

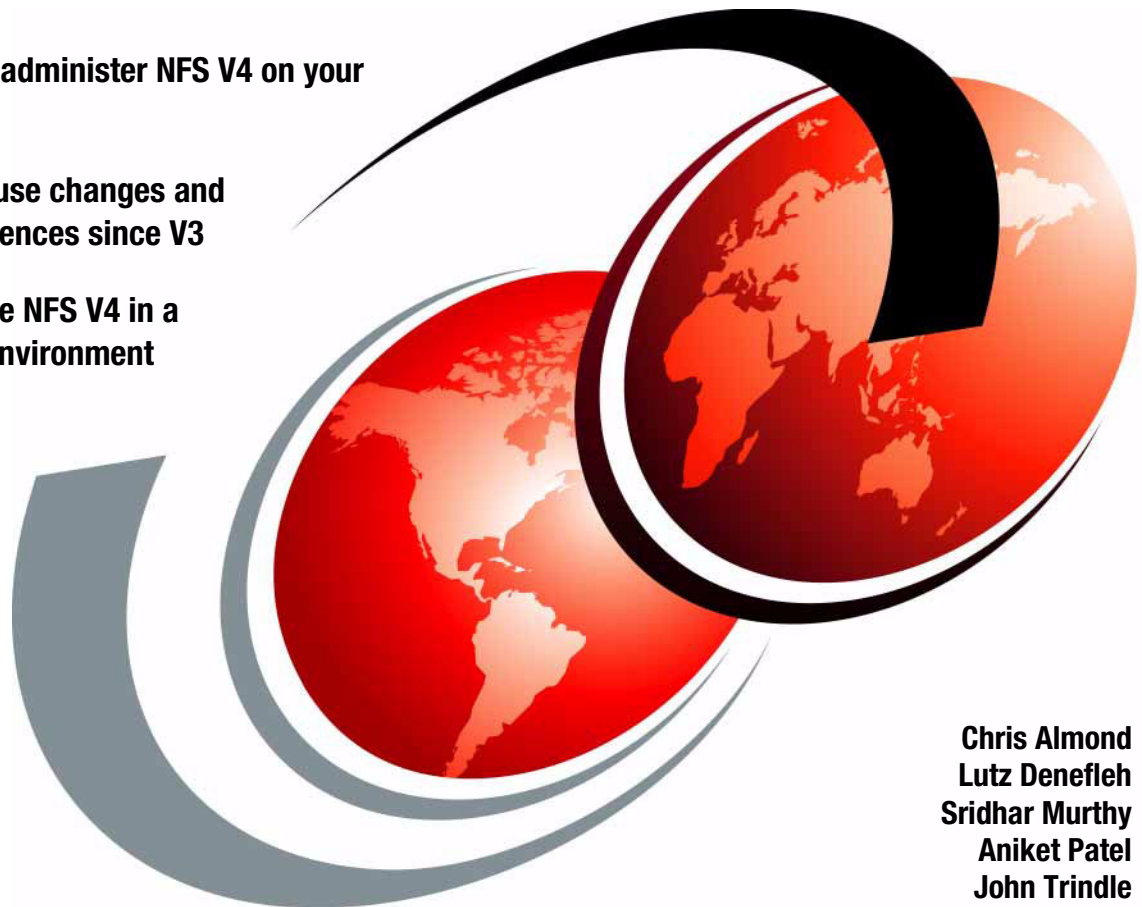
Securing NFS in AIX

An Introduction to NFS V4 in AIX 5L Version 5.3

Set up and administer NFS V4 on your systems

Command use changes and other differences since V3

Learn to use NFS V4 in a clustered environment



Chris Almond
Lutz Deneffle
Sridhar Murthy
Aniket Patel
John Trindle



International Technical Support Organization

Securing NFS in AIX

November 2004

Note: Before using this information and the product it supports, read the information in “Notices” on page ix.

First Edition (November 2004)

This edition applies to NFS Volume 4 running on AIX 5L.

© Copyright International Business Machines Corporation 2004. All rights reserved.

Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Trademarks	x
Preface	xi
The team that wrote this redbook	xiii
Acknowledgements	xiv
Become a published author	xv
Comments welcome	xv
Part 1. NFS V4 fundamentals	1
Chapter 1. NFS Version 4 overview	3
1.1 What is NFS?	4
1.2 NFS V2 and NFS V3 History	4
1.3 NFS V4 design motivations	5
1.4 Objectives of NFS V4 (RFC3530)	5
1.5 AIX 5.3 specific implementation of NFS V4	6
1.5.1 Mandatory features	6
1.5.2 Optional features	7
1.6 Planning and implementation considerations	7
1.6.1 Pre-implementation design considerations	7
1.7 Looking ahead to the rest of the book	8
Chapter 2. What's new in NFS V4?	11
2.1 How NFS works	12
2.2 Protocols used by NFS	12
2.2.1 UDP or TCP	13
2.2.2 Remote Procedure Call (RPC)	14
2.2.3 eXternal Data Representation (XDR)	14
2.3 NFS daemons	14
2.3.1 The portmap daemon	15
2.3.2 The rpc.mountd daemon	16
2.3.3 The rpc.statd daemon	16
2.3.4 The rpc.lockd daemon	16
2.3.5 The nfsd daemon	16
2.3.6 The block I/O daemon (biod)	17
2.4 NFS V3	17
2.5 The NFS Lock Manager protocol	19
2.6 NFS V4	20

2.6.1	Attribute classes	20
2.6.2	Username to UID mapping	24
2.6.3	Better namespace handling	25
2.6.4	Built-in security	27
2.6.5	Client-side caching and delegation	28
2.6.6	Compound RPC procedures	29
2.6.7	File locking	29
2.6.8	Internationalization	30
2.6.9	Volatile file handles	30
2.7	AIX 5L v5.3 implementation of NFS V4	30
2.8	NFS V4 supported features in AIX 5.3	31
2.8.1	Mandatory feature support	31
2.8.2	Other unsupported features	32
2.8.3	Optional feature support	32
2.8.4	NFS4 ACL	32
2.8.5	AIXC ACLs	33
2.8.6	External name space (exname)	34
2.8.7	Protocol differences: server exporting and client mounting	35
2.8.8	NFS files	36
2.8.9	Restricting NFS port ranges	41
2.8.10	Use of NFS_NOBODY	41
2.9	NFS daemons, files, and commands: a quick reference	41
Chapter 3. Enhanced security in NFS V4		45
3.1	General security concepts and terminology	46
3.1.1	Broad security categories	46
3.1.2	Information security components	46
3.1.3	RPC security flavors	47
3.1.4	RPCSEC_GSS protection levels	47
3.1.5	RPCSEC_GSS protection mechanisms	47
3.1.6	Looking ahead to the rest of the chapter	48
3.2	NFS V4 user/group identification	48
3.2.1	User identity management options	48
3.2.2	User/group identities and NFS V4	50
3.3	NFS V4 user authentication	59
3.3.1	AUTH_SYS user authentication	59
3.3.2	RPCSEC_GSS user authentication using Kerberos	59
3.4	NFS V4 user authorization	62
3.4.1	Standard UNIX file permissions	63
3.4.2	AIXC ACLs	63
3.4.3	NFS V4 ACLs: description	65
3.4.4	NFS V4 ACLs: ACL evaluation	69
3.4.5	NFS V4 ACLs: administration	72

3.4.6 NFS V4 ACLs: permissions scenarios	85
3.4.7 NFS V4 ACLs: NFS V3 clients	87
3.5 NFS V4 host identification	87
3.5.1 Basic host identification	87
3.5.2 Kerberos host identification	88
3.6 NFS V4 host authentication	88
3.7 NFS V4 host authorization	88
Part 2. Implementing NFS V4	91
Chapter 4. Planning for NFS V4	93
4.1 Deployment of NFS V4 in general	95
4.2 Mandatory requirements	97
4.2.1 What is your name resolution type?	97
4.2.2 Choosing your NFS domain	98
4.3 Identification methods	99
4.3.1 Selecting the user/group repository	100
4.3.2 Other identification considerations	101
4.4 NFS Authentication methods	102
4.4.1 AUTH_SYS method	102
4.4.2 Deploying Kerberos	102
4.4.3 Default types of encryption for KDC and security flavors	107
4.4.4 NFS client considerations when using Kerberos	108
4.4.5 Deployment of LDAP	109
4.5 Authorization methods	110
4.5.1 Choosing your user authorization method	110
4.5.2 Other user authorization considerations	111
4.6 Choosing the appropriate file system types	111
4.7 NFS protocols and namespace considerations	112
4.7.1 Pseudo-root FS - alias tree versus classic model	115
4.8 Sizing and capacity planning considerations	115
4.9 Migration considerations	116
Chapter 5. Sample implementation scenarios	119
5.1 Setup of the sample environment	121
5.1.1 PATH variable for NAS deployment	121
5.1.2 syslogd settings	121
5.2 Using NFS V4 as you did with NFS V3	122
5.3 How to unmount an exported NFS V4 file system	123
5.4 Setting up the NFS domain name	124
5.5 The pseudo-root FS	124
5.5.1 Setting up the pseudo-root FS on an NFS V4 server	125
5.5.2 Advantages of using the NFS V4 pseudo-root	128
5.5.3 Setting up the alias tree extension on an NFS V4 server	131

5.6	Setting up the NAS with a legacy database	134
5.6.1	Setup of a KDC server	135
5.6.2	Installing the IBM NAS file sets	135
5.6.3	Initial basic KDC functions test	138
5.6.4	Create user principals on the KDC server	139
5.6.5	Create the NFS server principals on the KDC server	141
5.7	Setting up an NFS V4 server with NAS on a different KDC server	142
5.7.1	Create the NFS server keytab file entry	142
5.7.2	Check the NFS V4 server before client access	143
5.7.3	Set up the NFS registry daemon	143
5.7.4	Set up the gssd daemon on the NFS V4 server	144
5.8	Setting up an NFS V4 client with NAS	145
5.8.1	General steps for all types of clients	145
5.8.2	Install the NAS client code	145
5.8.3	Set up the NFS domain	146
5.8.4	Set up the NFS domain-to-realm map	146
5.8.5	Full client installation steps	147
5.8.6	Slim client installation steps	150
5.8.7	Configuring RPCSEC_GSS on the clients	154
5.9	Preparing the system for Tivoli Directory Server and Kerberos V5	155
5.9.1	Set up procedure	156
5.9.2	Configure IBM Tivoli Directory Server	160
5.9.3	Configure the KDC server with LDAP backend	165
5.9.4	Configure the NFS V4 client for integrated login services	170
5.10	Integrating NFS V4 with a Linux client	176
5.10.1	NFS server and client setup	177
5.10.2	Read-only NFS V4 mount	182
5.10.3	Read/write NFS V4 mounts on Linux	185
5.10.4	Pseudo-file system in NFS V4 Linux client	187
5.11	Windows KDC and NFS V4 AIX 5.3	190
5.12	Setting up Kerberos cross-realm access	199
5.12.1	Add the krbtgt service principal to every KDC server	200
5.12.2	Kerberos configuration file changes on the KDC server, NFS V4 client and server	203
5.12.3	Add NFS domain-to-realm map on NFS V4 client and server	204
5.12.4	Client access verification	204
5.12.5	Client access mount using cross-realms	205
	Chapter 6. Problem determination	207
6.1	Problem determination tools and techniques	208
6.2	AIX problem determination tools and aids for NFS	208
6.2.1	Enabling syslogd	208
6.2.2	Using iptrace and ipreport	210

6.2.3	Using the fuser command	210
6.2.4	Using the rpcinfo command	210
6.2.5	Using the showmount command	211
6.2.6	Using the nfs4cl command	211
6.2.7	Using the nfsstat command	211
6.2.8	Using the errpt command	211
6.3	IBM NAS problem determination tools	211
6.4	Tivoli Directory Server problem determination tools	212
6.5	Third-party problem determination tools	212
6.5.1	Using the lsof command	212
6.5.2	Using the Ethereal utility	212
6.6	General NFS V4 problems	213
6.6.1	Warning: EIM is not configured	213
6.6.2	Realm is already mapped to domain	214
6.7	Exporting file systems	215
6.7.1	Exportfs: cannot change the v4 root.	215
6.7.2	Exportfs: /<path>: Invalid argument	215
6.7.3	Exportfs: /var/<logfile>: Too many levels of symbolic links.	217
6.8	Mount problems	218
6.8.1	General mount problem	218
6.8.2	Pseudo-root and nfs4cl problems	219
6.8.3	'vers' mount option error: "...Program not registered"	219
6.8.4	'vers' mount option error: "...server <name> not responding"	220
6.8.5	Mount command hangs - no system response	220
6.8.6	Mount with sec=krb5: "vmount: The file access permissions do not allow the specified action"	221
6.8.7	Mount with sec=krb5: "RPC: 1832-016 Unknown host..."	223
6.8.8	File and directory access: cd, ls, etc. return "permission denied"	226
6.8.9	File and directory access: file ownership is "nobody:nobody"	229
6.8.10	NAS problem: kadmin: "Unable to initialize kadmin interface"	231
6.9	GSS-API error codes	232
6.9.1	Major GSS-API error codes	232
6.9.2	Kerberos v5 status codes	234

Part 3. Appendices 241

Appendix A. Kerberos	243
Overview	244
Kerberos keys and initial setup	245
Authenticating to the Kerberos server	246
Authenticating to an application server	247
Kerberos terminology	251
Where to find more information about Kerberos	252

IBM Redbooks	252
Other IBM publications	252
Non-IBM publications	253
Other information sources	253
Appendix B. Sample scripts, files, and output	255
Sample administrative scripts	256
Change the pseudo-root FS sample script	256
Create a KDC server with NFS V4 server	256
Create a full client with legacy KDC server backend	258
Create a Full Client with KDC and LDAP backend	259
Script to copy ACLs to an entire directory structure	260
Windows command script to run ktpass	262
Script to gather additional information for local AIX software support.	263
Sample client Kerberos configuration files	268
Kerberos configuration file /etc/krb5/krb5.conf with legacy backend	268
Kerberos configuration file /etc/krb5/krb5.conf with LDAP backend	269
Kerberos configuration file /etc/krb5/krb5.conf with Windows Active Directory backend	269
LDIF sample file for KDC	270
Sample iptrace output	271
Successful authentication during mount request	271
Unsuccessful authentication during mount request	281
Appendix C. AIX 5.3 NFS quick reference	287
NFS configuration files	288
NFS daemons	290
NFS commands	291
Export options	293
mount command options	294
nfso command options	295
Abbreviations and acronyms	297
Glossary	299
Related publications	301
IBM Redbooks	301
Other publications	301
Online resources	302
How to get IBM Redbooks	302
Help from IBM	303
Index	305

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law. INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

AFS®

AIX®

AIX 5L™

Domino®

DB2®

DFS™

DYNIX/ptx®

IBM®

iSeries™

Lotus®

MQSeries®


Notes®

OS/2®

POWERparallel®

pSeries®

Redbooks™

Redbooks (logo) ™

RS/6000®

Tivoli®

WebSphere®

The following terms are trademarks of other companies:

Microsoft, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

Preface

Network File System Version 4 (NFS V4) is the latest defined client-to-server protocol for NFS. A significant upgrade of NFS V3, it was defined under the IETF framework by many contributors. NFS V4 introduces a major changes to the way NFS has been implemented and used up until now, including stronger security, wide area network sharing, and broader platform adaptability.

This IBM® Redbook is intended to provide a broad understanding of NFS V4 and specific AIX® NFS V4 implementation details. It discusses considerations for deployment of NFS V4, with a focus on exploiting the stronger security features of the new protocol.

In the initial implementation of NFS V4 in AIX 5.3, the most important functional differences are related to security. Chapter 3 and parts of the planning and implementation chapters in Part 2 cover this topic in detail.

Part 1 NFS V4 Fundamentals

Chapter 1. NFS Version 4 overview

This chapter provides a high-level discussion defining NFS, the evolution of the protocol, design motivations for NFS V4, features implemented in AIX 5.3, and an overview of planning and implementation design decisions.

Read this chapter for an introduction to Version 4 protocol design requirements. It includes an important section that introduces the planning and implementation design decisions necessary for successful deployment.

Chapter 2. What's new in NFS V4?

This chapter provides a more detailed analysis of the differences between NFS V4 and prior versions. Name space handling and psuedo file systems are discussed in detail. It also provides reference information demonstrating differences in supporting files and daemons.

Read the first part of the chapter for more detailed information on the NFS protocol design and evolution from Version 2 to Version 4. In later parts of the chapter, learn specific technical differences in the implementation of NFS V4.

Chapter 3. Enhanced security in NFS V4

This chapter provides a deep dive into the enhanced security features available in NFS V4.

Read this chapter for details on integration options to support identification, authentication, and authorization of users and hosts in an NFS V4 environment. Very important differences between standard UNIX® permissions and the new NFS V4 access control lists (ACLs) are described and demonstrated with sample scenarios. Discussion of ACL management and ACL inheritance strategies is also included.

Part 2 Implementing NFS V4

Chapter 4. Planning for NFS V4

This chapter analyzes the deployment planning challenges for NFS V4 and provides some decision support tools for the system architect responsible for this task.

Read this chapter to learn methods for planning a deployment. There are useful tools (decision tree flow charts, sample before and after migration scenarios) that should help the system designer to understand how to map the NFS V4 migration challenges to their own environment.

Chapter 5. Sample implementation scenarios

This chapter documents actual sample installation, implementation, and deployment tasks carried out to support the various NFS V4 configurations tested as part of the development process for this book.

Refer to this chapter for how-to examples showing NFS V4 setup. See the beginning of the chapter for a more detailed list of documented configurations.

Chapter 6. Problem determination

This chapter begins by describing some best-practice setup and configuration methods for capturing information in case you need to debug error conditions when deploying NFS V4. Also included are some actual error conditions and the information-gathering methods that led to solutions for those conditions.

Read the beginning of this chapter for useful tips on configuring NFSv4 test systems. The latter part of the chapter could help you find solutions to error conditions you may encounter.

Part 3 Appendices

Appendix A. Kerberos

This appendix provides detailed background information on the inner workings of the Kerberos authentication system.

Appendix B. Sample scripts, files, and output

This appendix contains sample administration scripts for automating NFS V4 setup and configuration, sample configuration files, and sample IPTrace output streams.

Appendix C. AIX 5.3 NFS quick reference

This appendix provides a quick reference to NFS. It includes comprehensive lists of NFS-related configuration files, system daemons, commands, exporting options, mount options, and nfsd options.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Austin Center.

Chris Almond is an ITSO Project Leader and IT Architect based in Austin, Texas. He has a total of 14 years of IT industry experience, including the past five with IBM. His experience includes UNIX/Linux® systems engineering, network engineering, Lotus® Domino®-based content management solutions, and WebSphere® Portal-based solution design.

Lutz Deneffle is a team leader at the PLM Technical Support Eastern Region Support Center in Mainz, Germany. He holds a Graduate Engineer degree in Fluid Dynamics. He has 16 years of experience in the Information Technology field and has worked at IBM for 15 years. His areas of expertise include solution implementations on RS/6000®, such as CATIA, Lotus Notes®/Domino, Tivoli®, DCE/DFS™, and Internet technologies. He is now responsible for the IBM internal infrastructure used by the PLM World Wide Technical Support.

Sridhar Murthy works for IBM as an IT Specialist in Dallas, Texas. He has 15 years of experience in the IT industry. He holds a Master's degree in Computer Science and a Bachelors' degree in Electrical Engineering. He is also an AIX Certified Advanced Technical Expert. His areas of expertise include solution implementations using Distributed Computing Environment (DCE), Distributed File System (DFS), Domino Servers, Tivoli Netview, and Network Authentication Service on AIX. He is currently evaluating methods for designing an architecture that will support migration of a 40,000+ user DEC/DFS environment to NFS V4.

Aniket Patel is a Team Leader at the IBM UK UNIX Support Centre. He has eight years of experience in UNIX and has worked at IBM for six years. Aniket graduated from Kingston University, UK, with a Bachelor of Science (Honours Degree) in Computer Science. His areas of expertise include UNIX Support on AIX, DYNIX/ptx®, and Linux, NFS, TCP/IP, SNA, X.25, DCE, Sendmail, MQSeries®, and the Microsoft® Windows® operating systems. Prior to recently

taking on the role of Team Leader, Aniket was responsible for a team of Network Specialists within the IBM UK UNIX Support Centre.

John Trindle works for Boeing in Seattle, Washington, as a systems design and integration specialist. He has worked with NFS on various UNIX systems (including SunOS/Solaris, DEC Ultrix, HP-UX, and most recently AIX) for 18 years. He holds a Bachelor of Science degree in General Engineering from the United States Military Academy, West Point, NY. His current areas of expertise include the AIX operating system and storage integration with AIX. He also designs and oversees several hierarchical storage management implementations using IBM Tivoli Storage Manager for Space Management.

Acknowledgements

The following people provided significant support for this project:

Carl Burnett, IBM AIX Kernel Architecture Team: for his thoughtful insights and overall guidance in helping us develop a content strategy for this book.

Ufuk Celikka, IBM AIX Security Development Team: for technical support during NFS V4 testing.

Bill McAllister, Team Leader IBM UK UNIX Support Centre: for his support in reviewing draft content and providing feedback during the project.

Brian L. McCorkle, IBM AIX NFS Development Team: for his continued patience and assistance while we tried to understand the implementation of NFS V4 on AIX. Brian's support enabled the team to meet its content goals for the book.

Dave Sheffield, Team Leader IBM AIX NFS Development Team: for facilitating access to Development Team resources whenever we needed it.

Steve Sipocz, IBM BCS Systems Integration Services: for technical support and review of our integration scenarios.

Betsy Thaggard
ITSO Editor, Austin Center

And other members of the AIX NFS Development Team and the AIX NAS Development Team, for their support on various technical issues.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners, and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us because we want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- ▶ Use the online **Contact us** review redbook form found at:

ibm.com/redbooks

- ▶ Send your comments in an e-mail to:

redbook@us.ibm.com

- ▶ Mail your comments to:

IBM Corporation, International Technical Support Organization
Dept. JN9B Building 905 3D004
11501 Burnet Road
Austin, Texas 78758-3493



Part 1

NFS V4 fundamentals



NFS Version 4 overview

The purpose of this chapter is to give a general overview of the Network File System (NFS) protocol, from NFS V1 to NFS V4.

We give you an introduction to the NFS protocol, beginning with a look at how NFS works. We then move on to a brief history of NFS followed by NFS V4. We also introduce you to some of the AIX 5.3 implementation specifics of NFS V4.

The following topics are discussed in this chapter:

- ▶ What is NFS?
- ▶ NFS V2 and NFS V3 history
- ▶ NFS V4 design motivations
- ▶ Objectives of NFS V4 (RFC3530)
- ▶ AIX 5.3 specific implementation of NFS V4
- ▶ Planning and implementation considerations

1.1 What is NFS?

In many environments, we would like to share files and programs among workstations in a local area network (LAN). Doing so requires programs that let us share these files, create new ones, carry out file locking, and manage ownership correctly.

Over the past few years there have been developments in multiple types of network capable file systems. These include the Apollo Domain, the Andrew File System (AFS®), the AT&T Remote File System (RFS), IBM DFS, and Sun Microsystems' Network File System (NFS). Each of these has had beneficial features and limiting drawbacks.

Of all the network file systems, NFS is probably the most widely used. NFS is available on almost all versions of UNIX, as well as on Apple Macintosh systems, MS-DOS, Windows, OS/2®, and VMS. It has continued to mature, and the latest revision, Version 4, will help to advance and expand its reach.

So, what is NFS? NFS is a distributed file system that enables users to access files and directories on remote servers as if they were local. For example, the user can use operating system commands to create, remove, read, write, and set file attributes for remote files and directories. NFS is independent of machine types, operating systems, and network architectures through the use of *Remote Procedure Calls (RPCs)*.

NFS operates on a client/server basis. An NFS server has files on a local disk, which are accessed through NFS on a client machine. To handle these operations, NFS consists of:

- ▶ Networking protocols
- ▶ Client and server daemons
- ▶ Kernel extensions

The kernel extensions are outside the scope of this book, but the protocols, daemons, planning, and implementation of NFS V4 will be discussed.

1.2 NFS V2 and NFS V3 History

NFS was first introduced by Sun Microsystems in the early 1980s.

NFS V1 was Sun's prototype version and was never released for public use.

NFS V2 was released in 1985 with the SunOS V2 operating system. Many UNIX vendors licensed this version of NFS from Sun. NFS V2 suffered many undocumented and subtle changes throughout its 10-year life. Some vendors

allowed NFS V2 to read or write more than 4 K bytes at a time; others increased the number of groups provided as part of the RPC authentication from 8 to 16. These minor changes created occasional incompatibilities between different NFS implementations; however, the protocol continued to provide an exceptional degree of compatibility between systems made by different vendors.

The NFS V3 specification was developed during July 1992. Working code for NFS V3 was introduced by some vendors in 1995 and was made widely available in 1996. Version 3 incorporated many performance improvements over Version 2 but did not significantly change the way that NFS worked or the security model used by the network file system.

1.3 NFS V4 design motivations

Increasingly, businesses have needed to secure and protect data. Earlier versions of NFS had weaknesses that kept them from meeting these needs. The following list is an example of areas that NFS V2 and NFS V3 have failed to address:

- ▶ Strong authentication to prevent malicious users from masquerading as a valid user of the system
- ▶ Fine-grained access control to make sure only the right people have access to sensitive data
- ▶ Encrypting data traffic to protect it from unauthorized disclosure as it travels over the network
- ▶ Uniquely identifying users in a large organization
- ▶ Good system and file I/O performance, including access from remote locations
- ▶ Being able to access shared data from many different platforms
- ▶ The use of an open systems design

1.4 Objectives of NFS V4 (RFC3530)

NFS V4 was originally defined under RFC3010 in December 2000. In April 2003 RFC3010 was superseded by RFC3530 to address some of the issues related to ease of implementation or clarification. The NFS V4 implementation on AIX 5.3 is based on RFC3530.

The main objectives of the Version 4 protocol focus on addressing the shortcomings of previous versions as listed in 1.3, “NFS V4 design motivations” on page 5, while achieving the following overall goals:

- ▶ Improved performance over the network
- ▶ Enhanced by adding security schemes built into the protocol
- ▶ Integrated locking support
- ▶ Cross-platform interoperability, including the Microsoft Windows environment
- ▶ Protocol extension support in a manner that does not invalidate backward compatibility
- ▶ Movement toward an Open Standard, managed by the IETF, whereas previous versions of NFS were proprietary

For more detailed information about the new features of the protocol, there are many references available. Refer to “Related publications” on page 301. One particularly useful reference for a more detailed discussion of Version 4 protocol functional design goals is a white paper titled *The NFS Version 4 Protocol* by Brian Pawlowski, et al., which can be found at:

<http://www.nluug.nl/events/sane2000/papers/pawlowski.pdf>

1.5 AIX 5.3 specific implementation of NFS V4

AIX 5.3 supports the latest NFS protocol, Version 4 (NFS V4). AIX also provides support for NFS Version 3 (NFS V3) and NFS Version 2 (NFS V2) client/server. It therefore provides backward compatibility with the existing install base of NFS clients and servers in a heterogeneous environment.

Vendor-specific implementation of RFC3530 into their product depends on several factors:

- ▶ Mandatory versus optional features described in the RFC
- ▶ Customer-specific requirements
- ▶ Derived business needs

The following features of the protocol are supported in AIX 5.3.

1.5.1 Mandatory features

The RFC has defined that the mandatory features are the minimal set of file or file system attributes that must be provided by the server and *must* be properly

represented by the server. All implementations of NFS V4, including AIX 5.3, must support these features to be compliant:

- ▶ GSS-API (RFC2078) with Kerberos V5 mechanism
- ▶ Identity mapping
- ▶ Pseudo-file system model

1.5.2 Optional features

The RFC specification also defines optional features. The term optional is used here, meaning that the RFC gives the vendor the option to implement the feature or not. AIX 5.3 implements the following optional features:

- ▶ NFS V4 Access Control Lists (ACL)
- ▶ Delegation
- ▶ Mapping of principals and file ownership attributes from one NFS V4 domain into another
- ▶ Data replication and migration

1.6 Planning and implementation considerations

Planning and implementing NFS V4 is more complex than for previous NFS versions. Because NFS V4 introduces new advanced capabilities, advanced planning will make deployment less painful and ongoing management easier. Therefore, it is essential that you choose the elements of your infrastructure carefully before beginning your implementation. To help with the decisions you need to make, we have dedicated a chapter to planning and one for implementation considerations. We also discuss in detail what we see as common migration paths. These are discussed in Chapter 5, “Sample implementation scenarios” on page 119.

1.6.1 Pre-implementation design considerations

1. How will you manage user identities? The options to choose from include:
 - a. LDAP [RFC2307]
 - b. NIS
 - c. Standard UNIX password and group files
2. Will you need to accommodate user identities from other organizations?
3. How will you authenticate users? Options here include:
 - a. Kerberos

- b. Standard UNIX password authentication
- 4. How will you control access to directories and files? Options here include:
 - a. Standard UNIX permissions
 - b. NFS V4 access control lists (ACLs)
 - If you choose to use ACLs, how would you structure your directories to maximize use of the inheritance features?
- 5. How will you structure your shared file system name space? Think about how you want to present your NFS exported file systems to the clients. Additional considerations include:
 - Do you want to make the server location transparent to the users? For example, will you need to move subdirectories between servers without changing where they appear in the client directory tree?
 - Do you have a large, flat directory structure (like home directories) where using distributed structures would help?
 - Do you have special NFS export restrictions that dictate directory structure?
 - Do your file and directory access control methods work better with a particular directory structure? For example, do you want to structure access controls grouped by organization or by role?
- 6. Do you require high availability? Services that might need to be designed for high availability include:
 - a. Directory/Identity services
 - b. Authentication services
 - c. File services (the actual NFS servers themselves)
- 7. What is your network architecture? Will you need to place servers near users in separate geographical locations?
- 8. How dispersed is your data access?

1.7 Looking ahead to the rest of the book

The questions highlighted in the pre-implementation considerations (outlined in the previous section) are difficult to answer until a significant amount of planning, research, and preparation is completed. It is the goal of this book to help prepare the reader to make these design decisions.

The next two chapters provide technical background on the NFS V4 protocol as compared to Version 3 (Chapter 2), then a detailed discussion of the new file system security control features that Version 4 provides (Chapter 3).

These chapters provide technical background to prepare the reader for planning an NFS V4 deployment. Before you can proceed with any implementation, you have to plan it, so Chapter 4, “Planning for NFS V4” on page 93 is the key chapter in this book. The chapter walks you through a design decision process for planning your NFS V4 infrastructure.

After your planning is complete, we look at some scenarios and how to implement them. The scenarios demonstrate how you could get to the final step of having a running NFS V4 infrastructure. We consider migration paths from a previous version of AIX and therefore an older version of NFS to AIX 5.3 and NFS V4. We also consider migration paths from an older version of AIX running NFS V3 to AIX 5.3 running NFS V3. In total, we look at three different migration paths.



What's new in NFS V4?

This chapter reveals the differences between NFS V3 and NFS V4, but first we look at NFS in general and how it has evolved over the years. The aim is to give those who are new to NFS a basic grounding in the protocol. Those who already know NFS can skip directly to 2.6, “NFS V4” on page 20.

Continuing from the NFS design discussion, we talk about the differences between the AIX v5.3 NFS V4 implementation and what the NFS V4 RFC RFC3530 describes. The reader is advised to refer to the NFS V4 RFC for a detailed explanation of the protocol. Detailed scenarios and implementation options will be discussed in the planning and implementation chapters.

The following topics are covered:

- ▶ How NFS works
- ▶ Protocols used by NFS
- ▶ NFS daemons
- ▶ NFS V3
- ▶ The NFS Lock Manager protocol
- ▶ NFS V4
- ▶ AIX5L v5.3 implementation of NFS V4
- ▶ NFS V4 support on AIX 5.3
- ▶ NFS files and commands: a quick reference

2.1 How NFS works

A server has files on a local disk, irrespective of file system type, that are accessed through NFS on a client machine. NFS is an application layer protocol that uses other underlying protocols defined in the TCP/IP model. NFS is built on top of the TCP/IP protocol stack over the *Remote Procedure Call (RPC)* protocol.

To handle this operation, NFS consists of:

- ▶ Networking protocols
- ▶ Client and server daemons

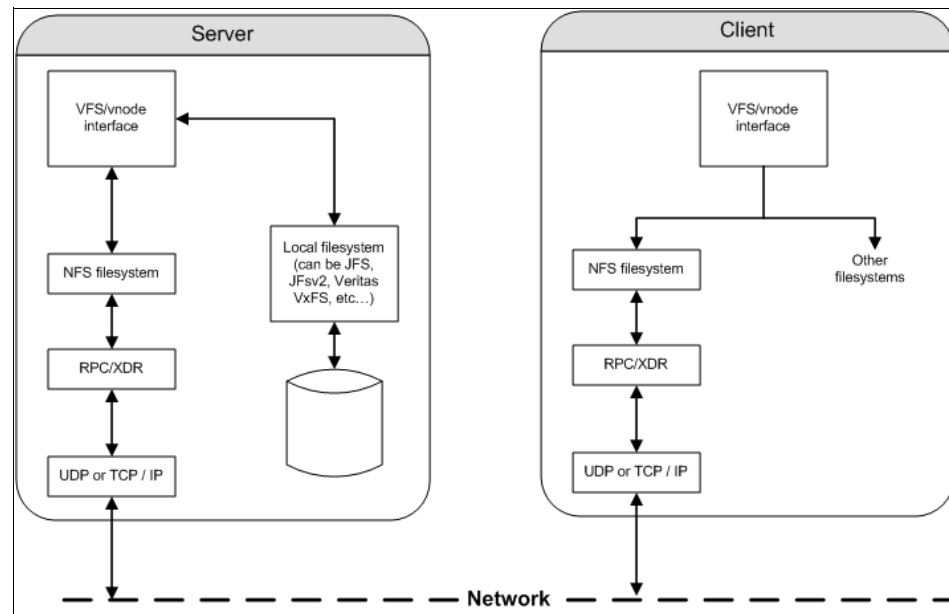


Figure 2-1 How NFS works

2.2 Protocols used by NFS

The NFS-specific protocols are Sun RPC and *External Data Representation (XDR)* protocol.

The *Open Systems Interconnection (OSI)* Reference Model is traditionally used as a general purpose reference for describing and comparing protocols. We assume that you are familiar with the OSI model. Figure 2-2 on page 13 attempts to show the components of the NFS protocol in relation to the OSI Reference Model. The mapping is only approximate and is intended as a conceptual guide.

7 Application	NFS	
6 Presentation		
5 Session	XDR	SunRPC
4 Transport	UDP	TCP
3 Network	IP	
2 Datalink	IEEE 802.2, etc,...	
1 Physical	Ethernet, Token Ring, etc,...	

Figure 2-2 Representing NFS with the seven-layer OSI model

2.2.1 UDP or TCP

Both NFS V2 and NFS V3 operate over the *User Datagram Protocol (UDP)* and the *Transmission Control Protocol (TCP)*. NFS V3 uses TCP by default. NFS V4 *only* uses TCP.

TCP is *stateful*, and NFS is *stateless*. A stateless server treats each request as an independent transaction, unrelated to any previous request. This simplifies the server design because it does not need to allocate storage to deal with conversations in progress or worry about freeing it if a client dies in mid-transaction. A disadvantage is that it may be necessary to include more information in each request, and this extra information will have to be interpreted by the server each time. UDP neither guarantees delivery nor does it require a connection. As a result, it is lightweight and efficient, but all error processing and retransmission must be taken care of by the application program.

A stateful server enables a host to send data in a continuous stream to another host. The transport service guarantees that all data will be delivered to the other end in the same order as sent and without duplication. Communication proceeds through three well-defined phases: connection establishment, data transfer, and connection release. TCP is a good example of this.

So, the statement that NFS is stateless would seem to be a contradiction and you would think it impossible for NFS to work over TCP. However, NFS and TCP

operate in a layer called Remote Procedure Call (RPC), and this hides the state issues of TCP from NFS.

2.2.2 Remote Procedure Call (RPC)

RPC is a library of procedures. These procedures enable one process (the client process) to direct another process (the server process) to execute procedure calls as though the client process had executed the calls in its own address space. Because the client and the server are two separate processes, they are not required to be on the same physical system, although they can be. The RPC call used is based on the file system action taken by the user. For example, when issuing an `ls -la` command on an NFS mounted directory, the long listing will be completed using RPCs `NFSPROC3_READDIR` or `NFSPROC3_READDIRPLUS`. The server in turn will send the output from the command through RPC back to the client. To the user, this transaction is totally transparent.

Note: The RPC described here is the Sun RPC and is not to be confused with the RPC used by products such as DCE. RPC services use TCP or UDP to transport data packets. NFS V4 only uses TCP as a transport protocol.

2.2.3 eXternal Data Representation (XDR)

Because the server and client processes can reside on two different physical systems, which may have completely different architectures, RPC must address the possibility that the two systems may not represent data in the same manner. Therefore, RPC uses data types defined by the XDR protocol. XDR is the specification for a standard representation of various data types. By using a standard data type representation, data can be interpreted correctly, even if the source of the data is a machine with a completely different architecture. XDR is used when the vnode points out that the accessed file or directory is not local, but resides on a remote system. A conversion of data into XDR format is needed before sending the data. Conversely, when it receives data, it converts it from XDR format into its own specific data type representation.

2.3 NFS daemons

A daemon is a process that runs continuously in the background and provides services to clients. Some of the daemons that enable NFS to work are:

- ▶ portmap
- ▶ nfsd
- ▶ rpc.mountd
- ▶ rpc.statd

- ▶ rpc.lockd
- ▶ biod

The rpc.mountd and nfsd daemons run only on the server, and the rpc.lockd and rpc.statd daemons run on the server and the client. The biod daemon runs only on the client.

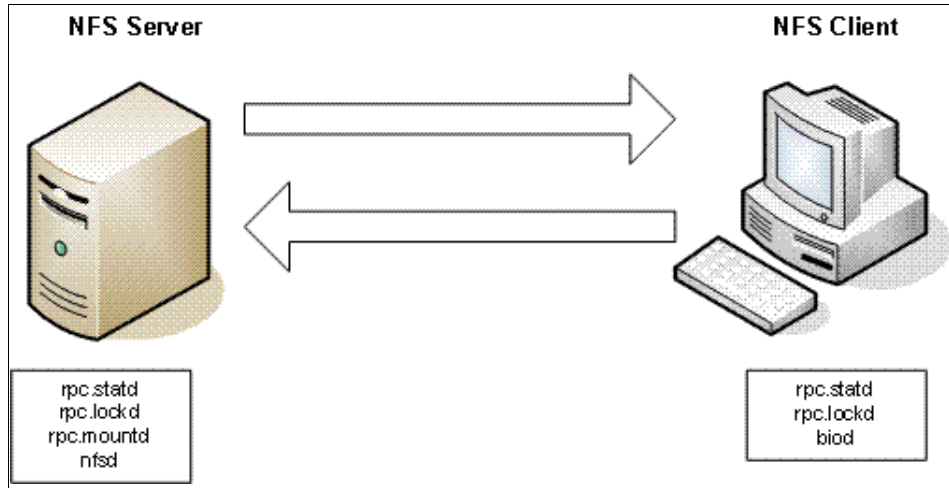


Figure 2-3 NFS client/server model with associated daemons

2.3.1 The portmap daemon

The portmap daemon converts RPC program numbers into Internet port numbers. When an RPC server starts up, it registers with the portmap daemon. The server tells the daemon which port number it is listening on and which RPC program numbers it serves. By this process, the portmap daemon knows the location of every registered port used by RPC servers on the host, and which programs are available on each of these ports. When mounting, the mount request starts with an RPC call named GETPORT that calls the portmap, which in turn informs the client of the port number that the called RPC server listens to. After this, the port number is used as reference for further communication. This is why the NFS daemons must be registered with the portmap daemon.

A client will only consult the portmap daemon once for each program the client tries to call. The portmap daemon tells the client which port to send the call to. The client stores this information for future reference. As standard RPC servers are normally started by the inetd daemon, the portmap daemon must be started before the inetd daemon is invoked.

2.3.2 The rpc.mountd daemon

The rpc.mountd daemon handles the actual mount service, which is needed when a client sends a mount request with an RPC procedure named MOUNTPROC3_MNT to the server. In addition, it provides a list of currently mounted file systems and the clients on which they are mounted.

2.3.3 The rpc.statd daemon

The rpc.statd daemon interacts with the rpc.lockd daemon to provide crash and recovery functions for the locking services on NFS.

Important: The rpc.statd daemon should always be started before the lockd daemon.

2.3.4 The rpc.lockd daemon

The rpc.lockd daemon processes lock requests that are either sent locally by the kernel or remotely by another lock daemon. The rpc.lockd daemon forwards lock requests for remote data to the server site lock daemon through the RPC package. The rpc.lockd daemon then asks the rpc.statd (status monitor) daemon for monitor service. The reply to the lock request is not sent to the kernel until both the statd daemon and the server site lockd daemon reply.

2.3.5 The nfsd daemon

The nfsd daemon runs on a server and manages certain aspects of the server configuration. The actual processing of client requests is handled by a multi-threaded kernel process rather than the nfsd daemon itself. Each daemon handles one request at a time. This means that on the server side, the receipt of any one NFS protocol request from a client requires the dedicated attention of an nfsd daemon until that request is satisfied, and the results of the request processing are sent back to the client. The nfsd daemons are the active agents providing NFS services.

The NFS daemons are inactive if there are no NFS requests to handle. When the NFS server receives RPC calls on the nfsd daemon's receive socket, the daemon is awakened to pick up the packet off the socket and invoke the requested operations.

2.3.6 The block I/O daemon (biobd)

The block I/O daemon (biobd) runs on all NFS client systems. When a user on a client wants to read or write to a file on a server, the biobd daemon sends this request to the server. For each read or write request, one biobd is requested. The biobd daemon is activated during system startup and runs continuously. The number of biobds are limited on a per-mount-point basis.

The actual work of reading and writing to a file on a server is handled by a multi-threaded kernel process with one thread assigned for each read or write request. The number of these threads changes dynamically depending on demand, but the biobd daemon controls the maximum number of these threads per mount.

2.4 NFS V3

Even though NFS V2 (discussed briefly in the introduction) was widely accepted, the protocol was not without its problems. This led to the introduction of NFS V3 RFC1813. The Version 2 protocol had the following major problems:

- ▶ A file size limit of 4 GB. Only files of up to 4 GB in size could be accessed. As computer environments began to grow and the need for larger data repositories became apparent, this limitation became a problem.
- ▶ Writes had to be synchronous. This led to write-intensive applications suffering performance problems. There were quite a few workarounds for this problem, but most violated the NFS V2 standard.

The main goals for the NFS V3 design were to solve these two problems. In 1994, Brian Pawlowski published a paper (referenced at end of this section) that provided an overview of the process the designers of the Version 3 protocol went through. This paper identified the following major areas where the protocol was enhanced:

- ▶ The 4 GB file restriction. All arguments within the protocol (such as file sizes) were changed from 32-bit to 64-bit.
- ▶ The write model was changed to introduce a write/commit phase. This enabled asynchronous writes.
- ▶ A new ACCESS procedure was introduced. This resolved permission-checking problems when mapping the ID of the superuser. This procedure is also important for correct behavior when ACLs exist on files.
- ▶ The 8 Kb write per procedure call limit was relaxed.
- ▶ The REaddirPLUS procedure was introduced. This returned both a file handle and attributes. This returns both directory entries and per-entry

attributes. REaddirPLUS is intended to reduce subsequent NFS LOOKUP calls for each directory entry.

- ▶ The file handle size was changed to a variable size, up to a maximum of 64 bytes. NFS V2 had a fixed file handle size of 32 bytes.
- ▶ The CREATE procedure was modified. This allowed for exclusive file creates.
- ▶ Filenames and pathnames were limited to 255 and 1024 characters respectively in NFS V2. This was changed in NFS V3 by variable length strings agreed on between the client and server.
- ▶ NFS V3 tightened the errors that could be returned from the server. All error values are iterated, and no errors outside the list are permitted.
- ▶ The blocksize field was removed. The blocks field was changed to *used* and recorded the total number of bytes used by the file.
- ▶ The NFS3ERR_JUKEBOX error type was introduced. This allowed for situations when a request is made to the server to read a file migrated to tape. Obviously, the time to read the data back from the tape will be considerable. This new error informs the client that the operation is in progress and the call should be retried.

Forcing writes from asynchronous to synchronous mode will affect performance. With NFS V3, the client can send a number of asynchronous WRITE requests that it can then commit to disk on the server at a later date by issuing a COMMIT request. After the server has received the COMMIT request, it cannot return until all data has been flushed to disk. The COMMIT request is similar to calling `fsync()`. The major difference is that the COMMIT request does not necessarily cover the data for the whole file. However, it does allow the client to flush all the data when a file is closed or to break up a large synchronous write request into smaller writes. These are performed asynchronously, but followed by a COMMIT request. This is an important enhancement, as it enables the file system on the server to coalesce a number of write requests into a single large write, making it more efficient. As the client is required to keep a copy of all data to be written to the file until a COMMIT is issued, asynchronous writes should not affect the crash/recovery properties of NFS.

The REaddirPLUS procedure, even though it is efficient, also presents problems. The procedure was introduced to minimize the number of over-the-wire LOOKUP requests when a REaddir procedure had been invoked. An example of this would be an `ls -F` request on a directory.

The implementation of REaddirPLUS is significantly more expensive than REaddir. The procedure should only be performed when first accessing the directory in order to populate the name cache, depending on the underlying operating system. It should then only be performed again in cases where the cache was invalidated for the directory due to a directory modification.

Many of the goals of NFS V3 were to improve performance. A number of different performance-related tests showed that NFS V3 did indeed meet its objectives very well, as documented in Pawlowski's 1994 white paper *NFS Version 3 Design and Implementation*, by Brian Pawlowski et al, found at:

<http://citeseer.ist.psu.edu/pawlowski94nfs.html>

2.5 The NFS Lock Manager protocol

File locking was omitted during the design of the NFS protocol. The main reason is that to support record locking, state would have to be maintained on the server. This would dramatically increase the complexity of NFS implementations.

File locking was not something that could be overlooked easily and was therefore implemented in SunOS as the *Network Lock Manager (NLM)*. NLM went through various iterations, with Version 3 being most widely used with NFS V2. With the introduction of NFS V3, the definition of NLM (Version 4) was included in the NFS specifications, but was still left as a separate protocol. The NLM also relied on the *Network Status Monitor* protocol. This was required so the clients and servers could be notified of a crash so that a lock state could be recovered.

Crash/recovery involves coordination between both the clients and server:

- | | |
|---------------------|---|
| Server crash | When locks are handed to clients, the server maintains a list of clients and the locks that they own. If the server were then to crash, these locks would then be lost. When the server reboots, a status monitor runs and sends a message to all known clients. The lock manager on each client is notified and given an opportunity to reclaim all locks it owns for the files on the server. There is a fixed amount of time in which the clients can respond. |
| Client crash | If the client crashes, any locks that the client holds on the server must be cleaned up. When the client recovers, it sends a message to the server to clean up its locks. Through the use of a client state number, which is incremented on reboots, the server can detect that the client has been rebooted and removes any locks that were held by the client before it crashed and rebooted. |

As stated earlier, the NLM was not widely adopted. The NFS protocol in Version 4 has been extended to include a file locking mechanism. You can find more information in 2.6.7, "File locking" on page 29.

2.6 NFS V4

NFS V4 introduces many substantial changes to the protocol. For example, the design of NFS predates the widespread adoption of the World Wide Web. It was originally designed for use in local area networks. With the advent of internetworking, there has been a greater need to use distributed file systems, such as NFS, in wide area networks. When NFS is used for distributed file systems across wide area networks, the security capabilities in pre-Version 4 implementations become apparent.

Important: Addressing existing security limitations in NFS is the most significant area of change introduced by Version 4. This redbook devotes an entire chapter to this topic: Chapter 3, “Enhanced security in NFS V4” on page 45.

NFS V4 goes a long way to overcome the shortcomings of NFS V2 and V3, and adds additional features left out in the NFS V3 implementation. NFS V4 is described in detail in RFC3530. The main changes that are discussed in this section are:

- ▶ Attribute classes
- ▶ User name to UID mapping
- ▶ Better namespace handling (pseudo-file systems)
- ▶ Built-in security
- ▶ Client-side caching and delegation
- ▶ Compound procedures
- ▶ File locking
- ▶ Internationalization
- ▶ Volatile file handles

2.6.1 Attribute classes

The set of attributes that were passed over-the-wire with earlier versions of NFS were very UNIX-oriented. This meant that the information returned by the server was sufficient to respond to a *stat()* call on the client. This made it difficult for non-UNIX systems to understand the protocol properly. To address this issue, NFS V4 introduces a new set of file attributes in three different classes:

- ▶ Mandatory
- ▶ Recommended
- ▶ Named

Mandatory attributes

Mandatory attributes are the minimal set of file or file system attributes that must be provided by the server and must be properly represented by the server. They must also be supported by every implementation. The mandatory set of attributes contain information such as the file type and size, information about the file handle expiration times, whether hard links and symbolic links are supported, and whether the file has named data streams and attributes. These are summarized in Table 2-1.

Table 2-1 Mandatory attributes and their description

Mandatory attribute	Description
Object type	The type of object (file, directory, symbolic link).
File handle expiration	Specifies file handle expiration behavior on the client.
Change indicator	The value created by the server that the client can use to determine if file data, directory contents, or attributes of a given object have been modified. The server may return the object's modify time for this attribute's value, but only if the file system object cannot be updated more frequently than the resolution of the modify time.
fsid	A unique file system identifier for the file system holding a given object. Contains major and minor components, each of which are unsigned 64-bit integers.
Lease duration	The time frame in which the leases at the server side end, in seconds.
Size	The size of a given object, in bytes.
UNIX LINK Support	Determines whether UNIX hard links are supported.
UNIX SYMLINK support	Determines whether UNIX symbolic links are supported.

Recommended attributes

The recommended attributes contain information such as the type of ACLs that the file system supports, the ACLs themselves, information about the owner and group, access timestamps, and quota attributes. They also contain information about the file system such as free space, total number of files, files available for use, and the file system limits such as maximum filename length and maximum number of links. These are summarized in Table 2-2 on page 22.

Table 2-2 Recommended attributes and their descriptions

Recommended attributes	Description
ACL	The Access Control List (ACL) for a given object.
Archive bit	Checks to see whether the file has been archived since the last time it was modified.
Case insensitive	Checks whether file names on a given file system are case-insensitive.
Case preservation	Checks whether the file name case in a given file system is preserved.
Change owner restricted	If this attribute is set to TRUE, the server will reject any request to change the owner or group associated with a given file if the caller is not a privileged user (for example, the root user in UNIX operating systems).
No file name truncation beyond maximum	Checks to ensure that either an error is returned or the name is truncated when the maximum file name size supported for a given object is exceeded,
File handle	An opaque data structure provided by the server to the client in response to a lookup request.
File ID	This attribute is a number uniquely identifying a file within a given file system.
Hidden	Checks to see whether a file is considered hidden with respect to the WIN32 API.
Maximum file size	The maximum file size supported for a file system for a given object.
Maximum number of links	The maximum number of links for a given object.
Maximum file name size	The maximum file name size supported for a given object.
Maximum read size	The maximum read size supported for a given object.
Maximum write size	The maximum write size supported for a given object. It should be supported if the file is writable. Lack of this attribute can lead to the client either wasting bandwidth or receiving poor performance.
MIME type	The MIME body type for a given object.

Recommended attributes	Description
UNIX mode bits	The UNIX-style permission bits for a given object.
Owner string	The string name of the owner for a given object.
Group string	The string name for the group ownership of a given object.
Modify time	The time of the last modification of a given object.
Create time	The creation time of a given object. This attribute does NOT have any relation to the traditional UNIX file attribute ctime or change time.
Access time	The time of last access for a given object.
Space available to user	The disk space, in bytes, available to a user on the file system containing a given object.
File system free space	The free disk space, in bytes, on the file system containing a given object.
File system total space	The total disk space, in bytes, on the file system containing a given object.
Space used by object	The number of file system bytes used by a given object.

Named attributes

Named attributes provide a mechanism to associate additional properties with a filesystem object (such as file or directory). A named attribute is an uninterpreted opaque stream of bytes with a name. Applications can use named attributes to place auxiliary application specific data on files. Multiple named attributes can exist on an object. The OPENATTR procedure is used to access named attributes for an object. NFS V4 organizes named attributes as a directory of attribute names. The REaddir and LOOKUP operations are used to obtain the attribute names. The READ, WRITE, and CREATE operations are then used to operate on the individual named attributes.

A client would access named attributes in the following way:

- ▶ The OPENATTR procedure sets the current filehandle to the named file attribute directory for the file object.
- ▶ The REaddir and REaddirPLUS procedures retrieve the file handles for the various named attributes associated with the original file system object.

Note: Named attributes are an optional protocol feature. Both the server and client must support named attributes for them to be used.

2.6.2 Username to UID mapping

With the Version 3 protocol and the commonly used AUTH_SYS RPC flavor, NFS requests contain a set of user credentials with a user ID (UID) and a list of group IDs (GIDs) to which the UID belongs. On the NFS server, these credentials are used to perform the permission checks that are part of UNIX file system access control (for example, verifying write permission to remove a file, or execute permission to search directories.) Additionally, filesystem object user and group ownership information is transferred as numeric integer values. Using integers to represent users and groups requires every client and server that would connect to each other to agree on user and group ID assignments. This UID to name mapping occurs only on the client.

NFS V4 departs from this model of identity representation. When transferring ownership information, it represents users and groups as strings in the following form:

```
user@nfs_domain
```

or

```
group@nfs_domain
```

Additionally, it is expected that the stronger RPCSEC-GSS RPC security flavor will be largely deployed with NFS V4. With RPCSEC-GSS, an opaque security token flows on the wire. From it, the receiver (NFS server) derives a string representation (principal) of the accessing identity. The server transforms the principal string into its native credentials. On UNIX systems, this results in a UID and corresponding list of member GIDs.

String-based identities require NFS clients and servers to translate between the protocol string attributes and the internal formats (UID and GID) that are used within the operating system.

Representing users and groups as strings provides added flexibility and removes the requirement that all systems utilize the same numeric ID space. It is expected that all systems share a common view of the user and group name space within a given NFS domain. The use of strings also opens the potential capability for interdomain NFS sharing when the capability exists to map an identity from a foreign domain into the receiver's local domain.

Important: For UID translation, NFS V4 requires that user and group names are consistent between the client and server to avoid potential access errors to secured data. However, the numeric UID space between systems does not have to match. We look at how this works in practice in 3.2.2, “User/group identities and NFS V4” on page 50.

2.6.3 Better namespace handling

NFS V2 and NFS V3 servers export a set of independent parts of their overall namespace and do not allow NFS clients to cross mountpoints on the server. This is because NFS expects all lookups to stay within a single file system. In NFS V4, the server provides a single root file handle through which clients can obtain file handles for any of the accessible exports.

NFS V4 no longer has a separate *mount* protocol. Instead of exporting a number of distinct exports, an NFS V4 client sees the NFS V4 server's exports as existing inside a single file tree, called the *nfsv4 pseudo-file system*. The pseudo-file system tree constructed by the server creates a single logical view of all the different exported file systems (Figure 2-4).

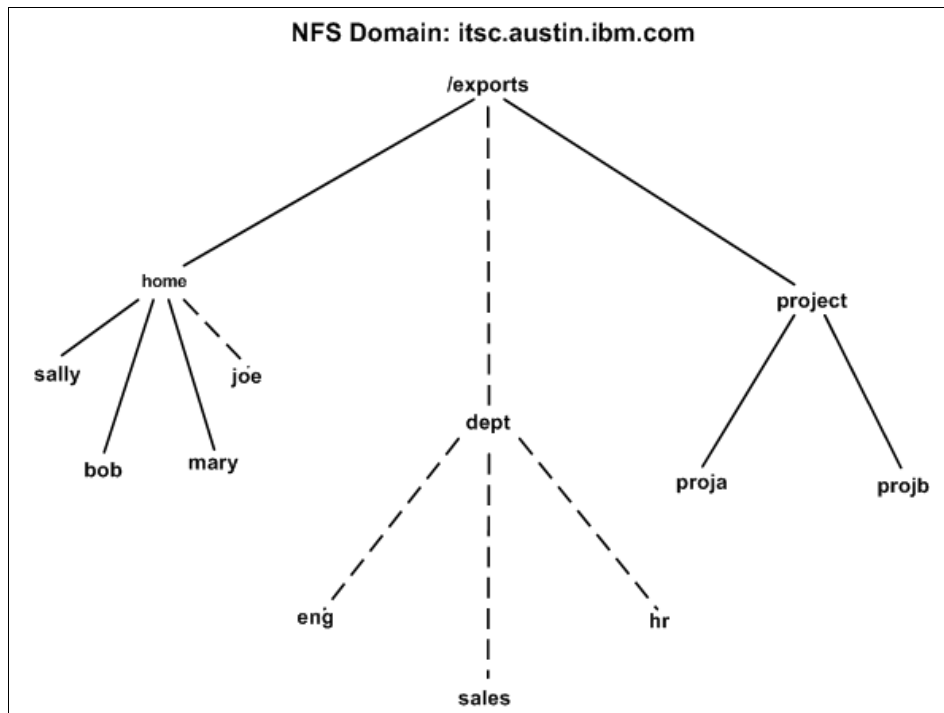


Figure 2-4 Pseudo-file systems: server view

In Figure 2-4 on page 25, we would like to export the following directories:

- ▶ /exports/home/sally
- ▶ /exports/home/bob
- ▶ /exports/home/mary
- ▶ /exports/project/proja
- ▶ /exports/project/projb

On the server, you must:

1. Set the pseudo-root node; in our case, we set this to /exports
2. Add the directories to export to the /etc/exports file:

```
/exports/home/sally -vers=4,ro
(exports/home/bob -vers=4,ro
(exports/home/mary -vers=4,ro
(exports/project -vers=4,ro
(exports/project/projA -vers=4,ro
(exports/project/projB -vers=4,ro
```

3. Run:

```
exportfs -va
```

4. On the client, mount the root export:

```
mount -o vers=4 <nfsv4_svr_name>:/ /<local_mount_point>
```

The contents of the newly mounted file system appear in Example 2-1 and Figure 2-5 on page 27.

Example 2-1 Client view of the pseudo-file system

```
# ls -al /nfs/*
/nfs/home:
total 26
drwxr-xr-x  4 root    system    5 Jul 28 11:26 .
drwxr-xr-x  3 root    system    4 Jul 28 11:26 ..
drwxr-xr-x  2 root    system   512 Jul 28 11:24 bob
drwxr-xr-x  2 root    system   512 Jul 28 11:25 mary
drwxr-xr-x  2 root    system   512 Jul 28 11:24 sally
/nfs/project:
total 18
drwxr-xr-x  3 root    system    4 Jul 28 11:26 .
drwxr-xr-x  3 root    system    4 Jul 28 11:26 ..
drwxr-xr-x  2 root    system   512 Jul 28 10:40 proja
drwxr-xr-x  2 root    system   512 Jul 28 10:40 projb
```

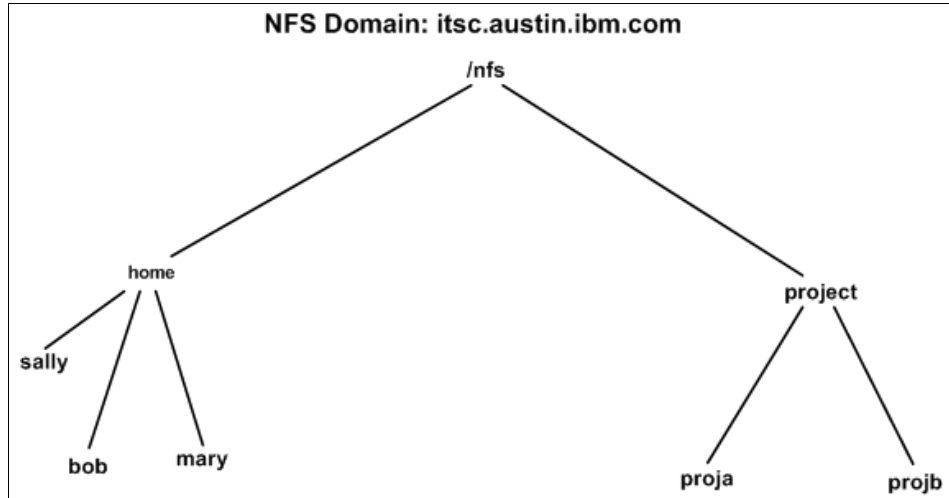


Figure 2-5 Pseudo-file systems: client view

The server has provided the client with a single view of the exported file systems. In NFS V4, a server's named space is a single hierarchy. In the example above, the export list hierarchy is not connected. When a server chooses to export a disconnected portion of its name space, the server creates a pseudo-file system to bridge the unexported portions of the name space, allowing a client to reach the export points from the single common root. A pseudo-file system is a structure containing only directories, created by the server having a unique fsid, that enables a client to browse the hierarchy of the exported file systems.

The client view of the pseudo-file system is limited to paths that lead to the exported file systems. Because `/home/joe` and `/dept` are not exported in the example, they do not appear on the client during browsing operations (Figure 2-5).

2.6.4 Built-in security

NFS has always relied on client-side authentication to provide security. This has generally not been a problem because NFS has largely been used within private networks. One of the objectives of the Version 4 protocol is to enable increased use of NFS to wide area networks.

The basic NFS security mechanisms are extended in NFS V4 through the mandated support of the RPCSEC_GSS RPC security flavor. RPCSEC_GSS is implemented at the RPC layer. It is capable of supporting different security mechanisms. Examples include Kerberos Version 5, and public-key-based mechanisms such as SPKM. NFS V4 requires that RPCSEC-GSS be provided

as an available RPC security flavor. It mandates that the Kerberos Version 5, SPKM, and LIPKEY security mechanisms be supported for full protocol compliance. It still allows the support and use of other RPC security flavors such as AUTH_SYS. A key weakness of the AUTH_SYS flavor has always been the ease with which hackers can forge and impersonate credentials.

RPCSEC_GSS is different from AUTH_SYS in two ways:

- ▶ RPCSEC_GSS goes beyond authentication to perform integrity checksums and encryption of the entire body of the RPC request and response.
- ▶ As RPCSEC_GSS encapsulates the GSS-API messaging tokens, it acts as a transport for mechanism-specific tokens for security flavors such as Kerberos. Adding new security mechanisms does not require rewriting significant portions of NFS or any other ONCRPC-based application.

All versions of NFS are capable of using RPCSEC_GSS. The difference is that while an implementation can claim to conform to NFS V2 and NFS V3 without implementing support for RPCSEC_GSS, a conforming implementation (one that claims to be based on RFC3530) of NFS V4 *must* implement security based on Kerberos Version 5 (as done in AIX 5.3) and LIPKEY. Kerberos V5 (KRB5) and LIPKEY are GSS-API conforming mechanisms.

Kerberos divides user communities into realms. Each realm has an administrator responsible for maintaining a database of principals or users, one master Key Distribution Center (KDC), and one or more slave KDCs that give user tickets to access services on specific hosts in a realm. Users in one realm can access services in another realm via trust relationships between the KDCs. Kerberos is a very good choice for enterprise work groups operating within an intranet. It provides centralized control, as well as single sign-on to the network. We will discuss Kerberos further in upcoming chapters.

