Problem Statement

4.1. Non-pNFS clients

The pNFS block layout [RFC5663] requires that storage systems allow both the server and multiple clients to access block storage devices. In principle, access control to block storage devices can be achieved via LUN masking techniques at the storage server as volumes are mounted. In practice this is not always done, and storage devices are often accessible independent of whether or not a client has mounted the associated pNFS file system. For this reason, it is desirable to enable clients to identify pNFS block storage in order to prevent non-pNFS access by the client.

4.2. Solution

The pNFS block layout specification allows a client to identify the storage devices associated with a mounted volume via the GETDEVICELIST and GETDEVICEINFO operations. These operations can only be issued once a client has established contact with the server, and thus do not allow a client to block non-pNFS access to pNFS storage devices in all cases. For environments in which it is

desirable for clients to block non-pNFS access to pNFS block storage, the following SHOULD be done:

- Each storage device dedicated to pNFS includes a GUID partition table (GPT).
- The pNFS Block Storage partitions are identified in the GPT with GUID e5b72a69-23e5-4b4d-b176-16532674fc34.
- NFS clients do not issue NFS operations for non-pNFS access to any storage identified as pNFS Block Storage by that GUID.

This enables an NFS client to prevent non-pNFS access to pNFS block storage immediately upon boot. As of 2010 most current OSs support GPT including FreeBSD, Linux and Solaris [GPT]. Block/volume storage is logically at a lower layer of the I/O stack than the network stack and hence during the client boot cycle access to block devices is possible before NFSv4 security can be enforced. The client MUST prevent WRITE I/Os to devices that will be later identified as pNFS devices. It is the responsibility of those administering and deploying pNFS with a block/volume storage access protocol to ensure that appropriate protection is provided to pNFS devices identifiable by the client. This protection is similar to the mechanism that protect GPT tables written on the block devices used by different operating systems.

4.3. pNFS clients

In this case the client MAY check if the devices in the GETDEVICEINFO list have pNFS signature at mount time. At I/O time the pNFS client MAY check the signature to validate that the block devices are valid and report to the server the error.

5. Changes to GETDEVICEINFO and or LAYOUTCOMMIT (RFC5661)

[Need to include the changes if we agree on the need for this]

6. Security Considerations

The security considerations of 2.6 [RFC5663] apply to this document. This document formalizes a mechanism which allows client operating systems to limit access to pNFS block storage devices. By doing so it allows for increase security in environments in which the client operating system is trusted. As with the issues discussed in the security section of 2.6 [RFC5663], in environments where the security

requirements are such that client-side protection from access to storage is not sufficient, pNFS block/volume storage layouts SHOULD NOT be used. Furthermore, in such environments, SAN mechanism (e.g., LUN mapping and/or masking) should be used to limit access.

7. IANA Considerations

There are no IANA considerations in this document beyond pNFS IANA Considerations are covered in [RFC5661].

8. Conclusions

This draft specifies additions to the pNFS block protocol addressing protection of disks used by pNFS clients for non-pNFS clients access.

9. References

9.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", http://tools.ietf.org/html/rfc5661, January 2010.

[RFC5663] Black, D., Glasgow, J., Fridella, S., "Parallel NFS (pNFS) Block/Volume Layout", http://tools.ietf.org/html/rfc5663, January 2010.

[RFC5664] Halevy, B., Welch, B., Zelenka, J., "Object-Based Parallel NFS (pNFS) Operations", http://tools.ietf.org/html/rfc5664, January 2010

[XDR] Eisler, M., "XDR: External Data Representation Standard", STD 67, RFC 4506, May 2006.

9.2. Informative References

[GPT] http://en.wikipedia.org/wiki/GUID_Partition_Table

[PAC] https://datatracker.ietf.org/doc/draft-ietf-nfsv4-pnfs-access-permissions-check/

This document was prepared using 2-Word-v2.0.template.dot.

Authors Addresses

Sorin Faibish (editor)
EMC Corporation
228 South Street
Hopkinton, MA 01748
US

Phone: +1 (508) 305-8545
Email: sfaibish@emc.com

Jason Glasgow
Google
5 Cambridge Center, Floors 3-6
Cambridge, MA 02142
US

Phone: +1 (617) 575 1599
Email: jglasgow@google.com

David L. Black
EMC Corporation
176 South Street
Hopkinton, MA 01748
US

Phone: +1 (508) 293-7953
Email: david.black@emc.com

NFSv4 T. Myklebust
Internet-Draft NetApp
Intended status: Standards Track March 5, 2011
Expires: September 6, 2011

Sharing change attribute implementation details with NFSv4 clients
draft-myklebust-nfsv4-change-attribute-type-00

Abstract

This document describes an extension to the NFSv4 protocol that allows the server to share information about the implementation of its change attribute with the client. The aim is to improve the client's ability to determine the order in which parallel updates to the same file were processed.

Keywords

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 described in [RFC2119].

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 http://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 September 6, 2011.

Copyright Notice

Table of Contents

1. Introduction 4
2. Definition of the 'change_attr_type' per-file system attribute 5
3. References 7
Author's Address 8

1. Introduction

Although both the NFSv4 [RFC3530] and NFSv4.1 protocol [RFC5661], define the change attribute as being mandatory to implement, there is little in the way of guidance. The only feature that is mandated by the spec is that the value must change whenever the file data or metadata change.

While this allows for a wide range of implementations, it also leaves the client with a conundrum: how does it determine which is the most recent value for the change attribute in a case where several RPC calls have been issued in parallel? In other words if two COMPOUNDs, both containing WRITE and GETATTR requests for the same file, have been issued in parallel, how does the client determine which of the two change attribute values returned in the replies to the GETATTR requests corresponds to the most recent state of the file? In some cases, the only recourse may be to send another COMPOUND containing a third GETATTR that is fully serialised with the first two.

In order to avoid this kind of inefficiency, we propose a method to allow the server to share details about how the change attribute is expected to evolve, so that the client may immediately determine which, out of the several change attribute values returned by the server, is the most recent.

2. Definition of the 'change_attr_type' per-file system attribute

```
enum change_attr_typeinfo = {
        NFS4_CHANGE_TYPE_IS_MONOTONIC_INCR         = 0,
        NFS4_CHANGE_TYPE_IS_VERSION_COUNTER        = 1,
        NFS4_CHANGE_TYPE_IS_VERSION_COUNTER_NOPNFS = 2,
        NFS4_CHANGE_TYPE_IS_TIME_METADATA          = 3,
        NFS4_CHANGE_TYPE_IS_UNDEFINED              = 4
};
```

Name	Id	Data Type	Acc
change_attr_type	XX	enum change_attr_typeinfo	R

The proposed solution is to enable the NFS server to provide additional information about how it expects the change attribute value to evolve after the file data or metadata has changed. To do so, we define a new recommended attribute, 'change_attr_type', which may take values from enum change_attr_typeinfo as follows:

NFS4_CHANGE_TYPE_IS_MONOTONIC_INCR: The change attribute value MUST monotonically increase for every atomic change to the file attributes, data or directory contents.

NFS4_CHANGE_TYPE_IS_VERSION_COUNTER: The change attribute value MUST be incremented by one unit for every atomic change to the file attributes, data or directory contents. This property is preserved when writing to pNFS data servers.

NFS4_CHANGE_TYPE_IS_VERSION_COUNTER_NOPNFS: The change attribute value MUST be incremented by one unit for every atomic change to the file attributes, data or directory contents. In the case where the client is writing to pNFS data servers, the number of increments is not guaranteed to exactly match the number of writes.

NFS4_CHANGE_TYPE_IS_TIME_METADATA: The change attribute is implemented as suggested in the NFSv4 spec [RFC3530] in terms of the time_metadata attribute.

NFS4_CHANGE_TYPE_IS_UNDEFINED: The change attribute does not take values that fit into any of these categories.

If either NFS4_CHANGE_TYPE_IS_MONOTONIC_INCR, NFS4_CHANGE_TYPE_IS_VERSION_COUNTER, or NFS4_CHANGE_TYPE_IS_TIME_METADATA are set, then the client knows at the very least that the change attribute is monotonically increasing, which is sufficient to resolve the question of which value is the most recent.

If the client sees the value NFS4_CHANGE_TYPE_IS_TIME_METADATA, then by inspecting the value of the 'time_delta' attribute it additionally has the option of detecting rogue server implementations that use time_metadata in violation of the spec.

Finally, if the client sees NFS4_CHANGE_TYPE_IS_VERSION_COUNTER, it has the ability to predict what the resulting change attribute value should be after a COMPOUND containing a SETATTR, WRITE, or CREATE. This again allows it to detect changes made in parallel by another client. The value NFS4_CHANGE_TYPE_IS_VERSION_COUNTER_NOPNFS permits the same, but only if the client is not doing pNFS WRITEs.