The *Low Infrastructure Public Key (LIPKEY)* system provides an SSL-like model and is equivalent security for use on the Internet. LIPKEY is a GSS-API security mechanism that uses a symmetric key cipher and server-side public key certificates. LIPKEY will not be discussed in any more detail as the NFS V4 implementation in the initial release of AIX 5L™ v5.3 does not support it.

2.6.5 Client-side caching and delegation

Most NFS client implementations do caching of both data and attributes to improve performance and reduce network traffic. Some vendors also support a technology known as *cacheefs* that allows extended caching of file and directory content data to persistent storage (disk) on the client system. This further increases the amount of data a client can cache.

With caching, some amount of server interaction is still required to maintain the required semantics of the NFS protocol. Clients must check with servers and open time to validate and flush cached information as appropriate. In addition, the client periodically polls the server while files are in use. Depending on the application environment, the network traffic associated with client cache maintenance can be modest. In less reliable or slower networks, this traffic can represent a performance restriction.

NFS V4 provides an optional protocol mechanism called delegation that can improve the caching of NFS. With delegations, the open time network traffic can be avoided as well as the periodic checks to servers. The reduction in network traffic can help increase the performance and scale of an NFS environment.

NFS V4, like its predecessors, has a weak cache consistency model. Clients are not guaranteed to see the most recent changes to data at a server. Delegations are optional and are granted at the NFS server's discretion. Without a delegation, the NFS V4 client operates similar to previous versions of NFS.

2.6.6 Compound RPC procedures

Many file-related operations over NFS require a large number of RPC calls. In a local area network this is not an issue; however, when operating in a wide area network, the effect on performance can be much more noticeable. NFS V4 introduces a compound RPC model where it is possible to combine multiple protocol operations into a single RPC request. This creates potential to more efficiently utilize network resources by reducing the total number of RPC transactions for a given workload.

2.6.7 File locking

We have already talked about the fact that file locking is not an integrated part of NFS V2 and NFS V3. NFS V4 provides both UNIX-level file locking functions and Windows-based share locking functions. This gives NFS V4 better interoperability in heterogeneous environments. File locking in its simplest form is the ability to block I/O operations by other applications on a file that contains a record lock. Methods for managing state ID facilitates this operation.

A *stateid* is a unique 64-bit object that defines the locking state of a specific file. When a client requests a lock, it presents a clientid and unique-per-client lock owner identification to identify the lock owner.

When a client first contacts a server, it presents an opaque structure identifying itself to the server. The opaque structure uniquely identifies a particular client. After the server receives the client's identifying data, it returns a 64-bit *clientid*. The clientid is unique and will not conflict with those previously generated.

2.6.8 Internationalization

Previous versions of the NFS protocol handled file names as an opaque byte stream. They were limited to a 7-bit US ASCII representation, but were commonly encoded in 8-bit ISO-Latin-1. There was no way to specify the type of encoding in the XDR. This limited the use of NFS in environments where there may be mixed character sets. In order to provide better support for internationalization, filesystem object names (such as files and directories) are encoded as *UTF-8* in NFS V4.

2.6.9 Volatile file handles

In previous versions of NFS, a *file handle* is a per-server unique identifier for a file system object that is opaque to the client. It is defined as having a value that is fixed for the lifetime of a file system object to which it refers. For example, a file handle on UNIX contains the inode number and the generation count. As the inodes are freed and reallocated, the generation count of the inode is incremented when reused. This ensures that a client file handle that refers to the old file cannot now be referred to the new file, even though the inode number stays the same.

NFS V4 divides file handles into two types: persistent and volatile. Persistent file handles describe the traditional file handle. Volatile file handles is a concept introduced in NFS V4, in which a client must cache the mapping between path name and file handle, and regenerate the file handle upon expiration.

2.7 AIX 5L v5.3 implementation of NFS V4

In this section, we discuss the parts of the NFS V4 protocol specification that IBM has chosen to implement. AIX 5.3 is the first version of AIX to introduce support for NFS V4. It is very important to note that it continues to support NFS V2 and NFS V3. In fact, in AIX 5.3, Version 3 is still the default NFS protocol version that is used in server exports and client mounts. This decision was primarily made to allow for an easier migration to AIX 5.3 from previous versions. As you already know, NFS V4 introduces some major changes, so it would not make sense to make Version 4 the default version. The *vers* option may be used with mounts and exports to specify NFS Version 4.

Note: In AIX 5.3, the default for exports is still NFS V3, not Version 4. You must explicitly declare an export for NFS V4 using the *vers* option. See Example 2-1 on page 26.

The initial AIX support places an emphasis on security, with support of the optional NFS V4 ACL model when using the AIX Enhanced Journaled File System. Support for managing access from foreign NFS V4 domains is also included. NFS V4 provides the RPCSEC_GSS RPC authentication flavor supporting the Kerberos 5 security mechanism with AIX 5.3. RPCSEC_GSS can also be used with the NFS V3 protocol.

Note: The AIX Enhanced Journaled File System is a JFS2 file system with the Extended Attributes Version 2 capability enabled to use NFS V4 ACLs.

When you first look at NFS on AIX V5.3, you will not see anything different. NFS on AIX V5.3 supports NFS V2, NFS V3, and NFS V4. For this reason, all of the daemons that you would see in previous versions of AIX are included in AIX 5.3. One side effect of incorporating NFS V4 into AIX 5.3 is that NFS V3 security controls can be extended by using some of the security features that are required to create a V4 conforming implementation.

2.8 NFS V4 supported features in AIX 5.3

This section offers detailed descriptions of various features of NFS V4 that are implemented in AIX 5L Version 5.3.

2.8.1 Mandatory feature support

The mandatory features of the NFS V4 protocol are supported as described in RFC3530, *with the following exceptions*:

- ▶ The UTF-8 requirements are not fully supported.

Specifically, the transmission of file names and file system strings such as symbolic link contents and directory entry names are not guaranteed to be in UTF-8 format. Transmission of NFS attribute strings, such as owner and owner group, are always in UTF-8 format. The NFS server and client do perform UTF-8 validation on incoming string data as defined in RFC3530. This checking may be administratively disabled using the `nfso` command. Disabling UTF-8 checking may be necessary to use NFS V4 in environments with non-UTF-8 configurations and data.

- ▶ The LIPKEY and SPKM-3 security mechanisms are *not* supported with RPCSEC_GSS authentication.

Note: Kerberos V5 is the only RPCSEC_GSS security mechanism that is supported in the implementation of NFS V4 in the initial release of AIX 5.3.

2.8.2 Other unsupported features

- ▶ Diskless client, Network Installation Management (NIM), and UDP are not supported over NFS V4.

We have mentioned before that exporting individual files is not supported on NFS V4, hence the lack of support for Diskless clients and NIM, which require individual files to be exported.

- ▶ Delegation is not supported in the initial AIX V5.3 implementation.

2.8.3 Optional feature support

The following optional features of NFS V4 are supported:

- ▶ NFS V4 ACLs are supported by both the client and server.

The NFS client supports management of NFS V4 ACLs using the **acledit**, **aclget**, and **aclput** utilities. The NFS server is capable of storing and retrieving NFS V4 ACLs in underlying file systems that support the NFS V4 ACL model. We offer more information in the next section.

- ▶ Support is provided to map principals and file ownership attributes from one NFS V4 domain into another.

This support is primarily intended for use at AIX NFS servers. It requires deployment of Enterprise Identity Mapping. The NFS mappings are managed using the **chnfsim** command.

- ▶ The AIX NFS V4 implementation supports two ACL types in underlying file systems: NFS V4 and AIXC.

Both of these ACL types are described below and in more detail in 3.4.2, “AIXC ACLs” on page 63 and 3.4.3, “NFS V4 ACLs: description” on page 65.

2.8.4 NFS4 ACL

The NFS4 ACL is the ACL defined by the NFS V4 protocol. It is platform-independent, so it can be supported by other vendors’ clients or servers. NFS V4 clients and servers are *not* required to support NFS4 ACL.

On an AIX server, if an underlying physical file system instance supports NFS4 ACLs, then the AIX NFS V4 server supports NFS4 ACL for that file system instance. The NFS client supports management of NFS V4 ACLs using the **acledit**, **aclget**, and **aclput** utilities. The NFS server is capable of storing and retrieving NFS V4 ACLs in underlying file systems that support the NFS V4 ACL model.

Most physical file system types on AIX do not support NFS4 ACL. These file system types include, but are not limited to: CFS, UDF, JFS, and JFS2 with extended attribute Version 1.

Note: Only JFS2 with extended attribute Version 2 (J2) supports NFS V4 ACLs.

See Chapter 3, “Enhanced security in NFS V4” on page 45 for more information about NFS V4 ACL support.

2.8.5 AIXC ACLs

The AIXC ACL is an AIX-proprietary ACL. It is not defined by the NFS V4 protocol, and it is understood only by AIX servers and clients.

On an NFS V4 server, AIXC ACL is supported when the underlying file system instance supports AIXC ACL. All instances of JFS and JFS2 support the AIXC ACL.

An NFS V4 or NFS V3 client has a mount option that enables or disables support for AIXC ACL. The default is to not support AIXC ACL. A user on an NFS V4 client file system can read and write AIXC ACL when both the client and the server are running AIX, the underlying physical file system instance on the server supports AIXC ACL, and the AIX client mounts the file system instance with AIXC ACL enabled. AIXC ACL support in NFS V4 is similar to the AIXC ACL support in AIX NFS V2 and NFS V3 implementations.

All instances of a JFS2 file system with extended attribute Version 2 support both AIXC ACL and NFS V4 ACL. A file in this type of file system may have mode bits only (no ACL), an NFS4 ACL, or an AIXC ACL. But it cannot have NFS4 ACL and AIXC ACL at the same time.

The **aclgettypes** command can be used to determine the ACL types that can be read and written on a file system instance. This command may return different output when it runs against a physical file system on an NFS V4 server locally than when it runs against the same file system on an NFS V4 client. For example, an NFS V4 file system instance on an NFS V4 server may support NFS V4 ACL and AIXC ACL, but the client is only configured to send and receive NFS V4 ACL (mounted with the **-noacl** option). In this case, when **aclgettypes** is executed from an NFS V4 client file system, only NFS V4 is returned. Also, if a user on the client requests an AIXC ACL, an error is returned.

The authoritative source for access checking lies in the underlying file system exported by the NFS server. The file system takes into consideration the file’s access controls (ACLs or permission bits), the caller’s credentials, and other local system restrictions that might apply.

Important: With AIXC or NFS V4 ACLs implemented in an exported file system, applications and users should not assume that examination of UNIX mode bits or ACLs alone can be used to conclusively predict access.

The `aclget`, `aclput`, and `acledit` commands can be used on the client to manipulate either NFS4 or AIXC ACLs. For more information, see Access Control Lists in *AIX 5L Version 5.3 Security Guide*, SC23-4907.

2.8.6 External name space (exname)

External name space (exname) is not part of the NFS V4 RFC. This is an option specific to AIX implementation. The `exname` option extends the pseudo-file system concept. The external name in your `/etc/exports` file must begin with the `nfsroot` name. But an `exname` export does not have to correspond to the server's root. Figure 2-6 explains how this works.

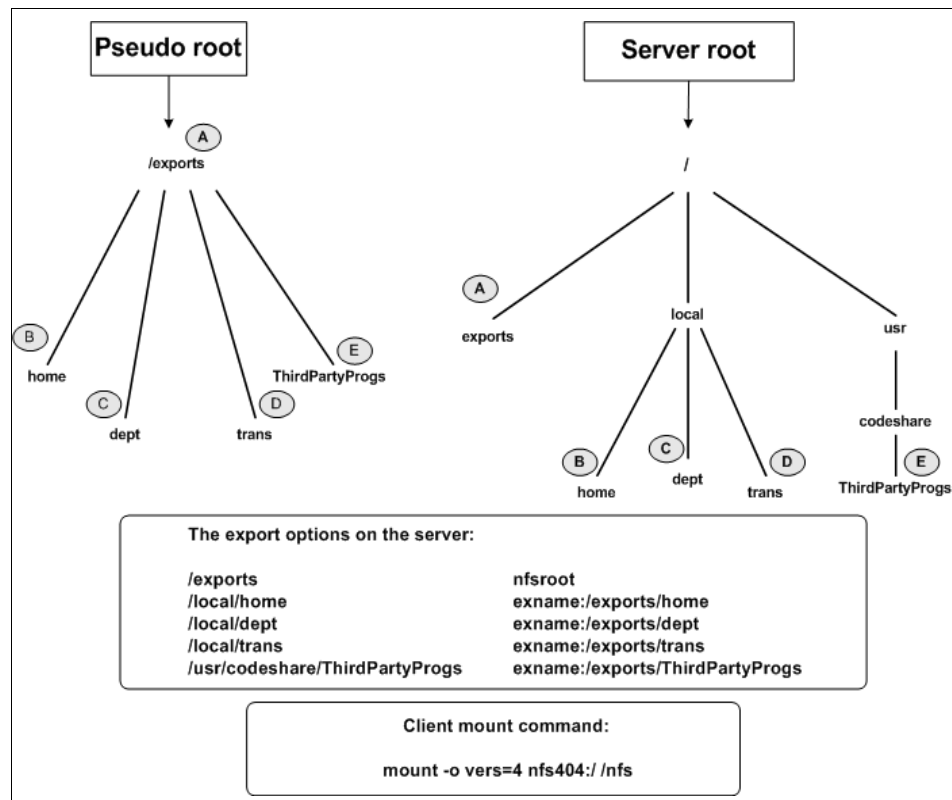


Figure 2-6 Representation of how the server builds the `exname` pseudo FS view

We want to render the view in such a way that the client sees what is represented by the pseudo-root part of Figure 2-6 on page 34. So, on the server, we want to export the following file systems:

```
/local/home  
/local/dept  
/local/trans  
/usr/codeshare/ThirdPartyProgs
```

We also want to make sure that we do not expose our server's file system tree to the client. How can we achieve this?

Attention: We must make sure that we set the pseudo-root on the server to `/exports`. When the server renders the pseudo FS view for the client, the directory or file under the `/exports` directory will be hidden. So, if you want to have directories and files user `/exports` available to the clients, you should either move them to another directory and export or choose a different directory to be the anchor for the pseudo-root.

Using the `exname` option, we create the `/etc/exports` file:

```
/local/trans -vers=4,rw,exname=/exports/trans  
/local/dept -vers=4,rw,exname=/exports/dept  
/local/home -vers=4,rw,exname=/exports/home  
/usr/codeshare/ThirdPartyProgs -vers=4,ro,exname=/exports/ThirdPartyProgs
```

We can see from this example that the `exname` option does not have the full path to the individual exports under **/exports**. In fact, you could specify the full path for all other exports. However, by following our example, the client is never shown the server's directory tree. This hides visibility of the actual server file system layout from the client.

If you intend to export a large number of directories under `/local` allowing the local component to be seen under `/exports`, then the `exname` option could also be used as shown:

```
/local/trans -vers=4,rw,exname=/exports/local/trans  
/local/dept -vers=4,rw,exname=/exports/local/dept  
/local/home -vers=4,rw,exname=/exports/local/home  
/usr/codeshare/ThirdPartyProgs  
-vers=4,ro,exname=/exports/local/ThirdPartyProgs
```

2.8.7 Protocol differences: server exporting and client mounting

For a full description of allowable export restrictions and export semantics, see the `exportfs` command description in *AIX 5L Version 5.3 Commands Reference*,

Volume 2, SC23-4889, and the /etc/exports file description in AIX 5L Version 5.3 Files Reference, SC23-4895.

Here are some differences between NFS V2, NFS V3, and NFS V4 in how mounts are handled. In NFS V2 and NFS V3, the server exported the directories that it wanted to make available for mounting. The NFS V2 or NFS V3 client then had to explicitly mount each export to which it wanted access.

With NFS V4, the server still specifies export controls for each server directory or file system to be exported for NFS access. From these export controls, the server renders a single directory tree of all the exported directories. This tree, a pseudo-file system, starts at the NFS V4 server's pseudo-root. The NFS V4 pseudo-file system model enables an NFS V4 client, depending on its implementation, to perform a single mount of the server's pseudo-root in order to access all of the server's exported data. The AIX NFS V4 client supports this feature. The actual content seen by the client is dependent on the server's export controls. (See 2.6.3, "Better namespace handling" on page 25.)

NFS V4 does *not* allow file-to-file mounting.

When a directory is exported by the server, that directory is only available to clients using the NFS V2 or NFS V3 protocol by default. To allow access by NFS V4, the export options must include the `vers` option. For details, see the **exports** command description in *AIX 5L Version 5.3 Commands Reference, Volume 2, SC23-4889*.

Client communication with the `rpc.mountd` daemon does not occur with NFS V4 mount processing. Operations in the core NFS V4 protocol are used to service client side mount operations. The NFS V4 server implementation does utilize the `rpc.mountd` daemon as part of handling NFS V4 access.

2.8.8 NFS files

This section includes information on the configuration files related to and used by NFS and the new Version 4 protocol.

- ▶ "/etc/exports file" on page 37
- ▶ "/etc/xtab file" on page 38
- ▶ "/etc/nfs/hostkey file" on page 38
- ▶ "/etc/nfs/local_domain file" on page 38
- ▶ "/etc/nfs/realm.map file" on page 39
- ▶ "/etc/nfs/princmap" on page 39
- ▶ "/etc/nfs/security_default" on page 40

/etc/exports file

The `/etc/exports` file indicates all directories that a server exports to its clients. Each line in the file specifies a single directory. The server automatically exports the listed directories each time the NFS server is started. These exported directories can then be mounted by clients. The syntax of a line in the `/etc/exports` file is:

```
directory -option[,option]
```

Important: NFS V4 does not support file exporting. If you need to export a specific file, export it as Version 2 or 3 (using the `vers=2` or `vers=3` options).

The directory is the full path name of the directory. Options can designate a simple flag such as `ro` (read only) or a list of host names. See the specific documentation of the `/etc/exports` file and the `export fs` command for a complete list of options and their descriptions.

Note: The `/etc/rc.nfs` script will not start the `nfsd` daemons or the `rpc.mountd` daemon if the `/etc/exports` file does not exist.

Here are entries from a sample `/etc/exports` file:

```
/exports/project/proja -ro,access=nfs401:nfs402:nfs403:nfs404
/exports/home -root=nfs404,access=nfs404
/var/tmp
/exports/dept/sales -vers=4,sec=krb5,access=allsales,root=allsales2
```

The first entry in this example specifies that the `/exports/project/proja` directory can be mounted by the systems named `nfs401`, `nfs402`, `nfs403`, and `nfs404`. These systems can read data and run programs from the directory, but cannot write to the directory.

The second entry in this example specifies that the `/exports/home` directory can be mounted by the system `nfs404` for read/write access. Additionally, `nfs404` may access data on the server as the root user.

The third entry in this example specifies that any client can mount the `/var/tmp` directory (with read/write access).

Attention: You will notice there is no access list specified for the `/var/tmp` entry. This means that *all* clients can mount this directory, and this is very insecure. However, you now have the ability to use the `sec=krb5` option, and this would take away the worry of openly exporting the directory.

The fourth entry allows access to the `/export/dept/sales` directory only to clients in the `allsales` netgroup using NFS V4 protocol and accessing the directory using Kerberos 5 (krb5) authentication. Root access is allowed only from `allsales2`.

Tip: Note the introduction of two new options: `vers` and `sec`.

/etc/xtab file

The `/etc/xtab` file has a format similar to the `/etc/exports` file and lists the currently exported directories. Whenever the `exportfs` command is run, the `/etc/xtab` file changes. This enables you to export a directory temporarily without having to change the `/etc/exports` file. If the temporarily exported directory is unexported, the directory is removed from the `/etc/xtab` file.

Important: The `/etc/xtab` file is updated automatically. You should *not* edit this file manually.

/etc/nfs/hostkey file

This file is used by the NFS server to specify the Kerberos host principal and the location of the keytab file. For instructions on how to configure and administer this file, see the `nfshostkey` command description in *AIX 5L Version 5.3 Commands Reference, Volume 4*, SC23-4891, or the command's `man` pages.

/etc/nfs/local_domain file

This file contains the local NFS domain of the system. It is implied that systems that share the same NFS local domain also share the same user and group registries. The `chnfsdom` command is used to administer this file:

```
# chnfsdom itsc.austin.ibm.com
#
# chnfsdom
Current local domain: itsc.austin.ibm.com
#
```

Important: You should not edit this file manually. Always use the `chnfsdom` command. The command automatically tells the `nfsrgyd` daemon that you have changed the local NFS domain. Also, a given NFS server or client machine can only belong to one NFS V4 domain.

For further information about the `chnfsdom` command, see the *AIX 5L Version 5.3 Commands Reference, Volume 1*, SC23-4888, and the command's `man` pages.

/etc/nfs/realm.map file

This file is used by the NFS registry daemon to map incoming Kerberos principals of the form `name@kerberos-realm` to the form `name@nfs-domain`. It then can resolve the `name@nfs-domain` to a local UNIX credential. This file provides a simple way to map Kerberos principals into the server's user registry. It is appropriate when clients in different Kerberos realms will be accessing the server, but the user namespace is global. For all Kerberos realms the server supports, the file contains lines in the following format:

```
realm1 nfs-domain
realm2 nfs-domain
```

If the Kerberos realm name is always the same as the server's NFS domain, this file is not needed.

Important:

- ▶ Multiple Kerberos realms can map to a single `nfs-domain`. The previous example demonstrates two realms mapping to one domain. However, a single realm *cannot* map to multiple `nfs` domains.
- ▶ We recommended that you do not edit the `/etc/nfs/realm.map` file manually. Always use the `chnfsrtd` command.

When the foreign identity mapping features of AIX NFS V4 support are used to facilitate inter-`nfs-domain` access, the mapping rules managed by the `chnfsim` utility allow mapping of realms into domains. In this case, `chnfsim` should be used instead of `chnfsrtd`.

See the `chnfsrtd` command description in *AIX 5L Version 5.3 Commands Reference, Volume 1*, SC23-4888, or the command's `man` pages for more details.

/etc/nfs/princmap

This file maps host names to Kerberos principals when the principal is not the fully qualified domain name of the server. It consists of any number of lines of the following format:

```
<host part of principal> <alias1> <alias2> ...
```

To add, edit, or remove entries from this file, use the `nfshostmap` command. For more information, see the `nfshostmap` command description in *AIX 5L Version 5.3 Commands Reference, Volume 4*, SC23-4891, or the command's `man(1)` pages.

/etc/nfs/security_default

The `/etc/nfs/security_default` file contains the list of security flavors that may be used by the NFS client, in the order in which they should be used. Use the `chnfssec` command to manage this file.

1. To create a new `/etc/nfs/security_default` file and tell the NFS client to first use `krb5`, then `krb5i`, and then `sys` security:

```
# chnfssec -a krb5,krb5i,sys
#
# cat /etc/nfs/security_default
krb5
krb5i
sys
#
```

2. To add a security flavor:

```
# chnfssec -a krb5,krb5i,krb5p,sys
#
# cat /etc/nfs/security_default
krb5
krb5i
krb5p
sys
```

Attention: If you add a security flavor to `/etc/nfs/security_default`, you must specify all of the existing methods *with* the new methods when running the `chnfssec` command. In this example, we add `krb5p` to `/etc/nfs/security_default`. As you can see, we have already used all the flavors in the file. Failure to do this (`chnfssec -a krb5p`) will result in the existing file being overwritten with the flavor specified in the command.

3. To remove a security flavor:

```
# chnfssec -r krb5p
#
# cat /etc/nfs/security_default
krb5
krb5i
sys
#
```

See the `chnfssec` command description in *AIX 5L Version 5.3 Commands Reference, Volume 1*, SC23-4888, or `man` pages for more information.

2.8.9 Restricting NFS port ranges

The NFS_PORT_RANGE environment variable may be used to limit the source port of network calls that the client makes to the server. This is very useful in a firewalled environment. Prior to the introduction of this feature, you had no control over what source port AIX would use, making firewall maintenance a difficult task. If used, this environment variable should be added to the /etc/environment file. The syntax of the environment variable is:

```
NFS_PORT_RANGE=udp[<starting_port>-<ending_port>]:tcp[<starting_port>-\  
<ending_port>]
```

In the following example, UDP packets sent by the client will have a source port in the range 3000 to 4000, and TCP connections will have a source port in the range 5000 to 6000. This means that you can now restrict traffic on all other higher range ports apart from the ones listed above.

```
NFS_PORT_RANGE=udp[3000-4000]:tcp[5000-6000]
```

2.8.10 Use of NFS_NOBODY

By default, and unauthenticated user is shown as user nobody belonging to group nobody. This can be changed by setting the NFS_NOBODY environment variable. This enables the user to be shown as whatever you set the NFS_NOBODY variable to.

2.9 NFS daemons, files, and commands: a quick reference

Table 2-3 lists the NFS daemons and their subsystem names. The following daemons are new to AIX and were specifically added to support NFS V4:

- ▶ /usr/sbin/gssd
- ▶ /usr/sbin/nfsrgyd

Table 2-3 List of NFS daemons

NFS daemons	Description
/usr/sbin/gssd	Services kernel requests for GSS operations.
/usr/sbin/nfsrgyd	Provides a names translation service for NFS servers and clients.
/usr/sbin/rpc.lockd	Processes lock requests through the RPC package.
/usr/sbin/rpc.statd	Provides crash-and-recovery functions for the locking services on NFS.

NFS daemons	Description
/usr/sbin/biod	Sends the client's read and write requests to the server. The biod daemon is SRC-controlled.
/usr/sbin/rpc.mountd	Answers requests from clients for file system mounts. The mountd daemon is SRC-controlled.
/usr/sbin/nfsd	Starts the daemons that handle a client's request for file system operations. nfsd is SRC-controlled.
/usr/sbin/portmap	Maps RPC program numbers to Internet port numbers. portmap is inetd-controlled.
/usr/sbin/rpc.rstatd	Returns performance statistics obtained from the kernel.
/usr/sbin/rpc.pcnfsd	Handles service requests from PC-NFS clients.

Table 2-4 gives an overview of all of the NFS files in AIX 5.3. The following files are new to AIX 5.3 and were introduced specifically for NFS V4:

- ▶ /etc/nfs/hostkey
- ▶ /etc/nfs/local_domain
- ▶ /etc/nfs/realm.map
- ▶ /etc/nfs/princmap
- ▶ /etc/nfs/security_default

The /etc/filesystems file also has new options for NFS V4.

Table 2-4 List of NFS files and those new to AIX 5.3

Files	Description
/etc/nfs/hostkey	Used by the NFS server to specify the Kerberos host principal and the location of the keytab file.
/etc/nfs/local_domain	Contains the local NFS domain of the system.
/etc/nfs/realm.map	Used by the NFS registry daemon to map incoming Kerberos principals of the form name@kerberos-realm to the form name@nfs-domain.
/etc/nfs/princmap	Maps host names to Kerberos principals when the principal is not the fully qualified domain name of the server.
/etc/nfs/security_default	Contains the list of security flavors that may be used by the NFS client, in the order in which they should be used.
/etc/filesystems	Lists all file systems that can potentially be mounted and their mounting configuration - persistent mounts.

Files	Description
/etc/bootparms	Lists the servers that diskless clients can use for booting from.
/etc/exports	Lists directories that can be exported to NFS clients.
/etc/networks	Contains information about networks on the Internet network.
/etc/pcnfsd.conf	Options for the rpc.pcnfsd daemon.
/etc/rpc	Contains database information for Remote Procedure Call (RPC) programs.
/etc/xtab	Lists directories that are currently exported.

Table 2-5 gives a list of all NFS commands in AIX 5.3. The following commands are new in AIX 5.3 and were specifically included for NFS V4:

- ▶ **/usr/sbin/chnfsdom**
- ▶ **/usr/sbin/chnfsrtd**
- ▶ **/usr/sbin/chnfssec**
- ▶ **/usr/sbin/nfshostkey**
- ▶ **/usr/sbin/nfshostmap**
- ▶ **/usr/sbin/nfs4cl**
- ▶ **/usr/sbin/chnfsim** (delivered in the **bos.cim.rte** fileset)

Table 2-5 List of NFS commands and those new to AIX 5.3

Commands	Description
/usr/sbin/chnfsdom	Changes the local NFS domain.
/usr/sbin/chnfsrtd	Changes NFS realm mappings.
/usr/sbin/chnfssec	Manages the /etc/nfs/security_default file.
/usr/sbin/nfshostkey	Configures the host key for an nfs server.
/usr/sbin/nfshostmap	Manages mapping of host names to Kerberos principles.
/usr/sbin/nfs4cl	Displays or modifies current NFS V4 statistics and properties.
/usr/sbin/chnfsim	Changes NFS foreign identity mappings.
/usr/sbin/chnfs	Starts a specified number of biod and nfsd daemons.
/usr/sbin/mknfs	Configures the system to run NFS and starts NFS daemons.

Commands	Description
<code>/usr/sbin/nfso</code>	Configures NFS network options.
<code>/usr/sbin/automount</code>	Mounts an NFS automatically on access.
<code>/usr/sbin/chnfsexp</code>	Changes the attributes of an NFS-exported directory.
<code>/usr/sbin/chnfsmnt</code>	Changes the attributes of an NFS-mounted directory.
<code>/usr/sbin/exportfs</code>	Exports and unexports directories to NFS clients.
<code>/usr/sbin/lsnfsexp</code>	Displays the characteristics of directories that are exported with NFS.
<code>/usr/sbin/lsnfsmnt</code>	Displays the characteristics of mounted NFS systems.
<code>/usr/sbin/mknfsexp</code>	Exports a directory.
<code>/usr/sbin/mknfsmnt</code>	Mounts a directory using NFS.
<code>/usr/sbin/rmnfs</code>	Changes the configuration to stop the NFS daemons.
<code>/usr/sbin/rpcinfo</code>	Reports the status of RPC servers.
<code>/usr/sbin/rmnfsexp</code>	Removes NFS-exported directories from a server's list of exports.
<code>/usr/sbin/rmnfsmnt</code>	Removes NFS-mounted file systems from a client's list of mounts.
<code>/usr/sbin/nfsstat</code>	Displays information about a machine's ability to receive calls.
<code>/usr/sbin/rpcgen</code>	Generates C code to implement an RPC protocol (delivered by <code>bos.net.nfs.server</code> fileset).



Enhanced security in NFS V4

This chapter describes the enhanced security options that are available in NFS V4 as released with the AIXL Version 5.3 operating system.

It starts by introducing general terminology, and then goes on to describe specific features.

The chapter includes the following sections:

- ▶ General security concepts and terminology
- ▶ NFS V4 user/group identification
- ▶ NFS V4 user authentication
- ▶ NFS V4 user authorization
- ▶ NFS V4 host identification
- ▶ NFS V4 host authentication
- ▶ NFS V4 host authorization

3.1 General security concepts and terminology

This section lays a foundation for the rest of the chapter by introducing terminology that will be used throughout the chapter.

3.1.1 Broad security categories

The topic of security can be very broad and far-reaching. A discussion of security measures might include items in the following categories:

- Physical security** Measures taken to control physical access to a facility or resource. Padlocks, fences, guards, and dogs are examples of physical security measures.
- Personnel security** Measures taken to help ensure that the people who are granted access to secured resources are reliable and are not likely to compromise the security of those resources. Security clearances and photo ID badges are examples of personnel security measures.
- Information security** Measures taken to protect important information from unauthorized disclosure, tampering, or destruction. Passwords, encryption, and file access permissions are examples of information security measures.

We will be talking in this chapter primarily about what we call *information* security, because NFS V4 is about sharing information. Organizations will not be able to properly protect their information resources without also implementing physical and personnel security measures, but we will not address those measures in this document.

3.1.2 Information security components

In an NFS V4 context, information security falls into the following areas:

- Identification** Uniquely establishes the identity of information system users, hosts, and services. Answers questions such as, “Who are the users or hosts that are trying to access shared data on my server?”
- Authentication** Confirms the identity of a system user, host, or service. Answers questions such as, “Is this user really who he or she claims to be?”

Authorization Controls what shared information each system user or other entity can access. Answers questions such as, “Does this user have the right to access a shared data object on my server?”

3.1.3 RPC security flavors

NFS V4 uses Sun’s Remote Procedure Call (RPC) protocol to communicate over the network between the client and the server. The IBM implementation enables you to use three different RPC security flavors:

- ▶ Basic UNIX security (AUTH_SYS, aka AUTH_UNIX)
- ▶ Diffie-Hellman security (AUTH_DH, aka AUTH_DES)
- ▶ RPCSEC_GSS security as defined in RFC2203

3.1.4 RPCSEC_GSS protection levels

When using RPCSEC_GSS security with RPCs, there are three levels of protection that can be applied to the RPCs as they are transmitted over the network between server and client:

Authentication	Validates the identity of RPC sender
Integrity	Validates that the contents of the RPC were not changed during transmission (also includes authentication)
Privacy	Prevents unauthorized viewing of data while it is in transit between client and server (also includes authentication and integrity)

Keep in mind that each increasing level of protection comes with a performance penalty. Choose the minimum level that meets your data protection requirements.

3.1.5 RPCSEC_GSS protection mechanisms

The NFS V4 standard (RFC 3530) requires that NFS implementations support three different RPCSEC_GSS mechanisms:

- ▶ Kerberos V5 (RFC1964)
- ▶ SPKM-3/LIPKEY (RFC2847)
- ▶ SPKM-3 on its own (RFC2847/RFC2025), for situations where the initiator (the client) is anonymous or has its own certificate.

In AIX 5.3, IBM has implemented Kerberos V5, but not SPKM/LIPKEY at this point in time.

IBM offers several forms of encryption with its Kerberos V5 implementation, among which are single DES and triple DES.

Note: Triple DES encryption gives the best protection, but you may need to use single DES to get better performance and interoperability.

3.1.6 Looking ahead to the rest of the chapter

The rest of this chapter compares different options for identification, authentication, and authorization, which are different for hosts and users. We discuss users first, and then hosts.

We use various Kerberos-related terms throughout this chapter. See Appendix A, “Kerberos” on page 243 for a description of Kerberos and its related terms.

3.2 NFS V4 user/group identification

In this section, we discuss three different methods for managing user and group identities: via the standard UNIX `/etc/passwd` and `/etc/group` files, via Sun Microsystems’ Network Information Services (NIS), and via the Lightweight Directory Access Protocol (LDAP) schema defined by RFC2307.

We also discuss how NFS V4 determines a user’s identity under `AUTH_SYS` authentication and `RPCSEC_GSS` (Kerberos) authentication.

3.2.1 User identity management options

UNIX implementations typically use a 32-bit integer to identify users and groups. These integers are referred to as UIDs for users and GIDs for groups. User and group ownership for system processes and file system objects are maintained in UID/GID form. NFS V2 and NFS V3 also use these UIDs and GIDs to identify users and groups.

People do not usually work directly with the numeric IDs; they work with text user and group names that are normally easier to associate with an actual individual or group. When presenting information about process and file ownership, the system translates the numeric IDs into the names. The relationship between the names and the IDs is maintained in a user registry, which can be standard UNIX, NIS, or LDAP.

Standard UNIX user registry

In standard UNIX, user-name-to-ID mappings are contained in the `/etc/passwd` file, and group-name-to-ID mappings are contained in the `/etc/group` file.

All but the smallest organizations will want to use a shared user registry, rather than maintaining separate `/etc/passwd` and `/etc/group` files on all hosts. Here is one reason why. All clients using data stored on an NFS server directory should use the same identifier to represent the same user or group. This is necessary to maintain consistent file ownership. For example, in an NFS V2 and NFS V3 environment, if UID 100 is Joe on one NFS client and Mary on another client, NFS files created by Joe from the first client will show as being owned by Mary on the 2nd client, and vice versa. To avoid this, the separate `/etc/passwd` and `/etc/group` files will need to be kept in sync on all NFS clients that access data on a common NFS server. This can be a very expensive and error-prone task.

NIS user registry

Sun Microsystems developed Network Information Services (NIS) to help centrally manage UID/GID- to-name mappings. With NIS, the user and group identity information is maintained in one place and is distributed to all of the machines participating in an NIS domain. Having all of an organization's NFS servers and clients participate in an NIS domain helps keep identity mappings consistent.

NIS has been widely used in conjunction with NFS, and it can work well for medium-sized organizations. It becomes problematic, however, when a large organization requires multiple NIS domains, or when two different organizations merge. Consolidating two different NIS domains can be a very difficult task.

More information about NIS can be found on Sun Microsystems' Web site:

<http://www.sun.com>

LDAP user registry

Directory services based on the Lightweight Directory Access Protocol (LDAP) can also be used to maintain user and group identities. LDAP-based directories are typically more scalable and secure than NIS. RFC2307 describes an LDAP schema that provides NIS-like functionality.

IBM Tivoli Directory Server is the IBM implementation of LDAP-based directory services. As of Version 5.1, the Tivoli Directory Server supports RFC2307 user and group identification. More information is available about the Tivoli Directory Server on the IBM Web site at:

<http://www.ibm.com/software/tivoli/products/directory-server>

3.2.2 User/group identities and NFS V4

NFS V4 handles user and group identities very differently from previous versions. NFS V2 and NFS V3 pass UIDs and GIDs between the client and server, whereas NFS V4 passes string names in the form `user@nfs_domain` or `group@nfs_domain`. We will describe identity mapping under three conditions:

- ▶ Single NFS domain using AUTH_SYS security
- ▶ Single NFS domain using Kerberos security
- ▶ Multiple NFS domains

We will use two sample operations in each case:

- ▶ Creating a file
- ▶ Requesting file ownership

Notes:

1. When we say “NFS” throughout the rest of this section, we mean NFS V4.
2. The examples in this section are not intended to depict everything that goes on in an NFS transaction. They are simplified to convey pertinent concepts.

Single NFS domain using AUTH_SYS security

Figure 3-1 shows how user information is passed from the NFS client to the server when creating a file.

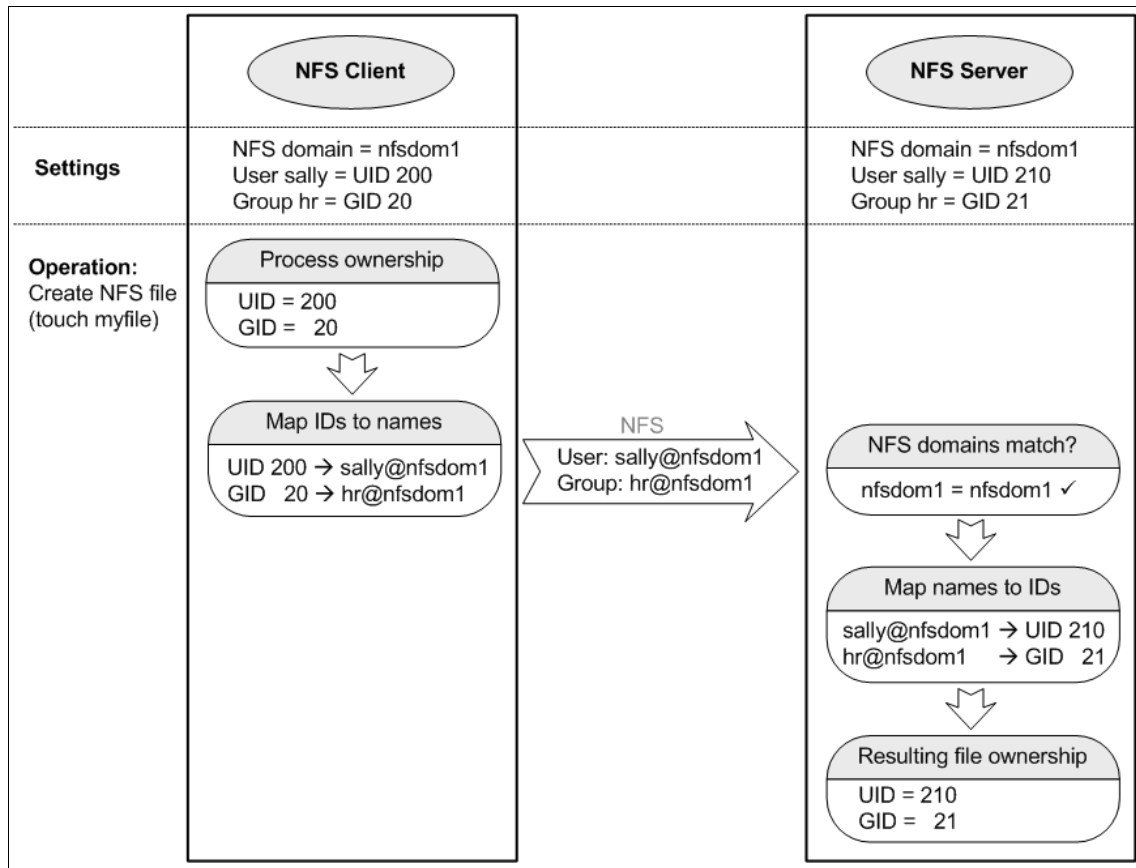


Figure 3-1 Creating a file under AUTH_SYS security

The ownership of the requesting user process is in the form of UID and GID. The NFS client translates the UID into a user name string and translates the GID into a group name string. These strings are then sent across to the server. The server checks to make sure that the NFS domains in the request match its own NFS domain. (If they didn't match, extra steps would be required. See the multiple domain discussion below.) The server then translates the strings back to UID/GID using its user registry, which may not be the same as the client's, and stores the UID/GID as ownership attributes of the file being created.

Important: For ownership to be transferred properly between client and server, the server must be able to convert the user and group names into IDs. If the server cannot do that, it will make the file owned by the user nobody, or the group nobody, or both. This is a key difference between NFS V4 and earlier versions. Prior to NFS V4, the server just took the provided UID/GID information and placed it in the file attributes. It did not have to do any translation.

Figure 3-2 shows how the user information is passed from the NFS server to the client when the client has requested ownership information for an existing file.

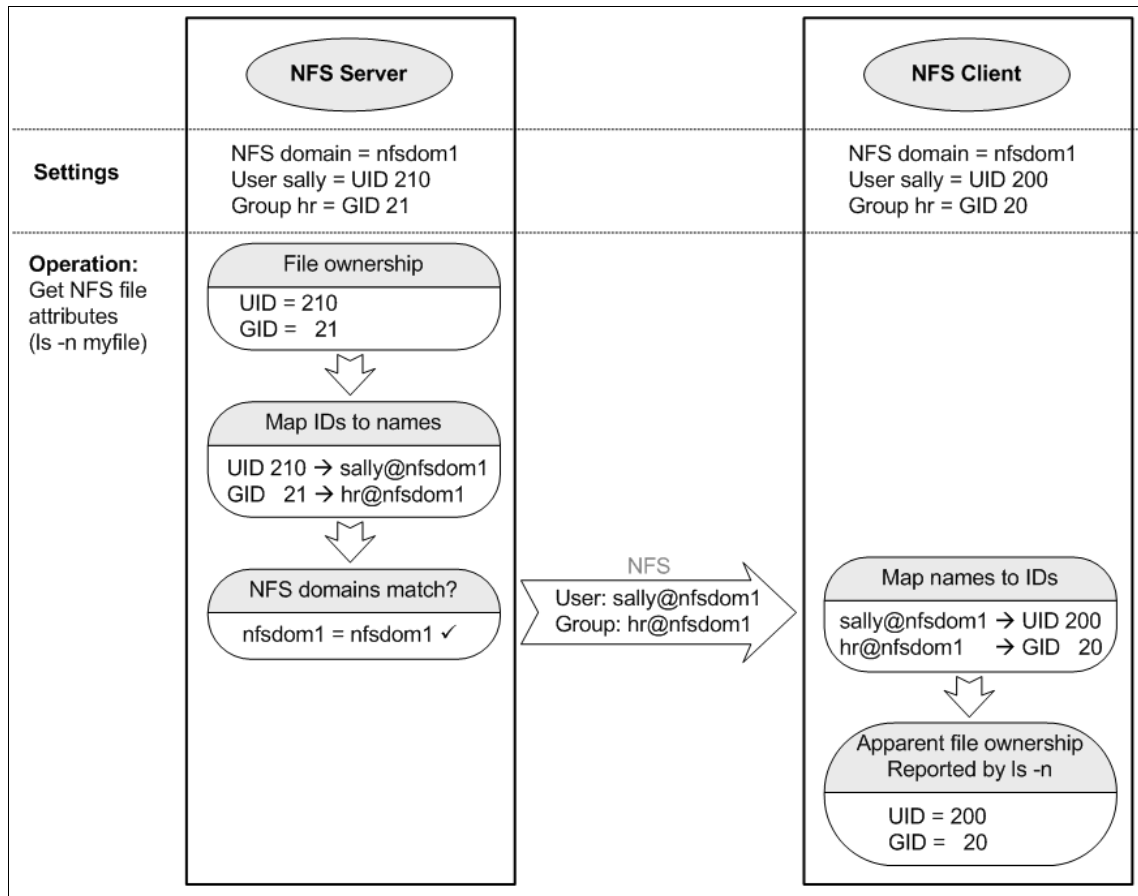


Figure 3-2 Getting file ownership under AUTH_SYS security

The ownership information that is stored with the file is in the form of UID and GID. The NFS server translates the UID into a user name string and the GID into

a group name string. It then checks to make sure that its NFS domain is the same as the requester's NFS domain. In this case they are the same, so the server can go ahead and send the owner and group strings across to the client. The client translates the strings back to UID/GID using its user registry, which may not be the same as the server's, and reports that information back to the requesting process.

The `ls -n` output would appear similar to this:

```
-rw-r--r--  1 200      20          255 Aug 11 11:05 myfile
```

If an `ls -l` command was issued, the client would take the UID/GID information and translate it back to names, resulting in output similar to:

```
-rw-r--r--  1 sally   hr          255 Aug 11 11:05 myfile
```

Single NFS domain using Kerberos security

Under Kerberos security, the NFS server does not take the client user's identity from the user and group strings in the NFS request. Instead, it determines the user's identity from the Kerberos principal that is part of the `RPCSEC_GSS` call. The principal identity is in the form `user@KERBEROS_REALM`. (The realm name is typically represented in all capital letters.)

Figure 3-3 on page 54 shows how an NFS server determines ownership when an NFS client requests to create a file in a directory that is under Kerberos security.

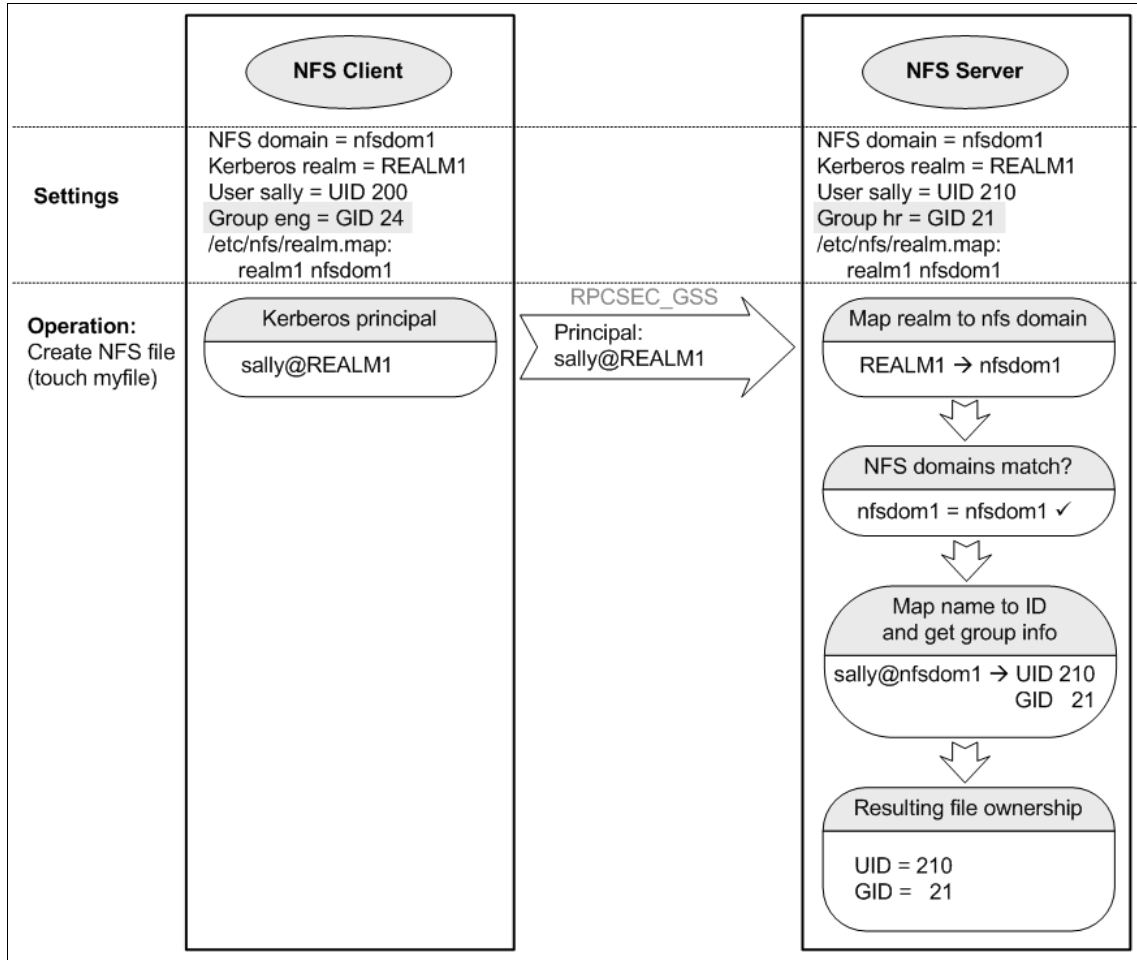


Figure 3-3 Creating a file under RPCSEC_GSS (Kerberos) security

The server takes the requesting principal's realm and maps it to an NFS domain via the contents of the /etc/nfs/realm.map file. It then checks to see whether the resulting NFS domain matches its own NFS domain. In this case, the domains match, so the server then looks up the user name in its user registry to determine the UID. The server also gets the user's GID from its user registry, not from the NFS request. It then places the UID and GID in the newly created file's attributes.

Attention: Determining the group ownership under Kerberos is an important difference from how NFS operates under AUTH_SYS security. With AUTH_SYS, the server gets the group information from the client via the NFS request. With Kerberos, the server only gets the user identity from the client. It gets the group information from the server's user registry.

Figure 3-3 illustrates this. Although the user sally's primary group is eng on the client, the created file is owned by the group hr because it is sally's primary group on the server. Using ls -l on the file on both the server and the client will show that it is owned by the 'hr' group.

Figure 3-4 on page 56 shows what happens when an NFS client requests an NFS server for ownership information for a file that is in a Kerberos-protected directory.

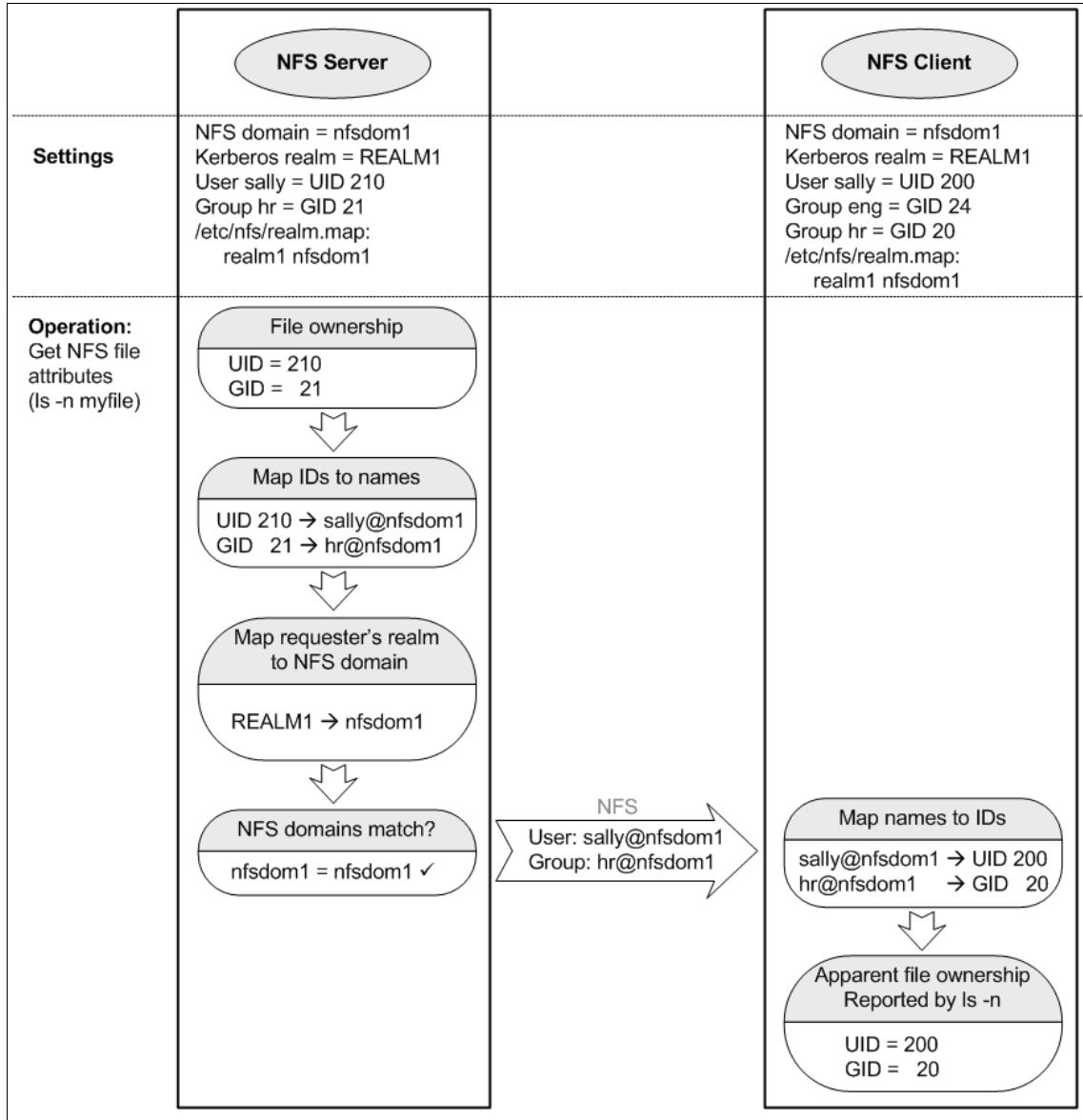


Figure 3-4 Getting file ownership under RPCSEC_GSS (Kerberos) security

The process is the same as the AUTH_SYS process in Figure 3-2 on page 52, except (again) that the client user's identity is determined from the Kerberos principal accompanying the request. The NFS server translates the principal's realm to an NFS domain to check to make sure that the requester's domain is the

same as the server's domain. Note that the file ownership information is still communicated through the NFS protocol.

Multiple NFS domains

If the client's NFS domain does not match the server's, name mapping must be made between the domains. User and group names in the client's domain must be mapped to user and group names in server's domain. This can be done via Enterprise Identity Mapping (EIM) that is available for free with AIX. (It is available on the base AIX CD, in the fileset bos.eim.rte.) See the IBM Redbook *Windows-based Single Signon and the EIM Framework on the IBM @server iSeries Server*, SG24-6975, for an overview of EIM and a description of how to configure it for use with the IBM Network Authentication Service. Although the book is written for iSeries™ servers, the concepts also apply to pSeries® servers and AIX.

If EIM is not in place, the NFS server will view the client identities as foreign, and it will map them to the user/group nobody.

Figure 3-5 on page 58 shows the process of creating a file under AUTH_SYS when EIM is used.

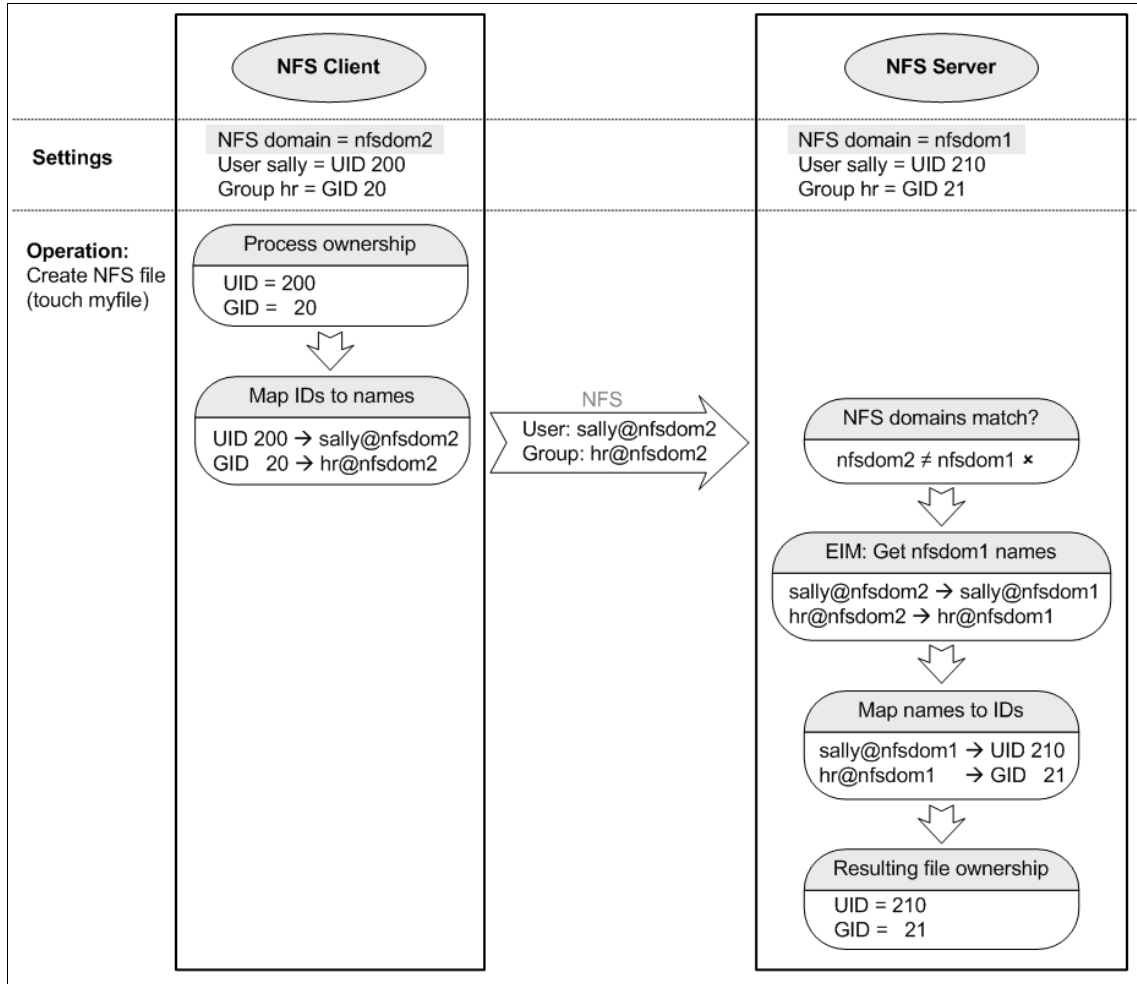


Figure 3-5 Creating a file under AUTH_SYS security: multiple domains

Note the extra EIM step after the server determines that the client's NFS domain does not match its own. The server requests from EIM which nfsdom1 user matches sally@nfsdom2, and which nfsdom1 group matches hr@nfsdom2. The figure shows the user and group names being the same between the two domains, but it is possible for them to be different. The user name sally@nfsdom2 might have mapped to the name sally2@nfsdom1, or it might even have mapped to mary@nfsdom1.

The other scenarios presented earlier in this section all also require the additional EIM step if the client's NFS domain does not equal the server's.

3.3 NFS V4 user authentication

By default, NFS uses the AUTH_SYS method to authenticate user identities. Alternatively, you can use RPCSEC_GSS (Kerberos) to authenticate users.

3.3.1 AUTH_SYS user authentication

Under the AUTH_SYS security flavor, the user is authenticated at the client, usually via a logon name and password. The NFS server trusts the user and group identities presented by its clients. If someone gains administrative control of an NFS client, or has control of a machine pretending to be a valid NFS client via IP address spoofing, it is easy to masquerade as any valid NFS user.

For example, if tight physical security cannot be maintained on a network, it would be easy for someone to bring in a Linux laptop, set the IP address to be the same as a valid NFS client, and connect the laptop to the network using the valid client's connection. When connected, the person can set up any user account desired on the laptop and gain access to any of the exported data on an NFS server.

Attention: Because of this vulnerability, you should not use AUTH_SYS user authentication if controlling access to your data is important.

3.3.2 RPCSEC_GSS user authentication using Kerberos

This section first gives a little background about Kerberos in general, and then it describes how Kerberos is used in NFS.

What is Kerberos?

The *AIX 5L Version 5.3 Security Guide* describes Kerberos as a network authentication service that provides a means of verifying the identities of principals (users and hosts) on physically insecure networks. Kerberos provides mutual authentication, data integrity, and privacy under the realistic assumption that network traffic is vulnerable to capture, examination, and substitution.

Kerberos tickets are credentials that verify a principal's identity. There are two types of tickets: ticket-granting and service. The ticket-granting ticket is for the initial identity request. When logging into a host system, a user needs something that verifies his or her identity, such as a password or a token. This password or token is used to obtain a ticket-granting ticket, which can then be used to request service tickets for specific services. This two-ticket method is called the *Kerberos third-party authentication model*.

The trusted third-party or intermediary in Kerberos is called the Key Distribution Center (KDC). The KDC issues all of the Kerberos tickets to the clients. Tickets have a predetermined lifetime and have to be renewed periodically.

The Kerberos database keeps a record of every principal; the record contains the name, private key, expiration date of the principal, and some administrative information about each principal. This database is maintained on the master KDC, and it can be replicated to one or more replica KDCs.

See Appendix A, “Kerberos” on page 243 for more information about what Kerberos is and how it works.

How does NFS use Kerberos?

When an NFS client and server are using Kerberos 5 (krb5) authentication, the client and server must establish a security context for NFS requests. The security context is a data structure that indicates that the client and server have completed a mutual authentication procedure. The context also contains the encryption keys that will be used for protecting exchanged data, if such protection has been requested. The security context has a lifetime and may need to be refreshed by the client from time to time. The client and server each maintain a cache of security contexts, one context per user/host combination.

A walkthrough of the authentication process helps explain how it works. Figure 3-6 on page 61 illustrates the steps involved in authenticating an NFS user request.

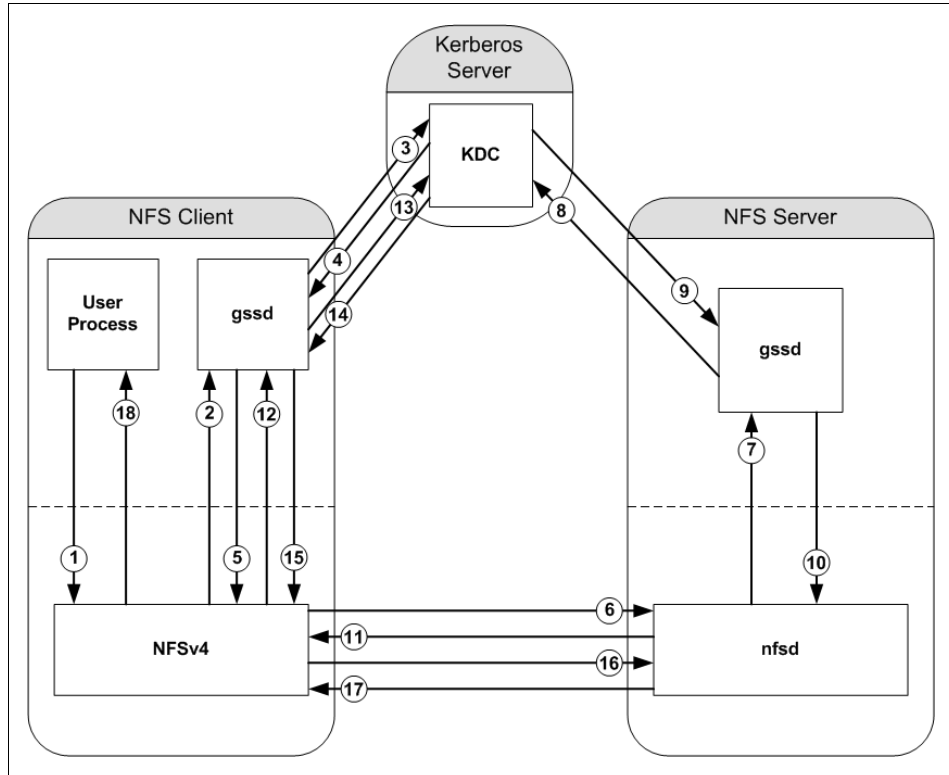


Figure 3-6 RPCSEC_GSS authentication flow

Here is a description of the authentication steps:

1. An authenticated user process on the NFS client attempts an operation on a Kerberos-protected NFS object.
2. The client NFS has no current context for the user/server combination. It calls out to the gssd to obtain an RPC security context with the target service (the server NFS). It gives the gssd the name of the service (machine principal of the NFS server) and the user's identity.
3. If the user has valid Kerberos credentials, the gssd then contacts the KDC to initiate context establishment to the target service.
4. If the service is known, the KDC returns an opaque token to the client gssd. This token will authenticate the user to the target.
5. The gssd returns the token to the client NFS.
6. The client NFS then sends the token to the server NFS in a NULL RPC call to request context establishment.

7. The server NFS passes the context token to the server gssd in a request to accept the context.
8. The server gssd passes the token to the KDC to verify the requesting user's identity.
9. If the KDC accepts the token, it returns another token to the server gssd and a context handle. This token will authenticate the server's identity to the requesting client.
10. The server gssd returns this token to the server NFS. The server NFS now has established context.
11. The server NFS returns the accept token to the client in response to the NULL RPC call.
12. The client passes the accept token to the client gssd in a second call to initiate the context.
13. The client gssd calls the KDC to verify the server's token.
14. If the KDC accepts the token, it returns a context handle to the client gssd.
15. The client gssd returns the context handle to the client NFS. The client now has established context.
16. The original NFS operation can now proceed under the established context.
17. The NFS server responds to the original operation.
18. Results of the original operation are returned to the user process.

3.4 NFS V4 user authorization

User authorization in an NFS context means controlling user access to directories and files in the exported file systems. This section describes two ways to control this access: via standard UNIX file permissions and via NFS V4 Access Control Lists (ACLs).

Note: Although you can also use AIX ACLs (known in AIX 5.3 as the AIXC ACL type), they are only supported on AIX systems, and they have not been widely adopted. We do not discuss them in detail in this document. For more about AIXC ACLs, see the *AIX 5L Version 5.3 Security Guide, SC23-4907*.

ACLs provide for more granular access control than standard UNIX file permissions. One of the main differences is in group access. Whereas standard UNIX permissions only provide access control for one group (the group that owns the file), ACLs enable different access permissions to be specified for multiple groups. ACLs also allow access permissions to be specified at a user level, but

controlling access on a per-user basis is usually not practical for organizations of any size.

3.4.1 Standard UNIX file permissions

The standard UNIX file permission model consists of granting read, write, and execute access to three categories of users:

User	The user who owns the file
Group	The group that owns the file
Other	Everyone else

This model frequently is inadequate when applied to a real-world organization structure. Individuals often operate in multiple roles, or different individuals operating in the same role may have attributes that require different data access.

For example, access to data is sometimes restricted based on company affiliation. Two people working on an engineering project might be from different companies. Although they both can have access to much of the engineering data, some of that data may be restricted to employees of one company and not the other. This sort of granular access control is very difficult, if not impossible, to implement using standard UNIX file permissions.

3.4.2 AIXC ACLs

An AIXC ACL has two parts:

- ▶ Base permissions, which map directly to the standard user/group/other UNIX file permissions
- ▶ Extended permissions, which enable you to control access to other specific users and groups

Here is an example of an AIXC ACL (in `aclget` format) that has extended permissions *disabled*:

```
*
* ACL_type  AIXC
*
attributes: SGID
base permissions
  owner(root):  rwx
  group(system): r-x
  others:  r-x
extended permissions
  disabled
```

Here is an example of an AIXC ACL that has extended permissions *enabled*:

```
*
* ACL_type  AIXC
*
attributes: SGID
base permissions
  owner(root):  rwx
  group(system): r-x
  others:  r-x
extended permissions
  enabled
  permit  rw-    u:sally
  deny    rwx    g:sales
```

AIXC ACLs from an NFS client's viewpoint

AIXC ACLs are supported on both NFS V3 and NFS V4 AIX clients. To be able to see and modify AIXC ACLs from an NFS client, you must mount your file systems with the `acl` option (`noacl` is the default).

There is one aspect in which an NFS V4 client handles AIXC ACLs differently from an NFS V3 client. An NFS V4 client depicts an AIXC ACL as an NFS V4 ACL if the AIXC ACL has extended permissions disabled, or if extended permissions are enabled but there are no entries in the extended permissions list. If the AIXC ACL has extended permissions enabled and there are entries in the extended permissions list, the NFS V4 client shows it as an AIXC ACL. An NFS V3 client always shows an AIXC ACL as an AIXC ACL.

For example, assume a file has the following AIXC ACL on the server:

```
*
* ACL_type  AIXC
*
attributes:
base permissions
  owner(root):  rw-
  group(system): r--
  others:  r--
extended permissions
  disabled
```

Then the NFS V4 client will show the ACL as:

```
*
* ACL_type  NFS4
*
*
* Owner: root
* Group: system
```

```

*
s: (OWNER@): a      rwpRwAdcCs
s: (OWNER@): d      xo
s: (GROUP@): a      rRadcs
s: (GROUP@): d      wpWxAco
s: (EVERYONE@): a    rRadcs
s: (EVERYONE@): d    wpWxAco

```

The *AIX 5L Version 5.3 Security Guide*, SC23-4907, has more about AIXC ACLs.

3.4.3 NFS V4 ACLs: description

NFS V4 ACLs are similar to Windows NTFS ACLs, but they are not identical. The developers of the NFS V4 standard chose the Windows ACLs model over POSIX ACLs because the Windows ACL model is both richer and more widely deployed. Although many UNIX vendors implemented ACLs based on the POSIX Draft ACL specification, those implementations tended to be proprietary, and the POSIX specification was never standardized.¹

Note: In order to use NFS V4 ACLs, the server file system must support them. As of this writing, AIX5L Version 5.3 only supports NFS V4 ACLs in two file system types: Enhanced Journaled File System (JFS2) with the extended attribute format set to Version 2 (EAv2), and General Parallel File System (GPFS). For more information about NFS V4 ACL support, see the *AIX 5L Version 5.3 Security Guide*, SC23-4907, and the *AIX 5L Differences Guide Version 5.3 Edition*, SG24-7463.

NFS V4 ACL format

According to the NFS V4 protocol specification, an NFS V4 ACL is an array of Access Control Entries (ACEs) that have four elements: a type, a set of flags, an access bit mask, and an identity. As implemented in the AIX JFS2 EAv2 file system, each ACL can be a maximum of 64 KB.

The textual representation of an NFS V4 ACL consists of a list of Access Control Entries (ACEs), one per line. Each ACE has four elements in the following format:

```
IDENTITY ACE_TYPE ACE_MASK ACE_FLAGS
```

IDENTITY has the format:

```
IDENTITY_type:(IDENTITY_name or IDENTITY_ID or IDENTITY_who):
```

Table 3-1 on page 66 lists possible values for IDENTITY_type.

¹ Pawlowski, Brian, Spencer Shepler, Carl Beame, Brent Callaghan, Michael Eisler, David Noveck, David Robinson, Robert Thurlow. *The NFS Version 4 Protocol*, SANE Conference, NL, 2000. <http://www.nluug.nl/events/sane2000/papers/pawlowski.pdf>.

Table 3-1 ACE IDENTITY_type values

IDENTITY_type	Description
u	user (IDENTITY_name or IDENTITY_ID)
g	group (IDENTITY_name or IDENTITY_ID)
s	special who string (IDENTITY_who)

IDENTITY_name is the user or group name.

IDENTITY_ID is the user or group numeric ID.

IDENTITY_who is a special who string that should be understood universally rather than in the context of a particular NFS domain. The string can be one of those shown in Table 3-2.

Table 3-2 ACE special who strings supported by AIX

IDENTITY_who	Description
OWNER@	The owner of the file
GROUP@	The group associated with the file
EVERYONE@	The world

The additional special who strings specified in RFC 3530 (shown in Table 3-3) are *not* currently supported in AIX.

Table 3-3 ACE special who strings not supported in AIX

IDENTITY_who	Description
ANONYMOUS@	Accessed without any authentication
AUTHENTICATED@	Any authenticated user (opposite of ANONYMOUS)
INTERACTIVE@	Accessed from an interactive session
NETWORK@	Accessed via the network
DIALUP@	Accessed via a dial-up connection
BATCH@	Accessed from a batch job
SERVICE@	Accessed from a system service

ACE_TYPE is a single character. Table 3-4 on page 67 shows possible values.

Table 3-4 ACE_TYPE values

ACE_TYPE	Description
a	Allow access
d	Deny access
l	Generate a system alarm when an access is attempted (currently not supported in AIX)
u	Generate an audit log entry when an access is attempted (currently not supported in AIX)

ACE_MASK is a set of permission flags (permission bits) that can be combined without any separator. Table 3-5 shows possible flags. Note that the flag values are case-sensitive.

Table 3-5 ACE_MASK values

ACE_MASK	RFC 3530 name	Description
r	READ_DATA or LIST_DIRECTORY	Permission to read the data of the file or list the contents of the directory
w	WRITE_DATA or ADD_FILE	Permission to modify the file's data or add a new file to the directory
p	APPEND_DATA or ADD_SUBDIRECTORY	Permission to append data to the file or add a new subdirectory to the directory
R	READ_NAMED_ATTRS	Permission to read the named attributes of the file or directory
W	WRITE_NAMED_ATTRS	Permission to write the named attributes of the file or directory
x	EXECUTE	Permission to execute the file or traverse the directory
D	DELETE_CHILD	Permission to delete files or subdirectories from within the directory
a	READ_ATTRIBUTES	Permission to read basic attributes (non-ACLs) of the file or directory
A	WRITE_ATTRIBUTES	Permission to change basic attributes (non-ACLs) of the file or directory
d	DELETE	Permission to delete the file or directory
c	READ_ACL	Permission to read the ACL of the file or directory

ACE_MASK	RFC 3530 name	Description
C	WRITE_ACL	Permission to change the ACL of the file or directory
o	WRITE_OWNER	Permission to change the owner of the file or directory
s	SYNCHRONIZE	Permission to access file locally at the server with synchronous reads and writes

ACE_FLAGS (optional) is a combination of one or more of the following two-letter flags without any separator. Four of the currently defined flags have to do with ACL inheritance (Table 3-6), and the other two have to do with auditing (Table 3-7). The inheritance flags only have meaning when applied to a directory. Auditing flags only have meaning when used with the audit or alarm ACE types.

Table 3-6 ACE_FLAG inheritance-related values

ACE_FLAG	RFC 3530 name	Description
fi	FILE_INHERIT	Indicates that this ACE should be added to each newly created non-directory file.
di	DIRECTORY_INHERIT	Indicates that this ACE should be added to each newly created subdirectory.
oi	INHERIT_ONLY	Indicates that this ACE does not apply to the current directory; it is only to be added to newly created files/subdirectories as specified by the above two flags.
ni	NO_PROPAGATE_INHERIT	Indicates that this ACE should be added to newly created files/subdirectories immediately under the directory, but subdirectories should not pass it on to their children.

Table 3-7 ACE_FLAG auditing-related values

ACE_FLAG	RFC 3530 name	Description
sf	SUCCESSFUL_ACCESS_ACE_FLAG	Generate audit or alarm when an access attempt succeeds.
ff	FAILED_ACCESS_ACE_FLAG	Generate audit or alarm when an access attempt fails.

NFS V4 ACL permission restrictions

Some permission bits are interrelated and must be used together under the following circumstances:

- ▶ The WRITE_DATA (w) and APPEND_DATA (p) bits must be specified together in a file's ACE, or in a directory's ACE that has the FILE_INHERIT flag set.

Special user permissions

In NFS V4, two classes of users have special access to files:

- ▶ The UNIX super user (UID=0) is allowed all access permissions, regardless of the ACE permission bit settings. (The only exception to this is execute permission.) This special access applies to processes running on the NFS server and processes running on NFS clients that have been given root access via the **exportfs** command.
- ▶ The owner of a file always has the permissions READ_ACL, WRITE_ACL, READ_ATTRIBUTES, and WRITE_ATTRIBUTES, regardless of the actual settings in the ACL.

3.4.4 NFS V4 ACLs: ACL evaluation

In order to properly use NFS V4 ACLs, it is important to understand how they are evaluated when determining whether an access request will be granted or denied.

Per the RFC 3530 NFS V4 standard and the *AIX 5L Version 5.3 Security Guide*, an AIX NFS V4 server evaluates the ACL list from the top down, applying the following rules:

- ▶ Only ACEs that have a who that matches the requester are considered. The credentials of the requester are not checked while processing the ACE with special who EVERYONE@.
- ▶ Each ACE is processed until all of the bits of the requester's access have been allowed or at least one of the requested bits not previously allowed has been denied.
- ▶ When a permission bit has been allowed, it is no longer considered in the processing of later ACEs.
- ▶ If a deny ACE_TYPE is encountered where the ACE_MASK has bits in common with not-yet-allowed bits in the request, access is denied, and the remaining ACEs are not processed.
- ▶ If the entire ACL has been processed and some of the requested access bits still have not been allowed, access is denied.

NFS V4 ACL evaluation examples

The following examples help illustrate ACL evaluation. For more examples, see the *AIX 5L Version 5.3 Security Guide*.

Given the following ACL on a file:

```
*
* ACL_type   NFS4
*
*
* Owner: sally
* Group: staff
*
g:sales:      d      wp
s:(OWNER@):  a      rRWDaAdcCs
s:(OWNER@):  d      wpo
s:(GROUP@):  a      rwpRxadcs
s:(GROUP@):  d      WDACo
s:(EVERYONE@): a    rwpRxadcs
s:(EVERYONE@): d    WDACo
```

If the user sally requests READ_DATA (r) and WRITE_DATA (w) access, the ACL evaluation will proceed as follows:

- ▶ The s:(OWNER@):a... ACE is processed because sally owns the file.
 - READ_DATA is allowed because that bit is set in the ACE_MASK.
 - WRITE_DATA is not yet allowed because it is not set in the ACE_MASK.
- ▶ The s:(OWNER@):d... ACE is processed because sally owns the file.
 - WRITE_DATA is denied because that bit is set in the ACE_MASK, and WRITE_DATA has not yet been allowed by a previous ACE.
- ▶ No further ACEs are processed, and the requested access is denied.

Notes:

1. In the previous example, even though the GROUP@ and EVERYONE@ ACEs allow WRITE_DATA access, Sally is denied WRITE_DATA access because it is specifically denied by the owner ACE.
2. The ACE order is important. If the group allow ACE had appeared in the list before the owner deny ACE, then Sally would be allowed write access to the file.

If the user sally, who is a member of the group staff, requests READ_DATA (r) and EXECUTE (x) access, the ACL evaluation will proceed as follows:

- ▶ The s:(OWNER@):a... ACE is processed because sally owns the file.
 - READ_DATA is allowed because that bit is set in the ACE_MASK.
 - EXECUTE is not yet allowed because it is not set in the ACE_MASK.
- ▶ The s:(OWNER@):d... ACE is processed because sally owns the file.
 - EXECUTE is not yet denied because it is not set in the ACE_MASK.
- ▶ The s:(GROUP@):a... ACE is processed because sally is a member of the group staff, which owns the file.
 - EXECUTE is allowed because that bit is set in the ACE_MASK.
- ▶ All requested permission bits have now been allowed. No further ACLs are processed, and the requested access is granted.

If the user joe, who is a member of the group sales, requests READ_DATA (r) and WRITE_DATA (w) access, the ACL evaluation will proceed as follows:

- ▶ The g:sales:d... ACE is processed because joe is a member of the group sales.
 - READ_DATA is not denied because it is not set in the ACE_MASK.
 - WRITE_DATA is denied because that bit is set in the ACE_MASK, and WRITE_DATA has not yet been allowed by a previous ACE.
- ▶ No further ACEs are processed, and the requested access is denied.

If joe requests just READ_DATA (r) access, the ACL evaluation will proceed as follows:

- ▶ The g:sales:d... ACE is processed because joe is a member of the group sales.
 - READ_DATA is not denied because it is not set in the ACE_MASK.
- ▶ The s:(EVERYONE@):a... ACE is processed.
 - READ_DATA is allowed because it is set in the ACE_MASK.
- ▶ All requested permission bits have now been allowed. No further ACLs are processed, and the requested access is granted.

Relationship between NFS V4 ACLs and UNIX permissions

Those familiar with standard UNIX read (r), write (w), and execute (x) permission bits may want to know how these bits correlate with the bits in the ACE_MASK. As you can probably tell from the ACE_MASK definitions above, the r, w, and x permission bits basically provide the same access as the standard UNIX permission bits.

For example, as with UNIX permissions, the w permission bit in a directory's ACL has to do with creating, deleting, and renaming files and subdirectories within that directory, rather than changing the contents of those files and subdirectories.

The difference comes into play when mapping the rwx bits to user (owner), group, and other. This mapping is unspecified in RFC 3530. Here is an example of the mapping we observe in AIX.

Consider a file with the following ACL:

```
*
* ACL_type   NFS4
*
*
* Owner: sally
* Group: staff
*
s:(OWNER@):  a    cCs
s:(OWNER@):  d    o
s:(GROUP@):  a    rRxadcS
s:(GROUP@):  d    wpWDACo
s:(EVERYONE@): a    rwpRxadcS
s:(EVERYONE@): d    WDACo
```

Applying `ls -l` to the file shows:

```
-rwxr-xrwx  1 sally  staff          0 Jul 30 11:20 testfile
```

Initially, you might think that the bits will map straight from OWNER@ to user, GROUP@ to group, and EVERYONE@ to other. As you can see from the example, this is not the case. Here you see that the user permissions show as rwx, where the s:OWNER@:a ACE has none of those bits set. Furthermore, even though the `ls -l` output makes it look like the user sally has write access to the file, she actually does not. Evaluating the ACEs from top down, write access is denied by the s:GROUP@:d entry.

Based on this, we must draw the conclusion that you cannot use the standard UNIX permissions bits to reliably predict access when using NFS V4 ACLs.

3.4.5 NFS V4 ACLs: administration

In AIX, NFS V4 ACLs (and AIXC ACLs) can be administered from either the NFS server or an NFS V4 client via the command line or via the AIX Web-based System Manager (WSM).

Manipulating ACLs via the command line

ACLs can be administered using the following commands:

aclget	Writes the textual representation of an ACL to standard output or to a named file.
aclput	Replaces the contents of an ACL from a textual representation provided either from standard input or from a named file.
acledit	Retrieves the ACL's textual representation into a text editor (specified by the EDITOR environment variable) and then replaces the ACL from the modified text.
aclconvert	Converts an ACL's format from either AIXC to NFS4 or from NFS4 to AIXC. The translation is not necessarily straightforward. <i>Use this with caution</i> , and make sure that the end result is what you intended.
aclgettypes	Returns a list of the ACL types supported by the file system that contains a given file or directory.

Note: The `aclconvert` and `aclgettypes` commands are new in AIX 5L V5.3.

Manipulating ACLs via WSM

You can also use the AIX Web-based System Manager (WSM) to manipulate ACLs. This may be easier for novice users to grasp, because it is GUI-based. We think, however, that WSM will be useful for only the most basic operations and that experienced administrators will mainly use the command line utilities.

Here is an example of how to use WSM to change an ACL:

1. Start up the WSM console and double-click the **File Systems** icon. Then double-click the **Overview and Tasks** icon. You will see a window that looks like Figure 3-7 on page 74.

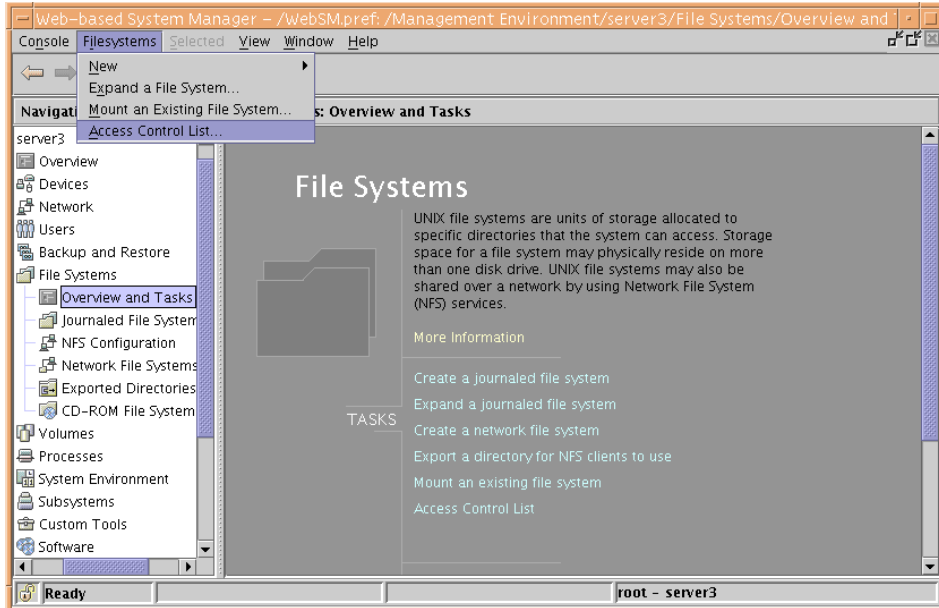


Figure 3-7 WSM File Systems → Overview and Tasks screen

2. Choose **Access Control List** either from the main window or from the **Filesystems** menu. This opens a window like Figure 3-8.

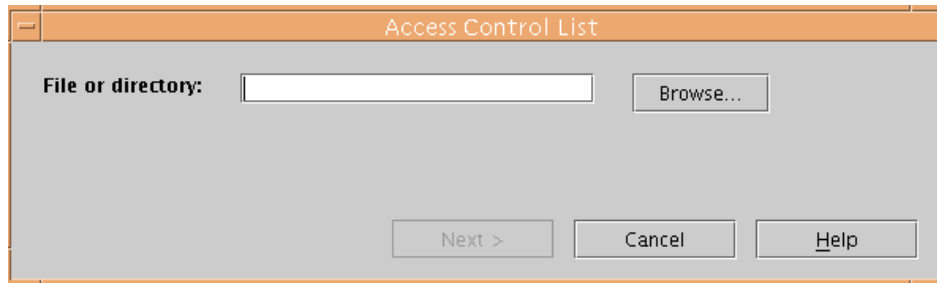


Figure 3-8 WSM ACL file or directory name prompt

3. Either: Type in the full path name of the file or directory whose ACL you would like to modify, or click **Browse** to choose the file or directory from the GUI. After you have entered the name, either type Enter or click **Next**.

4. This opens a window similar to Figure 3-9. Make sure that **Edit ACL** is selected and click **Next**.

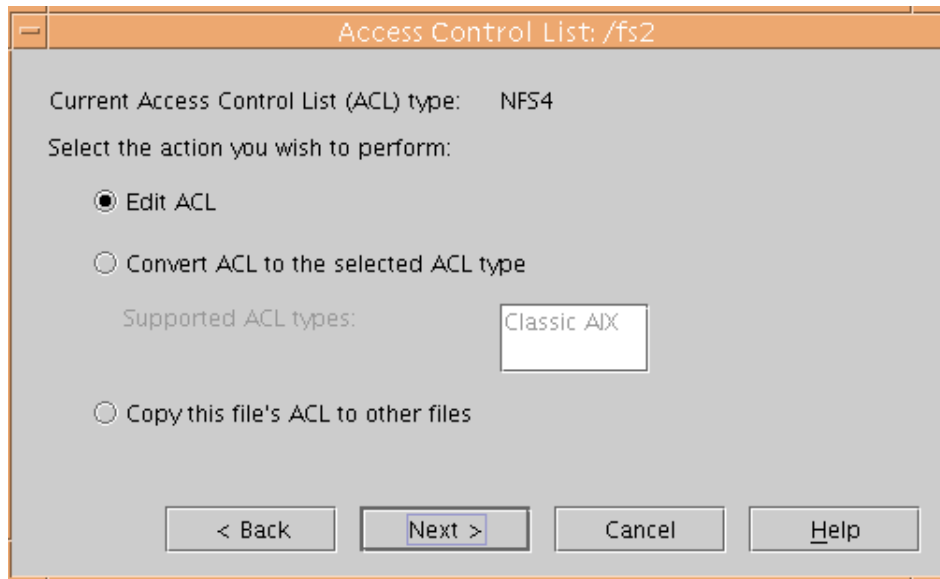


Figure 3-9 WSM ACL operation selection

5. The window shown in Figure 3-10 opens. Select the ACE you would like to change and click **Edit**.

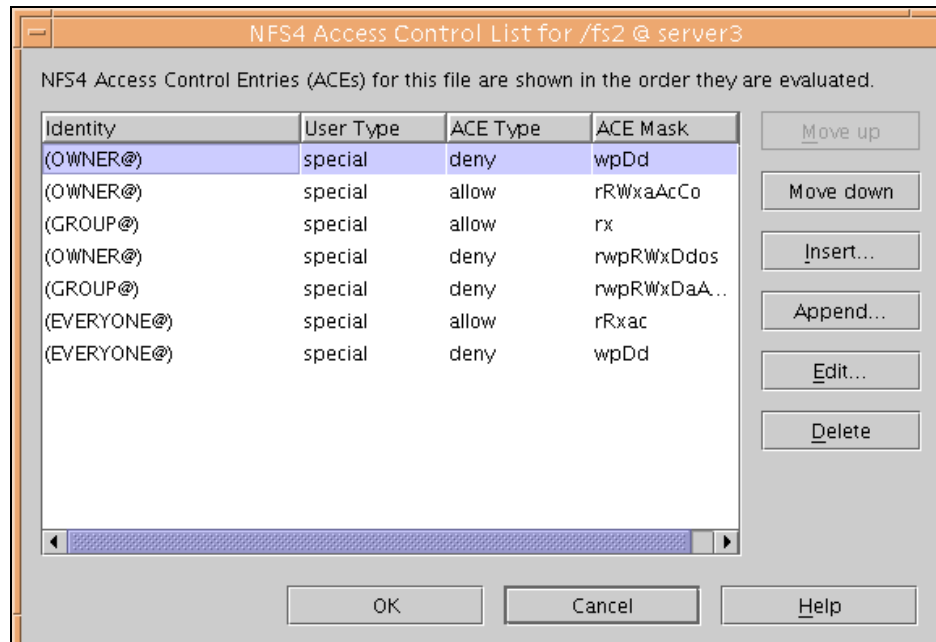


Figure 3-10 WSM ACL edit screen

- The ensuing window has two tabs: General and Access Mask. Under the General tab, you can set the user type and identity, the ACE type, and the ACE flags (inheritance, audit and alarm). If you select the **Access Mask** tab, you will see a window like Figure 3-11.

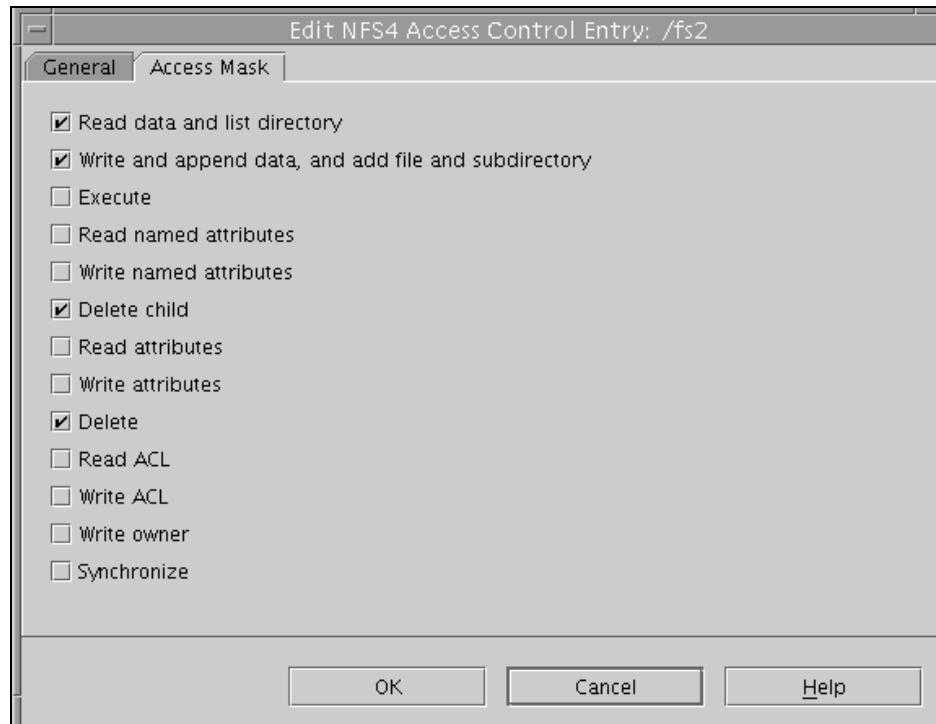


Figure 3-11 WSM ACE mask screen

- Make sure that the selected access mask entries are the ones you want and click **OK**. You will be returned to the ACL edit screen (Figure 3-10 on page 76).
- Repeat steps 5 and 6 for other ACEs that you want to change. After you have finished with all of the ACEs that you want to change, click **OK** on the ACL edit screen. You will see a pop-up window that indicates the status of the operation. When you are done viewing the status, click **Close** on that window.

This concludes the WSM example.

Use of the **chmod** command

When working with files and directories that have NFS V4 ACLs, **chmod** can only be used to set UNIX permission bits that are outside of what is stored in the ACL. These are the setuid bit, the setgid bit, and the sticky bit (Table 3-8 on page 78).

Table 3-8 *Chmod operations compatible with NFS V4 ACLs*

Chmod command	Description
chmod u+s chmod u-s	Set or unset the setuid bit.
chmod g+s chmod g-s	Set or unset the setgid bit.
chmod +t chmod -t	Set or unset the sticky bit.

Important: Using the **chmod** command to manipulate the rwx permission bits, either in octal form (for example, 755) or in symbolic form (u+x) replaces the NFS V4 ACL with an AIXC ACL, wiping out the original permissions that were on the file or directory.

Never use the octal form of the **chmod** command if you are using NFS V4 ACLs. Even if you think that you are leaving the rwx bits alone, using the octal form will replace the NFS V4 ACL with an AIXC ACL.

Note: If you use **chmod** to manipulate rwx permission bits on an NFS client and then (again on the client) run **aclget** on the file, the ACL will still appear to be an NFS V4 ACL. However, it will be an AIXC ACL on the NFS server. The NFS protocol translates AIXC ACLs that have extended permissions disabled to look like NFS V4 ACLs at the client.

ACL inheritance and umask

Does the UNIX umask have any impact on inherited ACL settings when creating a new file or directory? The answer is no. The umask has no effect on inherited ACL settings when using NFS V4 ACLs.

For example, if a directory has the following ACL, a file created in that directory will have the same ACL, even if the umask is set to 777.

```
*
* ACL_type   NFS4
*
*
* Owner: root
* Group: system
*
s:(OWNER@):  a      rwpRWxDaAdcCs  fidi
s:(OWNER@):  d      o          fidi
s:(GROUP@):  a      rwpRxadcS      fidi
```



```

s:(GROUP@): d      WDACo  fidi
s:(EVERYONE@): a    rwpRxadcs  fidi
s:(EVERYONE@): d      WDACo  fidi

```

ACL inheritance and move vs. copy

Table 3-9 describes the impact that the UNIX `mv`, `cp`, and `cp -p` commands have on a file's ACL (assuming that the destination file system also supports NFS V4 ACLs).

Table 3-9 UNIX commands and impact on ACLs

Command	Resulting file ACL
<code>mv</code>	The file retains the same ACL that it had in the original location if the source and target file systems are the same. If not, then ACL assignment occurs as per <code>cp</code> below.
<code>cp</code>	The file inherits its ACL from the directory where it is being placed, just as if it were a newly created file.
<code>cp -p</code>	The file retains the same ACL it had in the original location.

Directory structure and ACLs

There are many different ways you can choose to organize your data into a directory structure and implement ACLs to control access to that data. For example, you might choose from two different methods to control read access to data:

1. Controlling access at the directory (container) level by maintaining uniform access permissions on files and subdirectories within a directory
or
2. Leaving access wide open at the directory level and setting unique access restrictions at each individual file.

Choosing a method depends on your requirements. Each method has its own characteristics, some of which are as follows.

Characteristics of method #1:

- ▶ It is easier for most people to keep track of permissions when files with like permissions are grouped together.
- ▶ Permissions can be easily changed via a bulk replacement of the ACLs. (See "Maintaining an existing directory structure" on page 82.)
- ▶ If a subset of files in a directory need to have their permissions changed, the files must be moved to a different directory. Directory location changes can be

disruptive to operations. (for example, symbolic links and other path name references might have to be updated.)

- ▶ A file could inadvertently inherit incorrect permissions if it is placed in the wrong directory.
- ▶ If a directory has less restrictive access permissions than a parent directory, an NFS client could possibly mount that directory on a path that has more open access. A user whose access would normally be blocked by the parent directory might then be able to access the directory via the mounted path.

Characteristics of method #2:

- ▶ Every file's permissions can be tailored to its unique access requirements.
- ▶ A file does not have to be moved when its permissions need to be different than the files around it.
- ▶ It is more difficult to keep track of the different file permissions.
- ▶ It is easy to mistakenly overwrite a file's permissions via a bulk update, and it is relatively complicated to detect a mistake and then to restore the correct permissions after a mistake has happened. (You would have to know via some external source what the original permissions were.)
- ▶ To prevent inadvertent access, each file would have to start out with the most restrictive set of permissions (via inheritance), requiring manual intervention by the user to share that file with others.

Maximizing the benefits of ACL inheritance

If you choose to implement method 1 on page 79, you should organize your directory structure to maximize the use of ACL inheritance. To do this, carefully plan out your directory structure so that files and subdirectories with the same access requirements are collocated under a single parent directory.

This example helps illustrate this:

An organization has three departments: engineering (eng), sales, and human resources (hr). People from each of the departments are working on two different projects: projA and projB. For business reasons, the two projects must be entirely separate, and people working on one project must not be able to access the data belonging to the other project.

Each department has its own directory for data, and each department creates separate project directories under its directory. The resulting directory structure is depicted in Figure 3-12 on page 81.

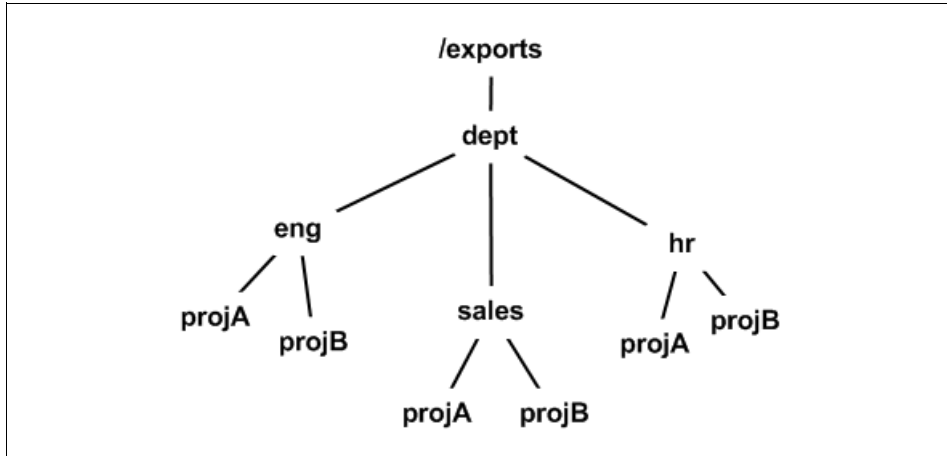


Figure 3-12 Directory structure that makes poor use of ACL inheritance

This structure has the project directories replicated under each department. If a permissions change has to be made to one of the projects, those changes must be made in three different places.

A directory structure that better lends itself to managing the project permissions is depicted in Figure 3-13.

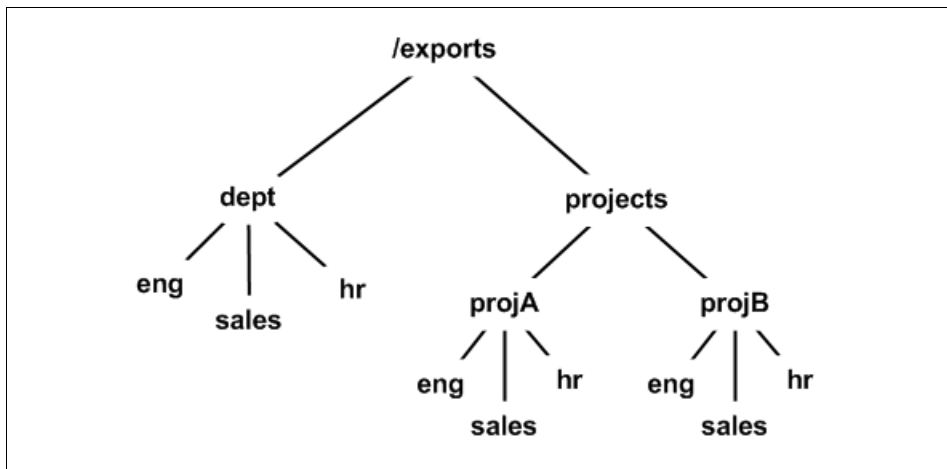


Figure 3-13 Directory structure that makes better use of ACL inheritance

This structure has a separate projects directory where the permissions for each special project can be managed in one place. The departments still put their own non-project-related data under the dept directory.

Maintaining an existing directory structure

Inheritance takes care of setting permissions for newly created files and directories, but it does not affect permissions for existing files and directories. No matter how carefully you plan ahead, you will eventually need to change permissions on an existing directory structure and all of the files it contains.

There are two possible ways to make a large-scale permissions change:

1. Make the change to one file or directory and then propagate the change to other files and directories by copying the whole ACL from the file or directory that you already changed.
2. Incrementally change the ACL for every file and directory.

The first method is simpler to implement, but all files and directories being changed will take on exactly the same permissions, eradicating any variation that may have existed. This may be a good thing or a bad thing, depending on your structure. The second method is much more complex to implement, but it does allow for other differences to exist in the ACLs. This is another example where carefully planning your directory structure around permissions requirements can make administration easier.

The rest of this section illustrates possible ways to implement the ACL propagation method #1 above.

It is possible to propagate an ACL to an entire directory structure using a combination of **aclget** and **aclput** as follows:

```
aclget dirname | aclput -R dirname
```

You can use a different source and directory name, or you can specify the same directory name for both source and destination to copy a directory's ACL to all of its descendants (including itself).

Caution: Only use **aclput -R** on a directory structure that has a uniform permissions structure. The command will make a wholesale replacement of all existing ACLs at and below the specified directory. Any variations in ACLs that previously existed will be lost.

Using the **aclget | aclput** combination is convenient, but there are drawbacks:

- ▶ If you mistype the name of the source directory, you will completely wipe out the permissions in the destination directory. This can be remedied quickly by reissuing the command with the correct source name, but meanwhile you will have blocked access to any user or application that tries to access the data.
- ▶ The **aclput -R** command stops at the first error it encounters, leaving the rest of the files untouched.

The sample script in Example 3-1 addresses these issues. It does not attempt to run `aclput` if either the source or destination does not exist, and it runs `aclput` on each individual file and directory so that all possible ACL changes will be made.

Example 3-1 Sample script for copying an ACL (with recursive option)

```
#!/usr/bin/ksh
#
# copy_acl.sh
#
# Copy the ACL for the given source file/directory to other files/directories
#

# Name of this script
scrname=${0##*/}

#
# Functions
#

function usage {
    echo "Usage: $scrname [-R] <source> <dest>"
    echo "  where"
    echo "    -R indicates a recursive copy"
    echo "        (copy ACL to all files and directories below and including"
    echo "        the destination.)"
    echo "  <source> = the name of the file or directory to copy the ACL from"
    echo "  <dest>   = the name of the file or directory to copy the ACL to"

    exit 1
}

if [[ $# -eq 0 ]]
then
    usage
fi

#
# Process input parameters
#

if [[ "$1" = "-R" ]]; then
    SETSUBTREE="true"
    shift
else
    SETSUBTREE="false"
fi

if [[ -n "$1" ]]; then
```

```

        SRC_NAME="$1"
    else
        usage
    fi

    if [[ -n "$2" ]]; then
        DEST_NAME="$2"
    else
        usage
    fi

    #
    # Initialize other variables
    #

    NBERR=0
    TMP_ACLFILE="/tmp/.AIXACL_$$"

    if [[ -e "${SRC_NAME}" ]]; then
        aclget -o "${TMP_ACLFILE}" "${SRC_NAME}"
        NBERR=$?
    else
        echo "Source \"${SRC_NAME}\" does not exist"
        NBERR=1
    fi

    if [[ "${NBERR}" -eq 0 ]]; then
        if [[ -e "${DEST_NAME}" ]]; then
            if [[ -d "${DEST_NAME}" && "${SETSUBTREE}" = "true" ]]; then
                find "${DEST_NAME}" -print | while read NAME
                do
                    aclput -i "${TMP_ACLFILE}" "${NAME}"
                    (( NBERR += $? ))
                    ls -dl "${NAME}"
                done
            else
                aclput -i "${TMP_ACLFILE}" "${DEST_NAME}"
                (( NBERR += $? ))
                ls -dl "${DEST_NAME}"
            fi
        else
            echo "Destination \"${DEST_NAME}\" does not exist"
            NBERR=1
        fi
    fi

    rm -f "${TMP_ACLFILE}"
    exit ${NBERR}

```

3.4.6 NFS V4 ACLs: permissions scenarios

The following scenarios help illustrate how NFS V4 ACLs can be applied.

1. Restricting home directory access to just the associated user, and not allowing users to change permissions and open up their home directories to others.
2. Making sure that a particular group is denied access to a set of data.

ACL scenario 1: home directories

Users' home directories can be a collector for all types of data. Users often place data in their home directory while working on it. The data may have come from a directory with strict access controls, and the home directory's permissions should not allow wider access to that data. One way to manage this is to lock down each home directory so that only its associated user can access it.

This is difficult to do with standard UNIX permissions. There are two basic options:

- ▶ Make the user the owner of the directory and allow only owner access.
Because the user owns the directory, he or she can change its permissions, which we do not want to allow.
- ▶ Create a group for each user, where the user is the only member; make the home directory owned by root and the user's group; and allow only owner and group access.
The user cannot change the directory permissions, but this option requires maintaining a whole set of groups, one for each user.

This is easier to do with NFS V4 ACLs. Make root the owner of the directory and add a user ACE to allow the user access to the directory. This is how that ACL would look:

```
*
* ACL_type   NFS4
*
*
* Owner: root
* Group: system
*
s:(OWNER@):  a      rwpRWxDaAdcCs  fidi
s:(OWNER@):  d      o      fidi
u:sally(sally@nfsdom1):  a      rwpRWxDaAdcCs
u:sally(sally@nfsdom1):  d      Co
s:(GROUP@):  d      rwpRWxDaAdcCos  fidi
s:(EVERYONE@):  d      rwpRWxDaAdcCos  fidi
```

(Note that the user ACEs do not have to be inherited because files that are created below the directory will be owned by the user.)

The user can open up permissions for files and subdirectories that he or she creates in the directory because the user owns them, but the home directory itself will still block access to those files.

Note: It might be possible to NFS mount a lower-level directory that has more open permissions and gain access to those files, but normally the mount operation is under system administrator control. If mounts are managed correctly, users will not be able to get directly at lower directories underneath the home directory.

ACL scenario 2: block a group's access

If you have two subcontractors working on a project, and you want to make sure that the subcontractors are not able to access each other's data, you can do the following:

Create a group for each subcontractor; put each subcontractor's data in a separate directory structure; and put an ACE at the top of the ACL that denies access to the other subcontractor. (It is important that the ACE be at the top of the list to prevent other ACEs from allowing access before the subcontractor's access is blocked.)

If the groups are company1 and company2, the ACL on company1's data would look like this:

```
*
* ACL_type   NFS4
*
*
* Owner: root
* Group: system
*
g:company2(company2@nfsdom1):      d      rwpRWxDaAdcCos  fidi
s:(OWNER@):      a      rwpRWxDaAdcCs   fidi
s:(OWNER@):      d      o      fidi
s:(GROUP@):      a      rRxadcs  fidi
s:(GROUP@):      d      wpWDACo  fidi
s:(EVERYONE@):   a      rRxadcs  fidi
s:(EVERYONE@):   d      wpWDACo  fidi
```

No matter what the rest of the ACEs are, company2 will be denied access to company1's data.

3.4.7 NFS V4 ACLs: NFS V3 clients

It is possible to make an NFS V3 mount of a file system that contains NFS V4 ACLs. This combination exhibits the following behavior:

- ▶ The NFS server will still grant or deny data access based on the NFS V4 ACLs.
- ▶ The NFS client will not be able to view or manipulate the ACLs directly. For example, an `aclget` command on the client returns the error:

```
aclget: The system call does not exist on this system.
```
- ▶ If the mount is made with the `acl` option (`noacl` is the default), the NFS client will be able to manipulate AIXC ACLs, but not NFS V4 ACLs.

The bottom line: You may have NFS V3 clients mount file systems that use NFS V4 ACLs. ACL inheritance and evaluation will work normally on the server. However, do not attempt to manipulate access permissions directly from the NFS V3 client. Any permissions change at the NFS V3 client will overwrite the NFS V4 ACL with an AIXC ACL.

Unfortunately, there is no good way to block a user on the NFS V3 client from running `chmod` or `aclput` on files or directories that he or she owns. You will have to publish policy and rely on well-behaved users. (You could completely disable the `chmod` and `aclput` commands on the client, but that would also disable them for other client file systems where using those commands is perfectly valid.)

Also keep in mind when using NFS V3 clients that the UIDs and GIDs have to match between server and client.

3.5 NFS V4 host identification

We discuss two forms of host identification: basic identification using IP address and host name, and Kerberos identification by machine principal.

3.5.1 Basic host identification

An NFS V4 server identifies client hosts by the IP address given in the RPC packets. The NFS server turns this IP address into a host name by way of the host resolver, which can get its information from the Domain Name Service (DNS), Network Information Services (NIS), or the local `/etc/hosts` file.

3.5.2 Kerberos host identification

Kerberos authentication uses a unique identifier called a *machine principal* to identify hosts. The machine principal is established when configuring a host into a Kerberos realm. The machine principal name is the fully qualified host name prefixed with `host/` (for example, `host/nfs402.itsc.austin.ibm.com`).

Another way that Kerberos indirectly identifies a host is through the NFS *service principal*. (This is the identification of the NFS service running on the host.) The service principal name is the fully qualified host name prefixed with `nfs/` (as in `nfs/nfs402.itsc.austin.ibm.com`). NFS clients using Kerberos authentication identify NFS servers with this service principal.

3.6 NFS V4 host authentication

NFS servers always identify client hosts by IP addresses and host names, regardless of the authentication method used. The value added with NFS V4 is that when Kerberos authentication is the only allowed security method for an exported directory (see the host authorization section next), the NFS client session must be properly authenticated before gaining access to any of the data in that directory.

Think of NFS V4 authentication of clients being mostly at the user level rather than at the host level. (See 3.3.2, “RPCSEC_GSS user authentication using Kerberos” on page 59.)

Kerberos does authenticate NFS server identities to the clients via the NFS service principal. (See the NFS service principal discussion in the previous section.)

3.7 NFS V4 host authorization

Host authorization in an NFS context means controlling which NFS client hosts can mount exported directories from the NFS server. This is accomplished in AIX with a combination of the `/etc/exports` file and the `exports` command.

Exporting directories from an NFS server is still fundamentally the same in NFS V4 as it was in NFS V3. The main difference is that NFS V4 has added the new security-related options shown in Table 3-10 on page 89.

Table 3-10 New /etc/exports security options for NFS V4

Option	Description
vers	Controls which version NFS mounts are allowed. Possible values are 2, 3, and 4. Versions 2 and 3 cannot be enforced separately. Specifying Version 2 or 3 allows access by clients using either NFS protocol versions 2 or 3. Version 4 can be specified independently and must be specified to allow access by clients using Version 4 protocol. The default is 2 and 3.
sec	Controls which security methods are allowed. Possible values are: sys (UNIX authentication) dh (DES authentication) krb5 (Kerberos, authentication only) krb5i (Kerberos, authentication, and integrity) krb5p (Kerberos, authentication, integrity, and privacy) none (Allow mount requests to proceed with anonymous credentials if the mount request uses an authentication flavor not specified in the export. Otherwise a weak auth error is returned. By default, all flavors are allowed.) In the absence of any sec option, sys (UNIX authentication) is assumed.

The *sec* option is unique in that it can appear more than once in the exports definition for a directory. This allows different *ro*, *rw*, *root*, and access options to be specified for the different security options. For example, hosts using the *sys* security method might only be allowed read access, while hosts using the *krb5* security method might be allowed read and write access.

Note: You cannot specify the same security option in more than one *sec=* stanza in a single exports definition.

Here is a sample /etc/exports line with the new NFS V4 security options:

```
/exports -vers=3:4,sec=krb5:krb5i:krb5p,rw,sec=sys:none,ro
```

For more about exporting directories, see the following publications:

- ▶ *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909
- ▶ The **exportfs** command in *AIX 5L Version 5.3 Commands Reference*
- ▶ The /etc/exports file in *AIX 5L Version 5.3 Files Reference*, SC23-4895



Part 2

Implementing NFS V4

In this part we introduce you to the planning and implementation methodologies that were used while preparing this book.



Planning for NFS V4

The most crucial part of any new software and hardware implementation project is the planning phase. If we get this part of the project right, then the design and implementation phases become easier. Therefore, in this chapter we discuss and provide detailed information about planning a reliable infrastructure for NFS V4 deployment. The underlying concepts are complex, but it is easy to plan, design, and implement when the concepts are well-understood.

Considerations that must be taken into account when beginning any planning phase include:

- ▶ Currently available and deployed hardware, software, and applications
 - Consider the following questions:
 - Do you have an inventory of your current assets and their usage?
 - Do you have a logical overview of your infrastructure?
- ▶ The future IT strategies of your company
 - Do you want to have centralized user management?
- ▶ Business-driven design and implementation issues
 - Do you have the need to exchange data with other customers or departments?
 - Do you have other (third-party) applications to be considered?

Important: Careful planning for your NFS V4 deployment is crucial if you want to properly use the enhanced security features provided by Kerberos Authentication as well as NFS V4 ACLs.

We also look at the necessary planning requirements if you want to use NFS V4 without some of the enhanced features, simply as a replacement to your current NFS V3 environment.

This chapter discusses the following:

- ▶ Deployment of NFS V4 in general
- ▶ Mandatory requirements
- ▶ Identification methods
- ▶ NFS Authentication methods
- ▶ Authorization methods
- ▶ Choosing the appropriate file system types
- ▶ NFS protocols and namespace considerations
- ▶ Sizing and capacity planning considerations
- ▶ Migration considerations

4.1 Deployment of NFS V4 in general

Deployment of NFS V4 should be planned in such a way that a smooth implementation and integration into the existing infrastructure can be achieved. Normally, this is done by:

1. Evaluating the basic conditions for deployment.
2. Defining the architectural design and logical layout.
3. Integrating the design in a sample environment.
4. Defining how the rollout will take place into your infrastructure.

Important: With NFS V4, the following two general design constraints should be taken into account while planning the deployment. These two topics reach a level of primary importance because of functional changes introduced by the NFS V4 standard:

- ▶ NFS domain and Identity Mapping
- ▶ Authentication

We have attempted to develop a logical approach to the planning process for implementing NFS V4. Figure 4-1 on page 96 shows a planning flow chart that we developed and used to build our test environment and to evaluate different migration scenarios. We recommend that you use this chart as a template when approaching your infrastructure planning.

For this redbook project, we did not have an existing environment to migrate from, so we planned and built one from scratch using the flow chart. However, most of you will be looking at this from an existing systems migration point of view. Some considerations in using the flow chart:

- ▶ The flow chart is designed to be modular. So, based on what you have already implemented in your existing environment, you can skip to the next step on the flow chart and continue with the planning.
- ▶ It was impractical to cover all possible permutations when considering a migration path, so in 4.9, “Migration considerations” on page 116, we walk you through some of the scenarios we consider to be most common.
- ▶ Certain parts of the flow chart may not easily plug into your environment, as they are new concepts introduced by NFS V4. On the whole, the flow chart should serve as a good building block for your planning considerations.

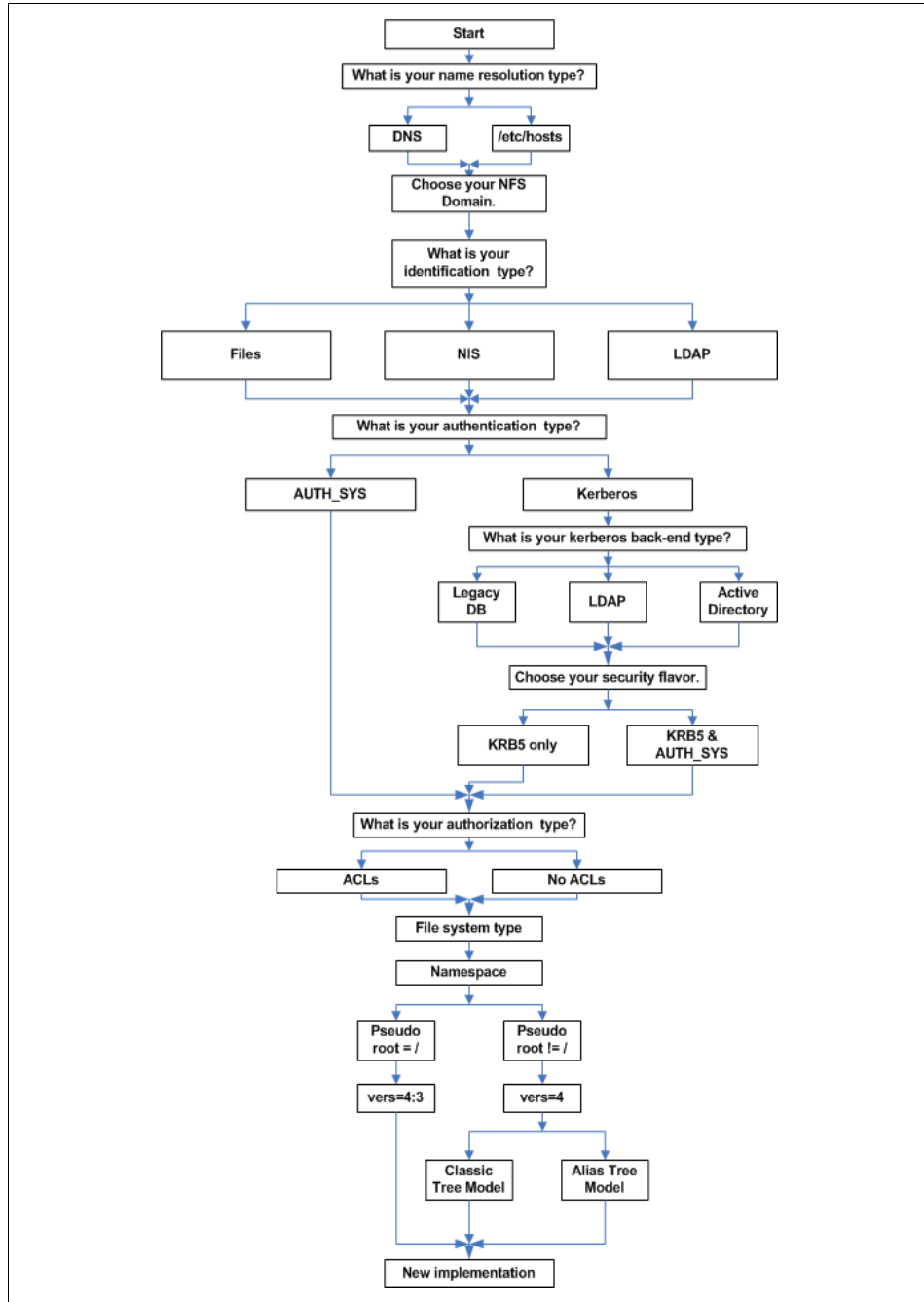


Figure 4-1 Flow chart: implementation process decisions

4.2 Mandatory requirements

The “What is your name resolution type?” and “Choose your NFS Domain” questions in the flow chart (Figure 4-1 on page 96) are two areas that dictate how you plan and implement your design.

4.2.1 What is your name resolution type?

Whatever name resolution option you choose (/etc/hosts or DNS), it must be in 100% working order. Name resolution plays a major part throughout the planning and implementation phase.

NFS V4–related name resolution

NFS V4 does not support referencing hosts by IP address. If applying NFS export restrictions based on client host names is desired, proper setup of host name resolution is essential for correct operation. Therefore, you should observe the following guidelines to avoid most name-resolution-related problems:

- ▶ Ensure valid forward and reverse name mappings for all systems on your network. The machines that will participate in your NFS domain or Kerberos realm must have resolvable name entries, forward and reverse. This means that if you are using DNS, an entry for all participating machines must exist that maps the host name to an IP address, and a reverse entry must exist for that IP address mapping it back to the original host name. If you are using /etc/hosts, the same applies, but you also have to ensure that all participating machines have the same common information within /etc/hosts file.
- ▶ Ensure fully qualified domain name (FQDN) to host name mapping. What a system considers its host name is also important. When NFS V4 or Kerberos creates the name of the host principal, it uses the output from the gethostname() call to resolve the so-called canonical name (CNAME) and generates the principal name. If this host principal does not match the host principal that is held in the KDC’s database, then the application will fail, saying it cannot find the principal.

For example, if a host resolves the CNAME bigserv for its host name bigserver, yet its host principal is host/bigserver.ibm.com@IBM.COM, it will attempt to find a host principal of host/bigserver@IBM.COM and fail.

- ▶ Ensure that both the localhost and loopback interface names (by default, IP address 127.0.0.1) can be resolved in any case.

You may take RFC1537, *Common DNS Data File Configuration Errors*, and RFC2181, *Clarifications to the DNS Specification* into account while planning your name resolution infrastructure.

Tip: Using the `/etc/hosts` file as the only form of name resolution on all systems will work, but using a DNS server provides a more reliable and easily managed solution to your name resolution. This gives you a single point to manage your name resolution rather than having to add or change entries on each host in your network.

Kerberos-related name resolution

Traditional UNIX Kerberos V5 implementations use a flat file (`/etc/krb5/krb5.conf` or `/etc/krb5.conf`) for hostname-to-Kerberos realm mapping. This is similar to the `/etc/hosts` file being used for the name-to-ipaddress mapping and vice versa. The Kerberos configuration file contains two major pieces of information.

- ▶ The DNS domain to Kerberos realm mappings
- ▶ A list of KDCs for each Kerberos realm

Obviously, this method does not scale, so just as DNS now serves the purpose of the old `/etc/hosts` file, DNS can also be used to provide Kerberos configuration.

Kerberos can use DNS as a service location protocol by using the DNS SRV record as defined in RFC2052. In addition, Kerberos can use a TXT record to locate the appropriate realm for a given host or domain name. These DNS entries are not required to run a Kerberos realm, but they do eliminate the need for manual configuration of clients. With these DNS records, Kerberos clients can find the appropriate KDCs without the use of a configuration file.

Windows will establish the necessary SRV records automatically when an Active Directory domain is created. Those using UNIX for their KDCs can create these DNS entries manually in their zone files as a convenience to the DNS clients.

4.2.2 Choosing your NFS domain

From a traditional point of view, sharing data between servers and clients is done by the use of NFS. All features of NFS V3 are fully supported by the NFS implementation in AIX 5.3.

If you do not wish to take advantage of the security enhancements offered by NFS V4, very little has to be installed or implemented. The changes that do have to be made are as follows:

- ▶ NFS V4 requires that the NFS domain be set on all servers and clients.
- ▶ NFS V4 changes the way users and groups are looked at. Previous versions of NFS used UIDs and GIDs. NFS V4 now changes this to `user@domain` and `group@domain`.

If you have not already done so, we recommend that you take the time to read through Chapter 2, “What’s new in NFS V4?” on page 11 and Chapter 3, “Enhanced security in NFS V4” on page 45 for a better understanding of what has changed in NFS V4 and the new security enhancements offered. The topics that are discussed from here on require you to have a basic understanding of the new concepts being brought in by NFS V4. The new protocol requires a major change in your mind set; you can no longer think along the lines of NFS V2 and NFS V3. We cannot stress this enough!

The NFS domain name is:

- ▶ By the standard defined by the RFC, bound to the DNS domain name.
- ▶ Case insensitive, upper-case characters will be treated as lower case during runtime.
- ▶ Upper-case characters will be converted automatically to lower case when using the preferred method to change or set the NFS domain: the **chnfsdom** *<NFS Domain Name>* command.

We recommend that you do not edit the `/etc/nfs/local_domain` file manually but if you do so and use upper-case characters, the **chnfsdom** command will show upper-case characters.

Important: The AIX implementation does not require the NFS domain to match your DNS domain. You can call your NFS domain pretty much anything you like, but keeping a relationship to your DNS domain simplifies managing the environment. It also helps make sure that the name is unique.

For simplified management, we logically linked our NFS domain to the DNS domain. In our sample environment our DNS domain was:

```
itsc.austin.ibm.com
```

Therefore our NFS domain became:

```
itsc.austin.ibm.com
```

4.3 Identification methods

For a fully supported login process in an AIX environment, you need both user identification and user authentication. In this section, we describe some details you should take into account while planning user identification in your environment. See 3.2, “NFS V4 user/group identification” on page 48 for general information on user identification.

4.3.1 Selecting the user/group repository

User identification comprises all the necessary information about what user IDs exist and what the attributes are for these user IDs. This information must be consistent so that you can accommodate the following considerations:

- ▶ Management of identities
How do you intend to manage creation, modification, and deletion of users and groups?
- ▶ Flexibility of the end user
Will end users each use only one system, or will they also log on to other systems using the same user name and password combination?
- ▶ Scalability / availability
How many NFS clients will you have? What happens if you have to integrate new systems into your environment? How much server downtime can you tolerate?
- ▶ Scope of data sharing
How widely will end users share data: only within each department, or throughout the entire organization?
- ▶ Access control
Access control decisions are based on user and group identity. To properly control access to a resource, you must make sure that all accessing entities are uniquely identified. With NFS V4, the identifiers are strings: `user@nfs_domain` or `group@nfs_domain`. If you also need to support NFS V3 clients in your environment, you will also need to make sure that the numeric UIDs and GIDs are unique.