3. References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119.

[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.

[RFC5661] Shepler, S., Eisler, M., and D. Noveck, "Network File System (NFS) Version 4 Minor Version 1 Protocol", RFC 5661.

Author's Address

Trond Myklebust
NetApp
3215 Bellflower Ct
Ann Arbor, MI 48103
USA

Phone: +1-734-662-6608
Email: Trond.Myklebust@netapp.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2011

D. Quigley

J. Morris
Red Hat, Inc.

March 14, 2011

MAC Security Label Support for NFSv4
draft-quigley-nfsv4-sec-label-03.txt

Abstract

This Internet-Draft describes additions to NFSv4 minor version one to support Mandatory Access Control systems. The current draft describes the mechanism for transporting a MAC security label using the NFSv4.1 protocol and the semantics required for that label. In addition to this it describes an example system of using this label in a fully MAC aware environment.

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 http://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 September 15, 2011.

Table of Contents

1. Introduction 3
2. Terms and Definitions 4
3. MAC Security Attribute 4
3.1. Interpreting security_attribute 5
3.2. Delegations 6
3.3. Permission Checking 6
3.4. Object Creation 6
4. MAC Security NFS Modes of Operation 6
4.1. Full Mode 7
4.1.1. Initial Labeling and Translation 7
4.1.2. Policy Enforcement 7
4.2. Smart Client Mode 8
4.2.1. Initial Labeling and Translation 8
4.2.2. Policy Enforcement 8
4.3. Smart Server Mode 9
4.3.1. Initial Labeling and Translation 9
4.3.2. Policy Enforcement 9
5. Examples 10
5.1. Multi-Level Security 10
5.1.1. Full Mode 10
5.1.2. Smart Client Mode 11
5.1.3. Smart Server Mode 11
6. Security Considerations 11
7. IANA Considerations 12
8. Normative References 12
Authors' Addresses 12

1. Introduction

Mandatory Access Control (MAC) systems have been mainstreamed in modern operating systems such as Linux (R), FreeBSD (R), Solaris (TM), and Windows Vista (R). MAC systems bind security attributes to subjects (processes) and objects within a system. These attributes are used with other information in the system to make access control decisions.

Access control models such as Unix permissions or Access Control Lists are commonly refered to as Discretionary Access Control (DAC) models. These systems base their access decisions on user identity and resource ownership. In contrast MAC models base their access control decisions on the label on the subject (usually a process) and the object it wishes to access. These labels may contain user identity information but usually contain additional information. In DAC systems users are free to specify the access rules for resources that they own. MAC models base their security decisions on a system wide policy established by an administrator or organization which the users do not have the ability to override. DAC systems offer no real protection against malicious or flawed software due to each program running with the full permissions of the user executing it. Inversely MAC models can confine malicious of flawed software and usually act at a finer granularity than their DAC counterparts.

People desire to use NFSv4 with these systems. A mechanism is required to provide security attribute information to NFSv4 clients and servers. This mechanism has the following requirements:

- Clients must be able to convey to the server the security attribute of the subject making the access request. The server may provide a mechanism to enforce MAC policy based on the requesting subject’s security attribute.

- Server must be able to store and retrieve the security attribute of exported files as requested by the client.

- Server must provide a mechanism for notifying clients of attribute changes of files on the server.

- Clients and Servers must be able to negotiate Label Formats and Domains of Interpretation (DOI) and provide a mechanism to translate between them as needed.

These four requirements are key to the system with only requirements two and three requiring changes to NFSv4.1. The ability to convey the security attribute of the subject as described in requirement one falls upon the RPC layer to implement. Requirement four allows

communication between different MAC implementations. The management of label formats, DOIs and the translation between them does not require any support from NFS on a protocol level and is out of the scope of this document.

2. Terms and 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 described in [RFC2119].

Label Format Specifier: An identifier used by the client to establish the syntactic format of the security label and the semantic meaning of its components. These specifiers exist in a registry associated with documents describing the format and semantics of the label.

Label Format Registry: The IANA registry containing all registered Label Format Specifiers along with references to the documents that describe the syntactic format and semantics of the security label.

Policy Identifier: An optional part of the definition of a Label Format Specifier which allows for clients and server to identify specific security policies.