User and group identities are maintained in some kind of repository. This repository typically is implemented through one of the following methods:

- ▶ Standard UNIX `/etc/passwd` and `/etc/group` files
- ▶ Network Information Services (NIS)
- ▶ Lightweight Directory Access Protocol (LDAP)

Use the following guidelines when choosing which method to use for your user/group repository.

Using `/etc/passwd` and `/etc/group` might be desirable if each user uses only one system. You will need to somehow maintain a central clearing house for user identities to make sure that you choose names and IDs that are unique.

If users log into multiple systems using the same user name and password, you will need to replicate your `/etc/passwd` and `/etc/group` information to all

participating systems. This becomes very difficult to manage when the number of systems is more than just a few. In this situation, you will probably want to use a centrally managed repository such as NIS or LDAP.

If you are using Kerberos authentication with NFS, we recommend that you choose LDAP, using the schema defined in RFC2307 as your user/group repository. LDAP supports Kerberos integrated logon, but NIS does not.

Note: The RFC2307 schema enables NIS maps to be imported into an LDAP directory. If your existing infrastructure uses NIS to manage user and group information, you may want to consider migrating to LDAP. After the NIS maps have been migrated, AIX 5L and other RFC2307 compliant platforms can use LDAP instead of NIS to access this information.

For further information on how this can be achieved we refer you to the following Technote: *AIX - Migrating NIS Maps into LDAP*, TIPS-0123, at:

<http://www.redbooks.ibm.com/abstracts/tips0123.html?0pen>

You can also use a combination of methods for your user/group repository. For example, if your security policy does not allow the same password to be used on multiple systems, you could maintain the password locally and still use a central repository such as NIS or LDAP for the rest of the user and group information.

4.3.2 Other identification considerations

If your organizational needs require that you operate across different NFS domains for data access, you must use identity mapping to map users and groups from one NFS domain to the other. Enterprise Identity Mapping (EIM) from IBM provides a way to manage and perform this cross-domain mapping.

Note: Use of EIM requires LDAP.

For more information about EIM, refer to the *AIX 5L version 5.3 Security Guide* and the IBM Redbook *Windows-based Single Signon and the EIM Framework on the IBM @server iSeries Server*, SG24-6975.

4.4 NFS Authentication methods

Before planning the NFS V4 authentication method used in your environment, we suggest reading 3.3, “NFS V4 user authentication” on page 59. With NFS V4, you have the following choice:

- ▶ AUTH_SYS
- ▶ Kerberos

4.4.1 AUTH_SYS method

By default, NFS uses the AUTH_SYS method to authenticate user identities.

Under the AUTH_SYS security flavor, the user is authenticated at the client, usually via a logon name and password. The NFS server trusts the user and group identities presented by its clients.

For example, if tight physical security cannot be maintained on a network, it would be easy for someone to bring in a Linux laptop, set the IP address to be the same as a valid NFS client, and connect the laptop to the network using the valid client’s connection. When connected, the person can set up any user account desired on the laptop. That way, someone who knows what they are doing can gain access to any of the exported data on an NFS server.

If someone gains administrative control of an NFS client, or has control of a machine pretending to be a valid NFS client via IP address spoofing, it is easy to masquerade as any valid NFS user.

Because of this vulnerability, you should not use AUTH_SYS user authentication if controlling access to your data is important.

4.4.2 Deploying Kerberos

To take advantage of features offered by NFS V4, we operate under a Kerberos authentication environment. If your organization has a Kerberos infrastructure in place, you can use the existing KDC. This can be done only if the installed version is compatible with the software requirements for NFS V4. IBM provides a detailed migration path for customers who have previous versions of Network Authentication Services (NAS). If you plan to set up a new Kerberos infrastructure to support NFS V4, we provide some guidelines in this section.

See Appendix A, “Kerberos” on page 243 for more information. Also, detailed information can be found in *IBM Network Authentication Service Version 1.4 for AIX, Linux, and Solaris Administrator’s and User’s Guide*. This document is made available with fileset krb5.doc.en_US on the AIX 5.3 Expansion CD.

The need for Kerberos

Kerberos has grown out of the need for a secure protocol that enables users to identify themselves to applications and services. Normally, information is transmitted across the network in an insecure form, which is usually just plain readable text. Kerberos removes many of the risks by encrypting passwords, reducing or eliminating the use of passwords, and providing optional methods to validate data integrity and encrypt data.

The main components of the Kerberos protocol implementation are:

- ▶ Key Distribution Center (KDC)
- ▶ Principals and realms
- ▶ Tickets
- ▶ Services

In addition to data encryption and integrity checking, you can use NFS V4 with Kerberos authentication to help control which clients can access exported directories on an NFS server. We demonstrate this in Figure 4-2.

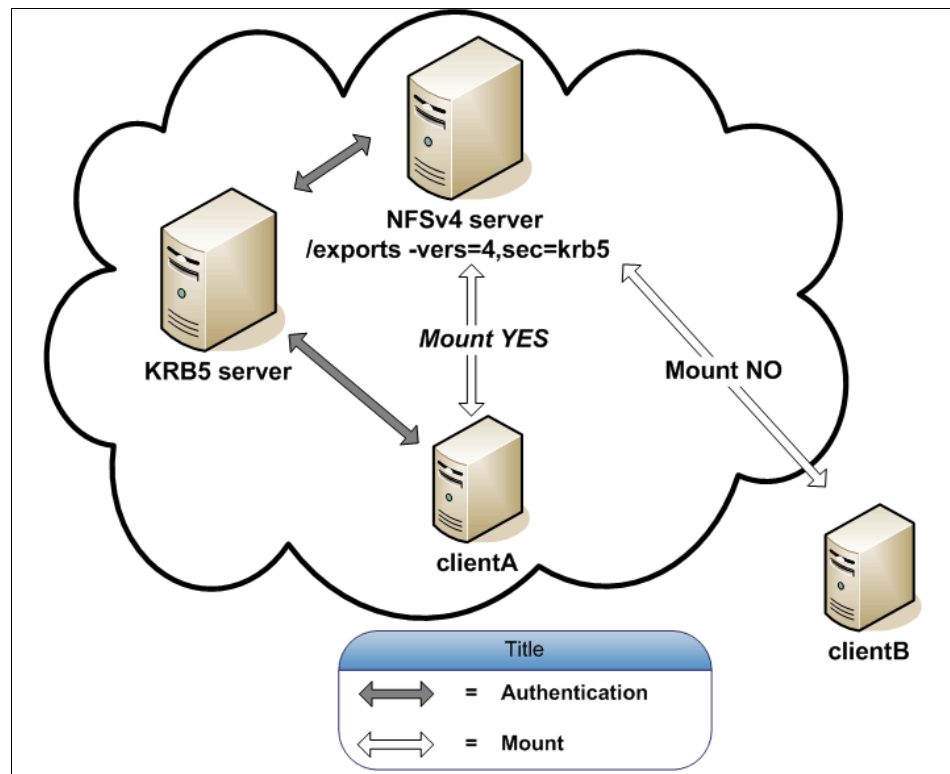


Figure 4-2 NFS V4 Kerberos authentication on system level

In Figure 4-2 on page 103, clientB cannot mount the exported file system because the client is not:

- ▶ In the Kerberos realm
- ▶ Authenticated to Kerberos

This authentication checking was not possible with NFS V3, and to achieve a similar control over mounts it is very common to use the `exportfs` option `-access=Client[:Client]` within the file `/etc/exports`. Sometimes the list of clients becomes very large and the server exports list must be changed every time a client is added, deleted, or renamed.

With NFS V4 and Kerberos authentication, with the option `sec=krb5`, the client is required to be authenticated before contacting the NFS V4 server. This means that even if clientA resides in the same Kerberos realm as the NFS V4 Server, a mount request will fail if clientA itself or the user is not authenticated.

KDC considerations

Authentication requests to the KDC can be easily handled with today's available processors, therefore a single or dual processor machine should suffice for thousands of clients.

Note: This applies only to UNIX-based systems running a KDC. Windows domain controllers function as much more than just a Kerberos KDC and may have a higher server-to-client ratio than a dedicated UNIX KDC.

The KDC server will be one of the most important servers in your network, so:

- ▶ The system should be running 24x7.
- ▶ Planning for disaster recovery should be taken into account. Your KDC database should be replicated if possible.
- ▶ The KDC server should ideally not be used for any other purpose because if the KDC is compromised, all Kerberos principals are compromised.

System time synchronization

Kerberos requires that the system time be reasonably close throughout the network (within five minutes by default). Before beginning with the implementation, you should set up a mechanism to automatically synchronize time throughout the network between the servers and the clients. The following are possible methods for synchronizing time:

- ▶ AIX timed daemon.
- ▶ NTP (Network Time Protocol, official home page at: <http://www.ntp.org/>)

- ▶ In a small business network, it should suffice to use the AIX **setclock** command, scheduled through a crontab entry, to synchronize the system time between the client and KDC server.
- ▶ On Windows OSs, use of NTP on the server and client is also supported.

Note: The default value for maximum clock skew is 300 seconds (five minutes). For security reasons, we recommend that you not change this value.

Multi-homed servers

Multi-homed servers use more than one physical network interface with different IP addresses to serve clients. There are several reasons why you would want to do this, including:

- ▶ Subnetworks
- ▶ Load balancing

The KDC server can only be bound to one IP address. This is defined during the setup of the KDC by providing the system host name.

Therefore, mapping between the client and server network interfaces as well as the KDC bound network interface has to be planned and managed.

Network infrastructure

You should take into consideration not only how many authentication clients you will be serving, but also where these clients are located. While the bandwidth requirements for Kerberos authentication are miniscule, the important metric for Kerberos performance is the network latency between clients and the Kerberos KDC. Each authentication exchange requires time for at least one full round trip between client and KDC. Users' authentication requests will become noticeably slow if this latency is long, for reasons such as:

- ▶ Going through a satellite uplink
- ▶ Traversing across DSL connected backbone

Consequently, you should position your KDCs so that they are as close to the clients' network as possible. To support geographically dispersed networks with possibly different types of connections, Kerberos implementations such as IBM Network Authentication Service are capable of using the replication mechanism to set up a KDC server and propagate the KDC database as needed.

Principals and realms

To understand how principal names are generated in Kerberos, we must first understand Kerberos realms. A Kerberos realm is often referred to as an administrative domain. A realm consists of members, which can be users,

servers, services, or network resources that are registered within a KDC's database. Each of these members has a unique identifier that is called a principal. The Kerberos realm is made up of the KDC and all of its principals.

Realm naming considerations

Although the realm can be any ASCII string, conventionally the realm name is the same as the domain name, in uppercase letters. If multiple realms are needed, use descriptive names that end with the domain name, such as CARPROJECT.AUSTIN.IBM.COM or POWERTRAIN.AUSTIN.IBM.COM.

Important: The normal conventions of naming Kerberos realms is to create them using upper case.

In our environment we created two realms:

```
REALM1.IBM.COM
```

and

```
REALM2.IBM.COM
```

Principal naming considerations

By default, every entity contained within a Kerberos installation, including individual users, computers, and services running on servers, has a principal associated with it. The principal is a unique identifier to which KDC assigns tickets. There are three parts of a principal name: the primary, the instance (optional) and the realm. The primary component of the principal name is separated from the instance by a forward slash (/) The realm is separated from the rest of the principal name by the at sign (@):

- ▶ In the case of a host principal, the user name is host/<hostname>@REALM
- ▶ For an NFS V4 service principal, it is nfs/<hostname>@REALM
- ▶ For a user, it is <username>@REALM

Because the principal is a unique identifier, you should plan a strategy for principal naming throughout your Kerberos environment. This is especially true for the NFS service principal names, because NFS V4 must have proper name resolution in place to compile the ticket request. See “NFS V4–related name resolution” on page 97.

Tip: Always use the FQDN for the NFS service principal.

We decided that all NFS V4 Server and NFS V4 full clients in our environment would have the following service principal name:

```
nfs/<hostname>.itsc.austin.ibm.com@REALM
```

User principal names can follow your current naming convention for user logon names. As of today, you should restrict principal names to the printable portion of the POSIX portable character set, which is equivalent to 7-bit ASCII.

4.4.3 Default types of encryption for KDC and security flavors

Several types of encryption are available for use with Network Authentication Service; by default, Triple-DES is used. Multiple encryption types can be used within a Kerberos realm. The NFS client and server will negotiate which of the supported encryption types to use.

NAS 1.4 supports the following types of encryption:

- ▶ aes128-cts, aes256-cts, des-cbc-crc, des-cbc-md4, des-cbc-md5, des3-cbc-sha1, arcfour-hmac, arcfour-hmac-exp

However, the NFS V4 implementation on AIX 5.3 supports only the following types of encryption:

- ▶ des-cbc-crc, des-cbc-md4, des-cbc-md5, des3-cbc-sha1

For the best performance and interoperability, we recommend that you use Single-DES as the standard encryption type on your installation. This may depend, of course, on your overall security strategy. If protecting packet privacy is critical, you might need to stay with Triple-DES encryption.

In addition to the chosen standard Kerberos encryption, you can select between the following security flavors, which are options used during exporting file systems and depend on your security needs:

- ▶ krb5
Use Kerberos V5 protocol to authenticate users before granting access to the shared file system.

Note: We only used krb5, and we recommend that you start with this.

- ▶ krb5i
Use Kerberos V5 authentication with integrity checking (checksums) to verify that the data has not been tampered with in transmission.
- ▶ krb5p
Use Kerberos V5 authentication, integrity checksums, and privacy protection (encryption) on the shared file data. This provides the most secure file sharing, as all traffic is encrypted.

Note that each increasing level of protection carries with it a performance penalty.

4.4.4 NFS client considerations when using Kerberos

This section discusses considerations you should take into account while planning and implementing your NFS V4 clients.

NFS V4 slim client versus full client

NFS V4 introduces the concept of a full client and a slim client similar to what is already used in the *Distributed Computing Environment* (DCE Version 3). This concept does not necessarily apply to other applications that use Kerberos authentication.

By default, every entity contained within a Kerberos installation (including individual users, computers, and services running on computers) has a principal associated with it. In the case of a host principal (machine principal), the principal name is `host/<hostname>@REALM`; for an NFS service principal, it is `nfs/<hostname>@REALM`.

In the AIX 5.3's NFS V4 implementation, two types of Kerberos clients are defined: slim clients and full clients.

- ▶ An NFS client without a dedicated NFS service principal is called a slim client.
- ▶ An NFS client with a dedicated NFS service principal in the form `nfs/<hostname>@REALM` is called a full client. The full client provides stronger security. However, it is the stronger Kerberos-based RPC security that this type of client provides that requires more administrative overhead: You should run the `config.krb5` command on each client with the Kerberos Administrative ID.

If you trust all systems connected to your internal network and authentication at a user level fulfills your needs, then using slim clients would suffice. The following reasons are why you would want to deploy slim clients in your infrastructure:

- ▶ Pre-installed new systems
- ▶ Mass rollout of new systems
- ▶ Unprompted upgrade of clients
- ▶ Share all the same Kerberos realm as well as NFS domain

In addition, there are several choices for installing the slim clients, and these are based on the way your systems are installed:

- ▶ Complete new installation (scratch installation)
- ▶ Cloning by use of a full system backup image

Attention: As this book is being written, a slim client is not capable of automatically mounting file systems that require `RPCSEC_GSS` authorization only. To enable a slim client to automatically mount file systems at boot time, the server exports definition must include `AUTH_SYS` in the first `sec=` stanza.

Kerberos integrated logon

User authentication at a system level is a process where a user claims to have a certain identity and the system has to check whether this is true. For this process, the system requires a unique piece of information about this user (usually a password). When users authenticate, the system challenges them by requesting that they type in their password. The user's response is then compared to the stored unique piece of information, and the request is accepted or denied depending on the outcome of this comparison.

AIX 5L supports load modules that are responsible for identification, for authentication, or both. You can use either one load module, which supports both, or you can specify one load module that is responsible for the identification part, and another that is responsible for authentication. Such a combination of two modules is called a compound module.

We recommend that you use a compound module comprised of `NAS 1.4` for authentication and `LDAP RFC2307` for identification. With `NAS` and `LDAP` in place and `KRB5LDAP` as the authentication module on the clients, you do not have to deal with creating users on every client.

For more about configuring the system to use these authentication modules, see 5.9.4, “Configure the NFS V4 client for integrated login services” on page 170.

For more details about using AIX directory integration to support user authentication and identification, refer to the Security chapter in the redbook *AIX 5L Version 5.3 Differences Guide*, SG24-7463.

4.4.5 Deployment of LDAP

LDAP can be used as a backend system to provide additional user identification information that is not covered by the KDC environment. Because the NFS V4 implementation of `RPCSEC_GSS` is based on the MIT Kerberos Standard RFC1510, all compliant third-party LDAP servers can be substituted for IBM Tivoli Directory Server.

4.5 Authorization methods

Authorization is used to control access to either client hosts or client users.

There is only one way to control access to client hosts: via the `/etc/exports` file and the `exportfs` command. This is described in 3.7, “NFS V4 host authorization” on page 88.

You have three options when it comes to controlling client user access to files and directories:

- ▶ Standard UNIX permissions
- ▶ AIXC ACLs
- ▶ NFS V4 ACLs

4.5.1 Choosing your user authorization method

When deciding which user authorization method to implement, think about your requirements for controlling access to data. Do you have simple or complex data access requirements?

Standard UNIX permissions enable you to control access to only three identities: the owning user, the owning group, and everyone else. If that is not sufficient to meet your access control requirements, then you should choose one of the ACL options. For example, if you have data where you need one group to have write access, one or more other groups to have read-only access, and everyone else to have no access at all, you will not be able to accomplish this using standard UNIX permissions.

If standard UNIX permissions do not meet your requirements, you can then choose to use AIXC ACLs or NFS V4 ACLs. If you choose NFS V4 ACLs, then make sure that you choose file system types on your server that support this. See 4.6, “Choosing the appropriate file system types” on page 111 for more information.

You should not use AIXC ACLs if your requirements include one of the following:

- ▶ You have non-AIX NFS clients that must be able to manipulate ACLs for data on your NFS server. AIXC ACLs are only supported in AIX.
- ▶ You require finer granularity access control than AIXC ACLs support. For example, AIXC ACLs do not provide a way for you to set up a directory where users can create files but not delete them after they are created.

Note: The choice of AIXC ACLs over NFS V4 ACLs should make sense in environments that already have experience with AIX ACL models and are running predominantly AIX platforms. Otherwise, use of NFS V4 ACLs is recommended as the default choice due to its compliance with open standards and to minimize potential interoperability issues with non-AIX platforms.

More details about permissions and ACLs can be found in 3.4, “NFS V4 user authorization” on page 62.

4.5.2 Other user authorization considerations

Maintaining access control lists is more complicated than maintaining standard UNIX permissions. You must understand how the ACLs are evaluated before you can correctly construct an ACL. (See 3.4.4, “NFS V4 ACLs: ACL evaluation” on page 69 for information about how NFS V4 ACLs are evaluated.)

Depending on how computer-savvy your users are, you will need to either educate them on how to properly maintain their own ACLs or set up ACL inheritance so that they do not have to worry about changing permissions. (See “Directory structure and ACLs” on page 79 for more about setting up ACL inheritance.)

What if you do not want to allow end users to change permissions, thus keeping that task under the control of designated system or data administrators? The system by default does not accommodate this. Files created by a user are owned by that user, and a file’s owner can always change its permissions. You would need to devise some way to make sure that all files are owned by an administrator account. One way is to run a periodic cron job that changes ownership of all files to that account. This leaves a window of time when a user can change permissions on a newly created file, but this can be remedied by the job that changes ownership. It can also make sure that the permissions are set as they should be after it changes ownership.

You can see from this discussion that you may need to implement additional administrative controls on top of those that are provided by default.

4.6 Choosing the appropriate file system types

AIX supports several different file system types, including:

- ▶ Journaled File System (JFS)
- ▶ Enhanced Journaled File System (JFS2)

- ▶ JFS2 with extended attribute format Version 2 (EAv2)
- ▶ General Parallel File System (GPFS)

The file system you can choose will be dictated by the authorization method you chose above. If you want to use NFS V4 ACLs, then your choice should be either JFS2 EAv2 or GPFS. This does not mean that your system cannot use the other file system types. The restriction only applies to the file systems where you want to use NFS V4 ACLs.

4.7 NFS protocols and namespace considerations

At this decision point, we are in the final planning stages. We now need to make decisions about whether we will be running in a mixed environment (NFS V3 and NFS V4) or creating a pure NFS V4 environment.

Namespace is a new concept to NFS V4. How do you go about implementing something like this in a production environment? We begin by considering what it means. We can then look at some ways that it can fit into your environment.

From an administrative point of view, a single namespace reduces the number of mounts a client has to make per server down to one. With previous versions of NFS, you had to mount all the required NFS exports from a server. A single namespace allows the server to render a single view (pseudo-FS) for the client to mount, hence the single mount. What does this mean to you? To gain the optimal benefits of this feature you have to consider your server's NFS exports layout.

Figure 4-3 on page 113 shows one way the single namespace could be implemented and how it could benefit you, using one NFS server and five NFS clients.

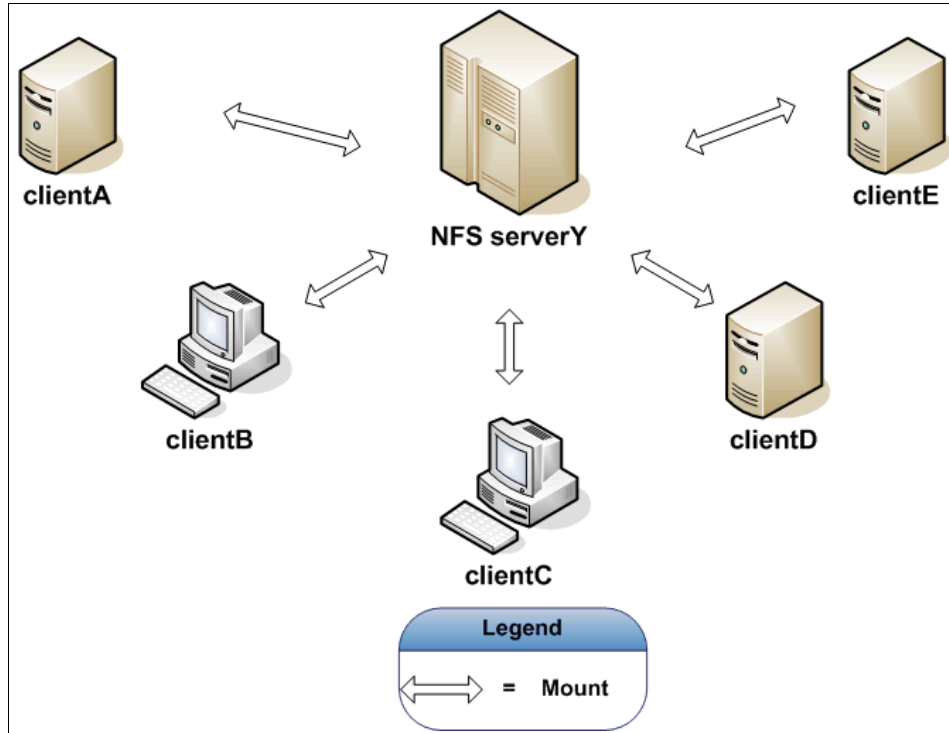


Figure 4-3 One NFS server serving five NFS clients

You want to export five file systems from your server. If you use NFS V3, then your server's exports file will look something like Example 4-1.

Example 4-1 /etc/exports file on NFS V3 server

```

/home -rw
/usr/codeshare/ThirdPartyProgs -rw
/usr/local -ro
/var/db2/v81/DB -rw
/exports/scratch -rw

```

The NFS V3 client would then have to mount each export individually, as shown in Example 4-2.

Example 4-2 The mount commands you would need to run on each NFS V3 client

```

mount serverY:/home /mount1
mount serverY:/usr/codeshare/ThirdPartyProgs /mount2
mount serverY:/usr/local /mount3

```

```
mount serverY:/var/db2/v81/DB /mount4
mount serverY:/exports/scratch /mount5
```

The NFS V3 server exported five file systems and the client had to mount five file systems to five different mount points.

Now we look at how a single namespace would simplify the export and mount process. We continue to use the directory structure defined in Example 4-1 on page 113 to carry this out. The server's `/etc/exports` file will look something like Example 4-3.

Example 4-3 Rendering a single view for the NFS V4 client on the NFS V4 server

```
/exports -nfsroot
/home -vers=4,rw,exname=/exports/home
/usr/codeshare/ThirdPartyProgs -vers=4,rw,exname=/exports/ThirdPartyProgs
/usr/local -vers=4,ro,exname=/exports/local
/var/db2/v81/DB -vers=4,rw,exname=/exports/DB
/exports/scratch -vers=4,rw,exname=/exports/scratch
```

What have we done to the `/etc/exports` file to make the single namespace concept into a practical implementation?

1. We need to set the pseudo-root on the server: This acts as the *glue* for all other file systems. We used the `/exports` directory.
2. We then export all the other file systems, as we did with the NFS V3 exports, but we add two new options:

vers=4	This tells the NFS server that the export is of type NFS V4.
exname	This is the AIX implementation extension of the single namespace concept. We have taken the file systems that we want to export and <i>glued</i> them to <code>/exports</code> (defined in the first line of the <code>/etc/exports</code> file).

If you look at the `exname` option (for all exports), we have also chosen to not show the parent directories of all the exported directories. This enables us to keep the server's directory structure private and unexposed to the clients.

We will look at this from a practical point after we have shown what the client must do to see the *logical view*. The command that the client has to run to mount the exported file systems is:

```
# mount -o vers=4 serverY:/ /nfs
```

The `mount` command tells `serverY` that the client is requesting a view of its single namespace, and after the server allows the client access to the view, the client will mount it onto the `/nfs` mount point locally.

This is how the view is represented to the client:

```
# ls /nfs
DB ThirdPartyProgs home local scratch
```

We can see that all the file systems we exported on the server are represented under the client's mount point (`/nfs`). If we now decide that we want to move the `/exports/scratch` file system, on the server, to a different location, say `/scratch`, and we also want to move `/home` to `/users/home`, we only have to make the appropriate changes to the server's `/etc/exports` file, and the client's mount command will remain the same.

Several other considerations must be taken in to account when allowing concurrent access to the same data with NFS V2, NFS V3, and NFS V4. NFS V3 access may receive errors due to the NFS V4 granted state. Also, NFS V3 performance may be affected when data is also exported for NFS V4 access (the use of `vers=3:4` in the `/etc/exports` file).

4.7.1 Pseudo-root FS - alias tree versus classic model

Implementation of pseudo-root FS or the AIX-specific alias tree implementation must be taken into account before deploying NFS V4. It introduces several changes compared to the classic model as with NFS V3. There is no need to migrate off the classic model when using NFS V4. Nevertheless, the alias tree model seems to be the best from our point of view. See 2.8.6, “External name space (exname)” on page 34.

While the pseudo-root FS - alias tree model delivers a good solution to build a Global Namespace, you have to think about the current logical as well as physical layout of your File systems on the NFS V4 Server. Nevertheless, you can adopt the pseudo-root FS - alias tree model to run with your current physical and logical layout.

Important: Either all Version 4 exports specify an external name, or none specify an external name.

4.8 Sizing and capacity planning considerations

Capacity planning of your infrastructure environment is crucial to your overall operation. It is not the purpose of this book to discuss this topic. For further reading see *Sizing and Capacity Planning*, SG24-7071.

4.9 Migration considerations

As already mentioned, we assume that, in most situations, a version of NFS is already deployed in your environment and therefore migration has to be taken into account. In this book, we cannot cover all possible migration paths in terms of how to migrate your current installation onto AIX 5.3. We have designed logical migration paths that can be used for migration planning, as shown in Figure 4-4 on page 117.

All logical migration paths have a normal NFS V3 deployment on AIX as the entry point, and end with deployment of Kerberos-based authentication. In addition, the identification method can be chosen between classic (using local `/etc/passwd`) or centralized, by the use of IBM Tivoli Directory Server or any other backend system.

Choosing the final Namespace model in terms of using, for example, pseudo-root FS goes with migration planning, which we do not cover in this book. This migration will affect your general physical as well as logical file system and mount structure of your server and client system but can be seen as independent to your security model on which we are focused.

Figure 4-4 on page 117 shows three possible logical migration paths indicated by the different numbers and arrows:

Dark arrows	Probable migration paths expected in production environments
Dashed arrows	Alternative paths for crossover
Number 1 target	Migration to NFS V4 without enhanced security
Number 2 target	Migration to NFS V4 with enhanced security using new authentication infrastructure
Number 3 target	Migration to NFS V4 with enhanced security by extending the available identification infrastructure

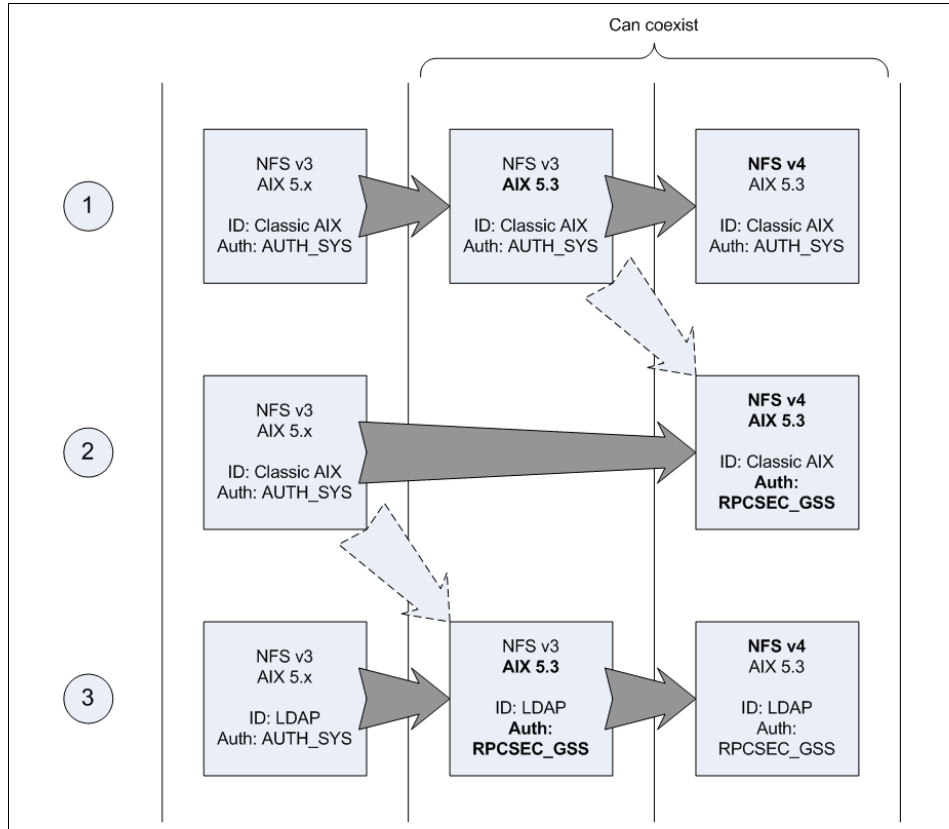


Figure 4-4 Possible NFS V4 migration scenarios

What does Can coexist at the top of Figure 4-4 mean?

- ▶ Server file systems can be exported to support NFS V4 and NFS V3 at the same time using the **exportfs** option `-vers=3:4`. This enables the client to mount the file system using either NFS V3 or NFS V4, and enables you to use clients that do not support NFS V4 to access the server file systems.
- ▶ With the AIX 5.3 NFS implementation, you can also use enhanced security (RPCSEC_GSS API) with NFS V3, but only if all servers and clients are running AIX 5.3.

At this point we discuss only the migration paths marked by the dark arrows.

1. In this case, migration to NFS V4 has been considered using a two-step approach by migrating all server systems first to AIX 5.3 without having NFS V4 deployed. After migrating all client systems to AIX 5.3, general deployment of NFS V4 can take place.

2. If you are able to migrate all of your systems to AIX 5.3, you may plan to integrate NFS V4 with enhanced security in one step. This implies paying particular attention when planning your authentication infrastructure.
3. If you already have centralized identification, migration using a two-step approach is reasonable. The first step introduces authentication services into your existing identification infrastructure, and step 2 migrates the existing NFS V3 environment into NFS V4.



Sample implementation scenarios

This chapter provides practical information about installation, administration, and deployment of NFS V4 implementation on AIX 5.3. You should get an idea of how NFS V4 can be implemented in your environment through the step-by-step scenarios that we used based on our own redbook development systems layout.

The following actions are demonstrated and discussed:

- ▶ **General setup**
What system level configurations are needed: PATH variables, Application Event logging and so forth.
- ▶ **Deploying NFS V4 in an classic NFS V3 manner without additional security enabled.**
We provide a sample that shows deployment of NFS V3 on AIX 5.3.
- ▶ **Setting up a pseudo-root file system and a global filespace**
We show how to set up and deploy the pseudo-root FS as well as the advantages of using the pseudo-root FS - alias tree model to provide a Global Filespace to the client.

- ▶ **NAS Service environment to support the additional NFS V4 security features**
This includes the required steps to set up a minimal NAS environment to deploy Kerberos on the server and client. We also show the steps required to integrate NFS V4 security flavors.
- ▶ **Additional LDAP backend integration into KDC with integrated logon**
This section guides you through the necessary steps to set up the environment to deploy integrated logon on your client system while running a KDC server with an LDAP backend for identification and authentication.

5.1 Setup of the sample environment

This section describes the setup used on our systems for compiling the samples. The setup may not adapt to your needs if you do not use NAS, but we believe that it is useful to provide the information here.

Important: It is important that you do not confuse the term NAS with Network Attached Storage. In this context NAS stands for *Network Authentication Service*.

5.1.1 PATH variable for NAS deployment

After installing the AIX 5.3 Base Operating System, the PATH environment variable must be adjusted to make the required commands available to the end user. In addition, correlations with other installed products such as DCE and Tivoli Netview should be considered while setting the PATH variable systemwide.

The Java14.sdk file set is automatically installed as a co-requisite of sysmgt.websm.rte 5.3.0.0. You can verify this by running the `installp -ugp Java14.sdk` command. The standard PATH variable in `/etc/environment` is set to:

```
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:/usr/java14/jre/bin:/usr/java14/bin:/usr/java131/jre/bin:/usr/java131/bin
```

We found that the NAS user commands, such as `kinit`, are not found in the PATH. Further, `kinit` is also installed with Java14.sdk.

Example 5-1 Sample output of different kinit versions with different PATH settings

```
# type kinit
kinit is /usr/java14/jre/bin/kinit
#
# type kinit
kinit is /usr/krb5/bin/kinit
#
```

Therefore, we changed the PATH variable in `/etc/environment` to:

```
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:/usr/krb5/bin:/usr/java14/jre/bin:/usr/java14/bin:/usr/java131/jre/bin:/usr/java131/bin
```

We distributed this file to all of our AIX NFS V4 servers and clients.

5.1.2 syslogd settings

By default, all NFS V4 related debug output is written to the syslog.

Tip: We recommend that you always enable syslog logging when carrying out your NFS setup. It will be beneficial in obtaining detailed messages for the errors you see when running NFS commands. You can disable the logging when you are satisfied that your environment is working as you expect it to.

To capture errors logged by the NFS daemons, we made the following changes to our systems:

1. A local file system was created and mounted to the `/var/nfsv4log` directory. (It is always better to have a separate file system for log daemons, so that your root file system is safeguarded from being filled up.)
2. The following stanza was added to the `/etc/syslog.conf` file:

```
*.debug /var/nfs4log/syslog.out rotate time 1d archive /var/nfs4log/archive/
```
3. We then refreshed the `syslogd` using the **refresh -s syslog** command to activate these changes immediately on the running system.

Using the setting for `syslogd` as given in step 2 may log more information as needed while running a system in production mode. Thus you can set back the log level to `ERROR` in the `/etc/syslog.conf` file to limit the amount of data written. The new stanza will now show:

```
*.error /var/nfs4log/syslog.out rotate time 1d archive /var/nfs4log/archive/
```

The changes will take affect only after you refresh the `syslog` daemon.

5.2 Using NFS V4 as you did with NFS V3

NFS V4 still supports the legacy file system export and mount model because it is common in NFS V3. There are no changes required to your server `/etc/exports` file or to the client `/etc/filesystems` file or automount map. This is true only if you are not planning to use the new features available with NFS V4.

Example 5-2 shows the server view of NFS V3 exported file systems.

Example 5-2 Server output of NFS V3 exported file systems on AIX 5.3

```
# pg /etc/exports
/exports -rw
/exports/home -rw
/exports/project/projA -ro
/exports/project/projB -ro
/usr/ldap -vers=3,ro
#
#exportfs -va
```

```
Exported /exports
Exported /exports/home
Exported /exports/project/projA
Exported /exports/project/projB
Exported /usr/ldap
#
```

Example 5-3 shows the client view for the NFS V3 mounted file systems from server nfs404.

Example 5-3 Client output for NFS V3 mounted file system on AIX 5.3

```
# showmount -e nfs404
export list for nfs404:
/exports          (everyone)
/exports/home     (everyone)
/exports/project/projA (everyone)
/exports/project/projB (everyone)
/usr/ldap         (everyone)
# mount nfs404:/exports /nfs
# mount nfs404:/exports/home /nfs/home
# mount nfs404:/exports/project/projA /nfs/project/projA
# mount nfs404:/exports/project/projB /nfs/project/projB
# cd /nfs/project/projB
# df .
Filesystem      512-blocks      Free %Used    Iused %Iused Mounted on
nfs404:/exports/project/projB      65536      64856    2%      5      1%
/nfs/project/projB
```

5.3 How to unmount an exported NFS V4 file system

With NFS V3 you could unmount any physical file systems NFS exported, on the NFS server, without first unexporting it. With NFS V4 this is no longer possible. As long as the file systems are exported they are locked within the server code and cannot be unmounted. To do so, you have to unexport the file system first.

Example 5-4 Unmount of exported file systems on NFS V4 server

```
# exportfs
/exports          -vers=4,rw
/exports/home     -vers=4,rw
/exports/project/projA -vers=4,ro
/usr/ldap         -vers=3,ro
#
# umount /exports/project/projB
umount: 0506-349 Cannot unmount /dev/fs1v01: The requested resource is busy.
```

```
#  
# exportfs -u /exports/project/projB  
# umount /exports/project/projB
```

5.4 Setting up the NFS domain name

As described in 4.2.2, “Choosing your NFS domain” on page 98, it is mandatory to have the NFS domain name set before you can use NFS V4. The current setting can be checked using `chnfsdom` or `smitty chnfsdom` commands.

If the NFS domain is not set, the output looks similar to Example 5-5.

Example 5-5 chnfsdom sample output

```
# chnfsdom  
Current local domain: N/A
```

In our sample environment, we already set the NFS domain so that the output looks like Example 5-6.

Example 5-6 Setting the NFS domain on the server

```
# chnfsdom  
Current local domain:  
nfs.ibm.com
```

Note: As this book is being written, changing the NFS domain does not recycle or start `nfsrgyd`. You have to manually start or recycle the daemon.

We started the `nfsrgyd` by using the command `startsrc -s nfsrgyd`.

Example 5-7 startup of the nfsrgyd daemon

```
# startsrc -s nfsrgyd  
0513-059 The nfsrgyd Subsystem has been started. Subsystem PID is 14496.
```

5.5 The pseudo-root FS

The mechanism and meaning of the NFS V4 pseudo-root has been described in 2.6.3, “Better namespace handling” on page 25. This chapter shows how to set up the pseudo-root and point out the advantages compared to the classic NFS V3 export model.

5.5.1 Setting up the pseudo-root FS on an NFS V4 server

The pseudo-root FS setup on an NFS V4 server can be achieved easily. This requires that the pseudo-root directory be available and that all exported file systems are locally mounted within this directory.

Note: There are several ways to change the pseudo-root on the server. We recommend using the `-nfsroot` option within the `/etc/exports` file; however, all NFS exported server file systems have to be unexported or this will not work.

By default, the root directory `/` is defined and exported as the pseudo-root FS from the server. This can be seen when the `nfsd -getnodes` command is executed on the running system. Example 5-8 shows the output.

Example 5-8 Output of command `nfsd -getnodes`

```
$nfsd -getnodes
#root:public
/://
$
```

Using the `-nfsroot` option in file `/etc/exports`

Perform these steps on the server to set the pseudo-root to `/exports` and export it to the clients:

1. Check the current settings for pseudo-root by running the `nfsd -getnodes` command.
2. Check that no NFS file systems are exported by using the `exportfs` command. If there are exported file systems, run the `exportfs -ua` command to unexport them.
3. Change the `/etc/exports` file so that the option `/exports -nfsroot` is listed in the first place.

Example 5-9 shows all commands and system responses when changing the pseudo-root FS using `-nfsroot` option.

Example 5-9 Changing the pseudo-root on an NFS V4 server using `-nfsroot`

```
# nfsd -getnodes
#root:public
/://
#
# exportfs -ua
# exportfs
exportfs: nothing exported
# pg /etc/exports
```

```

#
/exports -nfsroot
/local/trans -vers=4,rw,exname=/exports/trans
/local/dept -vers=4,rw,exname=/exports/dept
/local/home -vers=4,rw,exname=/exports/home
/usr/codeshare/ThirdPartyProgs -vers=4,ro,exname=/exports/ThirdPartyProgs
#
# exportfs -va
exported /local/trans
exported /local/dept
exported /local/home
exported /usr/codeshare/ThirdPartyProgs
# nfsd -getnodes
#root:public
/exports:/exports
#
#ps -ef |grep nfsd
root 270550 159906 0 17:12:13 - 0:00 /usr/sbin/nfsd -root / 3891

```

Important: Although the `nfsd` process still shows the option `-root` as `/` the actual pseudo-root FS has been changed to `/exports`. This has been verified by running the `nfsd -getnodes` command.

Changing the nfsd daemon

Perform these steps on the server to set the pseudo-root to `/exports` and export it to the clients:

1. Check the current settings for pseudo-root by running the `nfsd -getnodes` command.
2. Check that no NFS file systems are exported by using the `exportfs` command. If there are exported file systems, run the `exportfs -ua` command to unexport them.
3. Stop all NFS server processes by running `/etc/nfs.clean`
4. Change the pseudo-root with `chnfs -r /exports` or `smitty chrootfh`

The `chnfs` command:

- a. Changes the entries within the AIX Subsystem Resource for `nfsd`. This can be verified using the `lssrc -Ss nfsd` command.
- b. Starts the `nfsd` in command line mode outside the control of the Subsystem Resource Controller.

We have two options to proceed:

- Reboot the system so that all changes take effect.

- Stop all NFS processes using `/etc/nfs.clean` and restart NFS using `/etc/rc.nfs`.

We decided to use the second option, and we continue with Step 6.

5. Verify the change using the `nfsd -getnodes` command.
6. Stop all NFS processes on the system using `/etc/nfs.clean`
7. Start all NFS server processes with `/etc/rc.nfs`

Example 5-10 shows all commands and system responses when changing the pseudo-root FS using `chnfs`.

Example 5-10 Changing the pseudo-root on an NFS V4 server using chnfs

```
# nfsd -getnodes
#root:public
/://
#
# exportfs
/exports -vers=4,rw
/exports/project/projA -vers=4,rw
/exports/project/projB -vers=4,rw
#
# exportfs -ua
#
# exportfs
exportfs: 1831-182 nothing exported
#
# /etc/nfs.clean
nfs_clean: Stopping NFS/NIS Daemons
0513-044 The nfsd Subsystem was requested to stop.
0513-044 The biod Subsystem was requested to stop.
0513-044 The rpc.lockd Subsystem was requested to stop.
0513-044 The rpc.statd Subsystem was requested to stop.
0513-044 The rpc.mountd Subsystem was requested to stop.
0513-085 The ypserv Subsystem is not on file.
0513-004 The Subsystem or Group, ypbind, is currently inoperative.
0513-085 The yppasswdd Subsystem is not on file.
0513-085 The ypupdated Subsystem is not on file.
#
# chnfs -r /exports
0513-077 Subsystem has been changed.
#
# nfsd -getnodes
#root:public
/exports:/exports
#
# /etc/nfs.clean
Stopping NFS/NIS Daemons
```

```

0513-004 The Subsystem or Group, nfsd, is currently inoperative.
0513-004 The Subsystem or Group, biod, is currently inoperative.
0513-004 The Subsystem or Group, rpc.lockd, is currently inoperative.
0513-004 The Subsystem or Group, rpc.statd, is currently inoperative.
0513-004 The Subsystem or Group, rpc.mountd, is currently inoperative.
0513-004 The Subsystem or Group, ypbind, is currently inoperative.
#
# /etc/rc.nfs
Starting NFS services:
0513-059 The biod Subsystem has been started. Subsystem PID is 266398.
0513-029 The nfsrgyd Subsystem is already active.
Multiple instances are not supported.
0513-059 The nfsd Subsystem has been started. Subsystem PID is 303296.
0513-059 The rpc.mountd Subsystem has been started. Subsystem PID is 258216.
0513-059 The rpc.statd Subsystem has been started. Subsystem PID is 262314.
0513-059 The rpc.lockd Subsystem has been started. Subsystem PID is 274644.
0513-029 The gssd Subsystem is already active.
Multiple instances are not supported.
Completed NFS services.
#
# ps -ef |grep nfs
    root 286874 204944  0 09:48:14    - 0:00 /usr/sbin/nfsrgyd
    root 303296 204944  0 09:53:06    - 0:00 /usr/sbin/nfsd -root /exports
3891
    root 352494 344286  0 09:53:25 pts/0 0:00 grep nfs
#
# lssrc -g nfs
Subsystem      Group          PID           Status
biod           nfs           274482        active
nfsd           nfs           266406        active
rpc.mountd     nfs           249996        active
nfsrgyd        nfs           172234        active
rpc.lockd      nfs           254144        active
rpc.statd      nfs           245884        active
gssd           nfs           0             inoperative
#

```

Note: The gssd daemon is not running because we are not using RPCSEC_GSS at this point in our sample environment.

A simple shell script called SetNewRootFS.ksh to achieve the above described changes is available in “Change the pseudo-root FS sample script” on page 256.

5.5.2 Advantages of using the NFS V4 pseudo-root

As already described in 2.6.3, “Better namespace handling” on page 25, there are several advantages to using NFS V4 with pseudo-root. This section shows

examples of these advantages. We assume that the NFS V4 server pseudo-root FS has already been set to /exports.

On the server, we created three file systems (Example 5-11), which were mounted under the /exports directory using following subdirectories:

- ▶ /exports/home
- ▶ /exports/project/projA
- ▶ /exports/project/projB

Example 5-11 Sample df output from the NFS V4 server

/dev/fs1v00	65536	64832	2%	5	1% /exports/project/projA
/dev/fs1v01	65536	64864	2%	4	1% /exports/project/projB
/dev/fs1v02	65536	64864	2%	4	1% /exports/home

To be able to export these file systems to the clients, we created the /etc/exports file on the server using **smitty mknfsexp** with the entries in Example 5-12.

Example 5-12 Output of file /etc/exports for pseudo-root

```
# pg /etc/exports
/exports -vers=4,rw
/exports/home -vers=4,rw
/exports/project/projA -vers=4,ro
/exports/project/projB -vers=4,ro
```

In Example 5-12, the entries are similar to the /etc/exports file used with NFS V3 except for the fact that option `vers=4` indicates that the export is of type NFS V4 only. In addition, you still have to export every local file system so that the NFS server is capable of internally rendering and crossing the file system borders.

The main changes are seen on the client side. They are:

1. Only a single mount command is required enable retrieval and changing of directory to all available file systems.
2. With NFS V3, the same functionality is not available, so to mount all file systems, a mount command for each exported file system is required.

In our example, only the following command is needed on the client to make the pseudo-root FS and all exported file systems beneath available to the client:

```
mount -o vers=4 nfs404:/ /nfs
```

On traversal of the mount, for example, from /nfs to /nfs/project/projA, no subsequent mounts are required as NFS V4 will handle this by use of the `fsid()`.

Note: This works only with NFS V4 exported file systems. If you use NFS V3 by default on the client, the **mount** command will fail with this message:

```
mount nfs404:/ /nfs
mount: 1831-011 access denied for nfs404:/
mount: 1831-008 giving up on:
nfs404:/
The file access permissions do not allow the specified action.
```

Example 5-13 shows the output of an NFS V4 client accessing the pseudo-root FS.

Example 5-13 Client output while accessing a pseudo-root FS

```
# mount -o vers=4 nfs404:/ /nfs
# cd /nfs
# ls
dept      home      project  tt
# cd project/projA
# ls -l
total 0
drwxr-xr-x  2 nobody  system    256 Jul 26 17:43 Targets
drwxr-xr-x  2 root    system    256 Jul 26 17:42 lost+found
#
# mount
node        mounted          mounted over vfs date options
-----
nfs404  /                /nfs             nfs4   Jul 29 14:56 vers=4
#
# pwd
/nfs/project/projA
#
# df .
Filesystem    512-blocks    Free %Used    Iused %Iused Mounted on
[NFSv4]        65536         64832  2%          5    1% .
#
# nfs4cl showfs
Server        Remote Path      fsid              Local Path
-----
nfs404                10:13             /nfs/projB
nfs404                10:12             /nfs/projA
nfs404                10:17             /nfs/home
nfs404  /                10:4              /nfs
```

You may have recognized that the command **df** is not useful in this context. That is why the **nfs4cl** command is recommended.

5.5.3 Setting up the alias tree extension on an NFS V4 server

With reference to the description in 4.7, “NFS protocols and namespace considerations” on page 112, the alias tree model seems to be the best model for using pseudo-root FS to create a global NFS namespace. The following steps show how to set up the alias tree model on one of our sample servers.

Note: The described pseudo-root FS setup cannot coexist with the alias tree model. You have to choose between the two models.

We created three new file systems on the server that does not share the same root mount path:

- ▶ /local/trans
- ▶ /local/trans1
- ▶ /usr/codeshare/ThirdPartyProgs

Example 5-14 Sample df output from NFS V4 server for alias tree model

/dev/exname_1lv	65536	63376	4%	20	1%	/local/trans
/dev/exname2_1lv	65536	63392	4%	18	1%	/local/trans1
/dev/exname3_1lv	65536	63392	4%	18	1%	

/usr/codeshare/ThirdPartyProgs

The second step is to generate a new /etc/exports file and include the exname option as shown in Example 5-15.

Example 5-15 Sample file /etc/exports using the exname option

```
/local -vers=4,rw,exname=/exports/local
/local/trans -vers=4,rw,exname=/exports/local/trans
/local1/trans1 -vers=4,rw,exname=/exports/local1/trans1
/usr/codeshare/ThirdPartyProgs
-vers=4,ro,exname=/exports/usr/codeshare/ThirdPartyProgs
```

Finally, we export the new file systems to the clients using the **exportfs -va** command as shown in Example 5-16.

Example 5-16 Exporting file systems using exportfs -va

```
root@nfs402 #exportfs -va
exportfs: 1831-187 re-exported /local
exportfs: 1831-187 re-exported /local/trans
exportfs: 1831-187 re-exported /local1/trans1
exportfs: 1831-187 re-exported /usr/codeshare/ThirdPartyProgs
```

You can use this command to mount the pseudo-root FS on the client:

```
mount -o vers=4,sec=krb5 nfs402:/ /nfs
```

The global namespace is available using the pseudo-root FS - alias tree model.

In Example 5-17 we show the access to the alias tree on a NFS V4 client.

Example 5-17 Client view of the alias tree

```
# mount
node          mounted      mounted over  vfs    date      options
-----
nfs402  /              /nfs          nfs4   Aug 02 17:26 vers=4,
```

```
# find /nfs -print
/nfs
/nfs/usr
/nfs/usr/codeshare
/nfs/usr/codeshare/ThirdPartyProgs
/nfs/local1
/nfs/local1/trans1
/nfs/local1/trans1/trash
/nfs/local
/nfs/local/trans
/nfs/local/trans/var
/nfs/local/trans/trash
/nfs/local/trans/translog
/nfs/local/trans1
# nfs4cl showfs
#
Server      Remote Path      fsid          Local Path
-----
nfs402      /                 10:15         /nfs/trans
nfs402      /                 10:16         /nfs/trans1
nfs402      /                 10:17         /nfs/ThirdPartyProgs
nfs402      /                 10:4          /nfs
nfs403      /exports/ac1test 10:13         /mntjt
# pwd
/nfs/local1/trans1
# df .
Filesystem  512-blocks    Free %Used    Iused %Iused Mounted on
[NFSv4]      65536         63392    4%         18     1% .
#
```

A more realistic example was discussed in 2.6.3, “Better namespace handling” on page 25 and will be covered in this chapter. On the server that does not share the same root mount path, we created four new file systems called:

- ▶ /local/trans
- ▶ /local/home
- ▶ /local/dept
- ▶ /usr/codeshare/ThirdPartyProgs

The intention is to map these directories on the client in a global namespace called /nfs with the following directory structure:

- ▶ /nfs/home
- ▶ /nfs/home
- ▶ /nfs/dept
- ▶ /nfs/ThirdPartyProgs

To do so we created the following /etc/exports file on the server and included the exname option shown in Example 5-18.

Example 5-18 Sample alias tree file /etc/exports file on the server

```
/local/trans -vers=4,rw,exname=/exports/trans  
/local/dept -vers=4,rw,exname=/exports/dept  
/local/home -vers=4,rw,exname=/exports/home  
/usr/codeshare/ThirdPartyProgs -vers=4,ro,exname=/exports/ThirdPartyProgs
```

To mount the pseudo-root FS on the client, the following command is used:

```
mount -o vers=4,sec=krb5 nfs402:/ /nfs
```

The Global Namespace is made available using the pseudo-root FS - alias tree model.

Example 5-19 shows the access to the alias tree on an NFS V4 client.

Example 5-19 Extended client view of the pseudo-root FS - alias tree model

```
#mount -o sec=krb5,vers=4 nfs404:/ /nfs  
#  
#cd /nfs  
#  
# find . -print  
.  
./ThirdPartyProgs  
./ThirdPartyProgs/bin  
./ThirdPartyProgs/src  
./ThirdPartyProgs/contrib  
./home  
./home/sally
```

```

./home/bob
./home/mary
./home/joe
./dept
./dept/eng
./dept/hr
./dept/sales
./trans
./trans/lost+found
#
# cd /nfs/home
#
# ls -ltr
total 40
drwxrwx---  2 root    system      512 Aug 12 18:41 lost+found
drwxr-sr-x  2 sally   staff       512 Aug 12 18:50 sally
drwxr-sr-x  2 bob     staff       512 Aug 12 18:50 bob
drwxr-sr-x  2 mary   staff       512 Aug 12 18:50 mary
drwxr-sr-x  2 joe    staff       512 Aug 12 18:50 joe
# mount
  node          mounted          mounted over    vfs      date          options
-----
      /dev/hd4      /
      /dev/hd2      /usr
      /dev/hd9var   /var
      /dev/hd3      /tmp
      /dev/hd1      /home
      /proc         /proc
      /dev/hd10opt  /opt
      /dev/log_lv   /var/nfs4log
nfs404  /              /nfs            nfs4     Aug 10 15:07 vers=4,sec=krb5
#
# nfs4cl showfs

Server      Remote Path          fsid            Local Path
-----
nfs404
nfs404
nfs404
nfs404
nfs404  /

```

5.6 Setting up the NAS with a legacy database

In this section we describe the sample setup of IBM NAS Version 1.4 on AIX with a KDC legacy database. The identification of the user will be done using

standard AIX local mechanisms: by using the `/etc/passwd` file entries. This means that we do not use a centralized user identification registry such as LDAP or NIS.

The following definitions are used in the example:

NFS and KDC server host name = nfs403, OS: AIX 5.3
NFS client host name = nfs406, OS: AIX 5.3
NFS Domain Name itsc.austin.ibm.com
Realm Name REALM1.IBM.COM

Figure 5-1 gives a better view of the setup.

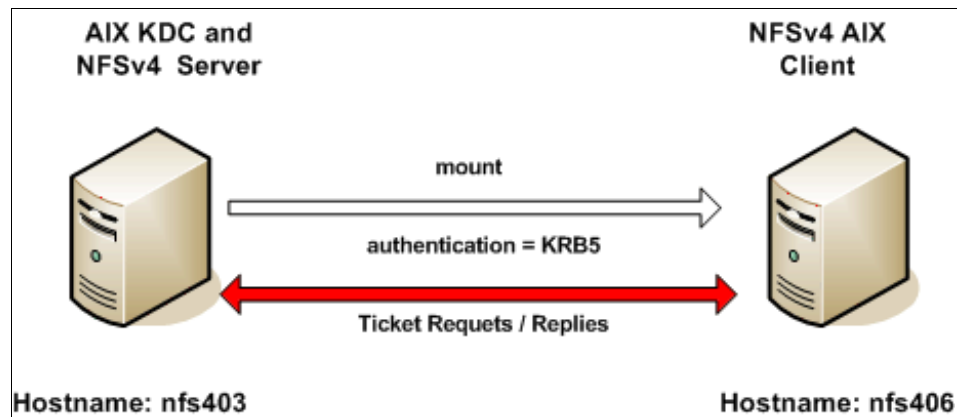


Figure 5-1 Sample AIX environment with AIX KDC server

5.6.1 Setup of a KDC server

In this section, we describe the minimal setup of the KDC server using IBM NAS V1.4 so that NFS V4 can use `RPCSEC_GSS` for authentication. Further information about IBM NAS and the KDC infrastructure can be found in *IBM Network Authentication Service Version 1.4 for AIX, Linux, and Solaris Administrator's and User's Guide*, which is delivered with the `krb5.doc.en_US` file set.

5.6.2 Installing the IBM NAS file sets

The IBM NAS Version 1.4 file sets are delivered with the AIX 5.3 Expansion CD. They can be installed using `smit` or the `installp` command. We used the command line interface for installation. The command we used was:

```
installp -aqXd . krb5.server modcrypt.base
```

Set up the KDC server

To configure NAS to use the legacy database, which is stored on the local file system, we used the following command:

```
mkkrb5srv -r REALM2.IBM.COM -d itsc.austin.ibm.com -s \  
nfs403.itsc.austin.ibm.com
```

While running this command, the system asks for a Master Database password and a password for the administrative principal called admin. Record the name and chosen password in an secure place as these principals are essential for your NAS environment.

Example 5-20 shows the output from the **mkkrb5srv** command with the options shown above.

Example 5-20 Output of command mkkrb5srv on the KDC server

```
# mkkrb5srv -r REALM2.IBM.COM -d itsc.austin.ibm.com \  
-s nfs403.itsc.austin.ibm.com
```

Fileset	Level	State	Description

Path: /usr/lib/objrepos			
krb5.server.rte	1.4.0.0	COMMITTED	Network Authentication Service Server
Path: /etc/objrepos			
krb5.server.rte	1.4.0.0	COMMITTED	Network Authentication Service Server

```
The -s option is not supported.  
The administration server will be the local host.  
Initializing configuration...  
Creating /etc/krb5/krb5_cfg_type...  
Creating /etc/krb5/krb5.conf...  
Creating /var/krb5/krb5kdc/kdc.conf...  
Creating database files...  
Initializing database '/var/krb5/krb5kdc/principal' for realm 'REALM2.IBM.COM'  
master key name 'K/M@REALM2.IBM.COM'  
You are prompted for the database Master Password.  
It is important that you DO NOT FORGET this password.  
Enter database Master Password:  
Re-enter database Master Password to verify:  
WARNING: no policy specified for admin/admin@REALM2.IBM.COM;  
defaulting to no policy. Note that policy may be overridden by  
ACL restrictions.  
Enter password for principal "admin/admin@REALM2.IBM.COM":  
Re-enter password for principal "admin/admin@REALM2.IBM.COM":  
Principal "admin/admin@REALM2.IBM.COM" created.  
Creating keytable...
```

```
Creating /var/krb5/krb5kdc/kadm5.ac1...
```

```
Starting krb5kdc...  
krb5kdc was started successfully.  
Starting kadmind...  
kadmind was started successfully.  
The command completed successfully.  
Restarting kadmind and krb5kdc
```

The same can be achieved by running the NAS command:

```
config.krb5 -S -d itsc.austin.ibm.com -r REALM2.IBM.COM
```

In addition, the two lines shown in Example 5-21 are added automatically to the `/etc/inittab` file so that the KDC server is automatically started after system reboot.

Example 5-21 /etc/inittab entries for NAS

```
# !sitab krb5kdc  
krb5kdc:2:once:/usr/krb5/sbin/krb5kdc  
#  
# !sitab kadm  
kadm:2:once:/usr/krb5/sbin/kadmind  
#
```

Attention: If you intend to install the KDC on an NFS V4 server, be aware that after activating the KDC server your NFS V4 server will not work properly until you have completed all further steps.

After successful installation and configuration of the KDC, on the server, the files shown in Figure 5-1 are installed in the `/etc/krb5` directory.

Table 5-1 Additional files installed at KDC configuration

krb5.conf	Contains general information for clients and servers. It must reside on each system that contains the administration server, a KDC, or a client. If two or more Network Authentication Service servers or clients reside on the same system, they must share the same <code>krb5.conf</code> file.
krb5_cfg_type	Determines the configuration type of the machine (master, slave, or client). This file must reside on the system that contains the administration server.

Example 5-22 on page 138 shows the contents of the `/etc/krb5/krb5.conf` file, and Example 5-23 on page 138 shows the contents of the `/etc/krb5/krb5_cfg_type` file.

Example 5-22 Sample /etc/krb5/krb5.conf file on the KDC server

```
# cat krb5.conf
[libdefaults]
    default_realm = REALM2.IBM.COM
    default_keytab_name = FILE:/etc/krb5/krb5.keytab
    default_tkt_encypes = des3-cbc-sha1 arcfour-hmac aes256-cts
des-cbc-md5 des-cbc-crc
    default_tgs_encypes = des3-cbc-sha1 arcfour-hmac aes256-cts
des-cbc-md5 des-cbc-crc

[realms]
    REALM2.IBM.COM = {
        kdc = nfs404.itsc.austin.ibm.com:88
        admin_server = nfs404.itsc.austin.ibm.com:749
        default_domain = ibm.com
    }

[domain_realm]
    .ibm.com = REALM2.IBM.COM
    nfs404.itsc.austin.ibm.com = REALM2.IBM.COM

[logging]
    kdc = FILE:/var/krb5/log/krb5kdc.log
    admin_server = FILE:/var/krb5/log/kadmin.log
    default = FILE:/var/krb5/log/krb5lib.log
#
```

Note: The `default_tkt_encypes` and `default_tgs_encypes` can be reduced to show only `des-cbc-crc` and `des-cbc-md5`, but keep in mind that this may only be correct with NFS V4.

Example 5-23 Sample contents of file /etc/krb5/krb5_cfg_type on the KDC server

```
# cat /etc/krb5/krb5_cfg_type
master
```

5.6.3 Initial basic KDC functions test

Before continuing, we test the initial setup of the KDC server by using the following command sequence to verify that all required processes for the KDC server have started:

```
ps -ef |grep krb |grep -v grep
```

This verifies the logon principal as admin:

```
kinit admin/admin@REALM2.IBM.COM
```

This verifies that a ticket has been granted to the admin principal:

```
klist
```

Example 5-24 shows the output of the above mentioned commands.

Example 5-24 Basic verification of KDC server

```
# ps -ef |grep krb |grep -v grep
  root 299160      1  0 18:20:51    -  0:00 /usr/krb5/sbin/krb5kdc
  root 315554      1  0 18:20:51    -  0:00 /usr/krb5/sbin/kadmind
#
# kinit admin/admin@REALM2.IBM.COM
Password for admin/admin@REALM2.IBM.COM:
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: admin/admin@REALM2.IBM.COM

Valid starting    Expires            Service principal
07/28/04 14:16:18  07/29/04 14:16:15  krbtgt/REALM2.IBM.COM@REALM2.IBM.COM
```

5.6.4 Create user principals on the KDC server

In this section, we create Kerberos user principals that match the existing UNIX user names. The principal name will be mapped to the user name by NFS to determine the UNIX credential associated with the principal. In general, principals can be added on the KDC server using this command:

```
kadmin.local -> add_principal (or addprinc)
```

Note: `kadmin.local` can only be run on the master KDC, whereas `kadmin` can be run on any machine that is part of the Kerberos realm. We use both variations of the command in examples in this chapter.

In Example 5-25, the principal named sally is added to the KDC database via command line. This requires that you have already authenticated to Kerberos using an administrative principal, for example admin/admin.

Note: We assume that the local AIX user sally already exists on the KDC server. In addition, this user name must also exist on all NFS V4 clients.

Example 5-25 Adding the principal named sally

```
# kadmin.local
kadmin.local: addprinc -e des-cbc-crc:normal sally
WARNING: no policy specified for sally@REALM2.IBM.COM;
```

defaulting to no policy. Note that policy may be overridden by ACL restrictions.

```
Enter password for principal "sally@REALM2.IBM.COM":  
Re-enter password for principal "sally@REALM2.IBM.COM":  
Principal "sally@REALM2.IBM.COM" created.  
#
```

To verify the newly created principal, we use the **kadmin.local** command as shown in Example 5-26. The **kadmin.local** interface also gives the options **listprincs**, **get_principal**, and **getprinc** to achieve the result.

Example 5-26 Output of kadmin.local getprinc command

```
kadmin.local: getprinc sally  
Principal: sally@REALM2.IBM.COM  
Expiration date: [never]  
Last password change: Wed Jul 28 14:58:25 CDT 2004  
Password expiration date: [none]  
Maximum ticket life: 1 day 00:00:00  
Maximum renewable life: 7 days 00:00:00  
Last modified: Wed Jul 28 14:58:25 CDT 2004 (admin/admin@REALM2.IBM.COM)  
Last successful authentication: [never]  
Last failed authentication: [never]  
Failed password attempts: 0  
Number of keys: 1  
Key: vno 1, DES cbc mode with CRC-32,  
no salt  
  
Attributes:  
  REQUIRES_PRE_AUTH  
Policy: [none]
```

Example 5-26 shows that the number of keys equals 1, and the standard encryption for this principal is set to Single-DES. You could have principals with different number of keys and standard encryption types within the KDC.

In the next step we verify that the user can be authenticated using the created principle sally as shown in Example 5-27.

Example 5-27 Verify the principal sally by use of kinit

```
# kinit sally@REALM2.IBM.COM  
Password for sally@REALM2.IBM.COM:  
#  
# klist  
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0  
Default principal: sally@REALM2.IBM.COM
```

```
Valid starting    Expires          Service principal
07/28/04 15:17:09 07/29/04 15:17:09 krbtgt/REALM2.IBM.COM@REALM2.IBM.COM
Renew until 07/29/04 15:17:22
```

This procedure has to be executed for every AIX user in your environment. For a large number of users you may want to create a script to automate this process. Example 5-28 should give you an idea of how to achieve this.

Example 5-28 User principal creation shell script

```
#!/bin/ksh
NAME=joe
NPASSWD="new01new"
/usr/krb5/sbin/kadmin.local <<EOF
add_principal -e des-cbc-crc:normal -pw ${NPASSWD} $NAME
EOF
```

Note: This script will run only if you are already authenticated as admin. Using this flag in a shell script can be dangerous if unauthorized users gain read access to this script. The script is only provided to give you an idea of how this can be achieved.

You also have the option to run the utility **mkseckrb5**, delivered with the AIX Base Operating System `bos.rte.security` file set, to import all existing users from the local system into the KDC database. For further details, see the commands reference chapter of the AIX 5.3 online documentation.

5.6.5 Create the NFS server principals on the KDC server

For each NFS server in your KDC environment, you must define a principal of type `nfs/<full_qualified_hostname>@REALM`, for example:

```
nfs/nfs404.itsc.austin.ibm.com@REALM1.ITSC.AUSTIN.IBM.COM
```

The setup for this principal is slightly different from the user principal setup, as we use a random password instead of one that is user-defined.

Example 5-29 Creation of NFS server principal `nfs/nfs@REALM`

```
# /usr/krb5/sbin/kadmin
kadmin.local: add_principal -e des-cbc-crc:normal -randkey
nfs/nfs404.itsc.austin.ibm.com
WARNING: no policy specified for nfs/nfs404.itsc.austin.ibm.com@REALM2.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.
Principal "nfs/nfs404.itsc.austin.ibm.com@REALM2.IBM.COM" created.
```

```
kadmin.local: getprinc nfs/nfs404.itsc.austin.ibm.com
Principal: nfs/nfs404.itsc.austin.ibm.com@REALM2.IBM.COM
Expiration date: [never]
Last password change: Wed Jul 28 16:21:44 CDT 2004
Password expiration date: [none]
Maximum ticket life: 1 day 00:00:00
Maximum renewable life: 7 days 00:00:00
Last modified: Wed Jul 28 16:21:44 CDT 2004 (admin/admin@REALM2.IBM.COM)
Last successful authentication: [never]
Last failed authentication: [never]
Failed password attempts: 0
Number of keys: 1
Key: vno 2, DES cbc mode with CRC-32,
no salt

Attributes:

Policy: [none]
```

5.7 Setting up an NFS V4 server with NAS on a different KDC server

In this section, we demonstrate the steps that are required to set up an NFS V4 server in your NAS environment by using the KDC server created in “Installing the IBM NAS file sets” on page 135 to act as a NFS V4 server as well.

5.7.1 Create the NFS server keytab file entry

As described previously, the principal for the NFS server has already been created as:

```
nfs/nfs404.itsc.austin.ibm.com@REALM1.IBM.COM
```

The next step is to set up a keytab entry on the NFS V4 server using the **kadmin** command as shown in Example 5-30.

Example 5-30 Sample ktadd to create an NFS server keytab file

```
kadmin: ktadd nfs/nfs404.itsc.austin.ibm.com
Entry for principal nfs/nfs404.itsc.austin.ibm.com with kvno 3, encryption type
Triple DES cbc mode with HMAC/sha1 added to keytab
WRFIELD:/etc/krb5/krb5.keytab.
Entry for principal nfs/nfs404.itsc.austin.ibm.com with kvno 3, encryption type
ArcFour with HMAC/md5 added to keytab WRFIELD:/etc/krb5/krb5.keytab.
```



```
Entry for principal nfs/nfs404.itsc.austin.ibm.com with kvno 3, encryption type
AES-256 CTS mode with 96-bit SHA-1 HMAC added to keytab
WRFILE:/etc/krb5/krb5.keytab.
Entry for principal nfs/nfs404.itsc.austin.ibm.com with kvno 3, encryption type
DES cbc mode with RSA-MD5 added to keytab WRFILE:/etc/krb5/krb5.keytab.
```

This command created the `/etc/krb5/krb5.keytab` file on the local system.

5.7.2 Check the NFS V4 server before client access

To be sure that the setup on the NFS V4 server is correct, we check the following:

- Are the keytab entries valid? Use the `ktutil` command.

Example 5-31 Check of valid keytab on the NFS V4 server

```
# /usr/krb5/sbin/ktutil
ktutil: read_kt /etc/krb5/krb5.keytab
ktutil: 1
slot  KVNO  Principal
-----
      1     3  nfs/nfs404.itsc.austin.ibm.com@REALM2.IBM.COM
      2     3  nfs/nfs404.itsc.austin.ibm.com@REALM2.IBM.COM
      3     3  nfs/nfs404.itsc.austin.ibm.com@REALM2.IBM.COM
      4     3  nfs/nfs404.itsc.austin.ibm.com@REALM2.IBM.COM
```

- Can the server principal get valid tickets? Use the `kinit` command.

Example 5-32 kinit using the server keytab file

```
# kinit -kt /etc/krb5/krb5.keytab\ nfs/nfs404.itsc.austin.ibm.com
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: nfs/nfs404.itsc.austin.ibm.com@REALM2.IBM.COM

Valid starting   Expires           Service principal
07/28/04 17:12:16  07/29/04 17:12:16  krbtgt/REALM2.IBM.COM@REALM2.IBM.COM
#
```

5.7.3 Set up the NFS registry daemon

The NFS domain name has to be set. The current setting can be checked using `chnfsdom` or `smitty chnfsdom` commands.

If the NFS domain is not set, the output appears as:

```
# chnfsdom
Current local domain: N/A
```

In our sample environment, we already set the NFS domain so that the output appears as:

```
# chnfsdom
Current local domain: nfs.ibm.com
```

Note: Changing the NFS domain does not recycle or start the `nfsrgyd`. You have to start and recycle the daemon manually.

We started the `nfsrgyd` by using the `startsrc -s nfsrgyd` command:

```
# startsrc -s nfsrgyd
0513-059 The nfsrgyd Subsystem has been started. Subsystem PID is 14496.
```

5.7.4 Set up the `gssd` daemon on the NFS V4 server

To enable NFS V4 using RPSEC-GSS, we have to create the map file between the server's keytab file and the NFS server principal. This is done using the `nfshostkey` command as shown in Example 5-33.

Example 5-33 Creating the server's hostkey map file

```
# nfshostkey -p nfs/nfs404.itsc.austin.ibm.com -f /etc/krb5/krb5.keytab
#
# nfshostkey -l
nfs/nfs404.itsc.austin.ibm.com
/etc/krb5/krb5.keytab
#
```

To set up the `gssd` daemon to start up automatically, run the `chnfs -S -B` or `smitty addsecurity` command. You have to repeat this setup for each NFS V4 server in your infrastructure.

The next step is to create the file `/etc/nfs/realm.map`. Use the `chnfsrtd` command. In our sample environment, the realm is `REALM2.IBM.COM` and the NFS domain is `itsc.austin.ibm.com`.

We used the command `chnfsrtd -a REALM2.IBM.COM itsc.austin.ibm.com` or `smit chnfsrtd`.

Example 5-34 Setting the realm.map file on the NFS V4 server

```
# cat /etc/nfs/realm.map
realm2.ibm.com itsc.austin.ibm.com
```

```
# chnfsrtd
realm2.ibm.com itsc.austin.ibm.com
```

Note: The realm entry in the `/etc/nfs/realm.map` file is not case-sensitive.

5.8 Setting up an NFS V4 client with NAS

In this section, we describe the steps that are required to set up a client system to use the created NAS and NFS V4 environment. We describe the basic required steps as well as the differences between a full client and slim client.

5.8.1 General steps for all types of clients

In general, the steps required to set up an NFS V4 client with NAS support are:

1. Install the NAS file set from the AIX 5.3 Expansion CD.
2. Set the NFS domain using `smit chnfsdom` or `chnfsdom` command.
3. Set the NFS domain-to-realm mapping using `chnfsrtd`.
4. Set up the NAS environment:
 - Full client: Create the NFS host principal and keytab file.
 - Slim client: Retrieve the required configuration files from a master.

For details about slim and full clients, see 4.4.4, “NFS client considerations when using Kerberos” on page 108 in this book.

5. Configure the `gssd` daemon after the Kerberos installation and verification have been completed.

5.8.2 Install the NAS client code

Next, we install IBM NAS Version 1.4 from the AIX 5.3 Expansion CD. Install the following file sets using the `smit` or `installp` commands:

krb5.client	Delivers all required client commands and libraries.
modcrypt.base	Delivers the required kernel extension for using NFS V4.

Example 5-35 shows the installation process and the file sets that are installed as part of the above.

Example 5-35 NAS client file set installation

```
# installp -aqXgYd . krb5.client modcrypt.base
```

```
+-----+
```

Installation Summary

Name	Level	Part	Event	Result
modcrypt.base.lib	5.3.0.0	USR	APPLY	SUCCESS
modcrypt.base.includes	5.3.0.0	USR	APPLY	SUCCESS
krb5.client.rte	1.4.0.0	USR	APPLY	SUCCESS
krb5.client.samples	1.4.0.0	USR	APPLY	SUCCESS
krb5.client.rte	1.4.0.0	ROOT	APPLY	SUCCESS

5.8.3 Set up the NFS domain

In Example 5-36, we set the NFS domain to `itsc.austin.ibm.com` by using the `chnfsdom` command.

Example 5-36 Client NFS domain setup

```
# chnfsdom
Current local domain: N/A
#
# chnfsdom itsc.austin.ibm.com
#
# chnfsdom
Current local domain: itsc.austin.ibm.com
#
```

5.8.4 Set up the NFS domain-to-realm map

The AIX 5.3 implementation of NFS V4 requires that you have a cross-realm definition between the NFS domain and the Kerberos realm that is used. This is done by generating a realm map file by using the `smit` or `chnfsrtd` commands.

Example 5-37 Client NFS domain-to-realm mapping

```
# chnfsrtd -a REALM1.IBM.COM itsc.austin.ibm.com
#
#chnfsrtd
realm1.ibm.com itsc.austin.ibm.com
#
```

At this point, you should start the necessary NFS processes by running the `/etc/rc.nfs` script. Before completing the client setup by enabling `RPSSEC_GSS`, we have to set up the NAS environment by using a full or slim NFS V4 Kerberos client, which we describe in the next two sections.

5.8.5 Full client installation steps

After installing the NAS client file sets, we configure the Kerberos client using the `mkkrb5clnt` command as shown in Example 5-38.

Example 5-38 mkkrb5clnt output

```
# mkkrb5clnt -d itsc.austin.ibm.com -r REALM1.IBM.COM -c\  
nfs407.itsc.austin.ibm.com -s nfs407.itsc.austin.ibm.com  
#  
Initializing configuration...  
Creating /etc/krb5/krb5_cfg_type...  
Creating /etc/krb5/krb5.conf...  
The command completed successfully.  
#
```

The same result can be achieved by using the `config.krb5` command with options `-C -d itsc.austin.ibm.com -r REALM1.IBM.COM -c nfs407.itsc.austin.ibm.com -s nfs407.itsc.austin.ibm.com`

The NFS server (or an NFS V4 Full Kerberos client), using `RPCSEC_GSS` security, must be able to acquire credentials for its service principal `nfs/<host_name>@REALM` to authenticate requests. This type of principal must be created using the `kadmin` command on either a server or the client during installation. We create this principal on the client. For this operation, you should know the Kerberos administrative principal (default is `admin/admin@REALM`) and the password for the administrative principal.

In addition, this operation verifies that the client can communicate with the KDC server.

Example 5-39 kinit to admin/admin

```
# kinit admin/admin  
Password for admin/admin@REALM1.IBM.COM:  
#  
# klist  
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0  
Default principal: admin/admin@REALM1.IBM.COM  
  
Valid starting Expires Service principal  
08/11/04 19:16:11 08/12/04 19:15:33 krbtgt/REALM1.IBM.COM@REALM1.IBM.COM  
#
```

The next step is to add the NFS service principal using the `kadmin` command. Example 5-40 on page 148 shows that we used the `kadmin: listprincs` commands to verify that no service principal of type

nfs/nfs403.itsc.austin.ibm.com has been registered for the host nfs403.itsc.austin.ibm.com. If this is the case, you can omit the principal creation and continue with creation of the keytab file.

Example 5-40 kadmin to add the client service principal

```
# /usr/krb5/sbin/kadmin
Authenticating as principal admin/admin@REALM1.IBM.COM with password.
Password for admin/admin@REALM1.IBM.COM:
kadmin: listprincs
joe@REALM1.IBM.COM
mary@REALM1.IBM.COM
bob@REALM1.IBM.COM
sally@REALM1.IBM.COM
host/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM
root/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM
admin/admin@REALM1.IBM.COM
kadmin:
kadmin: add_principal -e des-cbc-crc:normal -randkey
nfs/nfs403.itsc.austin.ibm.com
WARNING: no policy specified for nfs/nfs403.itsc.austin.ibm.com@REALM1.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.
Principal "nfs/nfs403.itsc.austin.ibm.com@REALM1.IBM.COM" created.
kadmin:
```

Now we create the keytab file so that the machine can request a valid ticket after reboot. This enables the machine to mount NFS V4 directories with the security flavor sec=krb5.

Example 5-41 Creating the client keytab file

```
kadmin: ktadd nfs/nfs403.itsc.austin.ibm.com
Entry for principal nfs/nfs403.itsc.austin.ibm.com with kvno 3, encryption type
Triple DES cbc mode with HMAC/sha1 added to keytab
WRFIL:/etc/krb5/krb5.keytab.
Entry for principal nfs/nfs403.itsc.austin.ibm.com with kvno 3, encryption type
ArcFour with HMAC/md5 added to keytab WRFIL:/etc/krb5/krb5.keytab.
Entry for principal nfs/nfs403.itsc.austin.ibm.com with kvno 3, encryption type
AES-256 CTS mode with 96-bit SHA-1 HMAC added to keytab
WRFIL:/etc/krb5/krb5.keytab.
Entry for principal nfs/nfs403.itsc.austin.ibm.com with kvno 3, encryption type
DES cbc mode with RSA-MD5 added to keytab WRFIL:/etc/krb5/krb5.keytab.
kadmin:
```

This step created the local keytab file krb5.keytab in the /etc/krb5 directory. If this keytab file is generated on another machine, such as a Windows KDC server,

you can transfer it using binary **ftp** to the correct location on the client and import the file using the **ktutil** command.

We verify the created keytab file using the **kinit** command (Example 5-42).

Example 5-42 kinit using the client keytab file

```
# kinit -kt /etc/krb5/krb5.keytab nfs/nfs403.itsc.austin.ibm.com
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: nfs/nfs403.itsc.austin.ibm.com@REALM1.IBM.COM

Valid starting    Expires          Service principal
08/11/04 19:47:27  08/12/04 19:47:27  krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
#
```

To enable NFS V4 to use RPSEC-GSS, we create a map between the client keytab file and the NFS service principal. This is done using the **nfshostkey** command (Example 5-43).

Example 5-43 hostkey map creation on the client

```
# nfshostkey -p nfs/nfs403.itsc.austin.ibm.com -f /etc/krb5/krb5.keytab
#
# nfshostkey -l
nfs/nfs403.itsc.austin.ibm.com
/etc/krb5/krb5.keytab
#
```

We again verify that the **nfshostkey** map file can be used to get a valid Kerberos ticket.

Example 5-44 Verification of the client nfs hostkey

```
# kdestroy
#
# kinit -kt `tail -n 1 /etc/nfs/hostkey` `head -n 1 /etc/nfs/hostkey`
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: nfs/nfs403.itsc.austin.ibm.com@REALM1.IBM.COM

Valid starting    Expires          Service principal
08/11/04 19:50:52  08/12/04 19:50:16  krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
#
```

The successful verification completes the full client installation.

5.8.6 Slim client installation steps

As described in “NFS V4 slim client versus full client” on page 108, a slim client does not have a service principal registered within the KDC server. In addition, you cannot create a keytab file, so no Kerberos ticket can be issued automatically on reboot.

Restriction: While writing this book, we learned that a slim client is not capable of automatically mounting file systems that require RPCSEC_GSS authorization only. In this case, the security flavor on the server must be at least `sec=sys:krb5(x)`.

Another recommended approach is to utilize the NFS V4 pseudo-FS model and use a single client mount of the server’s exported pseudo-root. The pseudo-root typically allows non-RPCSEC-GSS access.

The following are reasons why you would want to deploy slim clients in your infrastructure:

- ▶ Pre-installed new systems
- ▶ Mass rollout of new systems
- ▶ Unprompted upgrade of clients

In addition, there are several choices for how to install the slim clients, based on the way your systems are installed:

- ▶ Complete new installation (scratch installation)
- ▶ Cloning by use of a full system backup image

In the following section, we show how a slim client can be installed:

- ▶ To be used as a master image for cloning purposes
- ▶ Using the NAS and NFS V4 configuration files from the master machine

The provided examples show the configuration of a slim client. They can be used if the requirement is met and all targeted systems are to be in the same NFS domain and Kerberos realm. Otherwise, the slim client will not function properly and you will have to manually change the Kerberos configuration, the NFS domain configuration, or both.

Configure a slim client for cloning

We decided to use the following method for cloning:

1. Install one client with all required file sets to use NAS as described in “Full client installation steps” on page 147.

2. Configure the Kerberos client using the `mkkrb5clnt` command, but do not create a service principal or a keytab file.
3. Set up the NFS domain and enable `RPCSEC_GSS`.
4. Create all required `/etc/filesystem` entries.
5. Create a full system backup image to be used by NIM.

Example 5-45 Slim client configuration for cloning

```
# lsipp -L krb5*
Fileset                               Level State Type Description (Uninstaller)
-----
krb5.client.rte                        1.4.0.0 C F Network Authentication
Service
krb5.client.samples                    1.4.0.0 C F Network Authentication
Service
Samples

#
# mkkrb5clnt -d itsc.austin.ibm.com -r REALM1.IBM.COM -c\
nfs407.itsc.austin.ibm.com
Initializing configuration...
Creating /etc/krb5/krb5_cfg_type...
Creating /etc/krb5/krb5.conf...
The command completed successfully.
#
# chnfsdom
Current local domain: N/A
#
# chnfsdom nfs.ibm.com
# chnfsdom
Current local domain: nfs.ibm.com
#
# chnfs -S -B
0513-059 The gssd Subsystem has been started. Subsystem PID is 14812.
#
# /etc/rc.nfs
Starting NFS services:
0513-059 The biod Subsystem has been started. Subsystem PID is 22284.
0513-059 The nfsrgyd Subsystem has been started. Subsystem PID is 21812.
0513-059 The rpc.statd Subsystem has been started. Subsystem PID is 21118.
0513-059 The rpc.lockd Subsystem has been started. Subsystem PID is 14522.
0513-029 The gssd Subsystem is already active.
Multiple instances are not supported.
Completed NFS services.
#
# klist
Unable to get cache name (ticket cache: /var/krb5/security/creds/krb5cc_0).
Status 0x96c73ac3 - No credentials cache found.
```

```
#
# ls /etc/krb5
krb5.conf          krb5_cfg_type
#
# ls /etc/nfs
local_domain
#
```

Configure a slim client on a preinstalled AIX 5.3 system

To be able to configure an already installed AIX 5.3 system as slim client, we used the following method:

1. Create a tar image of the basic NAS and NFS configuration files on the master system as described above (Example 5-46).

This is the minimal list of files that are needed on a slim client. If your environment requires the need to have, for example, Enterprise Identity Mapping, there will be more files in the /etc/nfs directory.

Example 5-46 Creating a tar image of the basic NAS and NFS configuration files

```
# ls /etc/nfs/* > /tmp/SlimClientInList
#
# ls /etc/krb5/* >> /tmp/SlimClientInList
#
# tar -cvf /tmp/SlimClientImage.tar -L /tmp/SlimClientInList
a /etc/nfs/local_domain 1 blocks.
a /etc/nfs/realm.map 1 blocks.
a /etc/krb5/krb5.conf 2 blocks.
a /etc/krb5/krb5_cfg_type 1 blocks.
#
```

2. Copy the tar image onto the target system.
3. Install the contents of the tar image on the system (Example 5-47).

Example 5-47 Installing the contents of the tar image onto the target system

```
# tar -xvf /tmp/SlimClientImage.tar
x /etc/nfs/local_domain, 12 bytes, 1 media blocks.
x /etc/nfs/realm.map, 68 bytes, 1 media blocks.
x /etc/krb5/krb5.conf, 864 bytes, 2 media blocks.
x /etc/krb5/krb5_cfg_type, 7 bytes, 1 media blocks.
#
# kinit sally
Password for sally@REALM1.IBM.COM:
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
```

Default principal: sally@REALM1.IBM.COM

```
Valid starting    Expires          Service principal
08/13/04 15:24:45 08/14/04 15:24:43 krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
#
```

Verification of the slim client

On the server, validate the NFS pseudo-root. Include, at the very least, the security flavor `sec=sys` to enable mount during startup on the slim client. The following file systems have been exported on the server:

- ▶ `/exports -vers=4,sec=sys:krb5,rw`
- ▶ `/exports/home -vers=4,sec=sys:krb5,rw`
- ▶ `/exports/project/projA -vers=4,sec=sys:krb5,ro`
- ▶ `/exports/project/projB -vers=4,sec=sys:krb5,ro`

The slim client will not be able to mount a file system using security flavor `sec=krb5` without having valid Kerberos tickets. To achieve automatic mount of the pseudo-root FS during startup, the security flavor `sec=sys` on the server and the client for the NFS pseudo-root FS has to be used.

We used `smitty mknfsmnt` to create NFS pseudo-root FS mount on the client as shown in Example 5-48.

Example 5-48 Slim client /etc/filesystems NFS mount

```
# tail /etc/filesystems
/nfs:
    dev          = "/"
    vfs          = nfs
    nodename     = nfs404
    mount        = true
    options      = bg,hard,intr,vers=4,sec=sys:krb5
    account      = false
```

The slim client is now rebooted and with UID root, we access the NFS remote mounted directory.

Example 5-49 Slim client access after reboot

```
# uptime
 12:22PM up 8 mins, 1 user, load average: 0.02, 0.26, 0.19
#
#klist
Unable to get cache name (ticket cache: /var/krb5/security/creds/krb5cc_0).
Status 0x96c73ac3 - No credentials cache found.
#
```

```

# id
uid=0(root) gid=0(system)
groups=2(bin),3(sys),7(security),8(cron),10(audit),11(lp)
#
#mount
node          mounted          mounted over    vfs      date          options
-----
                /dev/hd4          /                jfs      Aug 12 12:13  rw,log=/dev/hd8
                /dev/hd2          /usr             jfs      Aug 12 12:13  rw,log=/dev/hd8
                /dev/hd9var       /var             jfs      Aug 12 12:13  rw,log=/dev/hd8
                /dev/hd3          /tmp             jfs      Aug 12 12:13  rw,log=/dev/hd8
                /dev/hd1          /home            jfs      Aug 12 12:14  rw,log=/dev/hd8
                /proc             /proc            procfs   Aug 12 12:14  rw
                /dev/hd10opt      /opt             jfs      Aug 12 12:14  rw,log=/dev/hd8
                /dev/log_lv       /var/nfs4log    jfs      Aug 12 12:14  rw,log=/dev/hd8
nfs404 /                /nfs             nfs4     Aug 12 12:14
rw,bg,hard,intr,vers=4,sec=sys:krb5
#
# cd /nfs
#
# ls
dept    home    local  project tmp
#

```

This system is now ready to be used as the master full system backup for prompted installation of the clients.

5.8.7 Configuring RPCSEC_GSS on the clients

If you want to use Kerberos authentication on the client, you must enable NFS V4 enhanced security on the client. You can enable enhanced security using **smi** **Configure NFS on This System** → **Enable RPCSEC_GSS** or by using the **chnfs -S -B** command:

```

# chnfs -S -B
0513-059 The gssd Subsystem has been started. Subsystem PID is 22158.
#

```

At this point, the client setup is finished and can mount and access directories with enhanced security.

Example 5-50 Verify the client using mount command

```

# mount -o vers=4,sec=krb5 nfs404:/ /nfs
#
# nfs4cl showfs /nfs

```

Server	Remote Path	fsid	Local Path
nfs404	/	10:4	/nfs

Current Server: nfs404:/nfs

options :
 rw,intr,rsize=32768,wsz=32768,timeo=300,retrans=5,biods=0,numclust=2,maxgroup
 s=0,acregmin=3,acregmax=60,acdirmin=30,acdirmax=60,sec=krb5
 #

The client is now ready for deploying NFS V4.

5.9 Preparing the system for Tivoli Directory Server and Kerberos V5

In this section, we lead you through the mandatory preparation steps on your AIX 5.3 system before you can deploy Kerberos V5 with the IBM Tivoli Directory Server LDAP backend.

Attention: As we write this book, there is a requirement for the system to be running in 64-bit kernel mode. This is required for the db2_08_01.ldap file set, which gives you the LDAP DB2® backend. This may change in future releases of the file set.

The following definitions are used in the example:

NFS server	hostname = nfs404, OS = AIX 5.3
NFS client	hostname = nfs405, OS: AIX 5.3
KDC and LDAP server	hostname = nfs407, OS: AIX 5.3
NFS Domain Name	itsc.austin.ibm.com
Realm Name	REALM1.IBM.COM

Figure 5-2 on page 156 gives a better view of the setup.

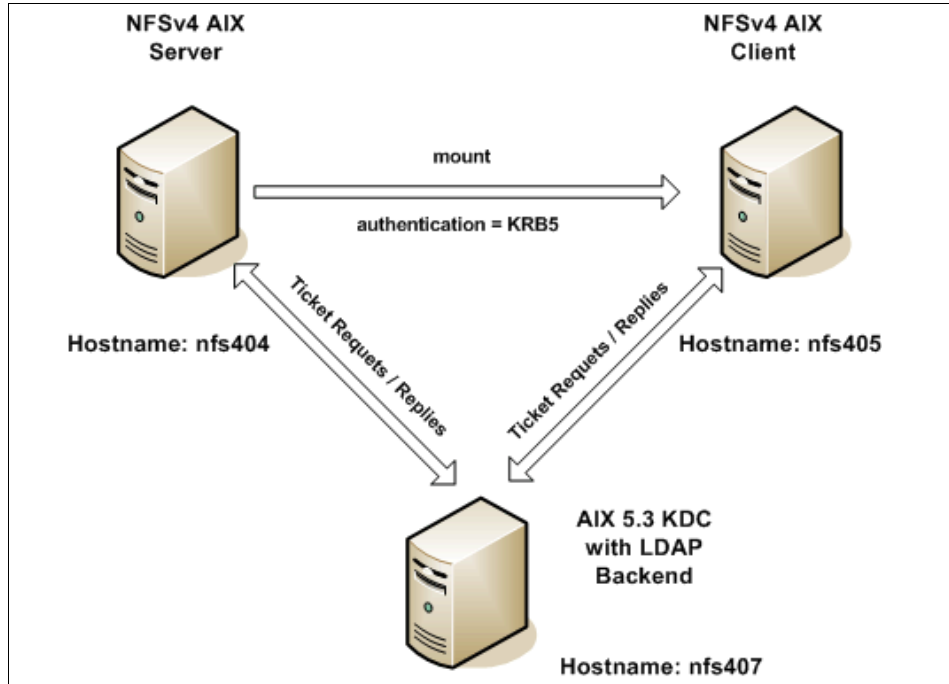


Figure 5-2 Sample AIX environment with an AIX KDC and LDAP backend

5.9.1 Set up procedure

The following procedure needs to be carried to ensure that your system is running a 64-bit kernel.

1. Check to see if your system is capable of running in 64-bit mode:

```
# bootinfo -y
64
#
```

If the result is 32, then your system is a 32-bit machine and you will not be able to proceed any further. If the result is as shown in the previous example, then continue with the rest of the steps.

2. We confirm what mode the system is running in:

```
# bootinfo -K
32
#
```

As can be seen from the output, the system is currently running in 32-bit mode. Before we can progress with the install, we have to configure it to run in 64-bit mode.

3. To change the system to run in 64-bit mode:

```
# cd /
#
# ln -sf /usr/lib/boot/unix_64 /unix
#
# ls -al /unix
lrwxrwxrwx 1 root system 21 Aug 04 15:30 /unix -> /usr/lib/boot/unix_64
#
# bosboot -ad /dev/ipldevice

bosboot: Boot image is 22469 512 byte blocks.
#
```

4. We now make sure that aio is enabled at system start.

```
# lsattr -El aio0
autoconfig defined STATE to be configured at system restart True
fastpath enable State of fast path True
kprocprio 39 Server PRIORITY True
maxreqs 4096 Maximum number of REQUESTS True
maxservers 10 MAXIMUM number of servers per cpu True
minservers 1 MINIMUM number of servers True
```

As the example shows, autoconfig is in a defined state for aio, and we need to change this to an *available* state. Use the following command and change STATE to be configured at system restart to available:

```
# smitty chgaio
```

5. We now reboot the system to allow for the 64-bit kernel.
6. After the system has rebooted, verify that it is running in 64-bit mode:

```
# bootinfo -K
64
#
```

7. We also confirm that aio is now in available state:

```
# lsattr -El aio0
autoconfig available STATE to be configured at system restart True
fastpath enable State of fast path True
kprocprio 39 Server PRIORITY True
maxreqs 4096 Maximum number of REQUESTS True
maxservers 10 MAXIMUM number of servers per cpu True
minservers 1 MINIMUM number of servers True
#
```

8. Next, we need to create a 350 MB file system so that we can install the DB2 binaries. (We want to keep the root file system as small as possible.) The new file system should be created to mount on to /usr/opt/db2_08_01. You will need to change the owner and group to bin respectively.
9. We also need to create a 200 MB file system to mount to /home/ldapdb2.
10. Create a group called dbsysadm and add the root user to this group.
11. Create a user called ldapdb2. This user must have these characteristics:
 - a. Primary group: dbsysadm
 - b. Password REGISTRY: files
 - c. HOME directory: /home/ldapdb2
 - d. Change the owner and group of /home/ldapdb2 to ldapdb2:dbsysadm

Important: Make sure that user ldapdb2 has a password assigned and can log on without any challenges before proceeding.

12. Add the following entries to /etc/services:

```
DB2_ldapdb2      60000/tcp
DB2_ldapdb2_1   60001/tcp
DB2_ldapdb2_2   60002/tcp
DB2_ldapdb2_END 60003/tcp
```

13. The following file sets must be installed for IBM Directory Server. They can be found on the AIX Base Installation media:

- ldap.client.adt
- ldap.client.rte
- ldap.html.en_US.config
- ldap.html.en_US.man
- ldap.msg.en_US
- ldap.server.cfg
- ldap.server.com
- ldap.server.java
- ldap.server.rte
- ldap.webdadmin
- ldap.client.rte
- ldap.server.cfg
- ldap.server.com
- db2_08_01.ldap

To install the file sets, this command was used:

```
# installp -aXYgd /dev/cd0 ldap.server.rte
```


At the end of the install, the screen in Example 5-51 should confirm a successful install.

Example 5-51 Successful install of ldap.rte.server

```

+-----+
                        Summaries:
+-----+

Installation Summary
-----
Name                               Level           Part           Event          Result
-----
ldap.client.rte                    5.2.0.0        USR             APPLY          SUCCESS
ldap.client.adt                    5.2.0.0        USR             APPLY          SUCCESS
ldap.client.rte                    5.2.0.0        ROOT           APPLY          SUCCESS
ldap.server.java                   5.2.0.0        USR             APPLY          SUCCESS
db2_08_01.client                   8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.cnvucs                   8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.conv                     8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.db2.rte                  8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.db2.samples              8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.essg                     8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.icuc                     8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.icut                     8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.jdbc                     8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.jhlp.en_US.iso8859      8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.cj                       8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.ldap                     8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.msg.en_US.iso8859       8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.pext                     8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.rep1                     8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.sqlproc                  8.1.1.16      USR             APPLY          SUCCESS
ldap.server.rte                    5.2.0.0        USR             APPLY          SUCCESS
ldap.server.com                    5.2.0.0        USR             APPLY          SUCCESS
ldap.server.cfg                    5.2.0.0        USR             APPLY          SUCCESS
ldap.server.com                    5.2.0.0        ROOT           APPLY          SUCCESS
ldap.server.cfg                    5.2.0.0        ROOT           APPLY          SUCCESS
db2_08_01.conn                     8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.cs.rte                   8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.das                       8.1.1.16      USR             APPLY          SUCCESS
db2_08_01.db2.engn                 8.1.1.16      USR             APPLY          SUCCESS
#

```

14. Install the Kerberos V5 file sets as described in 5.6.2, “Installing the IBM NAS file sets” on page 135.

Note: If you need SSL capabilities, install the AIX Certificate and SSL base file sets (gskta for 64-bit kernel, gksa for 32-bit kernel) and the crypto file sets from the AIX 5.3 Expansion Pack. Both gskta and gksa are required. gksa is used by the LDAP client apps such as ldapsearch.

We have now successfully installed all of the software that is required to support KDC and RFC2307 with IBM Directory Server.

5.9.2 Configure IBM Tivoli Directory Server

After installing the required programs, we now proceed with the initial configuration:

1. Ensure that **ibmslapd** is not running; otherwise the command in the next step would fail with the following message:

```
08/13/04 15:55:15 Non-SSL port initialized to 389.  
08/13/04 15:55:16 Attempt to bind failed; errno 67 (Address already in  
use).  
08/13/04 15:55:16 SocketInit failed for port 389.
```

2. We start with configuring the directory.

Example 5-52 Configuration of the Directory Server

```
# mksecdap -s -a cn=admin -p succ3ss -S rfc2307aix -u ALL  
Filesystem size changed to 212992  
ldapdb2's New password:  
Enter the new password again:
```

You have chosen the following actions:

Administrator DN 'cn=admin' and password will be set.

Setting administrator DN 'cn=admin' and password.

Set administrator DN 'cn=admin' and password.

IBM Tivoli Directory Server Configuration complete.

You have chosen the following actions:

Database 'ldapdb2' will be configured in instance 'ldapdb2'.

Configuring IBM Tivoli Directory Server Database.

Creating instance: 'ldapdb2'.

Created instance: 'ldapdb2'.

Cataloging instance node: 'ldapdb2'.

Cataloged instance node: 'ldapdb2'.

```
Starting database manager for instance: 'ldapdb2'.
Started database manager for instance: 'ldapdb2'.
Creating database: 'ldapdb2'.
Created database: 'ldapdb2'.
Updating the database: 'ldapdb2'
Updated the database: 'ldapdb2'
Updating the database manager: 'ldapdb2'
Updated the database manager: 'ldapdb2'
Enabling multi-page file allocation: 'ldapdb2'
Enabled multi-page file allocation: 'ldapdb2'
Configuring database: 'ldapdb2'
Configured database: 'ldapdb2'
Adding local loop back to database: 'ldapdb2'.
Added local loop back to database: 'ldapdb2'.
Stopping database manager for instance: 'ldapdb2'.
Stopped database manager for instance: 'ldapdb2'.
Starting database manager for instance: 'ldapdb2'.
Started database manager for instance: 'ldapdb2'.
Configured IBM Tivoli Directory Server Database.
```

```
IBM Tivoli Directory Server Configuration complete.
Server starting.
```

```
Plugin of type EXTENDEDOP is successfully loaded from libevent.a.
Plugin of type EXTENDEDOP is successfully loaded from libtranext.a.
Plugin of type EXTENDEDOP is successfully loaded from libldaprepl.a.
Plugin of type PREOPERATION is successfully loaded from libDSP.a.
Plugin of type PREOPERATION is successfully loaded from libDigest.a.
Plugin of type EXTENDEDOP is successfully loaded from libevent.a.
Plugin of type EXTENDEDOP is successfully loaded from libtranext.a.
Plugin of type AUDIT is successfully loaded from /lib/libldapaudit.a.
Plugin of type AUDIT is successfully loaded from
/usr/ccs/lib/libsecldapaudit64.a(shr.o).
Plugin of type EXTENDEDOP is successfully loaded from libevent.a.
Plugin of type EXTENDEDOP is successfully loaded from libtranext.a.
Plugin of type DATABASE is successfully loaded from /lib/libback-rdbm.a.
Plugin of type REPLICATION is successfully loaded from /lib/libldaprepl.a.
Plugin of type EXTENDEDOP is successfully loaded from /lib/libback-rdbm.a.
Plugin of type EXTENDEDOP is successfully loaded from libevent.a.
Plugin of type DATABASE is successfully loaded from /lib/libback-config.a.
Plugin of type EXTENDEDOP is successfully loaded from liblog.a.
Non-SSL port initialized to 389.
Migrating users and groups to LDAP server.
#
```

-
3. Although the DB2 database is now configured and running, we need to set the database to autostart after reboot of the system.

Example 5-53 Change autostart setting for DB2 instance ldapdb2

```
# su - ldapdb2
$
$ db2iauto -on ldapdb2
$
$ db2set
DB2COMM=TCPIP
DB2AUTOSTART=YES
$
```

4. We now tune the DB2 database.

Example 5-54 Tuning of the DB2 database

```
# su - ldapdb2
$
$ db2 update db cfg for ldapdb2 using DBHEAP 20000
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
$
$ db2 update db cfg for ldapdb2 using SORTHEAP 5000
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
$
$ db2 update db cfg for ldapdb2 using APPLHEAPSZ 10000
DB20000I The UPDATE DATABASE CONFIGURATION command completed successfully.
$
$ exit
#
```

5. We now stop the IBM Tivoli Directory Server to facilitate the addition of the container object for Kerberos:

```
# ibmdirctl -D cn=admin -w succ3ss stop
Stop operation succeeded
#
```

6. To be able to provide an LDAP backend to the KDC, run this command:

```
# ldapcfg -q -s "o=IBM,c=US"
#
```

We do this so that we do not have to use a legacy or file-based backend. For a detailed description of why this is done, refer to the *NAS Admin Guide IBM Network Authentication Service Version 1.4 for AIX, Linux, and Solaris Administrator's and User's Guide* delivered with the krb5.doc.en_US file set.

7. We now start the IBM Directory Server to enable us to add the Kerberos schema:

```
# ibmdirctl -D cn=admin -w succ3ss start
Start operation succeeded
#
```

8. We add the schema:

```
# ldapmodify -h localhost -D cn=admin -w succ3ss -f\  
/usr/krb5/ldif/IBM.KRB.schema.ldif -v -c  
#
```

9. We now create a schema for the KDC realm. Create /usr/ldap/etc/realm_add_ibm.ldif. (The NAS Admin Guide is used as reference for creating the schema file.) Add the following lines to the new file.

Example 5-55 Sample LDIF file for the KDC realm

```
# The suffix "ou=, o=, c=" should be defined before attempting to  
# load this data. Or change the suffix to be an already defined object.  
# Change all references of YOURHOSTNAME.DOMAINNAME to be your realm name  
#  
# version: 1
```

```
dn: o=IBM, c=US  
objectclass: top  
objectclass: organization  
o: IBM
```

```
dn: krbrealmName-V2=REALM1.IBM.COM, o=IBM, c=US  
objectclass: KrbRealm-V2  
objectclass: KrbRealmExt  
krbrealmName-V2: REALM1.IBM.COM  
krbprincSubtree: krbrealmName-V2=REALM1.IBM.COM, o=IBM, c=US  
krbDeleteType: 3
```

```
dn: cn=principal, krbrealmName-V2=REALM1.IBM.COM, o=IBM, c=US  
objectclass: container  
cn: principal
```

```
dn: cn=policy, krbrealmName-V2=REALM1.IBM.COM, o=IBM, c=US  
objectclass: container  
cn: policy
```

The sample file contents are shown in “LDIF sample file for KDC” on page 270.

10. We now modify the dn: o=IBM, c=US container by adding the schema.

Example 5-56 Adding the modified realm schema into LDAP

```
# ldapmodify -a -h localhost -D cn=admin -w succ3ss -f\  
/usr/ldap/etc/realm_add_ibm.ldif -v -c  
ldap_init(localhost, 389)  
add objectclass:
```

```

        BINARY (3 bytes) top
        BINARY (12 bytes) organization
add o:
        BINARY (4 bytes) IBM
adding new entry o=IBM, c=US

add objectclass:
        BINARY (11 bytes) KrbRealm-V2
        BINARY (11 bytes) KrbRealmExt
add krbrealmName-V2:
        BINARY (15 bytes) REALM1.IBM.COM
add krbprincSubtree:
        BINARY (45 bytes) krbrealmName-V2=REALM1.IBM.COM, o=IBM, c=US
add krbDeleteType:
        BINARY (1 bytes) 3
adding new entry krbrealmName-V2=REALM1.IBM.COM, o=IBM, c=US

add objectclass:
        BINARY (9 bytes) container
add cn:
        BINARY (9 bytes) principal
adding new entry cn=principal, krbrealmName-V2=REALM1.IBM.COM, o=IBM, c=US

add objectclass:
        BINARY (9 bytes) container
add cn:
        BINARY (6 bytes) policy
adding new entry cn=policy, krbrealmName-V2=REALM1.IBM.COM, o=IBM, c=U
#

```

11. At this point we can verify the IBM Directory using a simple ldap query that shows all available container names in the newly created LDAP directory.

Example 5-57 Verify the LDAP naming contexts

```

# ldapsearch -b "" -s base "objectclass=*" namingcontexts

namingcontexts=CN=SCHEMA
namingcontexts=CN=LOCALHOST
namingcontexts=CN=AIXDATA
namingcontexts=CN=PWDPOLICY
namingcontexts=CN=IBMPOLICIES
namingcontexts=O=IBM,C=US
#

```

5.9.3 Configure the KDC server with LDAP backend

1. Make sure that the NAS V1.4 server file sets are installed on the system. The next step is to configure the KDC server.

Example 5-58 Creating the KDC server using mkkrb5srv

```
# mkkrb5srv -r REALM1.IBM.COM -s nfs407.itsc.austin.ibm.com \  
-d itsc.austin.ibm.com -a admin/admin -l nfs407.itsc.austin.ibm.com \  
-u "cn=admin" -p succ3ss
```

Fileset	Level	State	Description

Path: /usr/lib/objrepos			
krb5.server.rte	1.4.0.0	COMMITTED	Network Authentication Service Server
Path: /etc/objrepos			
krb5.server.rte	1.4.0.0	COMMITTED	Network Authentication Service Server

The -s option is not supported.

The administration server will be the local host.

Initializing configuration...

Creating /etc/krb5/krb5_cfg_type...

Creating /etc/krb5/krb5.conf...

Creating /var/krb5/krb5kdc/kdc.conf...

Creating database files...

Initializing database 'LDAP' for realm 'REALM1.IBM.COM'

master key name 'K/M@REALM1.IBM.COM'

Attempting to bind to one or more LDAP servers. This may take a while...

You are prompted for the database Master Password.

Enter database Master Password:

Re-enter database Master Password to verify:

Attempting to bind to one or more LDAP servers. This may take a while...

WARNING: no policy specified for admin/admin@REALM1.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.

Enter password for principal "admin/admin@REALM1.IBM.COM":

Re-enter password for principal "admin/admin@REALM1.IBM.COM":

Principal "admin/admin@REALM1.IBM.COM" created.

Creating keytable...

Attempting to bind to one or more LDAP servers. This may take a while...

Creating /var/krb5/krb5kdc/kadm5.ac1...

Starting krb5kdc...

Attempting to bind to one or more LDAP servers. This may take a while...

krb5kdc was started successfully.

Starting kadmind...

Attempting to bind to one or more LDAP servers. This may take a while...

kadmind was started successfully.

```
The command completed successfully.
Restarting kadmind and krb5kdc
Attempting to bind to one or more LDAP servers. This may take a while...
Attempting to bind to one or more LDAP servers. This may take a while...
#
```

2. Check that the KDC server process krb5kdc and kadmind has been started.

Example 5-59 Verify the KDC server processes

```
# ps -ef |grep krb5 |grep -v grep
  root 290896      1   0 13:57:30    -   0:00 /usr/krb5/sbin/krb5kdc
  root 512076      1   0 13:57:30    -   0:00 /usr/krb5/sbin/kadmind
#
```

3. Now we log on to the KDC to check that a ticket can be issued.

Example 5-60 Verification of the KDC administrator

```
# kinit admin/admin
Password for admin/admin@REALM1.IBM.COM:
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: admin/admin@REALM1.IBM.COM

Valid starting    Expires          Service principal
08/05/04 14:07:07    08/06/04 14:07:03    krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
#
```

4. To be able to add, modify, and delete users and groups using the **mkgroup/mkuser -R KRB5LDAP** commands, we have to add the following authentication grammar to the `/usr/lib/security/methods.cfg` file and append the following lines.

Example 5-61 Authentication grammar in file /usr/lib/security/methods.cfg

```
LDAP:
    program = /usr/lib/security/LDAP
    program_64 =/usr/lib/security/LDAP64
KRB5:
    program = /usr/lib/security/KRB5

KRB5LDAP:
    options = db=LDAP,auth=KRB5
```

5. In addition, the local LDAP security client daemon must be running on the system. This can be accomplished with the following command.

Example 5-62 Enable the local security client daemon

```
# mksecldap -c -h nfs407.itsc.austin.ibm.com -a cn=admin -p succ3ss
#
```

6. To verify the LDAP security client daemon, you can use the `ls-secldapclntd` command.

Example 5-63 Output of command ls-secldapclntd

```
# /usr/sbin/ls-secldapclntd
ldapservers=nfs407.itsc.austin.ibm.com
ldapport=389
ldapversion=3
userbasedn=ou=aixuser,cn=aixsecdb,cn=aixdata
groupbasedn=ou=aixgroup,cn=aixsecdb,cn=aixdata
idbasedn=cn=aixid,ou=system,cn=aixsecdb,cn=aixdata
usercachesize=1000
usercacheused=0
groupcachesize=100
groupcacheused=0
cachetimeout=300
heartbeat=300
numberofthread=10
alwaysmaster=no
authtype=UNIX_AUTH
searchmode=ALL
defaultentrylocation=LDAP
ldaptimeout=60
userobjectclass=account,posixaccount,shadowaccount,aixauxaccount
groupobjectclass=posixgroup,aixauxgroup
#
```

At this point, the KDC and IBM Directory Server initial setup is complete.

In order to create users in the Kerberos realms and store the user/group identification information in the RFC2307, we run the following commands:

1. We added the root user to the realm to enable the AIX `mkuser` command to access the LDAP backend database.

Example 5-64 Creation of the root principal

```
# kinit admin/admin
Password for admin/admin@REALM1.IBM.COM:
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: admin/admin@REALM1.IBM.COM
```

```

Valid starting Expires Service principal
08/05/04 16:45:38 08/06/04 16:45:36 krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
#
# /usr/krb5/sbin/kadmin.local
Attempting to bind to one or more LDAP servers. This may take a while...
kadmin.local:
kadmin.local: add_principal root/nfs407.itsc.austin.ibm.com
WARNING: no policy specified for
root/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.
Enter password for principal "root/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM":
Re-enter password for principal
"root/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM":
Principal "root/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM" created.
kadmin.local: exit
#
# kdestroy
#

```

2. The next step is to verify the newly created principal. In addition, we need to have valid tickets to perform all further user creation steps (shown in Example 5-65).

Example 5-65 Verify the newly created principle

```

# kinit root/nfs407.itsc.austin.ibm.com
Password for root/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM:
#
# /usr/krb5/bin/klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: root/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM:

Valid starting Expires Service principal
08/06/04 10:39:24 08/07/04 10:39:23 krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
08/06/04 10:39:45 08/06/04 13:39:45
root/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM
#

```

3. Now we can create the users and groups as in Example 5-66.

Example 5-66 Creation of user and group in the KDC with LDAP backend

```

# mkgroup -R KRB5LDAP -a id=1400 eng
#
# mkuser -R KRB5LDAP id='6023' pgrp='staff' groups='eng' \ home='/home/sally'
shell='/bin/ksh' gecos='NFS V4 KDC Test user sally' sally
#
# passwd -R KRB5LDAP sally

```

```
sally's Old password:
sally's New password:
Enter the new password again:
#
```

4. Verify that the user sally is created correctly by executing the `lsuser sally` command.

Example 5-67 Output of command lsuser for user sally

```
# lsuser -R KRB5LDAP sally
sally id=6023 pgrp=staff groups=staff,eng home=/home/sally shell=/bin/ksh
gecos=NFSv4 KDC Test user sally login=true su=true rlogin=true telnet=true
daemon=true admin=false sugroups=ALL admgroups= tpath=nosak ttys=ALL expires=0
auth1=SYSTEM auth2=NONE umask=22 registry=KRB5LDAP SYSTEM=KRB5LDAP OR compat
logintimes= loginretries=0 pldwarntime=0 account_locked=false minage=0 maxage=0
maxexpired=-1 minalpha=0 minother=0 mindiff=0 maxrepeats=8 minlen=0
histexpire=0 histsize=0 pwdchecks= dictionlist= fsize=2097151 cpu=-1
data=262144 stack=65536 core=2097151 rss=65536 nofiles=2000 time_last_login=0
time_last_unsuccessful_login=0 unsuccessful_login_count=0 roles=
krb5_principal=sally@REALM1.IBM.COM krb5_principal_name=sally@REALM1.IBM.COM
krb5_realm=REALM1.IBM.COM maxage=0 expires=0 krb5_last_pwd_change=1091805538
admchk=false krb5_attributes=requires_preauth
krb5_mod_name=root/nfs407.itsc.austin.ibm.com@REALM1.IBM.COM
krb5_mod_date=1091805538 krb5_kvno=2 krb5_mkvno=0
krb5_max_renewable_life=604800 time_last_login=0 time_last_unsuccessful_login=0
unsuccessful_login_count=0 krb5_names=sally:nfs407.itsc.austin.ibm.com
#
```

5. To verify that the user ID is valid, try to log on as sally and display the Kerberos ticket details.

Example 5-68 Integrated logon with user sally

```
# telnet nfs407
Trying...
Connected to nfs407.itsc.austin.ibm.com.
Escape character is '^]'.

telnet (nfs407)
AIX Version 5
(C) Copyrights by IBM and by others 1982, 2004.
login: sally
sally's Password:
$
$ /usr/krb5/bin/klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_sally@REALM1.IBM.COM_6023
Default principal: sally@REALM1.IBM.COM
```

```

Valid starting    Expires          Service principal
08/06/04 10:39:24 08/07/04 10:39:23  krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
08/06/04 10:39:45 08/06/04 13:39:45  kadmin/admin@REALM1.IBM.COM
$

```

The user sally can log on to any machine in the Kerberos realm REALM1.IBM.COM without being defined as a local user (if the client has been set up with the previously mentioned authentication grammar). The user's identification will be stored in LDAP RFC2307.

Also, because user sally has a Kerberos ticket granted automatically during the AIX logon process, she can have access to all NFS V4 mounted file systems with `sec=krb5` option if provided ACLs are set for access.

All further steps to set up an NFS V4 server were described in previous sections.

5.9.4 Configure the NFS V4 client for integrated login services

We installed AIX 5.3 using the base install media. We now install the following additional file sets from the AIX 5.3 Expansion Pack:

- ▶ krb5.client.rte
- ▶ modcrypt.base

Example 5-69 Output of NAS file set installation verification

```

#lslpp -l | grep -i krb5
  krb5.client.rte          1.4.0.0  COMMITTED  Network Authentication Service
  krb5.client.samples     1.4.0.0  COMMITTED  Network Authentication Service
#
#lslpp -l |grep modcrypt
  modcrypt.base.includes  5.3.0.0  COMMITTED  Cryptographic Library Include
  modcrypt.base.lib       5.3.0.0  COMMITTED  Cryptographic Library
                                     (libmodcrypt.a)
#

```

1. The plan is to use AIX integrated login within KDC and IBM Directory Server as the LDAP backend, so we install the LDAP client software from the AIX 5.3. Base media.

Example 5-70 Output of LDAP client file set installation

```

# installp -agXd . ldap.client
Installation Summary
-----

```

Name	Level	Part	Event	Result
ldap.client.rte	5.2.0.0	USR	APPLY	SUCCESS

ldap.client.adt	5.2.0.0	USR	APPLY	SUCCESS
ldap.client.rte	5.2.0.0	ROOT	APPLY	SUCCESS
X11.adt.lib	5.3.0.0	USR	APPLY	SUCCESS
#				

2. Check that the system uses the correct PATH to locate the NAS binaries.

Example 5-71 Output of command type kinit

```
# type kinit
kinit is /usr/krb5/bin/kinit
#
```

3. Now we can configure this system into our NAS infrastructure as a client system using the **mkkrb5clnt** command, delivered by AIX BOS file set bos.rte.security. (nfs407.itsc.austin.ibm.com is the KDC serving the Kerberos realm REALM1.IBM.COM and also the IBM Tivoli Directory Server.)

Example 5-72 Output of Kerberos client configuration using mkkrb5clnt

```
# mkkrb5clnt -c nfs407.itsc.austin.ibm.com -r REALM1.IBM.COM -s
nfs407.itsc.austin.ibm.com -d itsc.austin.ibm.com -l nfs407.itsc.austin.ibm.com
-i files -A -K -T
Initializing configuration...
Creating /etc/krb5/krb5_cfg_type...
Creating /etc/krb5/krb5.conf...
The command completed successfully.
Password for admin/admin@REALM1.IBM.COM:
Configuring fully integrated login
Authenticating as principal admin/admin with existing credentials.
WARNING: no policy specified for
host/nfs405.itsc.austin.ibm.com@REALM1.IBM.COM;
  defaulting to no policy. Note that policy may be overridden by
  ACL restrictions.
Principal "host/nfs405.itsc.austin.ibm.com@REALM1.IBM.COM" created.

Administration credentials NOT DESTROYED.
Making root a Kerberos administrator
Authenticating as principal admin/admin with existing credentials.
WARNING: no policy specified for
root/nfs405.itsc.austin.ibm.com@REALM1.IBM.COM;
  defaulting to no policy. Note that policy may be overridden by
  ACL restrictions.
Enter password for principal "root/nfs405.itsc.austin.ibm.com@REALM1.IBM.COM":
Re-enter password for principal
"root/nfs405.itsc.austin.ibm.com@REALM1.IBM.COM":
Principal "root/nfs405.itsc.austin.ibm.com@REALM1.IBM.COM" created.
```

Administration credentials NOT DESTROYED.

Configuring Kerberos as the default authentication scheme
Cleaning administrator credentials and exiting.
#

4. We verify that the newly created principal can be used to authenticate with Kerberos.

Example 5-73 Principal root verification

```
# kinit root/nfs405.itsc.austin.ibm.com
Password for root/nfs405.itsc.austin.ibm.com@REALM1.IBM.COM:
root@nfs405 [ppc] klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: root/nfs405.itsc.austin.ibm.com@REALM1.IBM.COM

Valid starting    Expires          Service principal
08/10/04 09:54:00  08/11/04 09:53:50  krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
#
```

Note: This principal cannot be used for NFS V4 authentication. You still must create the service principal of type `nfs/<FQDN>@REALM`.

If using the `config.krb5` command, delivered with IBM NAS V1.4, the configuration will be valid, but the principal `nfs/<FQDN>@REALM` will not be created.

5. The next step is to add an NFS service (machine) principal and create a keytab file for the NFS V4 client `nfs402.itsc.austin.ibm.com`. This is done by carrying out the following process.

Example 5-74 Adding the NFS service principal

```
# kadmin -p admin/admin
Authenticating as principal admin/admin with password.
Password for admin/admin@REALM1.IBM.COM:
kadmin:
kadmin: add_principal -randkey -e des-cbc-crc:normal \
nfs/nfs402.itsc.austin.ibm.com
WARNING: no policy specified for nfs/nfs402.itsc.austin.ibm.com@REALM1.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.
Principal "nfs/nfs402.itsc.austin.ibm.com@REALM1.IBM.COM" created.
kadmin:
kadmin: ktadd nfs/nfs402.itsc.austin.ibm.com
Entry for principal nfs/nfs402.itsc.austin.ibm.com with kvno 3, encryption type
DES cbc mode with CRC-32 added to keytab
WRFILE:/etc/krb5/krb5.keytab.
```

```
kadmin:
kadmin: quit
#
```

6. Verify that the machine principal can log on using the keytab file that we just generated.

Example 5-75 Verification of the NFS service principal

```
# kinit -t /etc/krb5/krb5.keytab nfs/nfs402.itsc.austin.ibm.com
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: nfs/nfs402.itsc.austin.ibm.com@REALM1.IBM.COM
Valid starting Expires Service principal
07/30/04 11:08:14 07/31/04 11:08:14 krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
#
```

7. To create the hostkey map file, change the NFS domain and enable `RPCSEC_GSS` by executing the commands shown in Example 5-76.

Example 5-76 Enable NFS V4 on the system nfs402

```
# mkdir -p /etc/nfs
#
# nfshostkey -p nfs/nfs402.itsc.austin.ibm.com -f /etc/krb5/krb5.keytab
#
# chnfsdom itsc.austin.ibm.com
#
# /usr/sbin/chnfsrtd -a 'REALM1.IBM.COM' 'nfs.ibm.com'
#
# chnfs -S -B
#
# /etc/rc.nfs
#
```

8. The next step is to change the authentication grammar on `nfs402.itsc.austin.ibm.com` so that all users, except the root user, use integrated logon. We decided to change the Password Registry for the root user to files and the authentication grammar to `compat`:

```
# chuser registry=files root
#
# chuser SYSTEM="compat" root
#
```

9. We also make the following changes to the default stanza in the `/etc/security/user` file:

```
SYSTEM = "compat"
```

Change the line to:

```
SYSTEM = "KRB5LDAP OR compat"
```

We also need to add the following line to the default stanza in /etc/security/user:

```
registry = KRB5LDAP
```

Both changes can be achieved by running the following command.

Example 5-77 Change default logon using chsec command

```
#chsec -f /etc/security/user -s default -a registry=KRB5LDAP
#
#chsec -f /etc/security/user -s default -a "SYSTEM=\"KRB5LDAP OR compat\""
#
```

10. Edit the /usr/lib/security/methods.cfg file and add the following lines.

Example 5-78 Add KRB5LDAP stanza to file /usr/lib/security/methods.cfg

```
LDAP:
    program = /usr/lib/security/LDAP
    program_64 = /usr/lib/security/LDAP64
KRB5:
    program = /usr/lib/security/KRB5

KRB5LDAP:
    options = db=LDAP,auth=KRB5
```

11. Communication to the IBM Directory Server from the client must be enabled:

```
#mksecldap -c -h nfs407.itsc.austin.ibm.com -a cn=admin -p succ3ss
#
```

With these settings in place, any principal that is defined in the realm REALM1.IBM.COM can log on to nfs402.itsc.austin.ibm.com. Before trying to log on to the system, we have to verify that the local LDAP security client daemon is running. Otherwise, the logon attempt would fail as the AIX login process will not be able to contact the LDAP server.

Example 5-79 Verification that secdapclntd is running

```
# /usr/sbin/ls-secdapclntd
ldapservers=nfs407.itsc.austin.ibm.com
ldapport=389
ldapversion=3
userbasedn=ou=People,cn=aixdata
groupbasedn=ou=Groups,cn=aixdata
idbasedn=cn=aixid,ou=System,cn=aixdata
usercachesize=1000
```



```

usercacheused=0
groupcachesize=100
groupcacheused=0
cachetimeout=300
heartbeatT=300
numberofthread=10
alwaysmaster=no
authtype=UNIX_AUTH
searchmode=ALL
defaultentrylocation=LDAP
ldaptimeout=60
userobjectclass=account,posixaccount,shadowaccount,aixauxaccount
groupobjectclass=posixgroup,aixauxgroup
#

```

The local LDAP security client daemon is running and we can try the integrated logon for user sally.