Domain of Interpretation: A Domain of Interpretation (DOI) represents an administrative security boundary, where all systems within the DOI have semantically coherent labeling. That is, a security attribute must always mean exactly the same thing anywhere within the DOI.

Object: An object is a passive resource within the system that we wish to be protected. Objects can be entities such as files, directories, pipes, sockets, and many other system resources relevant to the protection of the system state.

Subject: A subject is an active entity usually a process which is requesting access to an object.

3. MAC Security Attribute

MAC models base access decisions on security attributes bound to subjects and objects. This information can range from a user identity for an identity based MAC model, sensitivity levels for Multi-level security, or a type for Type Enforcement. These models base their decisions on different criteria but the semantics of the security attribute remain the same. The semantics required by the security attributes are listed below:

- Must provide flexibility with respect to MAC model.
- Must provide the ability to atomically set security information upon object creation
- Must provide the ability to enforce access control decisions both on the client and the server
- Must not expose an object to either the client or server name space before its security information has been bound to it.

NFSv4 provides several options for implementing the security attribute. The first option is to implement the security attribute as a named attribute. Named attributes provide flexibility since they are treated as an opaque field but lack a way to atomically set the attribute on creation. In addition, named attributes themselves are file system objects which need to be assigned a security attribute. This raises the question of how to assign security attributes to the file and directories used to hold the security attribute for the file in question. The inability to atomically assign the security attribute on file creation and the necessity to assign security attributes to its subcomponents makes named attributes unacceptable as a method for storing security attributes.

The second option is to implement the security attribute as a recommended attribute as defined in section 5.2 of RFC 3530. Recommended attributes have a fixed format and semantics, which conflicts with the flexible nature of the security attribute. To resolve this the security attribute consists of two components. The first component is a Label Format Specifier(LFS) as defined in [XXX: Insert Doc here] to allow for interoperability between MAC mechanisms. The second component is an opaque field which the actual security attribute data. To allow for various MAC models NFSv4 should be used solely as a transport mechanism for the security attribute. It is the responsibility of the endpoints to consume the security attribute and make access decisions based on their respective models. In addition, creation of objects through OPEN and CREATE allows for the security attribute to be specified upon creation. By providing an atomic create and set operation for the security attribute it is possible to enforce the second and fourth requirements. To this end the attribute number XXX:ATTRNUM is requested for the security_attribute recommended attribute.

3.1. Interpreting security_attribute

The security_attribute field contains two components the first component being an LFS. The LFS serves to provide the receiving end with the information necessary to translate the security attribute

into a form that is usable by the endpoint. Label Formats assigned an LFS may optionally choose to include a Policy Identifier (PI) field to allow for complex policy deployments. The Label Format Specifier and Label Format Registry are described in detail in [XXX: Insert RFC Here]. The translation used to interpret the security attribute is not specified as part of the protocol as it may depend on various factors. The second component is an opaque section which contains the data of the attribute. This component is dependent on the MAC model to interpret and enforce. The character '@' is reserved as a separator between the LFS and opaque section of the security attribute.

3.2. Delegations

In the event that a security attribute is changed on the server while a client holds a delegation on the file, the client should follow the existing protocol with respect to attribute changes. It should flush all changes back to the server and relinquish the delegation.

3.3. Permission Checking

It is not feasible to enumerate all possible MAC models and even levels of protection within a subset of these models. This means that the NFSv4 client and servers can not be expected to directly make access control decisions based on the security attribute. Instead NFSv4 should defer permission checking on this attribute to the host system. These checks are performed in addition to existing DAC and ACL checks outlined in the NFSv4 protocol. Section 4 gives a specific example of how the security attribute is handled under a particular MAC model.

3.4. Object Creation

When creating files in NFSv4 the OPEN and CREATE operations are used. One of the parameters to these operations is an fattr4 structure containing the attributes the file is to be created with. This allows NFSv4 to atomically set the security attribute of files upon creation. When a client is MAC aware it must always provide the initial security attribute upon file creation. In the event that the server is the only MAC aware entity in the system it should ignore the security attribute specified by the client and instead make the determination itself. A more in depth explanation can be found in Section 4.

4. MAC Security NFS Modes of Operation

A system using Labeled NFS may operate in three modes. The first

mode provides the most protection and is called "full" mode. In this mode both the client and server implement a MAC model allowing each end to make an access control decision. The remaining two modes are variations on each other and are called "smart client" and "smart server" modes. In these modes one end of the connection is not implementing a MAC model and because of this these operating modes offer less protection than full mode.