Example 5-80 Sample integrated logon output

```

# telnet loopback
AIX Version 5
(C) Copyrights by IBM and by others 1982, 2004.
login: sally
sally's Password:
*****
*                                                                 *
*                                                                 *
* Welcome to AIX Version 5.3!                                     *
*                                                                 *
*                                                                 *
* Please see the README file in /usr/lpp/bos for information pertinent to *
* this release of the AIX Operating System.                       *
*                                                                 *
*                                                                 *
*****
$
$ id
uid=6023(sally) gid=1(staff) groups=1400(eng)
$
$ lsuser -a registry sally
sally registry=KRB5LDAP
$
$ klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_sally@REALM1.IBM.COM_6023
Default principal: sally@REALM1.IBM.COM

```

```

Valid starting    Expires          Service principal
08/09/04 18:35:26 08/10/04 18:35:25  krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
08/09/04 18:35:34 08/09/04 21:35:34  kadmin/admin@REALM1.IBM.COM
$

```

The user sally can log on to any machine in the Kerberos realm REALM1.IBM.COM without being defined as a local user if the client has been set up with the previously mentioned authentication grammar. The user's identification will be stored in LDAP RFC2307.

Also, because user sally has a Kerberos ticket granted automatically during the AIX logon process, she can have access to all NFS V4 mounted file systems with `sec=krb5` option if ACLs are set for access.

5.10 Integrating NFS V4 with a Linux client

NFS V4 is shipped with the Linux kernel Version 2.6.5. For the purposes of our example, we use Fedora Core 2 with kernel Version 2.6.5.1.358. This is the current downloadable version of Fedora Core at the time of writing this book.

Note: As we write this book, the unmodified Fedora Core 2 Linux does *not* contain a complete and working RPCSEC_GSS (Kerberos 5, LIPKEY, SPKM-3) implementation. This will probably happen in Fedora Core 3 Linux. Therefore we are unable to test this functionality in this book. You are welcome to attempt to patch the 2.6.5.x kernel to a later kernel manually. This should give you the missing functionality.

Linux was installed from the install media with the NFS services chosen at install time. In this section, we look at setting up some basic NFS V4 scenarios using the AUTH_SYS security mechanism.

The following definitions will be used in the example:

NFS server	hostname = nfs403, OS = AIX 5.3
NFS client	hostname = nfs408, OS: Fedora Core 2 with v2.6.5.1.358 kernel.
NFS Domain Name	itsc.austin.ibm.com
Realm Name	REALM1.IBM.COM

Figure 5-3 on page 177 gives a better view of the setup.

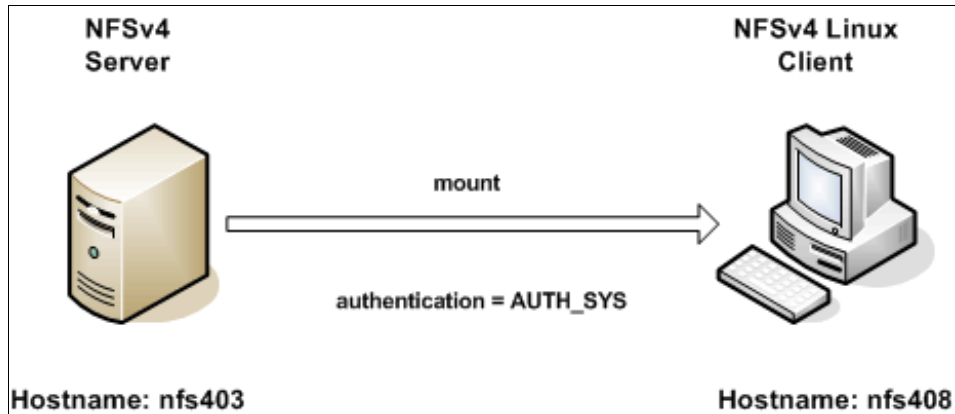


Figure 5-3 Sample environment

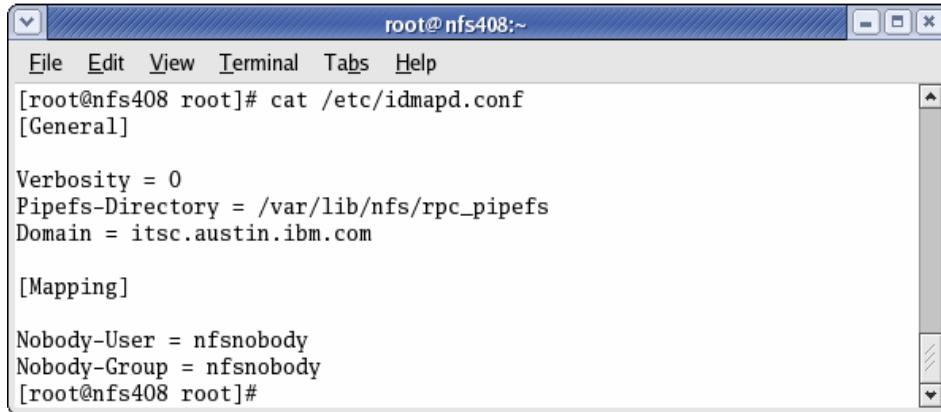
There are many configuration files that make up a Linux NFS client and you should make sure that they exist before you proceed:

- ▶ `/etc/fstab`
- ▶ `/etc/auto.master`
- ▶ `/etc/idmapd.conf`
- ▶ `/etc/gssapi_mech.conf`
- ▶ `/etc/init.d/portmap`
- ▶ `/etc/init.d/rpcidmapd`
- ▶ `/etc/init.d/rpcgssd` (required on the client when `RPCSEC_GSS` is used)

5.10.1 NFS server and client setup

The ID mapper daemon is required on the client. It maps NFS V4 `username@domain` user strings back and forth into numerous UIDs and GIDs. The client and server domains must match. This is set in the `/etc/idmapd.conf` file.

1. The domain on our server is `itsc.austin.ibm.com`, so we modify `/etc/idmapd.conf` to reflect this. Figure 5-4 on page 178 shows this.



```
root@nfs408:~
File Edit View Terminal Tabs Help
[root@nfs408 root]# cat /etc/idmapd.conf
[General]

Verbosity = 0
Pipefs-Directory = /var/lib/nfs/rpc_pipefs
Domain = itsc.austin.ibm.com

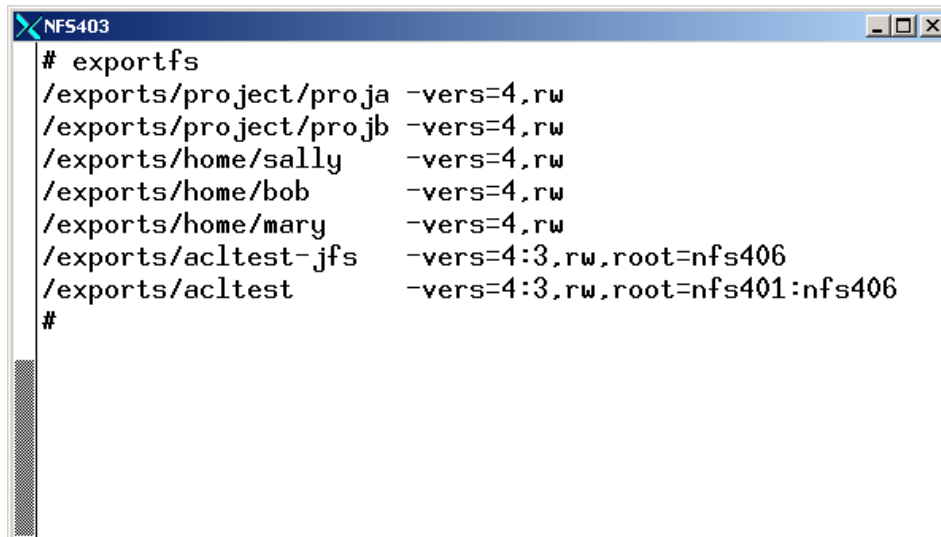
[Mapping]

Nobody-User = nfsnobody
Nobody-Group = nfsnobody
[root@nfs408 root]#
```

Figure 5-4 /etc/idmapd.conf showing the nfsdomain

2. Our NFS server has the directories exported as shown in Figure 5-5. You can edit /etc/exports using either a text editor or **smitty mknfsexp** to create your exports list.

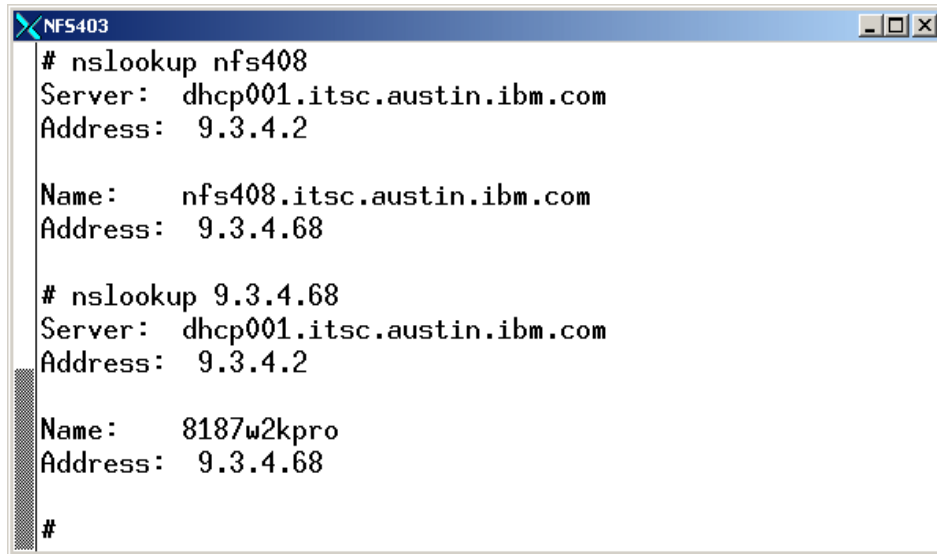
Tip: Manually editing /etc/exports is easier if you need to export more than one file system. Run **exportfs -va** after applying your changes to the exports file. With **smitty**, add one file system at a time.



```
NFS403
# exportfs
/exports/project/proja -vers=4,rw
/exports/project/projb -vers=4,rw
/exports/home/sally -vers=4,rw
/exports/home/bob -vers=4,rw
/exports/home/mary -vers=4,rw
/exports/acltest-jfs -vers=4:3,rw,root=nfs406
/exports/acltest -vers=4:3,rw,root=nfs401:nfs406
#
```

Figure 5-5 Samples exported directories on the NFS server

Name resolution between the server and client must work. This is an NFS V4 requirement.



```
NFS403
# nslookup nfs408
Server:  dhcp001.itsc.austin.ibm.com
Address:  9.3.4.2

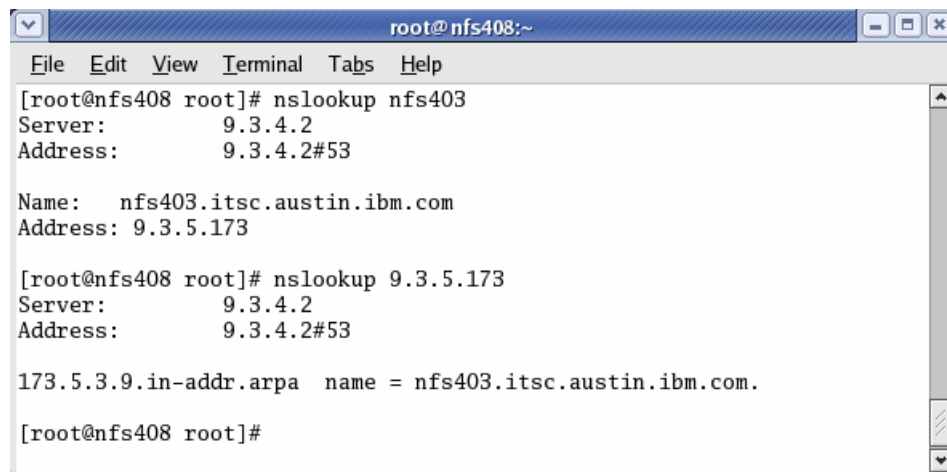
Name:    nfs408.itsc.austin.ibm.com
Address:  9.3.4.68

# nslookup 9.3.4.68
Server:  dhcp001.itsc.austin.ibm.com
Address:  9.3.4.2

Name:    8187w2kpro
Address:  9.3.4.68

#
```

Figure 5-6 NFS server name resolution of NFS client



```
root@nfs408:~
File Edit View Terminal Tabs Help
[root@nfs408 root]# nslookup nfs403
Server:          9.3.4.2
Address:         9.3.4.2#53

Name:   nfs403.itsc.austin.ibm.com
Address: 9.3.5.173

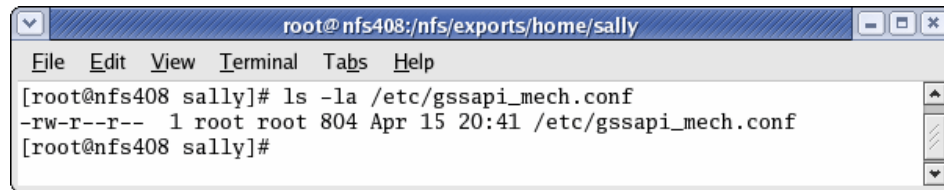
[root@nfs408 root]# nslookup 9.3.5.173
Server:          9.3.4.2
Address:         9.3.4.2#53

173.5.3.9.in-addr.arpa  name = nfs403.itsc.austin.ibm.com.

[root@nfs408 root]#
```

Figure 5-7 NFS client name resolution of NFS server

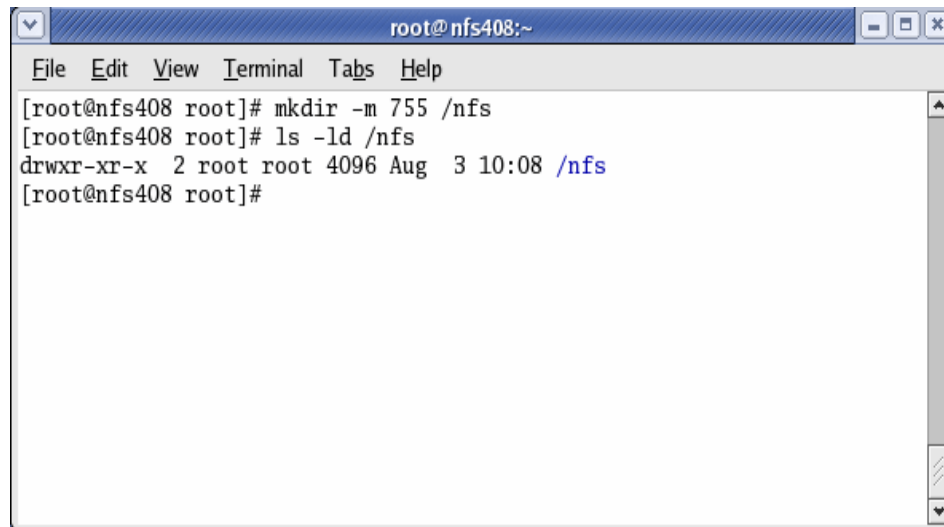
3. Make sure that the `/etc/gssapi_mech.conf` file exists. It should have been installed by default.



```
root@nfs408:/nfs/exports/home/sally
File Edit View Terminal Tabs Help
[root@nfs408 sally]# ls -la /etc/gssapi_mech.conf
-rw-r--r-- 1 root root 804 Apr 15 20:41 /etc/gssapi_mech.conf
[root@nfs408 sally]#
```

Figure 5-8 `/etc/gssapi_mech.conf` on NFS client

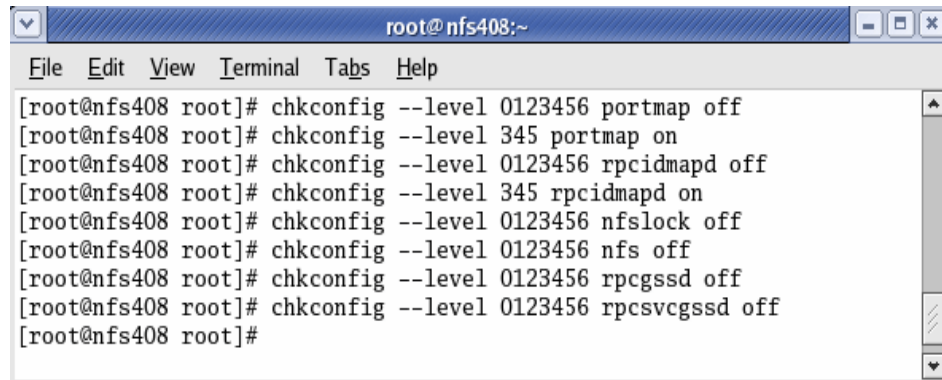
4. We use `/nfs` as our NFS mount point.



```
root@nfs408:~
File Edit View Terminal Tabs Help
[root@nfs408 root]# mkdir -m 755 /nfs
[root@nfs408 root]# ls -ld /nfs
drwxr-xr-x 2 root root 4096 Aug  3 10:08 /nfs
[root@nfs408 root]#
```

Figure 5-9 NFS mount point on the NFS client

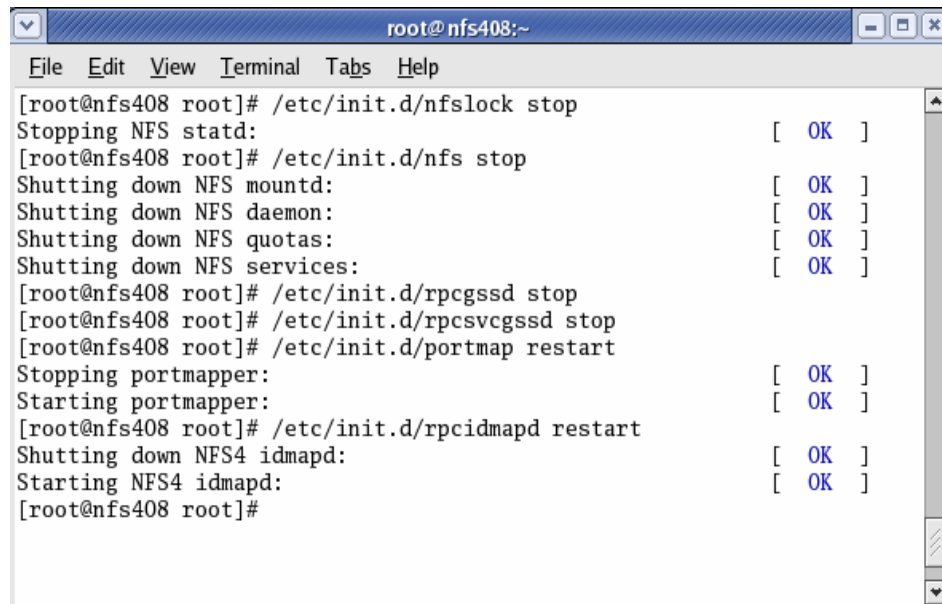
5. Ensure that all NFS-related scripts start and stop automatically. The **chkconfig** utility can be used for this.



```
root@nfs408:~
File Edit View Terminal Tabs Help
[root@nfs408 root]# chkconfig --level 0123456 portmap off
[root@nfs408 root]# chkconfig --level 345 portmap on
[root@nfs408 root]# chkconfig --level 0123456 rpcidmapd off
[root@nfs408 root]# chkconfig --level 345 rpcidmapd on
[root@nfs408 root]# chkconfig --level 0123456 nfslock off
[root@nfs408 root]# chkconfig --level 0123456 nfs off
[root@nfs408 root]# chkconfig --level 0123456 rpcgssd off
[root@nfs408 root]# chkconfig --level 0123456 rpcsvcgssd off
[root@nfs408 root]#
```

Figure 5-10 *chkconfig* to make sure all services start and stop automatically

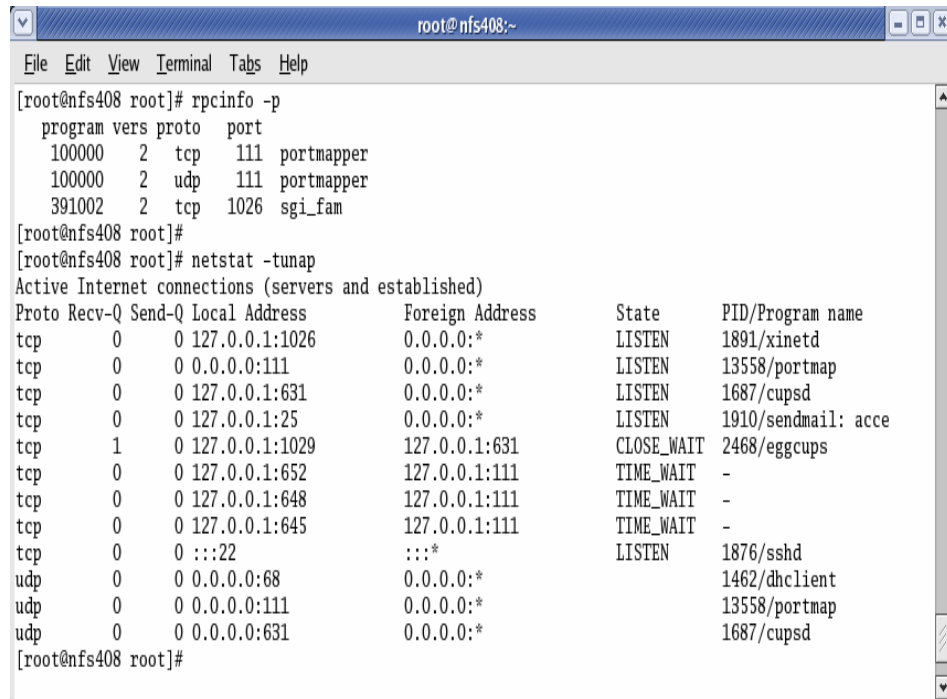
6. Ensure that all of the right daemons are restarted or stopped on the NFS client.



```
root@nfs408:~
File Edit View Terminal Tabs Help
[root@nfs408 root]# /etc/init.d/nfslock stop
Stopping NFS statd: [ OK ]
[root@nfs408 root]# /etc/init.d/nfs stop
Shutting down NFS mountd: [ OK ]
Shutting down NFS daemon: [ OK ]
Shutting down NFS quotas: [ OK ]
Shutting down NFS services: [ OK ]
[root@nfs408 root]# /etc/init.d/rpcgssd stop
[root@nfs408 root]# /etc/init.d/rpcsvcgssd stop
[root@nfs408 root]# /etc/init.d/portmap restart
Stopping portmapper: [ OK ]
Starting portmapper: [ OK ]
[root@nfs408 root]# /etc/init.d/rpcidmapd restart
Shutting down NFS4 idmapd: [ OK ]
Starting NFS4 idmapd: [ OK ]
[root@nfs408 root]#
```

Figure 5-11 *Confirm the correct daemons are restarted or stopped on the NFS client*

Figure 5-12 tells what NFS-related daemons are running on the client, and what UDP and TCP ports they are listening on.



```
root@nfs408:~  
File Edit View Terminal Tabs Help  
[root@nfs408 root]# rpcinfo -p  
program vers proto port  
100000 2 tcp 111 portmapper  
100000 2 udp 111 portmapper  
391002 2 tcp 1026 sgi_fam  
[root@nfs408 root]#  
[root@nfs408 root]# netstat -tunap  
Active Internet connections (servers and established)  
Proto Recv-Q Send-Q Local Address Foreign Address State PID/Program name  
tcp 0 0 127.0.0.1:1026 0.0.0.0:* LISTEN 1891/xinetd  
tcp 0 0 0.0.0.0:111 0.0.0.0:* LISTEN 13558/portmap  
tcp 0 0 127.0.0.1:631 0.0.0.0:* LISTEN 1687/cupsd  
tcp 0 0 127.0.0.1:25 0.0.0.0:* LISTEN 1910/sendmail: acce  
tcp 1 0 127.0.0.1:1029 127.0.0.1:631 CLOSE_WAIT 2468/eggcup  
tcp 0 0 127.0.0.1:652 127.0.0.1:111 TIME_WAIT -  
tcp 0 0 127.0.0.1:648 127.0.0.1:111 TIME_WAIT -  
tcp 0 0 127.0.0.1:645 127.0.0.1:111 TIME_WAIT -  
tcp 0 0 :::22 :::* LISTEN 1876/sshd  
udp 0 0 0.0.0.0:68 0.0.0.0:* 1462/dhclient  
udp 0 0 0.0.0.0:111 0.0.0.0:* 13558/portmap  
udp 0 0 0.0.0.0:631 0.0.0.0:* 1687/cupsd  
[root@nfs408 root]#
```

Figure 5-12 Checking NFS daemons and what ports they are listening on

5.10.2 Read-only NFS V4 mount

We first look at carrying out a read-only NFS V4 mount.

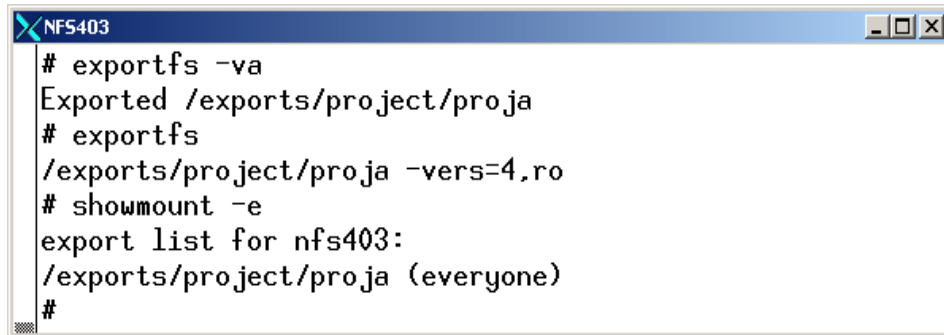
1. On the server, modify `/etc/exports` and add a file system to export in read-only mode.



```
NFS403  
# cat /etc/exports  
/exports/project/proja -vers=4,ro  
#
```

Figure 5-13 `/etc/exports` on the AIX NFS server

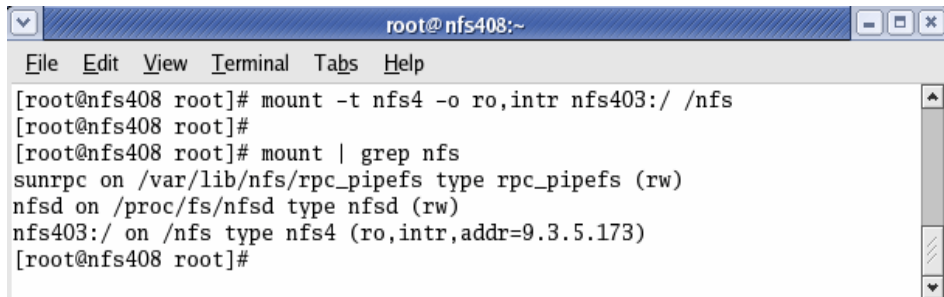
2. Export the directory specified on /etc/exports.



```
NFS403
# exportfs -va
Exported /exports/project/proja
# exportfs
/exportfs/project/proja -vers=4,ro
# showmount -e
export list for nfs403:
/exportfs/project/proja (everyone)
#
```

Figure 5-14 Exporting the directory on the NFS server

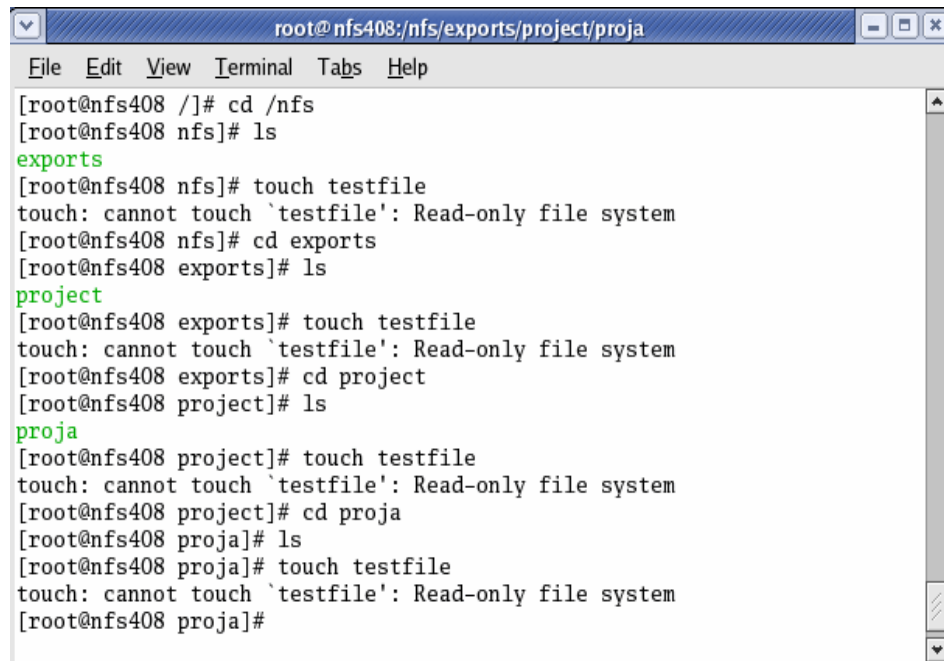
3. Mount the exported directory on the NFS client. We first mount it manually and then show how the /etc/fstab file can be modified to allow mounts via directory name.



```
root@nfs408:~
File Edit View Terminal Tabs Help
[root@nfs408 root]# mount -t nfs4 -o ro,intr nfs403:/ /nfs
[root@nfs408 root]#
[root@nfs408 root]# mount | grep nfs
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
nfsd on /proc/fs/nfsd type nfsd (rw)
nfs403:/ on /nfs type nfs4 (ro,intr,addr=9.3.5.173)
[root@nfs408 root]#
```

Figure 5-15 Manually mounting an NFS V4 file system in read-only mode

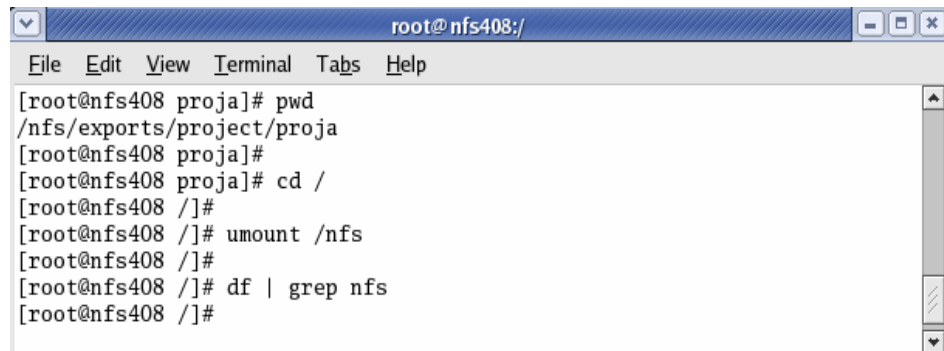
4. Look at the file system that we have just mounted. If you try to write to the directories, you will get an error. (Remember, we mounted the directory in read-only mode.)



```
root@nfs408:/nfs/exports/project/proja
File Edit View Terminal Tabs Help
[root@nfs408 /]# cd /nfs
[root@nfs408 nfs]# ls
exports
[root@nfs408 nfs]# touch testfile
touch: cannot touch `testfile': Read-only file system
[root@nfs408 nfs]# cd exports
[root@nfs408 exports]# ls
project
[root@nfs408 exports]# touch testfile
touch: cannot touch `testfile': Read-only file system
[root@nfs408 exports]# cd project
[root@nfs408 project]# ls
proja
[root@nfs408 project]# touch testfile
touch: cannot touch `testfile': Read-only file system
[root@nfs408 project]# cd proja
[root@nfs408 proja]# ls
[root@nfs408 proja]# touch testfile
touch: cannot touch `testfile': Read-only file system
[root@nfs408 proja]#
```

Figure 5-16 Checking to see whether read-only mode works

5. Unmount the NFS file system that was mounted in the previous steps.



```
root@nfs408:/
File Edit View Terminal Tabs Help
[root@nfs408 proja]# pwd
/nfs/exports/project/proja
[root@nfs408 proja]#
[root@nfs408 proja]# cd /
[root@nfs408 /]#
[root@nfs408 /]# umount /nfs
[root@nfs408 /]#
[root@nfs408 /]# df | grep nfs
[root@nfs408 /]#
```

Figure 5-17 Unmounting the NFS V4 file system

6. Append the following line to `/etc/fstab`:

```
nfs403:/ /nfs nfs4 ro,hard,intr,proto=tcp,port=2049,noauto 0 0
```

7. Mount the NFS V4 file system using the information in `/etc/fstab`.



```
root@nfs408:/
File Edit View Terminal Tabs Help
[root@nfs408 /]# mount -v /nfs
nfs403:/ on /nfs type nfs4 (ro,hard,intr,proto=tcp,port=2049,addr=9.3.5.173)
[root@nfs408 /]#
```

Figure 5-18 NFS mount using entry in `/etc/fstab`

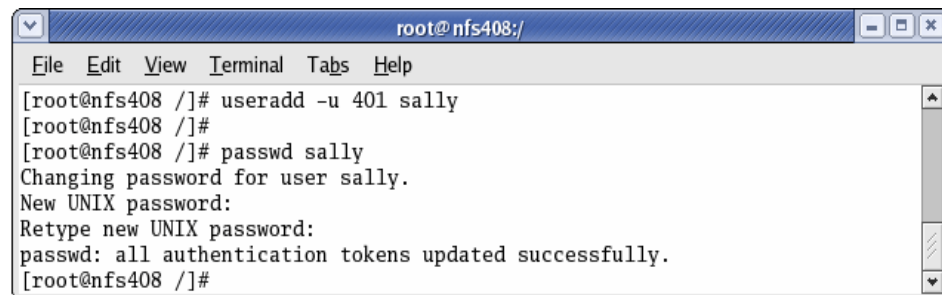
As you can see, Linux can mount AIX NFS V4 exports without any problems. We now unmount the file system as done in step 5 on page 184 to continue with the next section.

5.10.3 Read/write NFS V4 mounts on Linux

We can now look at the most common type of NFS mount: the read/write mount. For this example, we add another export to our server. We create the directory with the sticky bit set. This enables any remote user to read and write to the directory. If the root user on the NFS V4 client writes to the directory, the ownership will be changed to `nfsnobody`. Root squashing is turned on by default (which means that the root user on the NFS client does not have root privileges on the NFS server).

The NFS server already has a user called `sally`, with UID 401, created on it. We create the same user and UID on our NFS V4 client. For this example, we are not using LDAP. If you use LDAP, you can use users created there for testing.

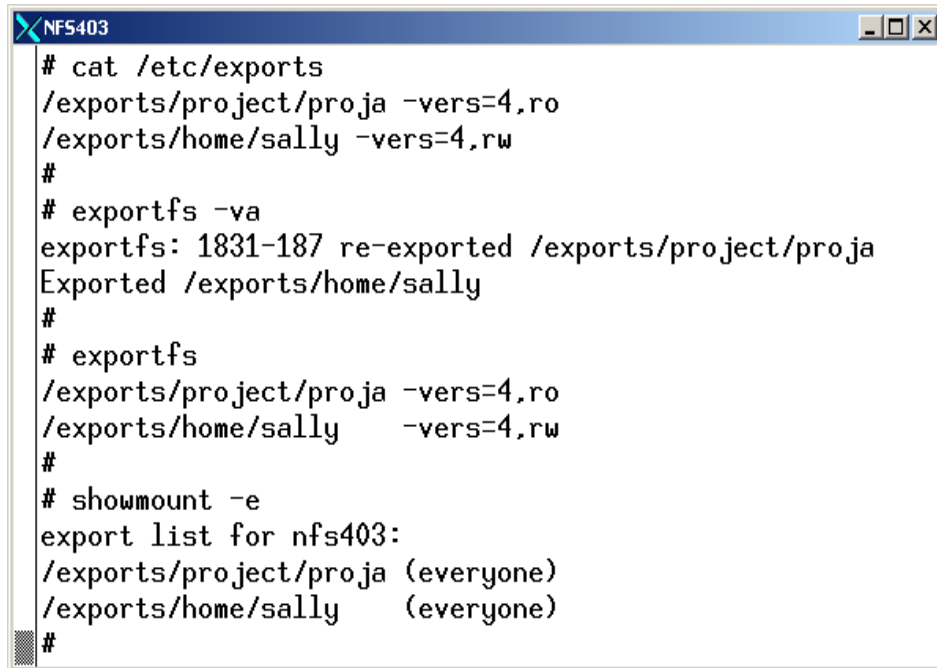
1. Create user `sally` on the NFS V4 client.



```
root@nfs408:/
File Edit View Terminal Tabs Help
[root@nfs408 /]# useradd -u 401 sally
[root@nfs408 /]#
[root@nfs408 /]# passwd sally
Changing password for user sally.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
[root@nfs408 /]#
```

Figure 5-19 Adding a user on the Linux client to match the NFS server

2. We modify `/etc/exports` on the server to export user sally's home directory and export the file system.



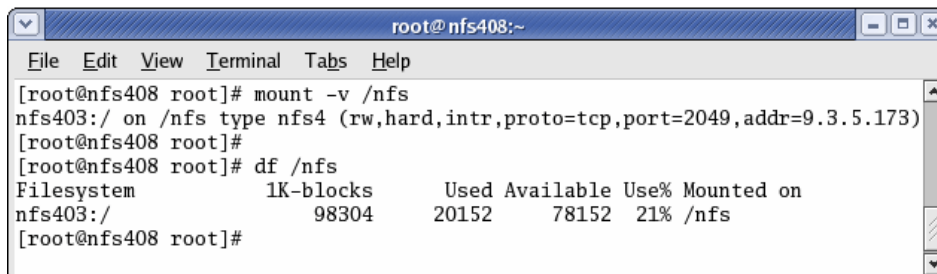
```
NFS403
# cat /etc/exports
/exports/project/proja -vers=4,ro
/exports/home/sally -vers=4,rw
#
# exportfs -va
exportfs: 1831-187 re-exported /exports/project/proja
Exported /exports/home/sally
#
# exportfs
/exports/project/proja -vers=4,ro
/exports/home/sally -vers=4,rw
#
# showmount -e
export list for nfs403:
/exports/project/proja (everyone)
/exports/home/sally (everyone)
#
```

Figure 5-20 Adding and exporting user sally's home directory on the NFS V4 server

3. Next, we modify `/etc/fstab` on the client so that it looks like this:

```
nfs403:/ /nfs nfs4 rw,hard,intr,proto=tcp,port=2049,noauto 0 0
```

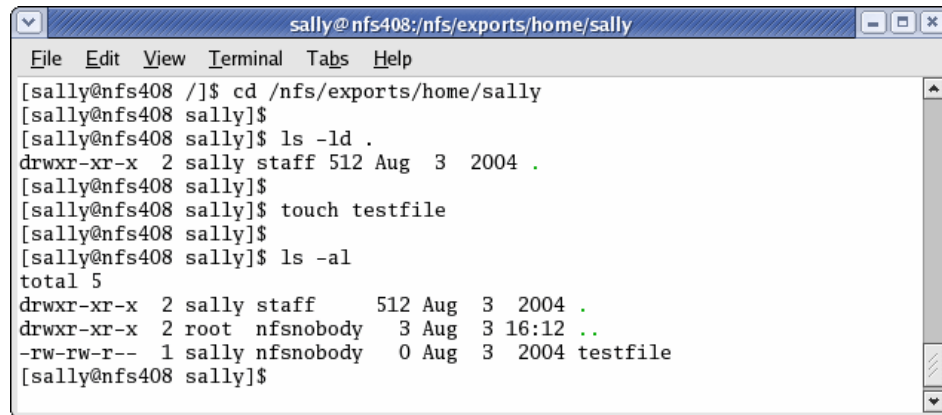
4. We then mount the file system.



```
root@nfs408:~
File Edit View Terminal Tabs Help
[root@nfs408 root]# mount -v /nfs
nfs403:/ on /nfs type nfs4 (rw,hard,intr,proto=tcp,port=2049,addr=9.3.5.173)
[root@nfs408 root]#
[root@nfs408 root]# df /nfs
Filesystem      1K-blocks      Used Available Use% Mounted on
nfs403:/          98304        20152    78152   21% /nfs
[root@nfs408 root]#
```

Figure 5-21 Mounting file system on NFS client

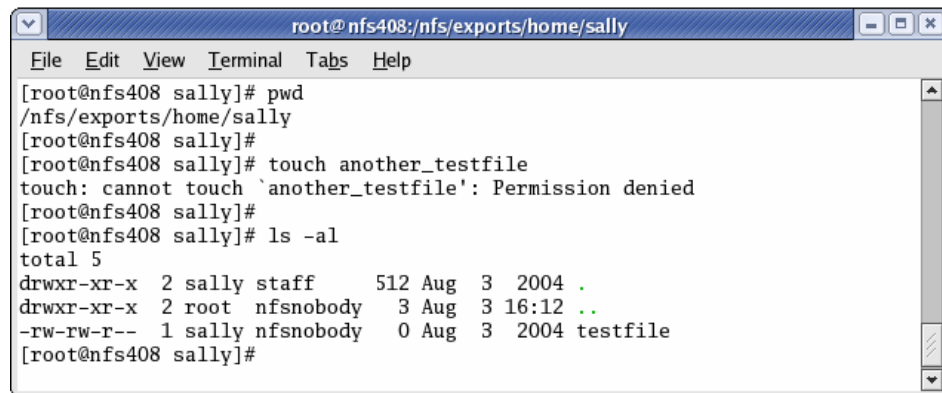
5. User sally should now be able to write to /nfs/exports/home/sally.



```
sally@ nfs408:/nfs/exports/home/sally
File Edit View Terminal Tabs Help
[sally@nfs408 ~]$ cd /nfs/exports/home/sally
[sally@nfs408 sally]$
[sally@nfs408 sally]$ ls -ld .
drwxr-xr-x 2 sally staff 512 Aug 3 2004 .
[sally@nfs408 sally]$
[sally@nfs408 sally]$ touch testfile
[sally@nfs408 sally]$
[sally@nfs408 sally]$ ls -al
total 5
drwxr-xr-x 2 sally staff 512 Aug 3 2004 .
drwxr-xr-x 2 root nfsnobody 3 Aug 3 16:12 ..
-rw-rw-r-- 1 sally nfsnobody 0 Aug 3 2004 testfile
[sally@nfs408 sally]$
```

Figure 5-22 Testing write as user sally

6. We can try writing to the directory owned by sally as root.



```
root@ nfs408:/nfs/exports/home/sally
File Edit View Terminal Tabs Help
[root@nfs408 sally]# pwd
/nfs/exports/home/sally
[root@nfs408 sally]#
[root@nfs408 sally]# touch another_testfile
touch: cannot touch `another_testfile': Permission denied
[root@nfs408 sally]#
[root@nfs408 sally]# ls -al
total 5
drwxr-xr-x 2 sally staff 512 Aug 3 2004 .
drwxr-xr-x 2 root nfsnobody 3 Aug 3 16:12 ..
-rw-rw-r-- 1 sally nfsnobody 0 Aug 3 2004 testfile
[root@nfs408 sally]#
```

Figure 5-23 Test on user sally's home directory as the root user

5.10.4 Pseudo-file system in NFS V4 Linux client

Remember, the pseudo-file system functionality in NFS V4 enables the server to present a single, seamless view of all exported file systems to an NFS V4 client.

Preparing the AIX NFS V4 server for pseudo-FS

Important: Ensure that the NFS V4 client has no directories mounted from the NFS V4 server before proceeding. Use the **umount** command to unmount it.

To set the NFS root on the server:

1. Unexport all file systems.

Example 5-81 Unexporting all file systems on the NFS server

```
# exportfs -vu
/exports/project/proja -vers=4,ro
/exports/home/sally -vers=4,rw
/exports/acctest -vers=4:3,rw,root=nfs401:nfs406
/exports/acctest-jfs -vers=4:3,rw,root=nfs406
#
```

2. Stop NFS.

Example 5-82 Shutting down the NFS daemons cleanly

```
# /etc/nfs.clean
nfs_clean: Stopping NFS/NIS Daemons
0513-044 The nfsd Subsystem was requested to stop.
0513-044 The biod Subsystem was requested to stop.
0513-006 The rpc.lockd was requested to stop
0513-044 The rpc.statd Subsystem was requested to stop.
0513-044 The rpc.mountd Subsystem was requested to stop.
0513-004 The Subsystem or Group, ybind, is currently inoperative.
#
```

3. Stop the nfsrgyd subsystem.

Example 5-83 Stopping the registry daemons

```
# stopsrc -s nfsrgyd
0513-044 The nfsrgyd Subsystem was requested to stop.
#
```

4. Change the NFS root to /exports and confirm the change.

Example 5-84 Setting the root node on the NFS server

```
# /usr/sbin/chnfs -r '/exports' '-B'
#
# nfsd -getnodes
#root:public
#
```

5. Restart the NFS daemons.

Example 5-85 Restarting the NFS daemons

```
# /etc/rc.nfs
Starting NFS services:
0513-059 The biod Subsystem has been started. Subsystem PID is 200780.
0513-059 The nfsrgyd Subsystem has been started. Subsystem PID is 307354.
0513-059 The nfsd Subsystem has been started. Subsystem PID is 315644.
0513-059 The rpc.mountd Subsystem has been started. Subsystem PID is 319678.
0513-059 The rpc.statd Subsystem has been started. Subsystem PID is 278604.
0513-029 The rpc.lockd Subsystem has been started. Subsystem PID 266398
Completed NFS services.
#
```

6. Export all of the file systems.

Example 5-86 Exporting all file systems on the NFS server

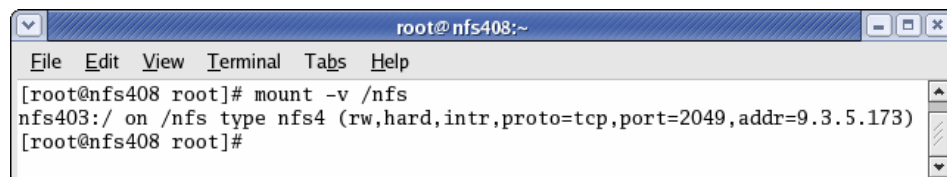
```
# exportfs -va
exportfs: 1831-187 re-exported /exports/project/proja
exportfs: 1831-187 re-exported /exports/home/sally
#
```

We are now ready to move on to the client.

Mounting the pseudo-file system on the NFS V4 Linux client

The process of mounting the pseudo-file system on the client is the same as in the previous examples. The entry in `/etc/fstab` will remain the same.

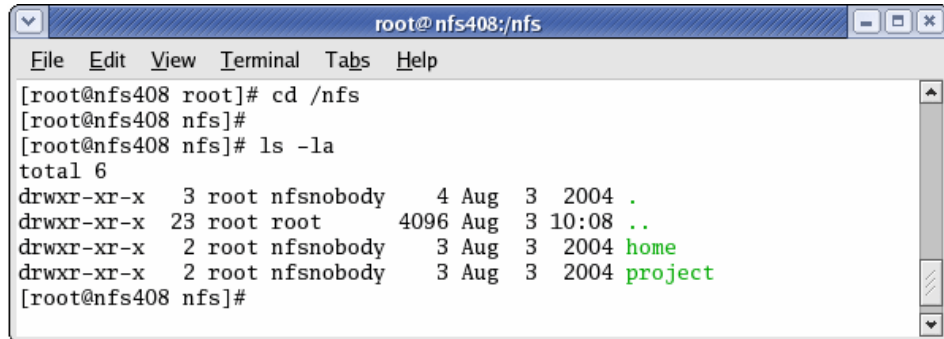
1. Mount the pseudo-file system on the client.

A terminal window titled "root@nfs408:~" with a menu bar containing "File", "Edit", "View", "Terminal", "Tabs", and "Help". The terminal output shows the command "mount -v /nfs" being executed, resulting in the message "nfs403:/ on /nfs type nfs4 (rw,hard,intr,proto=tcp,port=2049,addr=9.3.5.173)". The prompt returns to "[root@nfs408 root]#".

```
root@nfs408:~
File Edit View Terminal Tabs Help
[root@nfs408 root]# mount -v /nfs
nfs403:/ on /nfs type nfs4 (rw,hard,intr,proto=tcp,port=2049,addr=9.3.5.173)
[root@nfs408 root]#
```

Figure 5-24 Mounting the pseudo-file system on the NFS V4 client

2. We can now traverse the mount point to see what we have underneath.

A terminal window titled 'root@nfs408:/nfs' showing the following commands and output:

```
[root@nfs408 root]# cd /nfs
[root@nfs408 nfs]#
[root@nfs408 nfs]# ls -la
total 6
drwxr-xr-x  3 root nfsnobody   4 Aug  3  2004 .
drwxr-xr-x 23 root root       4096 Aug  3 10:08 ..
drwxr-xr-x  2 root nfsnobody   3 Aug  3  2004 home
drwxr-xr-x  2 root nfsnobody   3 Aug  3  2004 project
[root@nfs408 nfs]#
```

Figure 5-25 Client view of the pseudo-root

The output shows that, although the server is exporting `/exports/home/sally` and `/export/project/proja`, the client only sees these as `/nfs/home/sally` and `/nfs/project/proja`. This is as a direct result of setting the `nfs root` on the server to `/exports`.

5.11 Windows KDC and NFS V4 AIX 5.3

In this section, we set up a basic Active Directory and using the built-in KDC on the AIX 5.3 systems with the NFS V4 RPCSEC_GSS security mechanism.

We installed Windows Server 2003 Standard Edition, which includes a KDC server built in with the Active Directory. The Windows server was installed from the install media with no additional options. We also installed the Windows Server 2003 Resource Kit and the Windows Server 2003 Support Tools.

The following definitions will be used in the example:

NFS server	hostname = nfs405, OS = AIX 5.3
NFS client	hostname = nfs403, OS: AIX 5.3
KDC server	hostname = nfs409.kdc.austin.ibm.com, OS: Windows 2003 Server Standard Edition
NFS Domain Name	itsc.austin.ibm.com
Realm Name	REALM1.IBM.COM

Figure 5-26 on page 191 gives a better view of the setup.

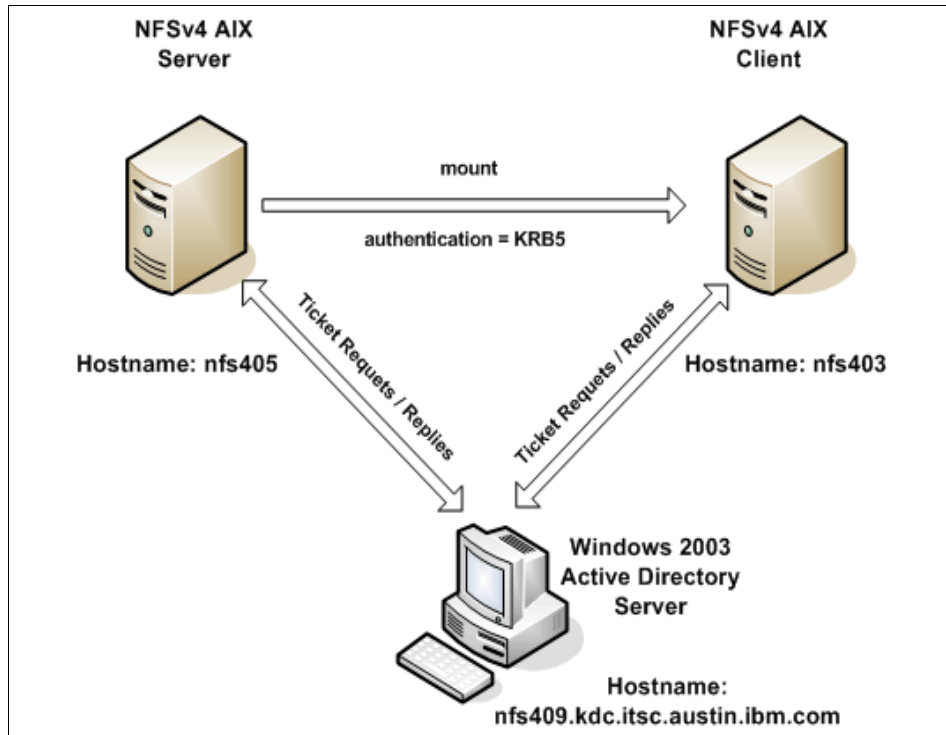


Figure 5-26 Sample AIX environment with Windows 2003 Active Directory

1. We installed Active Directory through the Windows Server configuration wizard on the Windows Server 2003 Standard Server.

Note: The Kerberos realm name is derived from the Active Directory domain name. The realm name is the domain name converted to uppercase (For the example.com domain name, the realm name is EXAMPLE.COM.) In our case this is KDC.ITSC.AUSTIN.IBM.COM on server nfs409.kdc.austin.ibm.com.

The following figures show the chosen setup.

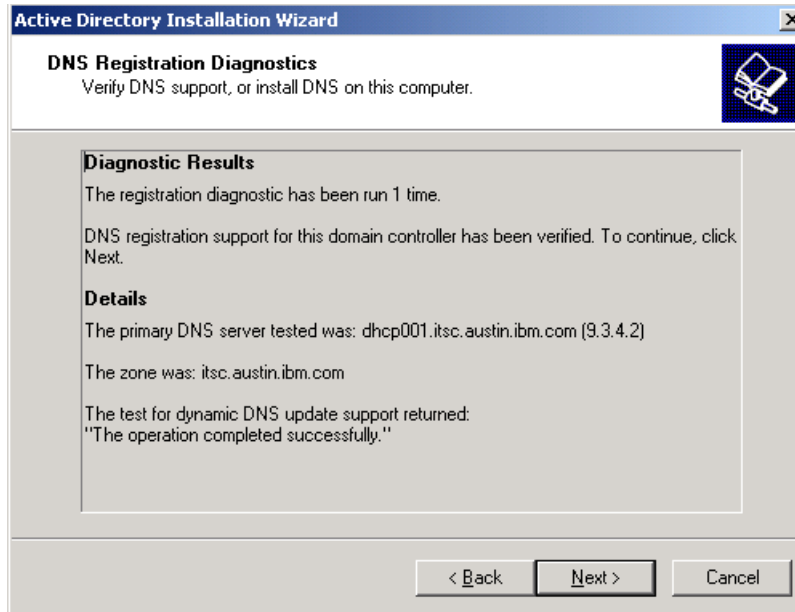


Figure 5-27 DNS Active Directory registration in Windows 2003 Server

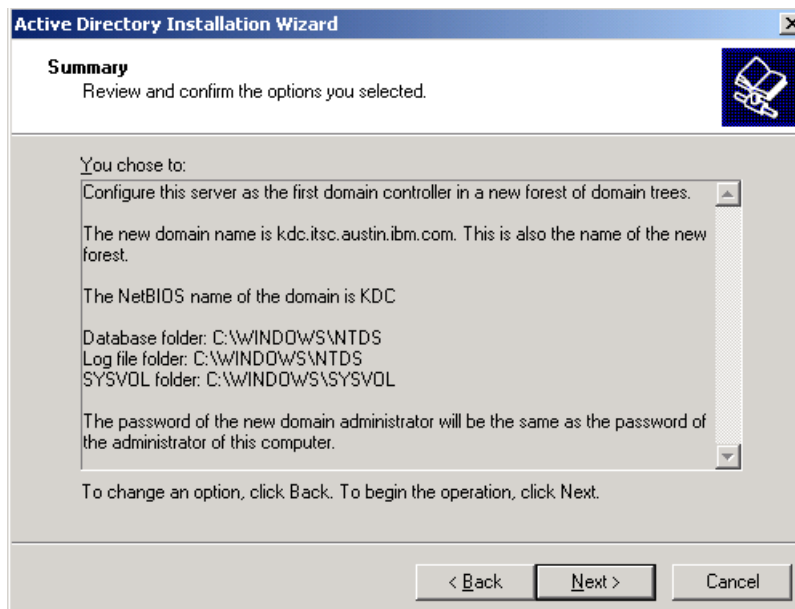


Figure 5-28 Active Directory Installation Wizard summary

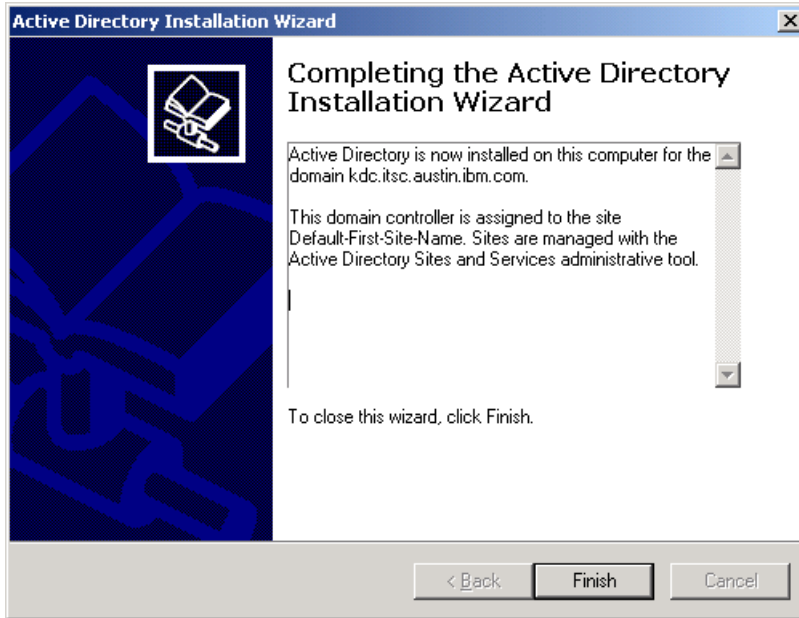


Figure 5-29 Active Directory Installation Wizard confirmation

2. Download and install the public Windows MIT Kerberos V5 utilities, which are available at:
<http://web.mit.edu/kerberos/www/>
3. We created a sample user called sa11y within Windows Active directory. This is achieved by carrying out the following steps:
 - a. Click **Start**.
 - b. Click **All Programs**.
 - c. Click **Administrative Tools**.
 - d. Click **Active Directory Users and Computers**.

4. We used the same method to create the user for the NFS V4 server (nfs405) and client (nfs403). See the following figures for details.

New Object - User

Create in: kdc.itsc.austin.ibm.com/Users

First name: sally Initials:

Last name: Test User for NFSv4

Full name: sally

User logon name: sally @kdc.itsc.austin.ibm.com

User logon name (pre-Windows 2000): KDC\sally

< Back Next > Cancel

Figure 5-30 User creation main panel

New Object - User

Create in: kdc.itsc.austin.ibm.com/Users

Password:

Confirm password:

User must change password at next logon

User cannot change password

Password never expires

Account is disabled

< Back Next > Cancel

Figure 5-31 User creation password panel

- Next we use the Leash Kerberos Ticket Manager (`leash32.exe`) delivered with the MIT Kerberos V5 utilities to get a ticket for user sally.

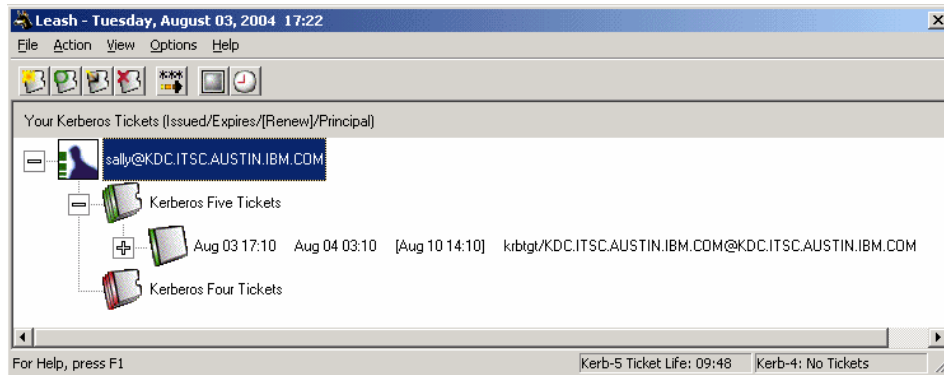


Figure 5-32 Kerberos ticket verification on Windows Server 2003

Use the following guidelines to configure the AIX NFS V4 server hosts.

Services running on UNIX or Linux systems can be configured with service instance accounts in the Active Directory. This allows full interoperability. Kerberos clients and servers on UNIX systems can authenticate using the Windows Server 2003 Kerberos server, and Windows 2000 Professional-based clients can authenticate to Kerberos services that support GSS-API.

Unlike Kerberos principal names, Windows Server 2003 account names are not multipart. Because of this, it is not possible to directly create an account of the name `nfs/UNIX_SRV@example.com`. Such a principal instance is created through the service principal name mappings.

Create a service instance account in the Active Directory:

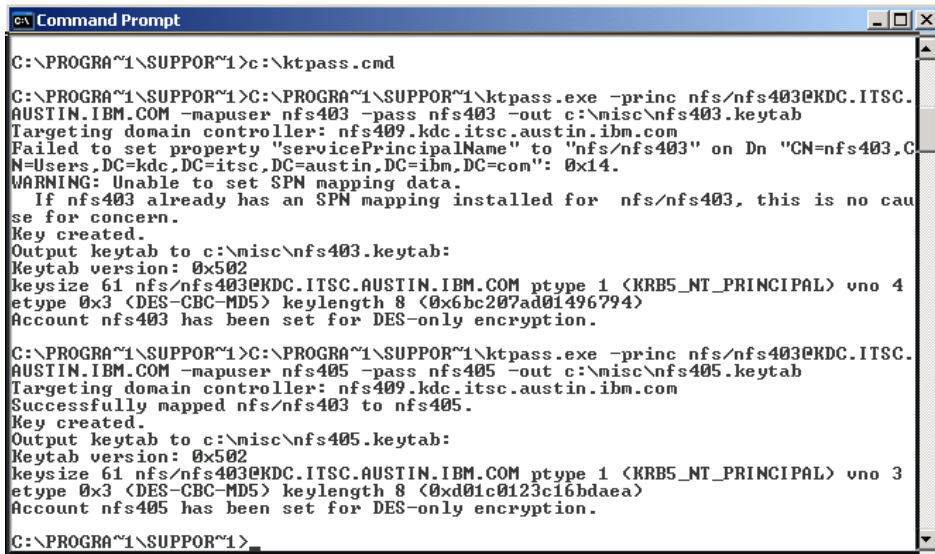
- Use the Active Directory Management tool to create a user account for the NFS V4 server; for example, create an account with the name `nfs405`.
- Use the `ktpass` tool to set up an identity mapping for the user account. Use this command (from the Windows COMMAND prompt):

```
ktpass -princ service-instance@REALM -mapuser account-name -pass
password -out UNIXmachine.keytab
```

The format of the Kerberos service-instance name is: `nfs/host@realm_name`, for example:

```
ktpass.exe -princ nfs/nfs405@KDC.ITSC.AUSTIN.IBM.COM -mapuser nfs405
-pass nfs405 -out c:\misc\nfs405.keytab
```

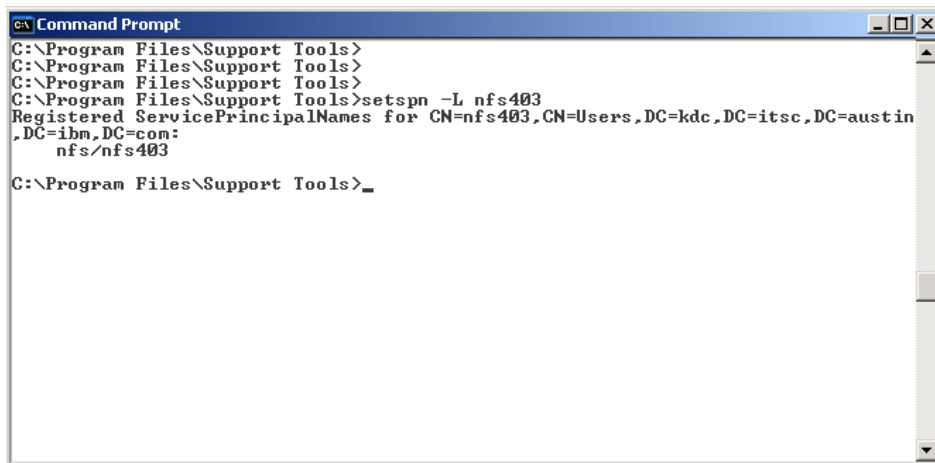
In this case, an account is created with the name `sample`, and a service principal name mapping is added for `sample/UNIX_SRV.example.com`. This is the purpose of using `ktpass` with the `-princ` and `-mapuser` switches.



```
C:\PROGRAM~1\SUPPORT~1>c:\ktpass.cmd
C:\PROGRAM~1\SUPPORT~1>C:\PROGRAM~1\SUPPORT~1\ktpass.exe -princ nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM -mapuser nfs403 -pass nfs403 -out c:\misc\nfs403.keytab
Targeting domain controller: nfs409.kdc.itsc.austin.ibm.com
Failed to set property "servicePrincipalName" to "nfs/nfs403" on Dn "CN=nfs403,CN=Users,DC=kdc,DC=itsc,DC=austin,DC=ibm,DC=com": 0x14.
WARNING: Unable to set SPN mapping data.
If nfs403 already has an SPN mapping installed for nfs/nfs403, this is no cause for concern.
Key created.
Output keytab to c:\misc\nfs403.keytab:
Keytab version: 0x502
keysize 61 nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM ptype 1 (KRB5_NT_PRINCIPAL) vno 4
etype 0x3 (DES-CBC-MD5) keylength 8 (0x6bc207ad01496794)
Account nfs403 has been set for DES-only encryption.
C:\PROGRAM~1\SUPPORT~1>C:\PROGRAM~1\SUPPORT~1\ktpass.exe -princ nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM -mapuser nfs405 -pass nfs405 -out c:\misc\nfs405.keytab
Targeting domain controller: nfs409.kdc.itsc.austin.ibm.com
Successfully mapped nfs/nfs403 to nfs405.
Key created.
Output keytab to c:\misc\nfs405.keytab:
Keytab version: 0x502
keysize 61 nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM ptype 1 (KRB5_NT_PRINCIPAL) vno 3
etype 0x3 (DES-CBC-MD5) keylength 8 (0xd01c0123c16bdaea)
Account nfs405 has been set for DES-only encryption.
C:\PROGRAM~1\SUPPORT~1>_
```

Figure 5-33 Sample output of `ktpass` tool

3. Verify the created SPN (service principal name) using the `setspn` command on Windows. The SPN must show the NFS service principal; otherwise, the systems will be not be able to authenticate with the KDC.



```
C:\Program Files\Support Tools>
C:\Program Files\Support Tools>
C:\Program Files\Support Tools>
C:\Program Files\Support Tools>setspn -L nfs403
Registered ServicePrincipalNames for CN=nfs403,CN=Users,DC=kdc,DC=itsc,DC=austin,DC=ibm,DC=com:
nfs/nfs403
C:\Program Files\Support Tools>_
```

Figure 5-34 Output of `setspn` tool

Note: On Windows, you cannot map multiple service instances to the same user account. Therefore we recommend that you do not add another instance with the same name.

4. Create the `/etc/krb5/krb5.conf` file on the AIX 5.3 server.

Attention: Be sure to have set all host names using the Fully Qualified Domain Name as used within your Active Directory; otherwise, the `gssd` daemon will not be able to authenticate with the Windows KDC. In our example, all systems used `<hostname>.kdc.austin.ibm.com`.

Example 5-87 Sample Kerberos configuration file for the Windows Active Directory

```
[libdefaults]
    default_realm = KDC.ITSC.AUSTIN.IBM.COM
    default_keytab_name = FILE:/etc/krb5/krb5.keytab
    default_tkt_enctypes = des-cbc-md5 des-cbc-crc
    default_tgs_enctypes = des-cbc-md5 des-cbc-crc
[realms]
    KDC.ITSC.AUSTIN.IBM.COM = {
        kdc = nfs409.kdc.itsc.austin.ibm.com:88
        admin_server = nfs409.kdc.itsc.austin.ibm.com:749
        default_domain = kdc.itsc.austin.ibm.com
    }
[domain_realm]
    .kdc.itsc.austin.ibm.com = KDC.ITSC.AUSTIN.IBM.COM
    nfs409.kdc.itsc.austin.ibm.com = KDC.ITSC.AUSTIN.IBM.COM
[logging]
    kdc = FILE:/var/krb5/log/krb5kdc.log
    admin_server = FILE:/var/krb5/log/kadmin.log
    default = FILE:/var/krb5/log/krb5lib.log
```

Note: The default encryption type entries, `default_txx_enctype`, are optional. However, if the Kerberos client receives an encryption type error, set the default encryption type to either `des-cbc-md5` or `des-cbc-crc`.

5. Merge the keytab file with the `/etc/krb5/krb5.keytab` file on the AIX System by copying the keytab file to the AIX system (for example, using binary FTP), then entering this command in the directory that it has been copied to:

```
/usr/krb5/sbin/ktutil
```

Then, at the **ktutil** command prompt, enter this command (where *UNIX-SRV.keytab* is the name of the keytab file you created in step 2 on page 195):

```
rkt UNIX_SRV.keytab
```

Exit **ktutil** by entering **quit** at the **ktutil** prompt.

Example 5-88 Adding the Windows keytab file to the AIX System

```
# /usr/krb5/sbin/ktutil
ktutil: rkt nfs403.keytab
ktutil: l
slot  KVNO  Principal
-----
      1      4      nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM
ktutil: wkt /etc/krb5/krb5.keytab
ktutil: quit
#
```

6. Verify that the system can get tickets.

Example 5-89 Verification of the keytab file

```
# kinit -kt /etc/krb5/krb5.keytab nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM

Valid starting   Expires           Service principal
08/04/04 17:05:20  08/05/04 03:05:20
krbtgt/KDC.ITSC.AUSTIN.IBM.COM@KDC.ITSC.AUSTIN.IBM.COM
Renew until 08/05/04 17:05:20
#
```

7. Set up your NFS V4 AIX server and client as already described.

Example 5-90 Configuration of the NFS V4 Windows KDC client

```
# chnfsdom nfs.ibm.com
#
#chnfsdom
Current local domain: nfs.ibm.com
#
#nfshostkey -p nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM -f /etc/krb5/krb5.keytab
#
#nfshostkey -l
nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM
/etc/krb5/krb5.keytab
#
```



```
#chnfsrtd -a KDC.ITSC.AUSTIN.IBM.COM nfs.ibm.com
#
#chnfsrtd
kdc.itsc.austin.ibm.com nfs.ibm.com
#
#chnfs -S -B
#
#/etc/rc.nfs
```

8. Now, we can mount the exported file system.

Example 5-91 Sample mount output

```
# mount -o vers=4,sec=krb5 nfs403:/exports/home /nfs
#
# kinit sally
Password for sally@KDC.ITSC.AUSTIN.IBM.COM:
#
# cd /nfs
#
# ls
bob          joe          lost+found  mary         sally
#
```

For further details about configuring AIX Integrated Login using Kerberos and Windows Active Directory, see the AIX 5.3 online documentation chapter about security and authenticating to AIX using Kerberos.

5.12 Setting up Kerberos cross-realm access

If you have several realms and you want to enable access throughout them, then you should activate *cross-realm access* on your server and client systems.

Note: Setting up cross-realm access is necessary to support systems belonging to multiple realms sharing access to secure data. Multiple realms can reside within a single NFS V4 domain to share user and group definitions. If the realms reside in different NFS V4 domains, then use of foreign domain identity mappings is likely required to provide access to secured data.

The following definitions will be used in the example:

NFS Domain Name	itsc.austin.ibm.com
Realm Name	REALM1.IBM.COM
KDC server	hostname = nfs407, OS = AIX 5.3
NFS client	hostname = nfs406, OS: AIX 5.3

Realm Name	REALM2.IBM.COM
KDC server	hostname = nfs403, OS = AIX 5.3
NFS server	hostname = nfs404, OS = AIX 5.3

Figure 5-35 gives a better view of the setup.

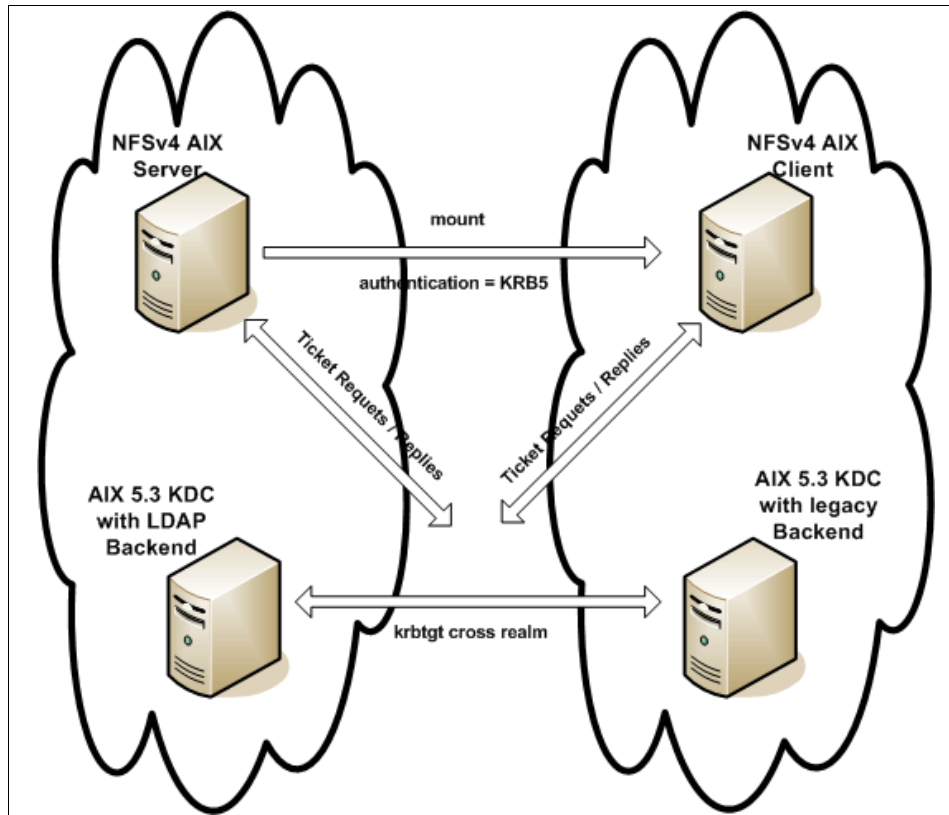


Figure 5-35 Sample cross realm layout

The goal of this sample is to show that a principal with tickets in REALM1.IBM.COM is capable of accessing NFS data exported from a server in REALM2.IBM.COM.

5.12.1 Add the krbtgt service principal to every KDC server

For a KDC in one realm to authenticate Kerberos users in a different realm, it must share a key with the KDC in the other realm. In both databases, there must be krbtgt service principals for realms. These principals should all have:

- ▶ The same passwords

- ▶ Key version numbers
- ▶ Encryption types

For example, if the administrators of REALM1.IBM.COM and REALM2.IBM.COM want to enable authentication across the realms, they would run the following commands on the KDCs in both realms:

```
kadmin: addprinc -requires_preauth krbtgt/REALM1.IBM.COM@REALM2.IBM.COM
and
kadmin: addprinc -requires_preauth krbtgt/REALM2.IBM.COM@REALM1.IBM.COM
```

Even if most principals in a realm are generally created with the `requires_preauth` flag enabled, this flag is not desirable on cross-realm authentication keys because doing so makes it impossible to disable preauthentication on a service-by-service basis. Disabling it as in the example above is recommended.

It is also very important that these principals have good passwords. MIT recommends that the TGT principal passwords be at least 26 characters of random ASCII text.

Example 5-92 shows the command and output as performed on our KDC server in realm REALM1.IBM.COM.

Example 5-92 kadmin output on nfs407 KDC server

```
kadmin: addprinc -requires_preauth krbtgt/REALM1.IBM.COM@REALM2.IBM.COM
WARNING: no policy specified for krbtgt/REALM1.IBM.COM@REALM2.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.
Enter password for principal "krbtgt/REALM1.IBM.COM@REALM2.IBM.COM":
Re-enter password for principal "krbtgt/REALM1.IBM.COM@REALM2.IBM.COM":
Principal "krbtgt/REALM1.IBM.COM@REALM2.IBM.COM" created.
kadmin: addprinc -requires_preauth krbtgt/REALM2.IBM.COM@REALM1.IBM.COM
WARNING: no policy specified for krbtgt/REALM2.IBM.COM@REALM1.IBM.COM;
defaulting to no policy. Note that policy may be overridden by
ACL restrictions.
Enter password for principal "krbtgt/REALM2.IBM.COM@REALM1.IBM.COM":
Re-enter password for principal "krbtgt/REALM2.IBM.COM@REALM1.IBM.COM":
Principal "krbtgt/REALM2.IBM.COM@REALM1.IBM.COM" created.
kadmin:
kadmin: getprinc krbtgt/REALM2.IBM.COM@REALM1.IBM.COM
Principal: krbtgt/REALM2.IBM.COM@REALM1.IBM.COM
Expiration date: [never]
Last password change: Thu Aug 19 18:14:06 CDT 2004
Password expiration date: [none]
Maximum ticket life: 1 day 00:00:00
Maximum renewable life: 7 days 00:00:00
Last modified: Thu Aug 19 18:14:07 CDT 2004 (admin/admin@REALM1.IBM.COM)
```

Last successful authentication: [never]
Last failed authentication: [never]
Failed password attempts: 0
Number of keys: 4
Key: vno 1, Triple DES cbc mode with HMAC/sha1,
no salt
Key: vno 1, ArcFour with HMAC/md5,
no salt
Key: vno 1, AES-256 CTS mode with 96-bit SHA-1 HMAC,
no salt
Key: vno 1, DES cbc mode with RSA-MD5,
no salt

Attributes:

Policy: [none]

The same has to be done on the KDC server in the other realm.

Example 5-93 kadmin output on nfs403 KDC server

```
# /usr/krb5/sbin/kadmin -p admin/admin
Authenticating as principal admin/admin with password.
Password for admin/admin@REALM2.IBM.COM:
kadmin: addprinc -requires_preauth krbtgt/REALM1.IBM.COM@REALM2.IBM.COM
WARNING: no policy specified for krbtgt/REALM1.IBM.COM@REALM2.IBM.COM;
        defaulting to no policy. Note that policy may be overridden by
        ACL restrictions.
Enter password for principal "krbtgt/REALM1.IBM.COM@REALM2.IBM.COM":
Re-enter password for principal "krbtgt/REALM1.IBM.COM@REALM2.IBM.COM":
Principal "krbtgt/REALM1.IBM.COM@REALM2.IBM.COM" created.
kadmin: addprinc -requires_preauth krbtgt/REALM2.IBM.COM@REALM1.IBM.COM
WARNING: no policy specified for krbtgt/REALM2.IBM.COM@REALM1.IBM.COM;
        defaulting to no policy. Note that policy may be overridden by
        ACL restrictions.
Enter password for principal "krbtgt/REALM2.IBM.COM@REALM1.IBM.COM":
Re-enter password for principal "krbtgt/REALM2.IBM.COM@REALM1.IBM.COM":
Principal "krbtgt/REALM2.IBM.COM@REALM1.IBM.COM" created.

kadmin: getprinc krbtgt/REALM2.IBM.COM@REALM1.IBM.COM
Principal: krbtgt/REALM2.IBM.COM@REALM1.IBM.COM
Expiration date: [never]
Last password change: Thu Aug 19 18:18:10 2004
Password expiration date: [none]
Maximum ticket life: 1 day 00:00:00
Maximum renewable life: 7 days 00:00:00
Last modified: Thu Aug 19 18:18:10 2004 (admin/admin@REALM2.IBM.COM)
Last successful authentication: [never]
Last failed authentication: [never]
```

```
Failed password attempts: 0
Number of keys: 4
Key: vno 1, Triple DES cbc mode with HMAC/sha1,
no salt
Key: vno 1, ArcFour with HMAC/md5,
no salt
Key: vno 1, AES-256 CTS mode with 96-bit SHA-1 HMAC,
no salt
Key: vno 1, DES cbc mode with RSA-MD5,
no salt
```

Attributes:

```
Policy: [none]
kadmin:
```

5.12.2 Kerberos configuration file changes on the KDC server, NFS V4 client and server

1. We have to add the new realm to `/etc/krb5/krb5.conf` using `vi` to modify the two stanzas `[realms]` and `[domain_realm]`. Example 5-94 shows the changed contents of the `realms` stanza.

Example 5-94 New realms stanza entries

```
[realms]
  REALM1.IBM.COM = {
    kdc = nfs407.itsc.austin.ibm.com:88
    admin_server = nfs407.itsc.austin.ibm.com:749
    default_domain = itsc.austin.ibm.com
  }

  REALM2.IBM.COM = {
    kdc = nfs403.itsc.austin.ibm.com:88
    admin_server = nfs403.itsc.austin.ibm.com:749
    default_domain = itsc.austin.ibm.com
  }
```

2. Now we change the `domain_realm` stanza as shown in Example 5-95.

Example 5-95 Correct domain_realm stanza for cross-realm access

```
[domain_realm]
  nfs407.itsc.austin.ibm.com = REALM1.IBM.COM
  nfs403.itsc.austin.ibm.com = REALM2.IBM.COM
```

Important: Do not try to map one DNS domain into a second Kerberos realm map. Example 5-96 shows an invalid entry in the `krb5.conf` file.

Example 5-96 Incorrect domain_realm stanza for cross-realm access

```
[domain_realm]
    .itsc.austin.ibm.com = REALM1.IBM.COM
    nfs407.itsc.austin.ibm.com = REALM1.IBM.COM
    .itsc.austin.ibm.com = REALM2.IBM.COM
    nfs403.itsc.austin.ibm.com = REALM2.IBM.COM
```

3. Now we restart the `gssd` daemon using `stopsrc -g gssd` and `startsrc -g gssd`.

Note: Due to internal kernel caching, it may take up to two minutes until `gssd` recognizes the change.

5.12.3 Add NFS domain-to-realm map on NFS V4 client and server

The NFS domain is the same, so we do not need identity mapping.

To enable the `nfsrgyd` daemon to map identification requests, we have to add an additional NFS domain-to-realm map into file `/etc/nfs/realm.map`; otherwise, the NFS user `nobody` will be mapped.

Example 5-97 Cross-realm NFS domain-to-realm map file on nfs406 client

```
# chnfsrtd -a realm2.ibm.com itsc.austin.ibm.com
#
# chnfsrtd
realm1.ibm.com itsc.austin.ibm.com
realm2.ibm.com itsc.austin.ibm.com
# pg /etc/nfs/realm.map
realm1.ibm.com itsc.austin.ibm.com
realm2.ibm.com itsc.austin.ibm.com
```

Note: You have to recycle the `nfsrgyd` daemon for the new mapping to be loaded.

5.12.4 Client access verification

Example 5-98 on page 205 shows verification from the client to a principal in the other realm using the `kinit` command.

Example 5-98 kinit into the new added realm

```
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: sally@REALM1.IBM.COM

Valid starting Expires Service principal
08/19/04 18:24:20 08/20/04 18:23:56 krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
08/19/04 18:24:24 08/20/04 18:23:56 krbtgt/REALM2.IBM.COM@REALM1.IBM.COM
#
# kinit mary@REALM2.IBM.COM
Password for mary@REALM2.IBM.COM:
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: mary@REALM2.IBM.COM

Valid starting Expires Service principal
08/19/04 18:34:36 08/20/04 18:34:36 krbtgt/REALM2.IBM.COM@REALM2.IBM.COM
Renew until 08/20/04 18:34:55
```

5.12.5 Client access mount using cross-realms

At this stage the Kerberos environment modification has been validated. But our target is to access the server in REALM2.IBM.COM without having to use `kinit` into the realm. That is shown in the next sample.

Example 5-99 Client mount to a server within realm2

```
# kdestroy
# klist
Unable to get cache name (ticket cache: /var/krb5/security/creds/krb5cc_0).
Status 0x96c73ac3 - No credentials cache found.
# mount
node mounted mounted over vfs date options
-----
/dev/hd4 / jfs Aug 19 18:28 rw,log=/dev/hd8
/dev/hd2 /usr jfs Aug 19 18:28 rw,log=/dev/hd8
/dev/hd9var /var jfs Aug 19 18:28 rw,log=/dev/hd8
/dev/hd3 /tmp jfs Aug 19 18:29 rw,log=/dev/hd8
/dev/hd1 /home jfs Aug 19 18:29 rw,log=/dev/hd8
/proc /proc procfs Aug 19 18:29 rw
/dev/hd10opt /opt jfs Aug 19 18:29 rw,log=/dev/hd8
/dev/log_lv /var/nfs4log jfs Aug 19 18:29 rw,log=/dev/hd8
/dev/home_lv /exports/home jfs Aug 19 18:29 rw,log=/dev/hd8
/dev/exname_lv /local/trans jfs Aug 19 18:29 rw,log=/dev/hd8
/dev/projA_lv /exports/project/projA jfs2 Aug 19 18:29
rw,log=/dev/log1v00
```

```

        /dev/projB_lv /exports/project/projB jfs2 Aug 19 18:29
rw,log=/dev/loglv00
#
# kinit sally
Password for sally@REALM1.IBM.COM:
# mount -o vers=4,sec=krb5 nfs404:/ /nfs
# mount
node          mounted          mounted over    vfs      date          options
-----
/dev/hd4      /                  /                jfs      Aug 19 18:28 rw,log=/dev/hd8
/dev/hd2      /usr              /usr             jfs      Aug 19 18:28 rw,log=/dev/hd8
/dev/hd9var   /var              /var             jfs      Aug 19 18:28 rw,log=/dev/hd8
/dev/hd3      /tmp              /tmp             jfs      Aug 19 18:29 rw,log=/dev/hd8
/dev/hd1      /home             /home            jfs      Aug 19 18:29 rw,log=/dev/hd8
/proc        /proc             /proc            procfs   Aug 19 18:29 rw
/dev/hd10opt  /opt              /opt             jfs      Aug 19 18:29 rw,log=/dev/hd8
/dev/log_lv   /var/nfs4log     /var/nfs4log    jfs      Aug 19 18:29 rw,log=/dev/hd8
/dev/home_lv  /exports/home    /exports/home    jfs      Aug 19 18:29 rw,log=/dev/hd8
/dev/exname_lv /local/trans     /local/trans     jfs      Aug 19 18:29 rw,log=/dev/hd8
/dev/projA_lv /exports/project/projA jfs2 Aug 19 18:29
rw,log=/dev/loglv00
        /dev/projB_lv /exports/project/projB jfs2 Aug 19 18:29
rw,log=/dev/loglv00
nfs404 /          /nfs              nfs4 Aug 20 14:36 vers=4,sec=krb5
#
# cd /nfs/exports2/home
# ls -ltr
total 48
drwxr-sr-x  2 sally  staff      512 Jul 30 19:42 sally
drwxr-sr-x  2 bob    staff      512 Jul 30 19:42 bob
drwxr-sr-x  2 mary   staff      512 Jul 30 19:42 mary
drwxr-sr-x  2 joe    staff      512 Jul 30 19:42 joe
drwxrwx---  2 root   system    512 Aug 16 11:09 lost+found
#

```



Problem determination

This chapter provides guidance in carrying out problem determination with NFS V4. It is beyond to the scope of this book to cover the possible problems, but we will guide you through the thought process you should consider when trying to debug NFS V4 problems.

We start by giving a brief introduction to the tools and aids available with AIX and other third-party tools we have found useful. We then guide you through issues that we encountered while writing the book and how we resolved them.

The sections covered in this chapter are as follows:

- ▶ Problem determination tools and techniques
- ▶ AIX problem determination tools and aids for NFS
- ▶ IBM NAS problem determination tools and aids
- ▶ IBM Tivoli Directory Server problem determination tools and aids
- ▶ Third-party problem determination tools and aids
- ▶ File system exports problem
- ▶ Mount problems
- ▶ Major GSS-API error codes
- ▶ Kerberos V5 status codes

6.1 Problem determination tools and techniques

Before we can carry out any kind of problem determination, we need to be familiar with the tools that are available with AIX and provided by third parties. Third-party tools such as Ethereal (free at <http://www.ethereal.com>) can be very useful when you need to examine a tcpdump or an iptrace. Of course, AIX provides all of the necessary tools to convert an iptrace to something readable to the human eye, but it is not always easy to look through a large trace.

You may find additional useful information on this topic in *Introduction to the IBM Problem Determination Tools*, SG24-6296.

6.2 AIX problem determination tools and aids for NFS

We begin by looking at the tools provided by AIX. For more about using these tools, refer to the relevant man pages and AIX system reference documentation.

6.2.1 Enabling syslogd

NFS uses the syslog to write its error and debug information. Before you carry out any problem determination, we recommend that you turn syslog logging on. The following steps take you through the process of enabling syslog logging (if it is not already enabled on your system).

Tip: We created a separate file system for the syslog output and mounted it to `/var/logs/`. It is always good to do this, as logging daemons are notorious at filling up file systems. Allowing the syslog daemon to write to the root file system, then having it subsequently fill up, can lead to service interruption.

1. Edit the `/etc/syslog.conf` file and add an entry:

```
*.debug /var/logs/syslog/syslog.out rotate time 1d archive \  
/var/logs/syslog/archive/
```

We used extra parameters in this example to enable us to automatically rotate the log on a daily basis.

2. We now create the `/var/logs/syslog` and `/var/logs/syslog/archive` directories. We also create the `/var/logs/syslog/syslog.out` log file (Example 6-1).

Example 6-1 Creating the necessary directories and log file

```
# mkdir /var/logs/syslog  
#  
# mkdir /var/logs/syslog/archive
```

```
#
# ls -ld /var/logs/syslog
drwxr-sr-x 3 root    sys 512 Aug 10 18:59 /var/logs/syslog
#
# ls -ld /var/logs/syslog/archive
drwxr-sr-x 2 root    sys 512 Aug 10 18:59 /var/logs/syslog/archive
#
# touch /var/logs/syslog/syslog.out
#
# ls -al /var/logs/syslog/syslog.out
-rw-r--r-- 1 root    sys 168 Aug 10 19:00 /var/logs/syslog/syslog.out
#
```

3. We tell the syslogd that we have made changes to its configuration file and that it should reread it.

```
# refresh -s syslogd
0513-095 The request for subsystem refresh was completed successfully.
#
```

Example 6-21 on page 220 shows sample entries in the syslog output file.

4. When you have completed your problem determination steps, you may want to disable syslog logging. Doing so is usually a personal preference. Some systems administrators prefer to have all logging turned on while others choose to do it on demand. To disable syslog logging, comment out the line that was added to `/etc/syslog.conf` in step 1 on page 208 and refresh the syslogd as shown in Example 6-2.

Example 6-2 Disabling syslogd logging

```
/*.debug /var/logs/syslog/syslog.out rotate time 1d archive \
/var/logs/syslog/archive/
#
# refresh -s syslogd
0513-095 The request for subsystem refresh was completed successfully.
#
```

Using the setting for syslogd shown in Example 6-2 may log more information than you need while running your system in production mode. You can set syslog to log only ERROR in the `/etc/syslog.conf` file. This limits the amount of data written to the log file. This is the new line in the `/etc/syslog.conf` file:

```
/*.error /var/logs/syslog/syslog.out rotate time 1d archive \
/var/logs/syslog/archive/
```

6.2.2 Using iptrace and ipreport

iptrace is very useful when trying to debug problems that are not obvious. It is not the purpose of this book to show you how to interpret an iptrace, but to show you how to use the available tools. Follow these steps to create an iptrace:

1. Run the following command to start an iptrace:

```
startsrc -s iptrace -a "-a -d [source host] -b [LogFile]"
```

2. Recreate the problem scenario.
3. Stop the iptrace by running:

```
stopsrc -s iptrace
```

We now decode the iptrace file into a readable format using the **ipreport** command. You can also use Ethereal to decode the trace file (see 6.5.2, “Using the Ethereal utility” on page 212). We recommend either of two ways to decode the iptrace file:

- ▶ The first method is to decode everything:

```
ipreport -v [LogFile] > [OutputFile]
```

- ▶ The second method is to decode RPC packets only:

```
ipreport -nsrv [LogFile] > [OutputFile]
```

If you are trying to debug NFS V2 or NFS V3 problems, then the second method would be the correct option to choose. For NFS V4, we recommend that you use the first method. This enables you to decode the Kerberos packets as well as all other relevant information.

6.2.3 Using the fuser command

The **fuser** command lists the process numbers of local processes that use the local or remote files specified by the file parameter. For block special devices, the command lists the processes that use any file on that device. It can be useful when trying to determine **mount** or **umount** problems.

6.2.4 Using the rpcinfo command

The **rpcinfo** command reports the status of remote procedure call (RPC) servers. It is very useful when you need to check whether a system is running the correct NFS programs.

6.2.5 Using the showmount command

The **showmount** command displays a list of all clients that have remotely mounted file systems. The **showmount -a** command cannot show any NFS V4 exported file systems. We recommend using the **nfs4cl** command on the NFS V4 client.

6.2.6 Using the nfs4cl command

The **nfs4cl** command displays or modifies current NFS V4 statistics and properties. See the **showmount** command in the previous section.

6.2.7 Using the nfsstat command

The **nfsstat** command displays statistical information about NFS and RPC calls.

6.2.8 Using the errpt command

The **errpt** command generates a report of logged errors from the error log. It is useful when you are trying to determine why a daemon is core dumping or not starting. For the most detailed output, run the command in the following way:

```
errpt -a | more
```

6.3 IBM NAS problem determination tools

By default, krb5kdc and kadmind log directly to a file specified in the `[logging]` stanza of the `krb5.conf` file. (The default location of this file is `/var/krb5/log/`.) Logging to syslog instead can give you more control over the amount of logged information. The `krb5kdc` process logs an informational message for every ticket it issues. On a busy KDC, this can cause the log file to grow rapidly. If you log to syslog, you can log only the KDC errors. Table 6-1 shows the default files.

Table 6-1 IBM NAS Server log files

Filename	Description
krb5kdc.log	Log file of the KDC server process krb5kdc
kadmin.log	Log file of the administrative server process kadmind

6.4 Tivoli Directory Server problem determination tools

When a problem occurs that appears to be related to the IBM Tivoli Directory Server, you should first check the following files for error messages. By default, Tivoli Directory Server uses the `/var/ldap` directory to generate and write its error and debug information. The files shown in Example 6-2 are used by default.

Table 6-2 IBM Tivoli Directory Server log files

Filename	Description
<code>ibmslapd.log</code>	Log file of the Tivoli Directory Server main server process
<code>db2cli.log</code>	Log file for Tivoli Directory Server DB2 client interface

If you want to have more information logged by Tivoli Directory Server, you can change the level of variable `ibm-slapdSysLogLevel` in the `/etc/ibmslapd.conf` file.

6.5 Third-party problem determination tools

Here we will discuss tools that complement available AIX tools.

Restriction: Third-party tools are not supported by IBM. If you choose to use these tools, you do so at your own risk.

6.5.1 Using the `lsof` command

`lsof` stands for *list open files*. It lists information about files that are currently open by processes. The main difference between `lsof` and `fuser` is that `lsof` takes files, file systems, and PIDs as arguments, but `fuser` only accepts files and file systems. If `fuser` does not give you any output, try using `lsof`.

`lsof` for AIX is available for download from the following sites:

<http://aixpdslib.seas.ucla.edu>
<http://www.bullfreeware.com>

6.5.2 Using the Ethereal utility

`Ethereal` is used by network professionals around the world for troubleshooting, analysis, software and protocol development, and education. It has all of the standard features you would expect in a protocol analyzer, and several features

not seen in any other product. Its open source license enables experts in the networking community to add enhancements.

Ethereal can be installed on a Windows or Linux system. It is capable of reading the **iptrace** output.

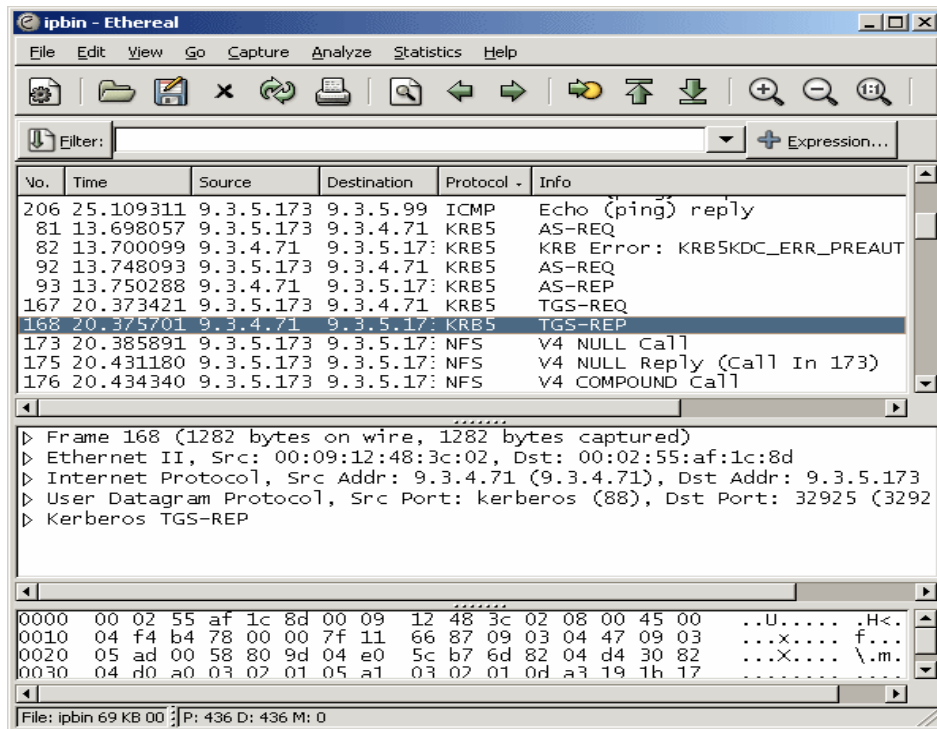


Figure 6-1 Sample screen shot of Ethereal

Ethereal is available for download from:

<http://www.ethereal.com>

6.6 General NFS V4 problems

The following messages may appear when using NFS V4.

6.6.1 Warning: EIM is not configured

After installing AIX 5.3 and activating an NFS V4 NFS domain the error entry shown in Example 6-3 on page 214 is logged into syslogd output file on the server and client side.

Example 6-3 EIM syslogd error entry

```
Aug 16 11:19:51 nfs403 syslog: nfsrgyd: dlopen(/usr/lib/libeim.a(shr.o))
failed: No such file or directory
Aug 16 11:19:51 nfs403 syslog: nfsrgyd: Warning: EIM is not configured
```

Problem determination steps

Not applicable

Solution

The message is telling you that Enterprise Identity Mapping (EIM) is not configured. If you are not using EIM, you can safely ignore the message. If you are using EIM, consult your EIM documentation for further assistance.

6.6.2 Realm is already mapped to domain

Trying to add a new NFS domain-to-realm map using `chnfsrtd` results in the message in Example 6-4.

Example 6-4 chnfsrtd message for duplicate domain

```
# chnfsrtd -a realm2.ibm.com nfs1.itsc.austin.ibm.com
Realm (realm2.ibm.com) is already mapped to domain (itsc.austin.ibm.com)
#
```

Problem determination steps

Use the `chnfsrtd` command without any argument to see the current settings.

Example 6-5 Output of chnfsrtd

```
# chnfsrtd
realm2.ibm.com nfs.itsc.austin.ibm.com
#
```

Solution

This behavior is expected. You cannot map the same realm to a second NFS domain.

6.7 Exporting file systems

The following error messages may appear when exporting file systems.

6.7.1 Exportfs: cannot change the v4 root...

We changed the `/etc/exports` file and added the following line to adopt the new pseudo-root FS:

```
/exports -nfsroot
```

The full `/etc/exports` file looked as shown in Example 6-6.

Example 6-6 The complete /etc/exports file for pseudo-root FS modification

```
/exports -nfsroot
/local/trans -vers=4,rw,exname=/exports/trans
/local/dept -vers=4,rw,exname=/exports/dept
/local/home -vers=4,rw,exname=/exports/home
/usr/codeshare/ThirdPartyProgs -vers=4,ro,exname=/exports/ThirdPartyProgs
```

After this, the `/etc/rc.nfs` command always fails with the message shown in Example 6-7.

Example 6-7 /etc/rc.nfs failure

```
# /etc/rc.nfs
Starting NFS services:
exportfs: cannot change the V4 root to /exports
```

Problem determination steps

Not applicable.

Solution

Check that the `/exports` directory exists. After creating the directory using command `mkdir /exports` everything worked fine.

6.7.2 Exportfs: /<path>: Invalid argument

After unexporting all file systems using the `exportfs -ua` command and subsequent export using the `exportfs -va` command, we get following output:

Example 6-8 Invalid argument while running exportfs -va

```
# exportfs -ua
# exportfs
```

```
exportfs: nothing exported
#
# exportfs -va
exportfs: /exports2: Invalid argument
exportfs: /exports2/home: Invalid argument
exportfs: /exports2/project/projA: Invalid argument
exportfs: /exports2/project/projB: Invalid argument
```

Problem determination steps

Check whether the pseudo-root FS has been set using the `nfsd -getnodes` command.

Example 6-9 Output of `nfsd -getnodes` command for invalid argument problem

```
# nfsd -getnodes
#root:public
/prootfs:/prootfs
#
```

Next step is to check whether the `/etc/exports` file contains the `-nfsroot` option.

Example 6-10 Check whether the `-nfsroot` option is in file `/etc/exports`

```
# pg /etc/exports
/exports2 -vers=4,sec=krb5:sys,rw
/exports2/home -vers=4,sec=krb5:sys,rw
/exports2/project/projA -vers=4,sec=krb5:sys,ro
/exports2/project/projB -vers=4,sec=krb5:sys,ro
```

Finally we verified that the `nfsd` daemon shows the `-root /` option.

Example 6-11 Check that `nfsd` is running with `-root /` option

```
# ps -ef |grep nfs |grep -v grep
   root 286924 159906  0 17:53:33    -  0:00 /usr/sbin/nfsd -root / 3891
```

Solution

The problem is caused by the missing `-nfsroot` option within the `/etc/exports` file. After adding the `/ -nfsroot` option to the `/etc/exports` file as shown in the next example, everything worked fine.

Example 6-12 Correct `/etc/exports` file with `-nfsroot` option set to `/`

```
# pg /etc/exports
/ -nfsroot
/exports2 -vers=4,sec=krb5:sys,rw
/exports2/home -vers=4,sec=krb5:sys,rw
```

```
/exports2/project/projA -vers=4,sec=krb5:sys,ro
/exports2/project/projB -vers=4,sec=krb5:sys,ro
#
# exportfs -va
exported /exports2
exported /exports2/home
exported /exports2/project/projA
exported /exports2/project/projB
#
# nfsd -getnodes
#root:public
/://
```

6.7.3 Exportfs: /var/<logfile>: Too many levels of symbolic links...

We changed the `/etc/exports` file to include this line to adopt the `exname` option:

```
/var/nfs4log -vers=4,sec=krb5:sys,ro,exname=/exports/logs
```

The full `/etc/exports` file looked as shown in Example 6-13.

Example 6-13 The complete /etc/exports file after modification

```
/exports -vers=4,sec=krb5:krb5i:krb5p:sys,rw
/exports/home -vers=4,sec=krb5:sys,rw
/exports/project/projA -vers=4,sec=krb5:sys,ro
/exports/project/projB -vers=4,sec=krb5:sys,ro
/var/nfs4log -vers=4,sec=krb5:sys,ro,exname=/exports/logs
```

After this, the `exportfs -va` command always fails with the message shown in Example 6-14.

Example 6-14 exportfs -va failure

```
exportfs: 1831-187 re-exported /exports
exportfs: 1831-187 re-exported /exports/home
exportfs: 1831-187 re-exported /exports/project/projA
exportfs: 1831-187 re-exported /exports/project/projB
exportfs: /var/nfs4log: There are too many levels of symbolic links to
translate a path name.
```

Problem determination steps

Check to see what the `nfsroot` is set to with the `nfsd -getnodes` command. The `exname` must begin with the `nfsroot`.

Solution

The use of the `exname` option with the pseudo-file system was incorrect. If you start using the `exname` option in the `/etc/exports` files you have to take the following into account:

1. Either all Version 4 exports must specify an external name, or none of the exports specify an external name.
2. The `nfsroot`, `/exports`, must not be exported.

After removing all non-`exname` entries from the `/etc/exports` file, the `exportfs` command ran without any further problems.

6.8 Mount problems

The NFS client can experience various types of mount problems, and in this section we look at the most common ones. Remember that if you are using NFS V4 with Kerberos, problem determination should always start at the client side. In this case using an `iptrace` should also be mandatory to break down the root cause of your problem. See “Sample `iptrace` output” on page 271 for a sample `iptrace` breakdown.

6.8.1 General mount problem

Problem

The mount on a client fails with access denied as shown in Example 6-15.

Example 6-15 Client mount returns access denied

```
# mount nfs403:/ /mnt
mount: 1831-011 access denied for nfs403:/
mount: 1831-008 giving up on:
nfs403:/
The file access permissions do not allow the specified action.
```

Problem determination steps

Check which NFS version has been used on the server to export the file system. In this case, the file system was exported using option:

```
/exports2 -vers=4,sec=krb5:sys,rw
```

Solution

Caused by use of the `vers=4` option on the client; by default, NFS V3 is used.

6.8.2 Pseudo-root and nfs4cl problems

Problem

This command works fine on the client:

```
mount -o vers=4,sec=krb5 nfs404:/ /nfs
```

However, this command output seems to be incorrect:

```
nfs4cl showfs
```

Example 6-16 Incorrect output from the nfs4cl showfs command

Server	Remote Path	fsid	Local Path
nfs404		10:13	/nfs/projB

Example 6-17 shows the output we expect to see.

Example 6-17 Correct output from the nfs4cl showfs command

Server	Remote Path	fsid	Local Path
nfs404		10:13	/nfs/project/projB

We see in Example 6-16 that in the Local Path column, **nfs4cl showfs** reports the mounted file system incorrectly. Example 6-17 shows the correct output.

Problem determination steps

Not applicable.

Solution

This is a problem with the **nfs4cl** command on AIX 5.3.

Note: As we wrote this book, this problem had been identified as a bug and should be resolved in a future Maintenance Level for AIX 5.3.

6.8.3 'vers' mount option error: "...Program not registered"

The **mount** command with option **vers=4** reports the error in Example 6-18.

Example 6-18 Errors when the mount command is used

```
# mount -o vers=4,sec=krb5 nfs404:/ /nfs
RPC: 1832-019 Program not registered
Verify the NFS local domain has been set, and the nfsrgyd process is running
```

Problem determination steps

Check that the NFS domain is set using the **chnfsdom** command, and that the associated NFS registry daemon **nfsrgyd** is running.

Example 6-19 Confirming the NFS domain is set and nfsrgyd is running

```
# chnfsdom
Current local domain: itsc.austin.ibm.com
#
# lssrc -s nfsrgyd
Subsystem      Group          PID           Status
nfsrgyd        nfs              
inoperative
#
```

Solution

Start the **nfsrgyd** using **/etc/rc.nfs**.

6.8.4 ‘vers’ mount option error: “...server <name> not responding”

If you try to mount with an unsupported NFS version number, the **mount** command will hang with message shown in Example 6-20.

Example 6-20 Mount command used with an incorrect NFS version number

```
# mount -o vers=5,sec=krb5 nfs402:/ /nfs
mount: 1831-010 server nfs402 not responding: RPC: Success
mount: retrying
```

Solution

Use a supported version number (Version 2, 3, or 4).

6.8.5 Mount command hangs - no system response

The **mount** command seems to hang without any message:

```
# mount -o vers=4,sec=krb5 nfs402:/ /nfs
```

Problem determination steps

Check the syslog entries and see if there are recently logged errors:

Example 6-21 Syslog entries showing output from gssd

```
Jul 30 20:24:50 localhost unix: kgss_init_sec_context returned
GSS_S_UNAVAILABLE
```

```
Jul 30 20:25:20 localhost unix: kgss_init_sec_context returned  
GSS_S_UNAVAILABLE
```

Solution

Restart the gssd subsystem or make sure it is running.

6.8.6 Mount with sec=krb5: “vmount: The file access permissions do not allow the specified action”

Running the following `mount` command returns the error shown in Example 6-22:

```
mount -o vers=4,sec=krb5 nfs404:/ /nfs
```

Example 6-22 Error returned by the mount command

```
mount: 1831-008 giving up on:  
nfs404:/  
vmount: The file access permissions do not allow the specified action.
```

This problem has multiple fail reasons even though this command works:

```
mount -o vers=4 nfs404:/ /nfs
```

Failure mode 1: problem determination steps

Check the syslog for errors similar to Example 6-23 and Example 6-24.

Example 6-23 Syslog entry on NFS V4 server if mounted from server

```
Jul 30 11:53:27 nfs404 unix: kgss_init_sec_context returned GSS_S_FAILURE  
KRB5_FCC_NOFILE
```

Example 6-24 Syslog entry on NFS V4 server if mounted from client

```
Jul 29 17:17:52 nfs404 syslog: accept_context failed, major=d0000,  
minor=96c73ae3  
Jul 29 17:17:52 nfs404 unix: kgss_accept_sec_context failed: GSS_S_FAILURE  
FFFFFFFF  
Jul 30 11:13:48 nfs404 syslog: nfsrgyd: Unable to map realm  
(itsc.austin.ibm.com) to a domain
```

Solution to failure mode 1

The `/etc/nfs/realm.map` file has been created incorrectly. Use the `chnfsrtd` command to recreate the file. This is the incorrect syntax for `/etc/nfs/realm.map`:

```
realm.map realm2.ibm.com itsc.austin.ibm.com
```

This is the correct syntax:

```
realm2.ibm.com itsc.austin.ibm.com
```

Failure mode 2: problem determination steps

Example 6-25 Errors from syslog output

```
Jul 30 16:02:45 nfs407 gssd[11968]: The user option is different from the
address family passed into the API.
Jul 30 16:02:45 nfs407 gssd[11968]: The user option is different from the
address family passed into the API.
Jul 30 16:02:45 nfs407 syslog: init_context failed, major=d0000, minor=96c73a07
Jul 30 16:02:45 nfs407 unix: kgss_init_sec_context returned GSS_S_FAILURE
KRB5KDC_ERR_S_PRINCIPAL_UNKNOWN
```

Solutions to failure mode 2

1.) *Incorrect hostkey entry*

There is an incorrect entry in the `/etc/nfs/hostkey` file for the NFS V4 server name. Use `nfshostkey -l` to check this:

Example 6-26 Using the nfshostkey command

```
# nfshostkey -l
nfs/nfs402.itsc.austin.ibm.com
/etc/krb5/krb5.keytab
#
```

The correct entry should be:

```
nfs/nfs407.itsc.austin.ibm.com
```

Carry out the following steps to resolve the problem:

a. Run the following command:

```
nfshostkey -p nfs/nfs407.itsc.austin.ibm.com -f\ /etc/krb5/krb5.keytab
```

b. Stop and restart NFS.

2.) *Incorrect nfshostmap entry*

There is an incorrect entry in the `/etc/nfs/princmap` file. Use the `nfshostmap -l` command to check this:

```
nfs407 nfs407.itsc.austin.ibm.com
```

This entry is incorrect. We do not have to map this host, as `nfs407` does not have a second interface. Carry out the following steps to resolve the problem:

1. Run the following command:

```
nfshostmap -d nfs407
```


2. Stop and restart the gssd subsystem:

```
stopsrc -s gssd
startsrc -s gssd
```

Restriction: As we write this book, we have found that the gssd takes approximately two minutes from the stop-and-start operation to update all local changes.

3.) *Incorrect credentials*

The credentials within the NAS credential cache file `/var/krb5/security/creds/krb5cc_0` are incorrect. Carry out the following steps to resolve the problem:

1. Unset the KRB5CCNAME variable with the following command:

```
unset KRB5CCNAME
```

2. To check the credentials available to the gssd daemon, run:

```
klist
```

3. Stop and restart the gssd subsystem:

```
stopsrc -s gssd
startsrc -s gssd
```

Failure mode 3: problem determination steps

The syslog file gives the entries shown in Example 6-27.

Example 6-27 KRB5_NO_TKT_IN_RLM in syslog file

```
Aug 16 11:14:50 nfs406 su: from root to sally at /dev/pts/0
Aug 16 11:14:50 nfs406 syslog: init_context failed, major=d0000, minor=96c73ab9
Aug 16 11:14:50 nfs406 unix: kgss_init_sec_context returned GSS_S_FAILURE
KRB5_NO_TKT_IN_RLM
```

Solution to failure mode 3

The client does not belong to the same Kerberos realm as the NFS V4 server. Thus you have to add the NFS domain to the `/etc/realm.map` file using the `chnfsrtd` or `smitty chnfsrtd` commands. After the change, this error will not be logged any more.

6.8.7 Mount with sec=krb5: “RPC: 1832-016 Unknown host...”

Running the following `mount` command returns the error shown in Example 6-28 on page 224, even though the NFS domain is set on the system:

```
mount -o vers=4,sec=krb5 nfs403:/ /nfs
```

Example 6-28 Unknown host error returned from the mount command

```
# mount -o vers=4,sec=krb5 nfs403:/ /nfs
RPC: 1832-016 Unknown host
Verify the NFS local domain has been set, and the nfsrgyd process is running
#
#chnfsdom
Current local domain: itsc.austin.ibm.com
#
# lssrc -s nfsrgyd
Subsystem      Group          PID           Status
nfsrgyd        nfs            16530         active
#
```

Problem determination steps

This message seems to indicate a local DNS/HOSTNAME resolution problem:

```
RPC: 1832-016 Unknown host
```

All other messages can be ignored at this time.

The local gssd must be able resolve the NFS V4 server as well as the KDC server, so we check whether the server names can be found using the **host** command as shown in Example 6-29.

Example 6-29 NFS V4 host IP / DNS test

```
# host nfs403
nfs403.itsc.austin.ibm.com is 9.3.5.173
#
# ping -c 1 nfs403
PING nfs403.itsc.austin.ibm.com (9.3.5.173): 56 data bytes
64 bytes from 9.3.5.173: icmp_seq=0 ttl=255 time=0 ms

--- nfs403.itsc.austin.ibm.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0/0/0 ms
#
```

This test passed, so the NFS server can be resolved. Next, we test whether the KDC server is known and resolvable (Example 6-30).

Example 6-30 Test to see whether the KDC server is known and accessible

```
# grep ":88" /etc/krb5/krb5.conf
      kdc = nfs409.kdc.itsc.austin.ibm.com:88
#
# host nfs409.kdc.itsc.austin.ibm.com
nfs409.kdc.itsc.austin.ibm.com is 9.3.4.71
```

```

# host nfs409
nfs409.itsc.austin.ibm.com is 9.3.5.179
# tn nfs409 88
Trying...
^C
# tn nfs409.kdc.itsc.austin.ibm.com 88
Trying...
Connected to nfs409.kdc.itsc.austin.ibm.com.
Escape character is '^T'.
^CConnection closed.
#

```

This test passed as well. We finally check the local host name and loopback interface using the **hostname** and **rpcinfo** commands (Example 6-31).

Example 6-31 Checking the local hostname and loopback interface

```

# hostname
nfs405.itsc.austin.ibm.com
#
root@nfs405 [scripts] host nfs405
nfs405.itsc.austin.ibm.com is 9.3.5.175
root@nfs405 [scripts] ping -c1 nfs405
PING nfs405.itsc.austin.ibm.com (9.3.5.175): 56 data bytes
64 bytes from 9.3.5.175: icmp_seq=0 ttl=255 time=0 ms

--- nfs405.itsc.austin.ibm.com ping statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max = 0/0/0 ms
root@nfs405 [scripts]
#
root@nfs405 [scripts] rpcinfo -p
  program vers proto  port  service
  100000    4   udp    111  portmapper
  100000    3   udp    111  portmapper
  100000    2   udp    111  portmapper
  100000    4   tcp    111  portmapper
  100000    3   tcp    111  portmapper
  100000    2   tcp    111  portmapper
  100003    2   udp    2049 nfs
  100003    3   udp    2049 nfs
  100003    2   tcp    2049 nfs
  100003    3   tcp    2049 nfs
  100003    4   tcp    2049 nfs
  200006    1   udp    2049
  200006    4   udp    2049
  200006    1   tcp    2049
  200006    4   tcp    2049
  400003    1   udp    33843

```

```

100005 1 tcp 32787 mountd
100005 1 udp 33847 mountd
100005 2 udp 33847 mountd
100005 3 udp 33847 mountd
400234 1 udp 33849
100024 1 tcp 32788 status
100024 1 udp 33851 status
100133 1 tcp 32788
100133 1 udp 33851
200001 1 tcp 32788
200001 1 udp 33851
200001 2 tcp 32788
200001 2 udp 33851
400005 1 udp 33848
100021 1 udp 33882 nlockmgr
100021 2 udp 33882 nlockmgr
100021 3 udp 33882 nlockmgr
100021 4 udp 33882 nlockmgr
100021 1 tcp 32789 nlockmgr
100021 2 tcp 32789 nlockmgr
100021 3 tcp 32789 nlockmgr
100021 4 tcp 32789 nlockmgr
#
#root@nfs405 [scripts] host loopback
host: name loopback NOT FOUND
#

```

Solution

The system's **loopback** interface name resolution failed (see RFC1537 for details). This indicates an incorrect entry on the DNS server or `/etc/hosts` setup. Correcting the system loopback entry resolves the problem and the `host` command returns the correct entry, shown in Example 6-32.

Example 6-32 Correct entry returned by the host command

```

#root@nfs405 [scripts] host loopback
loopback is 127.0.0.1, Aliases: localhost
root@nfs405 [scripts] mount -o vers=4,sec=krb5 nfs402:/ /nfs
#

```

6.8.8 File and directory access: `cd`, `ls`, etc. return “permission denied”

A NFS client cannot access the mounted files and directories. Several commands, such as `cd` or `ls`, return a permission-denied message (Example 6-33 on page 227).

Example 6-33 Permission denied messages from several commands

```
# nfs4cl showfs

Server      Remote Path      fsid            Local Path
-----      -
nfs403      /exports/home    10:12          /nfs
#
# cd /nfs
ksh: /nfs: permission denied
#
# ls -l /nfs
/nfs: No permission
#
```

Failure mode 1: problem determination steps

1. Check the syslog entries on the client, as shown in Example 6-34.

Example 6-34 Syslog entries seen as a result of the permissions problem

```
Aug 6 09:17:30 nfs403 syslog: init_context failed, major=d0000, minor=96c73ab9
Aug 6 09:17:30 nfs403 unix: kgss_init_sec_context returned GSS_S_FAILURE
KRB5_NO_TKT_IN_RLM
```

These errors indicate that no valid ticket has been issued to the user.

2. You can also check the ticket status by running the `klist` command.

Example 6-35 Checking the ticket status using klist

```
#date
Tue Aug 10 15:00:52 CDT 2004
#
#klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: sally@REALM2.IBM.COM

Valid starting    Expires          Service principal
08/02/04 11:14:12    08/03/04 11:14:11    krbtgt/REALM2.IBM.COM@REALM2.IBM.COM
08/02/04 11:14:19    08/03/04 11:14:11
nfs/nfs402.itsc.austin.ibm.com@REALM2.IBM.COM
```

Solution to failure mode 1

Request a new ticket using the `kinit` command to resolve the problem (Example 6-36 on page 228).

Example 6-36 Requesting a new ticket for the user using the kinit command

```
# kinit sally
Password for sally@KDC.ITSC.AUSTIN.IBM.COM:
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: sally@KDC.ITSC.AUSTIN.IBM.COM

Valid starting Expires Service principal
08/06/04 09:27:06 08/06/04 19:27:13
krbtgt/KDC.ITSC.AUSTIN.IBM.COM@KDC.ITSC.AUSTIN.IBM.COM
Renew until 08/07/04 09:27:06

#
# cd /nfs
#
# ls
bob          joe          lost+found  mary        sally
#
```

Failure mode 2: problem determination steps

Check the syslog log file and check whether you can see an entry similar to the one in Example 6-37.

Example 6-37 Error showing the gssd being unable to access a valid ticket cache

```
Aug 10 12:25:50 nfs404 unix: kgss_init_sec_context returned GSS_S_FAILURE
KRB5_FCC_NOFILE
```

The error indicates that the gssd process is unable to access a valid ticket cache file for the principal.

Check to see whether the KRB5CCNAME environment variable has been set.

Example 6-38 Has the KRB5CCNAME environment variable been set?

```
$ set |grep -i krb
AUTHSTATE=KRB5LDAP
KRB5CCNAME=FILE:/var/krb5/security/creds/krb5cc_sally@REALM1.IBM.COM_6023
PATH=/usr/bin:/etc:/usr/sbin:/usr/ucb:/usr/bin/X11:/sbin:/usr/krb5/bin:/usr/java14/jre/bin:/usr/java14/bin:/usr/java131/jre/bin:/usr/java131/bin
```

Solution to failure mode 2

Change the user environment by unsetting the KRB5CCNAME environment variable and obtaining a new ticket, as shown in Example 6-39 on page 229.

Example 6-39 Unsetting KRB5CCNAME and obtaining a new ticket

```
$ unset KRB5CCNAME
$
$ klist
Unable to get cache name (ticket cache: /var/krb5/security/creds/krb5cc_6023).
    Status 0x96c73ac3 - No credentials cache found.
$
$ kinit sally
Password for sally@REALM1.IBM.COM:
$
$ cd /nfs
$
```

6.8.9 File and directory access: file ownership is “nobody:nobody”

A newly created file has ownership nobody:nobody instead of the format username:groupname. The server and client are in the same NFS domain, and the identity of the user is valid. Example 6-40 shows the problem.

Example 6-40 The nobody:nobody ownership issue

```
# tn nfs406
Trying...
Connected to nfs406.itsc.austin.ibm.com.
Escape character is '^T'.
AIX Version 5
(C) Copyrights by IBM and by others 1982, 2004.
login: sally
sally's Password:
$
$ id
uid=6023(sally) gid=1(staff) groups=1400(eng)
$
$ klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_6023
Default principal: sally@REALM1.IBM.COM

Valid starting    Expires          Service principal
08/11/04 11:06:29 08/12/04 11:06:23  krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
08/11/04 11:06:34 08/12/04 11:06:23  nfs/nfs404.itsc.austin.ibm.com@REALM1.IBM.COM
$
$ mount
   node          mounted          mounted over    vfs      date          options
-----
nfs404  /              /nfs             nfs4      Aug 11 10:53  vers=4,sec=krb5
$ cd /nfs/tmp
```

```
$
$ touch thisisatest
$
$ ls -l thisisatest
-rw-r--r--  1 nobody  nobody           0 Aug 11 10:54 thisisatest
$
```

Problem determination steps

On the NFS server, the syslog log file gives entries as shown in Example 6-41.

Example 6-41 NFS V4 registry daemon entries in the syslog log file

```
Aug 11 11:09:22 nfs404 syslog: nfsrgyd: Unable to map local user (sally) to a
foreign user
Aug 11 11:09:22 nfs404 syslog: nfsrgyd: Unable to map local group (staff) to a
foreign group
```

Solution

The realm to NFS domain mapping in the `/etc/nfs/realm.map` file is incorrect. Carry out the following steps to resolve the problem:

1. Change the mapping using the `chnfsrtd` command to make the change from:

```
realm1.ibm.com nfstest.itsc.austin.ibm.com
```

to

```
realm1.ibm.com itsc.austin.ibm.com
```

2. Stop and start the `nfsrgyd` on NFS server `nfs404`:

```
stopsrc -s nfsrgyd
startsrc -s nfsrgyd
```

Example 6-42 shows the test we carried out to prove that the issue had been resolved.

Example 6-42 The resolved issue after steps 1 and 2 have been carried out

```
$ cd /nfs
$
$ touch thisisatest2
$
$ ls -l thisisatest2
-rw-r--r--  1 sally   staff           0 Aug 11 11:08 thisisatest2
$
```

6.8.10 NAS problem: kadmin: “Unable to initialize kadmin interface”

After installing the NAS file sets and configuration of the NAS client, `kadmin` no longer works and results in the error shown in Example 6-43.

Example 6-43 kadmin initialization error

```
# /usr/krb5/sbin/kadmin
Authenticating as principal root/admin@REALM1.IBM.COM with password.
Unable to initialize kadmin interface.
      Status 0x29c2500 - Operation failed for unspecified reason.
```

Problem determination steps

See whether you can log on to Kerberos using the principal `admin/admin`.

Example 6-44 kinit verification with principal admin/admin

```
# kinit admin/admin
Password for admin/admin@REALM1.IBM.COM:
#
# klist
Ticket cache: FILE:/var/krb5/security/creds/krb5cc_0
Default principal: admin/admin@REALM1.IBM.COM

Valid starting    Expires          Service principal
08/12/04 10:20:04  08/13/04 10:20:01  krbtgt/REALM1.IBM.COM@REALM1.IBM.COM
```

Solution

The problem is caused by the fact that principal `root/admin` is not known as an administrative principal. So this message is normal and we recommend that you use the `kadmin` command with the `-p admin/admin` option.

Example 6-45 kadmin verification with -p option

```
# /usr/krb5/sbin/kadmin -p admin/admin
Authenticating as principal admin/admin with password.
Password for admin/admin@REALM1.IBM.COM:
kadmin:
```

6.9 GSS-API error codes

Each GSS-API function returns two status codes: a major status code and a minor status code. Major status codes indicate behavior of the GSS-API itself.

For example, if an application attempts to transmit a message after a security context has expired, GSS-API returns a major status code of `GSS_S_CONTEXT_EXPIRED`. Major status codes are listed among the GSS-API status codes.

Minor status codes are returned by the underlying security mechanisms that are supported by a given implementation of GSS-API. Every GSS-API function takes as the first argument a `minor_status` or `minor_stat` parameter. An application can examine this parameter when the function returns, successfully or not, to see the status that is returned by the underlying mechanism.

Example 6-46 GSS-API syslog entries

```
Aug 16 11:14:50 nfs406 syslog: init_context failed, major=d0000, minor=96c73ab9
Aug 16 11:14:50 nfs406 unix: kgss_init_sec_context returned GSS_S_FAILURE
KRB5_NO_TKT_IN_RLM
```

In Example 6-46, the following important information is provided:

- ▶ The GSS subroutine called `kgss_init_sec_context` returned with GSS-API:
 - Major status: `GSS_S_FAILURE`
 - Minor status: `KRB5_NO_TKT_IN_RLM`

The following two sections provide a quick messages reference. A more detailed list can be found in *IBM Network Authentication Service Version 1.4 for AIX, Linux, Solaris and Windows Application Development Reference*, available in the `krb5.doc.en_US` file set.

6.9.1 Major GSS-API error codes

The following table lists calling errors returned by GSS-API.

Table 6-3 GSS-API routine failures

Error code	Description
<code>GSS_S_BAD_MECH</code>	An unsupported mechanism was requested.
<code>GSS_S_BAD_NAME</code>	An invalid name was supplied.