4.1. Full Mode

Full mode environments consist of MAC aware NFSv4 servers and clients and may be composed of mixed MAC models and policies. The system requires that both the client and server have an opportunity to perform an access control check based on all relevant information within the network. The file object security attribute is provided using the mechanism described in Section 3. The security attribute of the subject making the request is transported at the RPC layer using the mechanism described in XXX: Insert RFC for RPC layer label draft.

4.1.1. Initial Labeling and Translation

The ability to create a file is an action that a MAC model may wish to mediate. The client is given the responsibility to determine the initial security attribute to be placed on a file. This allows the client to make a decision as to the acceptable security attributes to create a file with before sending the request to the server. Once the server receives the creation request from the client it may choose to evaluate if the security attribute is acceptable.

Security attributes on the client and server may vary based on MAC model and policy. To handle this the security attribute field has an LFS component. This component is a mechanism for the host to identify the format and meaning of the opaque portion of the security attribute. A full mode environment may contain hosts operating in several different LFSs and DOIs. In this case a mechanism for translating the opaque portion of the security attribute is needed. The actual translation function will vary based on MAC model and policy and is out of the scope of this document. If a translation is unavailable for a given LFS and DOI then the request SHOULD be denied. Another recource is to allow the host to provide a fallback mapping for unknown security attributes.

4.1.2. Policy Enforcement

In full mode access control decisions are made by both the clients and servers. When a client makes a request it takes the security attribute from the requesting process and makes an access control

decision based on that attribute and the security attribute of the object it is trying to access. If the client denies that access an RPC call to the server is never made. If however the access is allowed the client will make a call to the NFS server.

When the server receives the request from the client it extracts the security attribute conveyed in the RPC request. The server then uses this security attribute and the attribute of the object the client is trying to access to make an access control decision. If the server's policy allows this access it will fulfill the client's request, otherwise it will return NFS4ERR_ACCESS

Implementations MAY validate security attributes supplied over the network to ensure that they are within a set of attributes permitted from a specific peer, and if not, reject them. Note that a system may permit a different set of attributes to be accepted from each peer. An example of this can be seen in Section 5.1.1.

4.2. Smart Client Mode

Smart client environments consist of NFSv4 servers that are not MAC aware but NFSv4 clients that are. Clients in this environment are may consist of groups implementing different MAC models policies. The system requires that all clients in the environment be responsible for access control checks. Due to the amount of trust placed in the clients this mode is only to be used in a trusted environment.

4.2.1. Initial Labeling and Translation

Just like in full mode the client is responsible for determining the initial label upon object creation. The server in smart client mode does not implement a MAC model, however, it may provide the ability to restrict the creation and labeling of object with certain labels based on different criteria as described in Section 4.1.2.

In a smart client environment a group of clients operate in a single DOI. This removes the need for the clients to maintain a set of DOI translations. Servers should provide a method to allow different groups of clients to access the server at the same time. However it should not let two groups of clients operating in different DOIs to access the same files.

4.2.2. Policy Enforcement

In smart client mode access control decisions are made by the clients. When a client accesses an object it obtains the security attribute of the object from the server and combines it with the

security attribute of the process making the request to make an access control decision. This check is in addition to the DAC checks provided by NFSv4 so this may fail based on the DAC criteria even if the MAC policy grants access. As the policy check is located on the client an access control denial should take the form that is native to the platform.

4.3. Smart Server Mode

Smart server environments consist of NFSv4 servers that are MAC aware and one or more MAC unaware clients. The server is the only entity enforcing policy, and may selectively provide standard NFS services to clients based on their authentication credentials and/or associated network attributes (e.g. IP address, network interface). The level of trust and access extended to a client in this mode is configuration-specific.

4.3.1. Initial Labeling and Translation

In smart server mode all labeling and access control decisions are performed by the NFSv4 server. In this environment the NFSv4 clients are not MAC aware so they can not provide input into the access control decision. This requires the server to determine the initial labeling of objects. Normally the subject to use in this calculation would originate from the client. Instead the NFSv4 server may choose to assign the subject security attribute based on their authentication credentials and/or associated network attributes (e.g. IP address, network interface).

In smart server mode security attributes are contained solely within the NFSv4 server. This means that all security attributes used in the system remain within a single LFS and DOI. Since security attributes will not cross DOIs or change format there is no need to provide any translation functionality above that which is needed internally by the MAC model.

4.3.2. Policy Enforcement

All access control decisions in smart server mode are made by the server. The server will assign the subject a security attribute based on some criteria (e.g. IP address, network interface). Using the newly calculated security attribute and the security attribute of the object being requested the MAC model makes the access control check and returns NFS4ERR_ACCESS on a denial and NFS4_OK on success. This check is done transparently to the client so if the MAC permission check fails the client may be unaware of the reason for the permission failure. When operating in this mode administrators attempting to debug permission failures should be aware to check the

MAC policy running on the server in addition to the DAC settings.

5. Examples

The section below outlines an example of the Labeled NFS operation modes using a traditional Multi Level Security(MLS) model where objects are given a sensitivity level (Unclassified, Secret, Top Secret etc..) and a category set. This is just one example of a possible MAC model and is not required by labeled NFS implementations.

5.1. Multi-Level Security

In a Multi-level security (MLS) system objects are generally assigned a sensitivity level, and a set of compartments. The sensitivity levels within the system are given an order ranging from lowest to highest classification level. Read access to an object is allowed when the sensitivity level of the subject "dominates" the object it wants to access. This means that the sensitivity level of the subject is higher than that of the object it wishes to access and that its set of compartments is a super-set of the compartments on the object.

The rest of the section will just use sensitivity levels. In general the example is a client that wishes to list the contents of a directory. The system defines the sensitivity levels Unclassified(U), Secret(S), and Top Secret(TS). The directory to be searched is labeled Top Secret which means access to read the directory will only be granted if the subject making the request is also labeled Top Secret.

5.1.1. Full Mode

In the first part of this example a process on the client is running at the Secret level. The process issues a readdir system call which enters the kernel. Before translating the readdir system call into a request to the NFSv4 server the host operating system will consult the MAC module to see if the operation is allowed. Since the process is operating at Secret and the directory to be accessed is labeled Top Secret the MAC module will deny the request and an error code is returned to user space.

Consider a second case where instead of running at Secret the process is running at Top Secret. In this case the sensitivity of the process is equal to or greater than that of the directory so the MAC module will allow the request. Now the readdir is translated into the necessary NFSv4 call to the server. For the RPC request the

client is using the proper credential to assert to the server that the process is running at Top Secret.

When the server receives the request it extracts the security label from the RPC session and retrieves the label on the directory. The server then checks with its MAC module if a Top Secret process is allowed to read the contents of the Top Secret directory. Since this is allowed by the policy then the server will return the appropriate information back to the client.

In this example the policy on the client and server were both the same. In the event that they were running different policies a translation of the labels might be needed. In this case it could be possible for a check to pass on the client and fail on the server. The server may consider additional information when making its policy decisions. For example the server could determine that a certain subnet is only cleared for data up to Secret classification. If that constraint was in place for the example above the client would still succeed, but the server would fail since the client is asserting a label that it is not able to use (Top Secret on a Secret network).

5.1.2. Smart Client Mode

In smart client mode the example is identical to the first part of a full mode operation. A process on the client labeled Secret wishes to access a Top Secret directory. As in the full mode example this is denied since Secret does not dominate Top Secret. If the process were operating at Top Secret it would pass the local access control check and the NFSv4 operation would proceed as in a normal NFSv4 environment.

5.1.3. Smart Server Mode

In a smart server mode the client behaves as if it were in a normal NFSv4 environment. Since the process on the client does not provide a security attribute the server must define a mechanism for labeling all requests from a client. Assume that the server is using the same criteria used in the full mode example. The server sees the request as coming from a subnet that is a Secret network. The server determines that all clients on that subnet will have their requests labeled with Secret. Since the directory on the server is labeled Top Secret and Secret does not dominate Top Secret the server would fail the request with NFS4ERR_ACCESS.

6. Security Considerations

Depending on the level of protection the MAC system offers there may

be a requirement to tightly bind the security attribute to the data. It must be taken into consideration that when used in a pNFS environment is it possible that the security attribute and file data will be stored on separate servers.

7. IANA Considerations

XXX: Add section about LFS and LFS Registry referecing the other document.

8. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.

Authors' Addresses

David P. Quigley

Email: dpquigl@davequigley.com

James Morris
Red Hat, Inc.

Email: jmorris@namei.org