Error code	Description
GSS_S_BAD_NAMETYPE	A supplied name was of an unsupported type.
GSS_S_BAD_BINDINGS	Incorrect channel bindings were supplied.
GSS_S_BAD_STATUS	An invalid status code was supplied.
GSS_S_BAD_SIG	A token had an invalid signature.
GSS_S_NO_CRED	No credentials were supplied, or the credentials were unavailable or inaccessible.
GSS_S_NO_CONTEXT	No context has been established.
GSS_S_DEFECTIVE_TOKEN	A token was invalid.
GSS_S_DEFECTIVE_CREDENTIAL	A credential was invalid.
GSS_S_CREDENTIALS_EXPIRED	The referenced credentials have expired.
GSS_S_CONTEXT_EXPIRED	The context has expired.
GSS_S_FAILURE	Miscellaneous failure. The underlying mechanism detected an error for which no specific GSS-API status code is defined. The mechanism-specific status code (minor-status code) provides more details about the error.
GSS_S_BAD_QOP	The quality-of-protection requested could not be provided.
GSS_S_UNAUTHORIZED	The operation is forbidden by local security policy.
GSS_S_UNAVAILABLE	The operation or option is unavailable.
GSS_S_DUPLICATE_ELEMENT	The requested credential element already exists.
GSS_S_NAME_NOT_MN	The provided name was not a mechanism name (MN).

6.9.2 Kerberos v5 status codes

The following tables list the status messages that can be returned by Kerberos V5 in the `minor_status` argument. For more about GSS-API status codes, see GSS-API Status Codes.

Messages returned in Kerberos v5 for KRB5KDC status codes

Table 6-4 Kerberos V5 KRB5KDC status codes

Minor status code	Description
KRB5KDC_ERR_BAD_PVNO	Requested protocol version not supported.
KRB5KDC_ERR_BADOPTION	KDC cannot fulfill requested option.
KRB5KDC_ERR_C_OLD_MAST_KVNO	Client's key is encrypted in an old master key.
KRB5KDC_ERR_C_PRINCIPAL_UNKNOWN	Client not found in Kerberos database.
KRB5KDC_ERR_CANNOT_POSTDATE	Ticket is ineligible for postdating.
KRB5KDC_ERR_CLIENT_NOTYET	Client not yet valid; try again later.
KRB5KDC_ERR_CLIENT_REVOKED	Client's credentials have been revoked.
KRB5KDC_ERR_ETYPE_NOSUPP	KDC has no support for encryption type.
KRB5KDC_ERR_KEY_EXP	Password has expired.
KRB5KDC_ERR_NAME_EXP	Client's entry in database has expired.
KRB5KDC_ERR_NEVER_VALID	Requested effective lifetime is negative or too short.
KRB5KDC_ERR_NONE	No error.
KRB5KDC_ERR_NULL_KEY	Client or server has a null key.
KRB5KDC_ERR_PADATA_TYPE_NOSUPP	KDC has no support for padata type.
KRB5KDC_ERR_POLICY	KDC policy rejects request.
KRB5KDC_ERR_PREAUTH_FAILED	Preauthentication failed.
KRB5KDC_ERR_PREAUTH_REQUIRED	Additional preauthentication required.

Minor status code	Description
KRB5KDC_ERR_PRINCIPAL_NOT_UNIQUE	Principal has multiple entries in Kerberos database.
KRB5KDC_ERR_S_OLD_MAST_KVNO	Server's key is encrypted in an old master key.
KRB5KDC_ERR_S_PRINCIPAL_UNKNOWN	Server not found in Kerberos database.
KRB5KDC_ERR_SERVER_NOMATCH	Requested server and ticket do not match.
KRB5KDC_ERR_SERVICE_EXP	Server's entry in database has expired.
KRB5KDC_ERR_SERVICE_NOTYET	Server not yet valid; try again later.
KRB5KDC_ERR_SERVICE_REVOKED.	Credentials for server have been revoked.
KRB5KDC_ERR_SUMTYPE_NOSUPP	KDC has no support for checksum type.
KRB5KDC_ERR_TGT_REVOKED	TGT has been revoked.
KRB5KDC_ERR_TRTYPE_NOSUPP	KDC has no support for transited type.

Messages returned in Kerberos V5 for KRB5KRB status code

Table 6-5 Kerberos V5 KRB5KRB status codes

Minor status codes	Description
KRB5KRB_AP_ERR_BAD_INTEGRITY	Decrypt integrity check failed.
KRB5KRB_AP_ERR_BADADDR	Incorrect net address.
KRB5KRB_AP_ERR_BADDIRECTION	Incorrect message direction.
KRB5KRB_AP_ERR_BADKEYVER.	Key version is not available.
KRB5KRB_AP_ERR_BADMATCH	Ticket and authenticator do not match.
KRB5KRB_AP_ERR_BADORDER	Message out of order.
KRB5KRB_AP_ERR_BADSEQ	Incorrect sequence number in message.
KRB5KRB_AP_ERR_BADVERSION	Protocol version mismatch.

Minor status codes	Description
KRB5KRB_AP_ERR_ILL_CR_TKT	Illegal cross-realm ticket.
KRB5KRB_AP_ERR_INAPP_CKSUM	Inappropriate type of checksum in message.
KRB5KRB_AP_ERR_METHOD	Alternative authentication method required.
KRB5KRB_AP_ERR_MODIFIED	Message stream modified.
KRB5KRB_AP_ERR_MSG_TYPE	Invalid message type.
KRB5KRB_AP_ERR_MUT_FAIL	Mutual authentication failed.
KRB5KRB_AP_ERR_NOKEY	Service key not available.
KRB5KRB_AP_ERR_NOT_US	The ticket is not for us.
KRB5KRB_AP_ERR_REPEAT	Request is a replay.
KRB5KRB_AP_ERR_SKEW	Clock skew too great.
KRB5KRB_AP_ERR_TKT_EXPIRED	Ticket expired.
KRB5KRB_AP_ERR_TKT_INVALID	Ticket has invalid flag set.
KRB5KRB_AP_ERR_TKT_NYV	Ticket not yet valid.
KRB5KRB_AP_ERR_V4_REPLY	Initial ticket response appears to be Version 4 error.
KRB5KRB_AP_WRONG_PRINC	Wrong principal in request.
KRB5KRB_ERR_FIELD_TOOLONG	Field is too long for this implementation.
KRB5KRB_ERR_GENERIC	Generic error.

Messages returned in Kerberos for KRB5 status codes

Table 6-6 Kerberos V5 general KRB5 status codes

Minor codes	Description
KRB5_BAD_ENCTYPE	Bad encryption type.
KRB5_BAD_KEYSIZE	Key size is incompatible with encryption type.
KRB5_BAD_MSIZ	Message size is incompatible with encryption type.

Minor codes	Description
KRB5_BADMSGTYPE	Invalid message type specified for encoding.
KRB5_CC_BADNAME	Credential cache name malformed.
KRB5_CC_END	End of credential cache reached.
KRB5_CC_FORMAT	Bad format in credentials cache.
KRB5_CC_IO	Credentials cache I/O operation failed.
KRB5_CC_NOMEM	No more memory to allocate (in credentials cache code).
KRB5_CC_NOTFOUND	Matching credential not found.
KRB5_CC_TYPE_EXISTS	Credentials cache type is already registered.
KRB5_CC_UNKNOWN_TYPE	Unknown credential cache type.
KRB5_CC_WRITE	Error writing to credentials cache file.
KRB5_CCACHE_BADVNO	Unsupported credentials cache format version number.
KRB5_CONF_NOT_CONFIGURED	Kerberos /etc/krb5/krb5.conf configuration file not configured.
KRB5_CONFIG_BADFORMAT	Improper format of Kerberos /etc/krb5/krb5 configuration file.
KRB5_CONFIG_CANTOPEN	Cannot open or find Kerberos /etc/krb5/krb5 configuration file.
KRB5_CONFIG_NODEFREALM	Configuration file /etc/krb5/krb5.conf does not specify default realm.
KRB5_CONFIG_NOTENUFSPACE	Insufficient space to return complete information.
KRB5_ERR_BAD_HOSTNAME	Host name cannot be canonicalized.
KRB5_ERR_HOST_REALM_UNKNOWN	Cannot determine realm for host.
KRB5_FCC_INTERNAL	Internal file credentials cache error.
KRB5_FCC_NOFILE	No credentials cache file found.
KRB5_FCC_PERM	Credentials cache file permissions incorrect.

Minor codes	Description
KRB5_FWD_BAD_PRINCIPAL	Bad principal name while trying to forward credentials.
KRB5_GET_IN_TKT_LOOP	Looping detected in krb5_get_in_tkt.
KRB5_IN_TKT_REALM_MISMATCH	Client/server realm mismatch in initial ticket request.
KRB5_KDC_UNREACH	Cannot contact any KDC for requested realm.
KRB5_KDCREP_MODIFIED	KDC reply did not match expectations.
KRB5_KDCREP_SKEW	Clock skew too great in KDC reply.
KRB5_KEYTAB_BADVNO	Unsupported key table format version number.
KRB5_KT_BADNAME	Key table name malformed.
KRB5_KT_END	End of key table reached.
KRB5_KT_IOERR	Error writing to key table.
KRB5_KT_KVNONOTFOUND	Key version number for principal in key table is incorrect.
KRB5_KT_NAME_TOOLONG	Keytab name too long.
KRB5_KT_NOTFOUND	Key table entry not found.
KRB5_KT_NOWRITE	Cannot write to specified key table.
KRB5_KT_TYPE_EXISTS	Key table type is already registered.
KRB5_KT_UNKNOWN_TYPE	Unknown key table type.
KRB5_MUTUAL_FAILED	Mutual authentication failed.
KRB5_NO_2ND_TKT	Request missing second ticket.
KRB5_NO_LOCALNAME	No local name found for principal name.
KRB5_NO_TKT_IN_RLM	Cannot find ticket for requested realm.
KRB5_NO_TKT_SUPPLIED	Request did not supply a ticket.
KRB5_NOCREDS_SUPPLIED	No credentials supplied to library routine.

Minor codes	Description
KRB5_PREAUTH_BAD_TYPE	Unsupported preauthentication type.
KRB5_PREAUTH_FAILED	Generic preauthentication failure.
KRB5_PREAUTH_NO_KEY	Required preauthentication key not supplied.
KRB5_PRINC_NOMATCH	Requested principal and ticket do not match.
KRB5_PROG_ATYPE_NOSUPP	Program lacks support for address type.
KRB5_PROG_ETYPE_NOSUPP	Program lacks support for encryption type.
KRB5_PROG_KEYTYPE_NOSUPP	Program lacks support for key type.
KRB5_PROG_SUMTYPE_NOSUPP	Program lacks support for checksum type.
KRB5_RC_IO_EOF	End-of-file on replay cache I/O.
KRB5_RC_IO_MALLOC	No more memory to allocate (in replay cache I/O code).
KRB5_RC_IO_PERM	Permission denied in replay cache code.
KRB5_RC_MALLOC	No more memory to allocate (in replay cache code).
KRB5_RC_PARSE	Replay cache name parse/format error.
KRB5_RC_REQUIRED	Message replay detection requires rcache parameter.
KRB5_RC_TYPE_EXISTS	Replay cache type is already registered.
KRB5_RC_TYPE_NOTFOUND	Replay cache type is unknown.
KRB5_RCACHE_BADVNO	Unsupported replay cache format version number.
KRB5_REALM_CANT_RESOLVE	Cannot resolve KDC for requested realm.
KRB5_REALM_UNKNOWN	Cannot find KDC for requested realm.

Minor codes	Description
KRB5_SAM_UNSUPPORTED	Bad SAM flags in obtain_sam_padata.
KRB5_SENDAUTH_BADAPPLVERS	Bad application version was sent (by sendauth).
KRB5_SENDAUTH_BDAUTHVERS	Bad sendauth version was sent.
KRB5_SENDAUTH_BADRESPONSE	Bad response (during sendauth exchange).
KRB5_SENDAUTH_REJECTED	Server rejected authentication (during sendauth exchange).
KRB5_SERVICE_UNKNOWN	Kerberos service unknown.
KRB5_SNAME_UNSUPP_NAMETYPE	Conversion to service principal undefined for name type.
KRB5_TKT_NOT_FORWARDABLE	Requesting ticket cannot get forwardable tickets.
KRB5_WRONG_ETYPE	Requested encryption type not used in message.

Messages returned in Kerberos v5 for KRB5DES status codes

Table 6-7 Kerberos V5 KRB5DES status codes

Minor codes	Description
KRB5DES_BAD_KEYPAR	DES key has bad parity.
KRB5DES_WEAK_KEY	DES key is a weak key.

Appendixes

This part includes the following sections:

- ▶ Appendix A, “Kerberos” on page 243
- ▶ Appendix B, “Sample scripts, files, and output” on page 255
- ▶ Appendix C, “AIX 5.3 NFS quick reference” on page 287



A

Kerberos

This appendix describes the Kerberos authentication method. It also provides a list of references that contain more in-depth information about Kerberos.

The appendix contains the following sections:

- ▶ Overview
- ▶ Kerberos keys and initial setup
- ▶ Authenticating to the Kerberos server
- ▶ Authenticating to an application server
- ▶ Kerberos terminology
- ▶ Where to find more information about Kerberos

The first four sections are reprinted from the IBM Redbook *The RS/6000 SP Inside Out*, SG24-5374.

Overview

Kerberos: Also spelled Cerberus, the watchdog of Hades, whose duty was to guard the entrance (against whom or what does not clearly appear)... It is known to have had three heads.

- Ambrose Bierce, *The Enlarged Devil's Dictionary*

Kerberos is a trusted third-party authentication system for use on physically insecure networks. It enables entities communicating over the network to prove their identity to each other while preventing eavesdropping or replay attacks. Figure 6-2 shows the three parties involved in the authentication. The Kerberos system was designed and developed in the 1980s by the Massachusetts Institute of Technology (MIT) as part of the Athena project. The current version of Kerberos is Version 5, which is standardized in RFC 1510, *The Kerberos Network Authentication Service (V5)*.

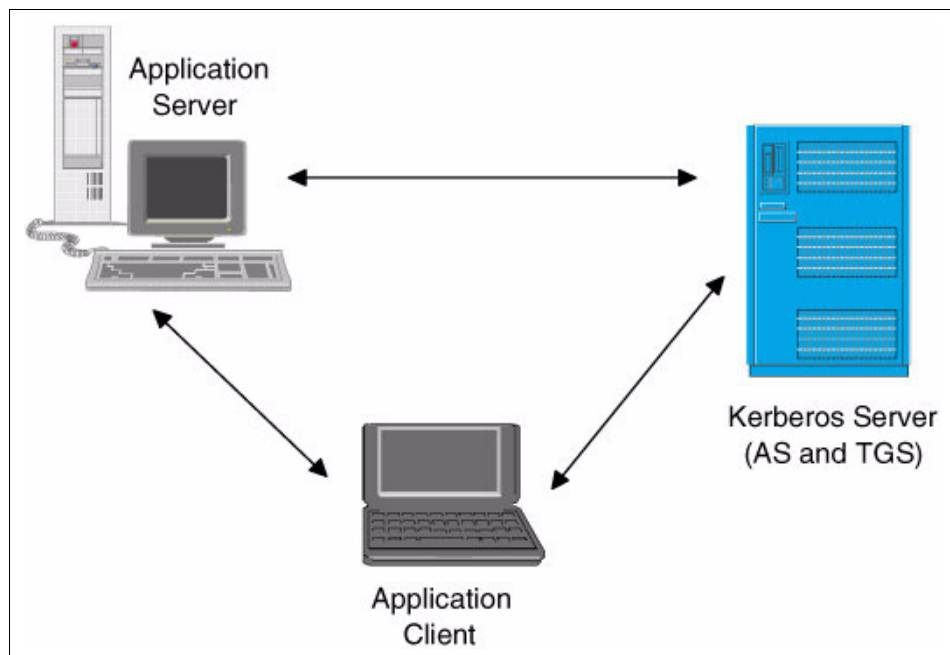


Figure 6-2 Partners in a third-party authentication

Kerberos provides two services to Kerberos principals (users or services): an authentication service (AS) and a ticket-granting service (TGS). Principals can prove their identity to the AS with a single sign-on and get a ticket-granting ticket (TGT) back from the AS. When one authenticated principal (the client) wants to

use the services of a second authenticated principal (the server), it can get a service ticket for this service by presenting its TGT to the Kerberos TGS. The service ticket is then sent from the client to the server, which can use it to verify the client's identity.

This section describes the protocol that Kerberos uses to provide these services, independent of a specific implementation. A more detailed rationale for the Kerberos design can be found in the MIT article *Designing an Authentication System: a Dialogue in Four Scenes*, which is available from:

<http://web.mit.edu/kerberos/www/dialogue.html>

Kerberos keys and initial setup

To encrypt the messages that are sent over the network, Kerberos uses a symmetric encryption method, normally the Data Encryption Standard (DES). This means that the same key is used to encrypt and decrypt a message, and consequently the two partners of a communication must share this key if they want to use encryption. The key is called a secret key for obvious reasons: it must be kept secret by the two parties; otherwise the encryption will not be effective.

This approach must be distinguished from public key cryptography, which is an asymmetric encryption method. There, two keys are used: a public key and a private key. A message encrypted with one of the keys can only be decrypted by the other key, not by the one that encrypted it. The public keys do not need to be kept secret (hence the name "public"), and a private key is known only to its owner. (It is not even to the communication partner as in the case of symmetric cryptography.) This has the advantage that no key must be transferred between the partners prior to the first use of encrypted messages.

With symmetric encryption, principals must provide a password to the Kerberos server before they can use the Kerberos services. The Kerberos server then encrypts it and stores the resulting key in its database. This key is the shared information that the Kerberos server and the principal can use to encrypt and decrypt the messages they send each other. Initially, two principals who want to communicate with each other do not share a key, and so cannot encrypt their messages. But since the Kerberos server knows the keys of all the principals, it is called a trusted third party and can securely provide a common session key to the two parties.

Obviously, the initial passwords have to be entered securely, if possible at the console of the Kerberos server machine. They might also be generated by the Kerberos server (especially if the principal is a host or service). In that case they must be securely transferred to the principal that stores (or remembers) them.

Authenticating to the Kerberos server

If a principal (typically a user) wants to use Kerberos services (for example, because it wants to use an application service that requires Kerberos authentication), it first has to prove its identity to the Kerberos server. This is done in the following way:

A command to sign on to the Kerberos system is issued on the application client, typically `kinit`. This command sends an authentication request to the Kerberos server, as shown in Figure A-1. This contains the type of service that is requested (here, the client wants to get service from the ticket-granting service), the client's (principal's) name, and the IP address of the client machine. This request is sent in plain text. Note that the principal's password is not sent in this packet, so there is no security exposure in sending the request in plain text.

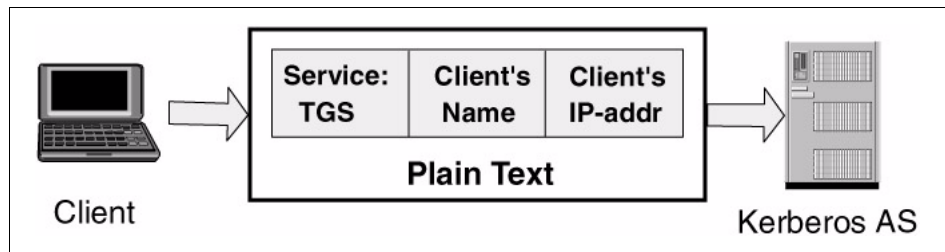


Figure A-1 Client's authentication request

The request is processed by the authentication server. Using the client's name, it looks up the corresponding key in the Kerberos database. It also generates a random session key to be shared by the client and the TGS, which will be used to encrypt all future communication of the client with the TGS. With this information, the AS constructs the ticket-granting ticket for the client, which (as with all Kerberos tickets) contains six parts:

1. The service for which the ticket is good (here, the TGS)
2. The client's (principal's) name
3. The client machine's IP address
4. A timestamp showing when the ticket was issued
5. The ticket lifetime (configurable in K5)
6. The session key for client/TGS communications

This ticket is encrypted with the secret key of the TGS, so only the TGS can decrypt it. Because the client needs to know the session key, the AS sends back a reply that contains both the TGT and the session key, all of which is encrypted by the client's secret key. This is shown in Figure A-2 on page 247.

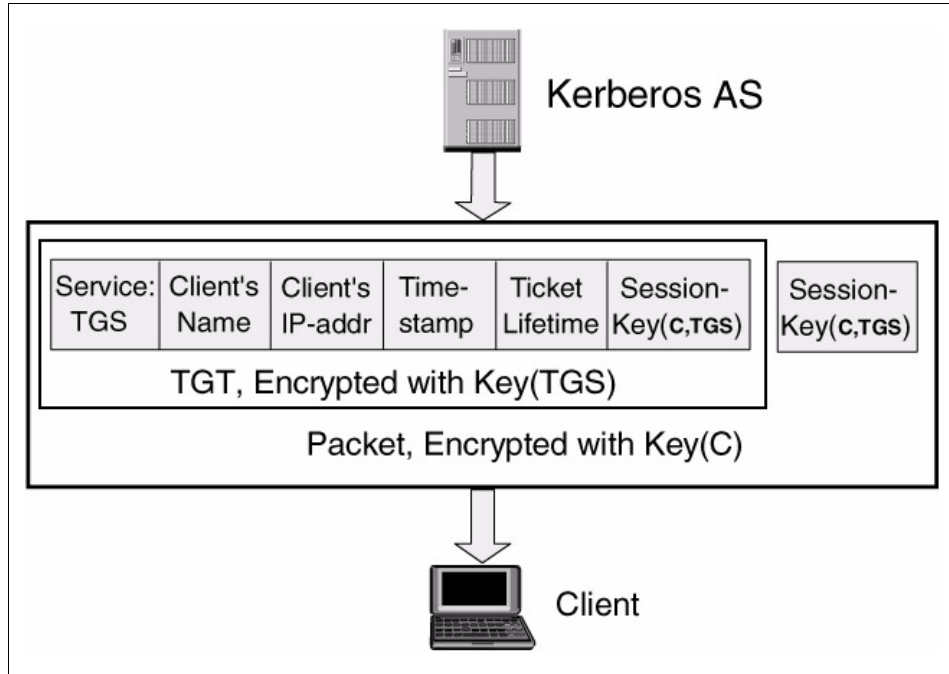


Figure A-2 Authentication server's reply: TGT

Now the sign-on command prompts the user for the password and generates a DES key from the password using the same algorithm as the Kerberos server. It then attempts to decrypt the reply message with that key. If this succeeds, the password has matched the one used to create the user's key in the Kerberos database, and the user has authenticated herself. If the decryption fails, the sign-on is rejected and the reply message is useless. Assuming success, the client now has the encrypted TGT and the session key for use with the TGS, and stores them both in a safe place. Note that the authentication has been done locally on the client machine, and the password has not been transferred over the network.

Authenticating to an application server

If the client now wants to access an application service that requires Kerberos authentication, it must get a service ticket from the TGS. The TGT that was obtained during the Kerberos sign-on can be used to authenticate the client to the TGS; there is no need to type in a password each time a service ticket is requested.

If the client sent only the (encrypted) TGT to the Kerberos TGS, this might be captured and replayed by an intruder who has impersonated the client machine. To protect the protocol against such attacks, the client also generates an authenticator which consists of three parts:

1. The client's (principal's) name
2. The client machine's IP address
3. A timestamp showing when the authenticator was created

The authenticator is encrypted with the session key that the client shares with the TGS. The client then sends a request to the TGS consisting of the name of the service for which a ticket is requested, the encrypted TGT, and the encrypted authenticator. This is shown in Figure A-3.

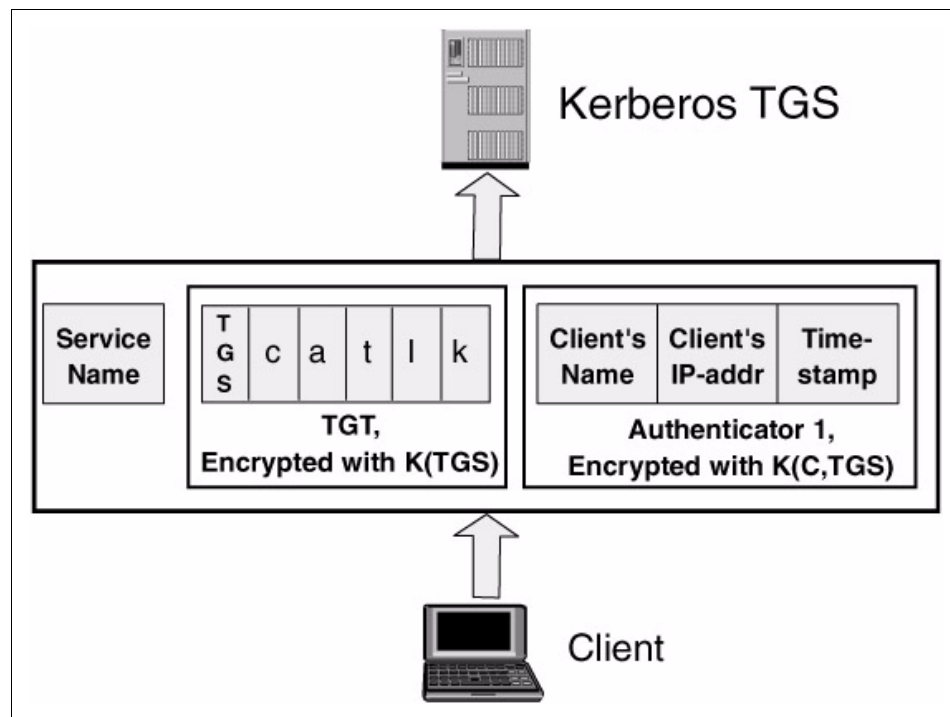


Figure A-3 Client's service ticket request

The TGS can decrypt the TGT because it is encrypted with its own secret key. In that ticket, it finds the session key to share with the client. It uses this session key to decrypt the authenticator, and can then compare the client's name and address in the TGT and the authenticator.

If the timestamp that the TGS finds in the authenticator differs from the current time by more than a prescribed difference (typically 5 minutes), a ticket replay attack is assumed and the request is discarded.

If all checks pass, the TGS generates a service ticket for the service indicated in the client's request. The structure of this service ticket is identical to the TGT described in "Authenticating to the Kerberos server" on page 246. The content differs in the service field (which now indicates the application service rather than the TGS), the timestamp, and the session key. The TGS generates a new, random key that the client and application service will share to encrypt their communications. One copy is put into the service ticket (for the server), and another copy is added to the reply package for the client since the client cannot decrypt the service ticket. The service ticket is encrypted with the secret key of the service, and the whole package is encrypted with the session key that the TGS and the client share. The resulting reply is shown in Figure A-4. Compare this to Figure A-2 on page 247.

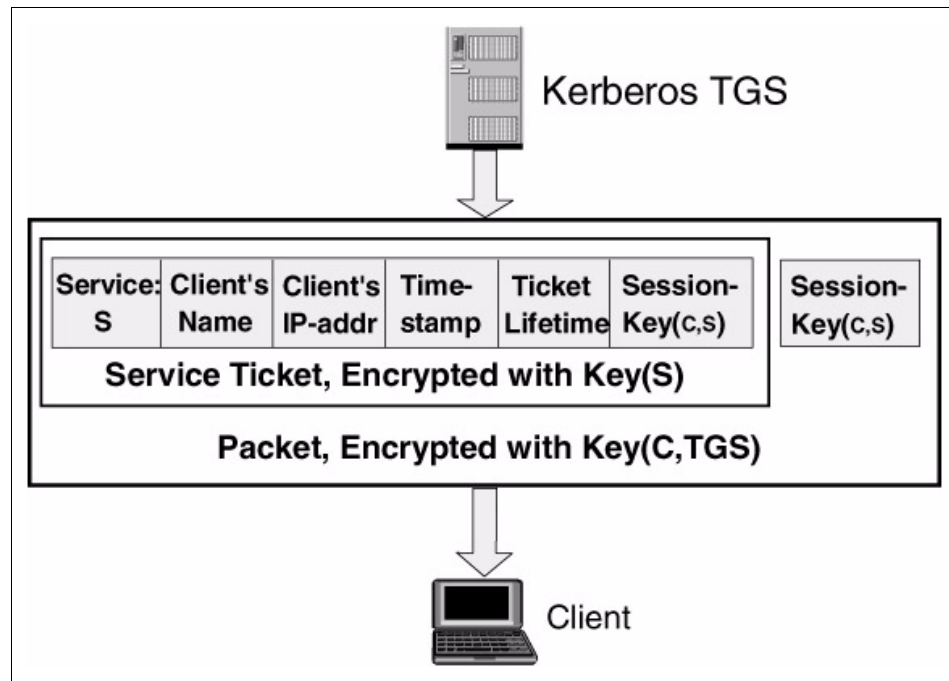


Figure A-4 Ticket-granting service's reply: service ticket

The client can decrypt this message using the session key it shares with the TGS. It then stores the encrypted service ticket and the session key to share with the application server, normally in the same ticket cache where it already has stored the TGT and session key for the TGS.

To actually request the application service, the client sends a request to that server that consists of the name of the requested service, the encrypted service ticket, and a newly generated authenticator to protect this message against replay attacks. The authenticator is encrypted with the session key that the client and the service share. The resulting application service request is shown in Figure A-5. Note the resemblance to the request for ticket-granting service in Figure A-3 on page 248.

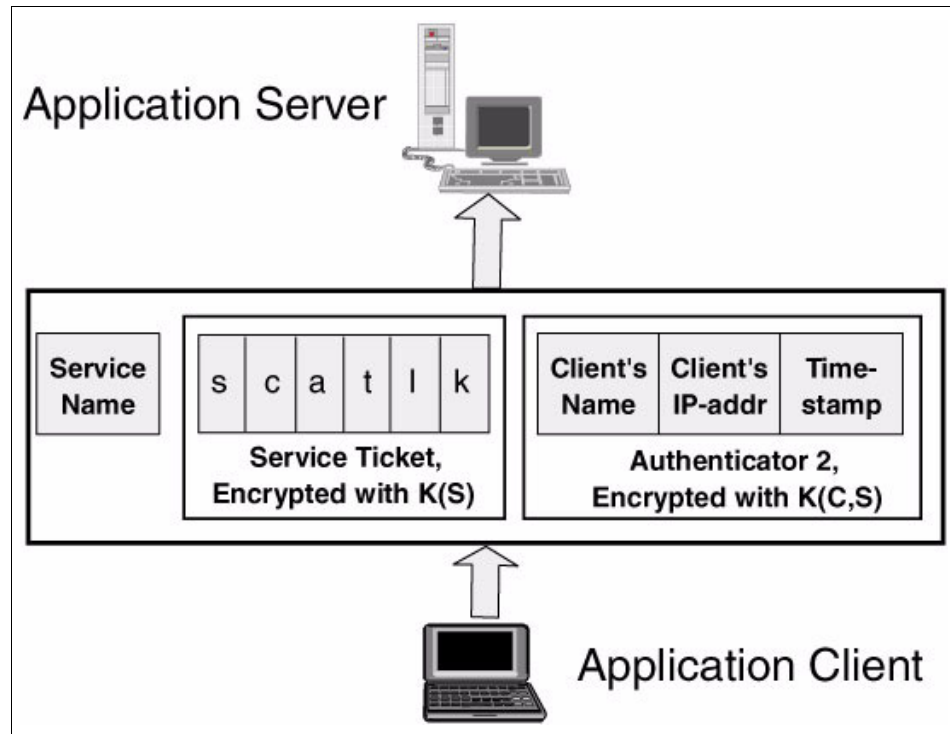


Figure A-5 Client's application service request

The application server decrypts the service ticket with its secret key, uses the enclosed session key to decrypt the authenticator, and checks the user's identity and the authenticator's timestamp. Again, this processing is the same as for the TGS processing the service ticket request. If all checks pass, the server performs the requested service on behalf of the user.

Authorization: Kerberos is only responsible for authenticating the two partners. Any authorization mechanism must be enforced by the application itself.

If the client requires mutual authentication (that is, the service has to prove its identity to the client), the server could send back a message that is encrypted by the session key it shares with the client, and application-dependent contents that the client can verify. Because the service can only know the session key if it was able to decrypt the service ticket, it must have known its secret key and so has proven its identity.

Kerberos terminology

The following terms are used when discussing Kerberos authentication:

Realm	A Kerberos domain that can consist of a number of machines providing authentication services.
Principal	A user or a service that uses authentication services and is identified in the authentication database. For example, <code>root.admin@REALM1.IBM.COM</code> , where <i>root</i> is the user identity and <i>admin</i> the instance.
Instance	<p>In the case of a <i>service instance</i>, it represents the occurrence of the server. Service example: <code>hardmon.sp21cw0</code>, where <i>hardmon</i> represents the service and <i>sp21cw0</i> represents the node providing the service.</p> <p>In the case of a <i>user</i>, the instance represents the Kerberos authority granted to the user. User example: <code>root.admin</code>, where <i>admin</i> represents a Kerberos authorization for administrative tasks in Kerberos.</p>
Authentication database	A set of files containing the definitions of the Kerberos authentication information. The authentication database is maintained at the Kerberos server.
Ticket	An encrypted message containing the identity of a user. A ticket is passed from a client to a server as soon as a Kerberos service is requested. Tickets have a predetermined lifetime and have to be renewed periodically.
Key	An eight-byte form of a user or service password stored in the authentication database. This password is associated with a Kerberos user or service principal, not a user ID.

Ticket-granting ticket

A ticket that is generated by the Kerberos authentication database as proof that Kerberos recognizes the user as an authorized user.

KDC

The trusted third-party or intermediary in Kerberos that issues all Kerberos tickets to the clients.

Where to find more information about Kerberos

Here is a list of references where you can find more information about Kerberos and related products.

IBM Redbooks

(Redbooks are available for purchase or download at <http://www.redbooks.ibm.com>.)

AIX 5L Version 5.2 Security Supplement, SG24-6066

This book includes a chapter titled “Exploiting Network Authentication Service.” The chapter discusses IBM Network Authentication Service Version 1.3 for AIX, which is the IBM AIX implementation of Kerberos.

RS/6000 SP System Management: Easy, Lean and Mean, GG24-2563

This book discusses Kerberos in the context of the IBM Scalable POWERparallel® System. Although the book is becoming dated (published in 1995), it contains useful overview material, and it has a useful step-by-step illustration of the ticket-granting process.

Other IBM publications

AIX 5L Version 5.3 Security Guide, SC23-4907

This manual has a chapter dedicated to Kerberos. The chapter describes how to implement and troubleshoot Kerberos authentication in AIX.

IBM Network Authentication Service Version 1.4 for AIX, Linux, and Solaris Administrator's and User's Guide

This manual describes how to plan for, install, configure, and administer the IBM Network Authentication Service Version 1.4.

Non-IBM publications

Kerberos: The Definitive Guide, by Jason Garman. O'Reilly & Associates, Inc., 2003. ISBN 0596004036

The author's own words describing this book (quoted from the preface): "This book is geared toward the system administrator who wants to establish a single sign-on network using Kerberos. This book is also useful for anyone interested in how Kerberos performs its magic: the first three chapters will be most helpful to these people."

Red Hat Enterprise Linux 3 Reference Guide, Red Hat, Inc., 2003

This book has a chapter dedicated to Kerberos, where it briefly describes Kerberos, discusses advantages and disadvantages, explains how it works, and defines terminology.

The book is available for download from <http://www.redhat.com>.

The following Requests for Comment (RFCs) specify standards for Kerberos implementations. The RFCs are available in text format at <http://www.ietf.org/rfc.html>. They are also available both in text and PDF format at <http://www.faqs.org/rfcs/index.html>.

RFC 1508	<i>Generic Security Service Application Program Interface</i>
RFC 1510	<i>The Kerberos Network Authentication Service (V5)</i>
RFC 1964	<i>The Kerberos Version 5 GSS-API Mechanism</i>

Other information sources

<http://web.mit.edu/kerberos/index.html>

This Web site is the main page for the MIT Kerberos project. It includes documentation and tutorials to provide a better understanding of the protocol and downloadable source code for interested developers.

<http://www.cmf.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html>

This is a comprehensive list of frequently asked questions compiled by Ken Hornstein of the U.S. Naval Research Laboratory. It also contains links to other useful sources of information.



Sample scripts, files, and output

This appendix provides copies of sample scripts from throughout the book.

It contains the following sections:

- ▶ Sample administrative scripts
- ▶ Sample client Kerberos configuration files
- ▶ LDIF sample file for KDC
- ▶ Sample iptrace output

Sample administrative scripts

In this section you will find sample scripts used during our testing and implementation scenarios. If you decide to use the scripts, you should use them only as templates. We do not recommend that you use the scripts in the provided form without first carefully reviewing the code and adding error checking and logging where needed. As provided, there is no form of error checking in the scripts and this may lead to unexpected results.

Change the pseudo-root FS sample script

The script in Example B-1 can be used to set the pseudo-root FS.

Example: B-1 Sample script to set/change the pseudo-root FS

```
#!/bin/ksh

NEWNFSROOT="/exports"
#
exportfs -ua
/etc/nfs.clean
#
chnfs -r ${NEWNFSROOT}
#
lssrc -Ss nfsd | grep ${NEWNFSROOT}
#
nfsd -getnodes
#
/etc/nfs.clean
#
/etc/rc.nfs
#
lssrc -g nfs
#
ps -ef | grep nfsd
#
exportfs
```

Create a KDC server with NFS V4 server

The script in Example B-2 can be used to create and configure your KDC server on the same system as your NFS V4 server.

Example: B-2 Sample script to create a KDC server with legacy database

```
#!/bin/ksh
HOST=$(hostname)
```

```
IREALM="REALM2.IBM.COM"
KDCSERV="nfs403.itsc.austin.ibm.com"
DNSDOM="itsc.austin.ibm.com"
NFSDOM="itsc.austin.ibm.com"
#
HOST="${HOST}.${DNSDOM}"
unset KRB5CCNAME
#
SECRET="succ3ss"
#
#config.krb5 -S -d ${DNSDOM} -r ${IREALM}
mkkrb5srv -r ${IREALM} -d ${DNSDOM} -s ${HOST}
#
kinit admin/admin
klist
#
/mnt/scripts/create_initial_kdc_user.ksh
#
/usr/krb5/sbin/kadmin -p admin/admin -w ${SECRET}<< EOF
add_principal -e des-cbc-crc:normal -randkey nfs/${HOST}
EOF
#
/usr/krb5/sbin/kadmin -p admin/admin -w ${SECRET}<< EOF
ktadd nfs/${HOST}
EOF
#
nfshostkey -p nfs/${HOST} -f /etc/krb5/krb5.keytab
nfshostkey -l
#
chfnfsdom ${NFSDOM}
chfnfsdom
#
chfnfsrtd -a ${IREALM} ${NFSDOM}
chfnfsrtd
#
exportfs -ua
/etc/nfs.clean
chfnfs -r /exports
/etc/nfs.clean
#
chfnfs -S -B
#
cp /mnt/scripts/exports.v4krb5.fullclients /etc/exports
#
/etc/rc.nfs
exportfs -va
```

Create a full client with legacy KDC server backend

This script can be used to create a full client with a legacy KDC server backend.

Example: B-3 Sample script to create a full client with a legacy backend

```
#!/bin/ksh
HOST=$(hostname)
IREALM="REALM2.IBM.COM"
KDCSERV="nfs402.itsc.austin.ibm.com"
DNSDOM="itsc.austin.ibm.com"
NFSDOM="itsc.austin.ibm.com"
#
HOST="${HOST}.${DNSDOM}"
#
SECRET="succ3ss"
#
unset KRB5CCNAME
#
setclock ${KDCSERV}
#
#config.krb5 -C -d $DNSDOM -r $IREALM -c $KDCSERV -s $KDCSERV
mkkrb5clnt -c $KDCSERV -r $IREALM -s $KDCSERV -d $DNSDOM
#
kinit admin/admin
klist
#
/usr/krb5/sbin/kadmin -p admin/admin -w ${SECRET}<< EOF
add_principal -e des-cbc-crc:normal -randkey nfs/${HOST}
EOF
#
/usr/krb5/sbin/kadmin -p admin/admin -w ${SECRET}<< EOF
ktadd nfs/${HOST}
EOF
#
nfshostkey -p nfs/${HOST} -f /etc/krb5/krb5.keytab
nfshostkey -l
#
chnfsdom ${NFSDOM}
chnfsdom
#
chnfsrtd -a ${IREALM} ${NFSDOM}
chnfsrtd
#
mkdir /nfs
#
chnfs -S -B
#
/etc/rc.nfs
```

Create a Full Client with KDC and LDAP backend

The script in Example B-4 performs all operations to configure an NFS V4 full Kerberos client with integrated login.

Example: B-4 Sample script to create a full client with KDC and LDAP backend

```
#!/bin/ksh
HOST=$(hostname)
IREALM="REALM1.IBM.COM"
KDCSERV="nfs407.itsc.austin.ibm.com"
LDAPSERV="nfs407.itsc.austin.ibm.com"
DNSDOM="itsc.austin.ibm.com"
NFSDOM="itsc.austin.ibm.com"
#
HOST="${HOST}.${DNSDOM}"
#
SECRET="succ3ss"
ADMINDN="cn=admin"
#
unset KRB5CCNAME
#
#synchronize the system time with the KDC Server
setclock ${KDCSERV}
#
mkkrb5clnt -c $KDCSERV -r $IREALM -s $KDCSERV -d $DNSDOM -l $LDAPSERV -i files
-A -K -T
#
mksecldap -c -h $LDAPSERV -a ${ADMINDN} -p ${SECRET}
#
/usr/sbin/ls-secldapclntd
#
echo "file /usr/lib/security/methods.cfg has to edited"
if grep -p ^KRB5LDAP /usr/lib/security/methods.cfg
then
  echo "file /usr/lib/security/methods.cfg contains KRB5LDAP no changes will
occurr"
else
  echo "file /usr/lib/security/methods.cfg need to be changed"
  cp /usr/lib/security/methods.cfg /usr/lib/security/methods.cfg.save
  echo "\nKRB5LDAP:\n\toptions = db=LDAP,auth=KRB5" >>
/usr/lib/security/methods.cfg
fi
#
chsec -f /etc/security/user -s default -a registry=KRB5LDAP
chsec -f /etc/security/user -s default -a "SYSTEM=\"KRB5LDAP OR compat\""
#
chuser registry=files root
chuser SYSTEM="compat" root
```

```

#
kinit admin/admin
klist
#
/usr/krb5/sbin/kadmin -p admin/admin -w ${SECRET}<< EOF
add_principal -e des-cbc-crc:normal -randkey nfs/${HOST}
EOF
#
/usr/krb5/sbin/kadmin -p admin/admin -w ${SECRET}<< EOF
ktadd nfs/${HOST}
EOF
#
nfshostkey -p nfs/${HOST} -f /etc/krb5/krb5.keytab
nfshostkey -l
#
kdestroy
kinit -kt /etc/krb5/krb5.keytab nfs/${HOST}
#
chnfsdom ${NFSDOM}
chnfsdom
#
chnfsrtd -a ${IREALM} ${NFSDOM}
chnfsrtd
#
mkdir /nfs
#
chnfs -S -B
#
/etc/rc.nfs

```

Script to copy ACLs to an entire directory structure

The script in Example B-5 copies an ACL from a source file or directory to a destination file or directory. If the `-r` option is specified and the destination is a directory, the ACL is also copied to all files or subdirectories underneath the destination directory.

Example: B-5 Sample script for copying an ACL (with recursive option)

```

#!/usr/bin/ksh
#
# copy_acl.sh
#
# Copy the ACL for the given source file/directory to other files/directories
#

# Name of this script
scrname=${0##*/}

```

```

#
# Functions
#

function usage {
    echo "Usage: $scrname [-r] <source> <dest>"
    echo "  where"
    echo "    -r indicates a recursive copy"
    echo "      (copy ACL to all files and directories below and including"
    echo "        the destination.)"
    echo "  <source> = the name of the file or directory to copy the ACL from"
    echo "  <dest>   = the name of the file or directory to copy the ACL to"

    exit 1
}

if [[ $# -eq 0 ]]
then
    usage
fi

#
# Process input parameters
#

if [[ "$1" = "-r" ]]; then
    SETSUBTREE="true"
    shift
else
    SETSUBTREE="false"
fi

if [[ -n "$1" ]]; then
    SRC_NAME="$1"
else
    usage
fi

if [[ -n "$2" ]]; then
    DEST_NAME="$2"
else
    usage
fi

#
# Initialize other variables
#

```

```

NBERR=0
TMP_ACLFILE="/tmp/.AIXACL_$$"

if [[ -e "${SRC_NAME}" ]]; then
    aclget -o "${TMP_ACLFILE}" "${SRC_NAME}"
    NBERR=$?
else
    echo "Source \"${SRC_NAME}\" does not exist"
    NBERR=1
fi

if [[ "${NBERR}" -eq 0 ]]; then
    if [[ -e "${DEST_NAME}" ]]; then
        if [[ -d "${DEST_NAME}" && "${SETSUBTREE}" = "true" ]]; then
            find "${DEST_NAME}" -print | while read NAME
            do
                aclput -i "${TMP_ACLFILE}" "${NAME}"
                (( NBERR += $? ))
                ls -dl "${NAME}"
            done
        else
            aclput -i "${TMP_ACLFILE}" "${DEST_NAME}"
            (( NBERR += $? ))
            ls -dl "${DEST_NAME}"
        fi
    else
        echo "Destination \"${DEST_NAME}\" does not exist"
        NBERR=1
    fi
fi

rm -f "${TMP_ACLFILE}"
exit ${NBERR}

```

Windows command script to run ktpass

The Windows command script in Example B-6 was used to add the NFS service principals for the created Windows Active Directory users.

Example: B-6 Sample ktpass_WinKDC command script

```

C:\PROGRA~1\SUPPOR~1\ktpass.exe -princ nfs/nfs403@KDC.ITSC.AUSTIN.IBM.COM
-mapuser nfs403 -pass nfs403 -out c:\misc\nfs403.keytab
C:\PROGRA~1\SUPPOR~1\ktpass.exe -princ nfs/nfs405@KDC.ITSC.AUSTIN.IBM.COM
-mapuser nfs405 -pass nfs405 -out c:\misc\nfs405.keytab

```

Script to gather additional information for local AIX software support

The script in Example B-7 gathers additional information that the AIX `snap` command does not gather. It is not intended as a substitute for the `snap` command but should be used as an addition to the `snap` command and provided to your local AIX Support Center.

Example: B-7 nfs_pd.script to gather additional information for IBM AIX support

```
#!/bin/ksh
# *****README*****
# SPECIAL NOTICES
#
# Information in this document is correct to the best of our
# knowledge at the time of this writing.
# Please use this information with care. IBM will not be
# responsible for damages of any kind resulting from its use.
# The use of this information is the sole responsibility of
# the customer and depends on the customer's ability to eval-
# uate and integrate this information into the customer's
# operational environment.
#
# This script will gather information about your networking environment
# so that IBM may attempt to determine why your computer is experiencing
# problems. You will need to run this script as the root user. If you
# do not have enough space in the /tmp filesystem, you may need to increase
# the size of this filesystem.
# In order to run the script, you will need to give it execute permissions.
# In order to do this, make sure your working directory is the directory
# where the script is located. Then issue -
# chmod +x nfs.script
#
# To run the script from this directory, type -
# ./nfs_pd.script
# *****
#
# In addition to this script, you may be asked to supply an iptrace with
# your testcase. If you are asked to do so, the syntax is as follows -
# startsrc -s iptrace -a "-a /tmp/iptrace.bin"
#
# When you wish to stop the trace, issue -
# stopsrc -s iptrace
# You can tar this file along with the rest of the test case and upload it
# following the instructions that the script outlines.
#
# *****

clear
echo ""
```

```

echo "*****WELCOME*****"
echo ""
echo "*****EXECUTING NFS SCRIPT*****"
echo ""
echo "This script will generate a TESTCASE which should be sent to IBM"
echo "to be analyzed by AIX Software Support."
echo ""
echo "Please enter your 5 digit PMR number: "
read pmr
echo "Please enter you 3 digit branch number: "
read branch
echo "Please enter you 3 digit country code: "
read country
echo ""
echo "*****Thank You*****"
echo ""
echo "*****Gathering TESTCASE...Please Standby...*****"
echo ""

if [ -d /tmp/ibm ] ; then
echo ""
echo "Unable to create the /tmp/ibm directory because it already exists."
echo ""
echo "Exiting the script."
echo ""
    exit
else
    mkdir /tmp/ibm
fi

/usr/bin/date > /tmp/ibm/date.start
/usr/bin/uptime > /tmp/ibm/uptime.out
/usr/bin/uname -M > /tmp/ibm/uname-M.out
/usr/bin/uname -n > /tmp/ibm/uname-n.out
/usr/bin/hostname > /tmp/ibm/hostname.out
/usr/sbin/bootinfo -K > /tmp/ibm/bootinfo-K.out

echo "Gathering config files ..."
/usr/bin/cp /etc/inetd.conf /tmp/ibm/inetd.conf
/usr/bin/cp /etc/services /tmp/ibm/services
/usr/bin/cp /etc/inittab /tmp/ibm/inittab
/usr/bin/cp /etc/hosts /tmp/ibm/hosts
/usr/bin/cp /etc/filesystems /tmp/ibm/filesystems

if [ -f /etc/auto* ]; then
    /usr/bin/cp /etc/auto* /tmp/ibm/
fi

if [ -f /etc/exports ] ; then

```

```

    /usr/bin/cp /etc/exports /tmp/ibm/exports
fi

if [ -f /etc/xtab ] ; then
    /usr/bin/cp /etc/xtab /tmp/ibm/xtab
fi

if [ -f /etc/rmtab ] ; then
    /usr/bin/cp /etc/rmtab /tmp/ibm/rmtab
fi

if [ -f /etc/netsvc.conf ] ; then
    /usr/bin/cp /etc/netsvc.conf /tmp/ibm/netsvc.conf
fi

if [ -f /etc/resolv.conf ] ; then
    /usr/bin/cp /etc/resolv.conf /tmp/ibm/resolv.conf
fi

if [ -f /etc/irs.conf ] ; then
    /usr/bin/cp /etc/irs.conf /tmp/ibm/irs.conf
fi

if [ -f /etc/rc.nfs ] ; then
    /usr/bin/cp /etc/rc.nfs /tmp/ibm/rc.nfs
fi

if [ -f /etc/nfs/hostkey ] ; then
    /usr/bin/cp /etc/nfs/hostkey /tmp/ibm/hostkey
fi

if [ -f /etc/nfs/local_domain ] ; then
    /usr/bin/cp /etc/nfs/local_domain /tmp/ibm/local_domain
fi

if [ -f /etc/nfs/realm.map ] ; then
    /usr/bin/cp /etc/nfs/realm.map /tmp/ibm/realm.map
fi

if [ -f /etc/nfs/princmap ] ; then
    /usr/bin/cp /etc/nfs/princmap /tmp/ibm/princmap
fi

if [ -f /etc/nfs/security_default ] ; then
    /usr/bin/cp /etc/nfs/security_default /tmp/ibm/security_default
fi

echo "Gathering TCP related command outputs ..."
/usr/bin/netstat -v > /tmp/ibm/netstat-v.out

```

```

/usr/bin/netstat -in > /tmp/ibm/netstat-in.out
/usr/bin/netstat -rn > /tmp/ibm/netstat-rn.out
/usr/bin/netstat -an > /tmp/ibm/netstat-an.out
/usr/bin/netstat -s > /tmp/ibm/netstat-s.out
/usr/bin/netstat -m > /tmp/ibm/netstat-m.out

echo "Gathering OS related command output ..."
/usr/sbin/instfix -i > /tmp/ibm/instfix.out 2>&1
grep AIX /tmp/ibm/instfix.out > /tmp/ibm/aix_m1
/usr/sbin/lssattr -El sys0 > /tmp/ibm/sys0.out
/usr/sbin/lssattr -El inet0 > /tmp/ibm/inet0.out
/usr/sbin/lssattr -El aio0 > /tmp/ibm/aio0.out
/usr/bin/lssrc -ls inetd > /tmp/ibm/inetd.out
/usr/bin/lssrc -a > /tmp/ibm/lssrc-a.out
/usr/bin/lssrc -g tcpip > /tmp/ibm/lssrc-tcp.status
/usr/bin/lssrc -g nfs > /tmp/ibm/lssrc-nfs.status
/usr/bin/lslpp -h > /tmp/ibm/lslpp-h.out
/usr/bin/lppchk -v > /tmp/ibm/lppchk-v.out
/usr/bin/lppchk -c bos.net.* > /tmp/ibm/lppchk-c.out
/usr/sbin/scls -l > /tmp/ibm/scls-l.out
/usr/bin/ls -l /etc/*.conf > /tmp/ibm/conf.out
/usr/bin/what /usr/lib/drivers/netinet > /tmp/ibm/what-netinet
/usr/bin/cp /etc/trcfmt /tmp/ibm/trcfmt
/usr/bin/trcnm -a > /tmp/ibm/namelist

/usr/bin/errpt -a > /tmp/ibm/errpt-a.out
/usr/bin/ps -ef > /tmp/ibm/ps-ef.out
/usr/bin/ps aux > /tmp/ibm/psaux.out
/usr/bin/env > /tmp/ibm/env.out
/usr/bin/env | grep -i krb > /tmp/ibm/env_krb.out
/usr/bin/df -k > /tmp/ibm/df-k.out
/usr/sbin/no -a > /tmp/ibm/no.out
/usr/sbin/nfso -a > /tmp/ibm/nfso.out
/usr/bin/rpcinfo -p > /tmp/ibm/rpcinfo.out
/usr/sbin/arp -an > /tmp/ibm/arp.out
/usr/sbin/lssdev -Cc adapter > /tmp/ibm/adapter.out
/usr/sbin/lssdev -Cc if > /tmp/ibm/lssdev-if.out
/usr/sbin/lscfg -v > /tmp/ibm/lscfg-v.out

echo "Gathering NFS related command output ..."
/usr/bin/showmount -e > /tmp/ibm/showmount-e.out
/usr/sbin/mount > /tmp/ibm/mount.out
/usr/sbin/nfsstat -m > /tmp/ibm/nfsstat-m.out
/usr/sbin/nfsstat -cr > /tmp/ibm/nfsstat-cr.out
/usr/sbin/nfs4cl showfs > /tmp/ibm/nfs4cl_showfs.out
/usr/sbin/nfs4cl showstat > /tmp/ibm/nfs4cl_showstat.out
/usr/sbin/chnfsdom > /tmp/ibm/chnfsdom.out

/usr/bin/netstat -s > /tmp/ibm/netstat-s.out2

```

```

/usr/bin/netstat -an > /tmp/ibm/netstat-an.out2

if [ -f `which lsof` ] ; then
    lsof > /tmp/ibm/lsof.out
fi

ADAPTERLIST="ent tok atm mpc fddi escon cat"
for i in $ADAPTERLIST
do
    for j in `usr/sbin/lscfg -Cc adapter|grep "^$i"|awk '{print $1}'`
    do
        /usr/sbin/lssattr -El $j > /tmp/ibm/lssattr-El.$j.out
        /usr/sbin/lscfg -v $j > /tmp/ibm/lscfg-v.$j.out
    done
done

for i in `usr/bin/netstat -in|grep -Ev ">:::1|Name|ink"|awk '{print $1}'`
do
    /etc/ifconfig $i > /tmp/ibm/ifconfig.$i.out
    /usr/sbin/lssattr -El $i > /tmp/ibm/lssattr-El.$i.out
done

/usr/bin/netstat -v > /tmp/ibm/netstat-v.out2
/usr/bin/netstat -in > /tmp/ibm/netstat-in.out2
/usr/bin/netstat -an > /tmp/ibm/netstat-an.out3
/usr/bin/netstat -s > /tmp/ibm/netstat-s.out3
/usr/bin/netstat -m > /tmp/ibm/netstat-m.out2
/usr/bin/netstat -v > /tmp/ibm/netstat-v.out3
/usr/bin/netstat -in > /tmp/ibm/netstat-in.out3
/usr/bin/netstat -an > /tmp/ibm/netstat-an.out4
/usr/bin/netstat -s > /tmp/ibm/netstat-s.out4
/usr/bin/netstat -m > /tmp/ibm/netstat-m.out3
/usr/sbin/nfs4cl showfs > /tmp/ibm/nfs4cl_showfs.out2
/usr/sbin/nfs4cl showstat > /tmp/ibm/nfs4cl_showstat.out2

/usr/bin/ps -ef > /tmp/ibm/ps-ef.out2
/usr/bin/ps aux > /tmp/ibm/psaux.out2

/usr/bin/date > /tmp/ibm/date.stop
cd /tmp/ibm
/usr/bin/tar -cf /tmp/$pmr.$branch.$country.tar *
cd /tmp
/usr/bin/compress $pmr.$branch.$country.tar
/usr/bin/rm -rf /tmp/ibm

clear
echo ""
echo "The script has completed. "
echo "There should be a "$pmr.$branch.$country".tar.Z file in your "

```

```

echo "/tmp directory. Please ftp this file to your local IBM Support "
echo "Centre - your local IBM Support representative should be able to "
echo "provide the relevant details."
echo ""
echo "*****Thank You*****"
echo ""
echo "Please be sure to use the bin mode within FTP, otherwise "
echo "the tar file may be corrupted."
echo ""
echo "*****Thank You For Using IBM AIX Software Support*****"
echo ""

```

Sample client Kerberos configuration files

This section provides the Kerberos configuration files from our test environment.

Kerberos configuration file `/etc/krb5/krb5.conf` with legacy backend

This is the sample configuration file used on clients with a legacy KDC database.

Example: B-8 Sample `/etc/krb5/krb5.conf` file with a legacy KDC database

```

[libdefaults]
    default_realm = REALM2.IBM.COM
    default_keytab_name = FILE:/etc/krb5/krb5.keytab
    default_tkt_enctypes = des3-cbc-sha1 arcfour-hmac aes256-cts
des-cbc-md5 des-cbc-crc
    default_tgs_enctypes = des3-cbc-sha1 arcfour-hmac aes256-cts
des-cbc-md5 des-cbc-crc

[realms]
    REALM2.IBM.COM = {
        kdc = nfs402.itsc.austin.ibm.com:88
        admin_server = nfs402.itsc.austin.ibm.com:749
        default_domain = ibm.com
    }

[domain_realm]
    .ibm.com = REALM2.IBM.COM
    nfs402.itsc.austin.ibm.com = REALM2.IBM.COM

[logging]
    kdc = FILE:/var/krb5/log/krb5kdc.log
    admin_server = FILE:/var/krb5/log/kadmin.log
    default = FILE:/var/krb5/log/krb5lib.log

```

Kerberos configuration file /etc/krb5/krb5.conf with LDAP backend

Example B-9 shows the sample configuration file used on the clients with Kerberos LDAP backend.

Example: B-9 Sample /etc/krb5/krb5.conf with KRB5 and LDAP backend

```
[libdefaults]
    default_realm = REALM1.IBM.COM
    default_keytab_name = FILE:/etc/krb5/krb5.keytab
    default_tkt_enctypes = des3-cbc-sha1 arcfour-hmac aes256-cts
des-cbc-md5 des-cbc-crc
    default_tgs_enctypes = des3-cbc-sha1 arcfour-hmac aes256-cts
des-cbc-md5 des-cbc-crc
    use_ldap_lookup = 1
    ldap_server = nfs407.itsc.austin.ibm.com

[realms]
    REALM1.IBM.COM = {
        kdc = nfs407.itsc.austin.ibm.com:88
        admin_server = nfs407.itsc.austin.ibm.com:749
        default_domain = itsc.austin.ibm.com
    }

[domain_realm]
    .itsc.austin.ibm.com = REALM1.IBM.COM
    nfs407.itsc.austin.ibm.com = REALM1.IBM.COM

[logging]
    kdc = FILE:/var/krb5/log/krb5kdc.log
    admin_server = FILE:/var/krb5/log/kadmin.log
    default = FILE:/var/krb5/log/krb5lib.log
```

Kerberos configuration file /etc/krb5/krb5.conf with Windows Active Directory backend

Example B-10 shows the sample configuration file used on clients with a Microsoft Windows Active Directory backend.

Example: B-10 Sample /etc/krb5/krb5.conf with Windows Active Directory backend

```
[libdefaults]
    default_realm = KDC.ITSC.AUSTIN.IBM.COM
    default_keytab_name = FILE:/etc/krb5/krb5.keytab
    default_tkt_enctypes = des-cbc-md5 des-cbc-crc
    default_tgs_enctypes = des-cbc-md5 des-cbc-crc
```

```

[realms]
    KDC.ITSC.AUSTIN.IBM.COM = {
        kdc = nfs409.kdc.itsc.austin.ibm.com:88
        admin_server = nfs409.kdc.itsc.austin.ibm.com:749
        default_domain = kdc.itsc.austin.ibm.com
    }

[domain_realm]
    .kdc.itsc.austin.ibm.com = KDC.ITSC.AUSTIN.IBM.COM
    nfs409.kdc.itsc.austin.ibm.com = KDC.ITSC.AUSTIN.IBM.COM

[logging]
    kdc = FILE:/var/krb5/log/krb5kdc.log
    admin_server = FILE:/var/krb5/log/kadmin.log
    default = FILE:/var/krb5/log/krb5lib.log

```

LDIF sample file for KDC

Example B-11 shows a sample LDIF file for the KDC LDAP container to work with integrated logon into AIX. It is used in 5.9, “Preparing the system for Tivoli Directory Server and Kerberos V5” on page 155.

Example: B-11 Sample realm_add_ibm.ldif file

```

# The suffix “ou=Austin, o=IBM, c=US” should be defined before attempting to
# load this data. Or change the suffix to be an already defined object.
# Change all references of YOURHOSTNAME.AUSTIN.IBM.COM to be your realm name
#
# version: 1

dn: o=IBM, c=US
objectclass: top
objectclass: organization
o: IBM

dn: krbrealmName-V2=REALM2.IBM.COM, o=IBM, c=US
objectclass: KrbRealm-V2
objectclass: KrbRealmExt
krbrealmName-V2: REALM2.IBM.COM
krbprincSubtree: krbrealmName-V2=REALM2.IBM.COM, o=IBM, c=US
krbDeleteType: 3

dn: cn=principal, krbrealmName-V2=REALM2.IBM.COM, o=IBM, c=US
objectclass: container
cn: principal

```



```
dn: cn=policy, krbrealmName-V2=REALM2.IBM.COM, o=IBM, c=US
objectclass: container
cn: policy
```

Sample iptrace output

In this section we provide some sample iptrace output taken while setting up and debugging our test environment.

Successful authentication during mount request

Example B-12 shows the iptrace taken during a successful authentication while carrying out an NFS V4 mount. We outlined the three major packages, which are:

- ▶ Kerberos ticket request KRB5 TGS-REQ
- ▶ Valid Kerberos ticket reply KRB5 TGS-REP
- ▶ The NFS V4 compound call

Example: B-12 Sample iptrace output showing successful authentication during mount

No.	Time	Source	Destination	Protocol Info
167	20.373421	9.3.5.173	9.3.4.71	KRB5

TGS-REQ

Frame 167 (1311 bytes on wire, 1311 bytes captured)

Arrival Time: Aug 6, 2004 15:30:07.127965000

Time delta from previous packet: 0.000276000 seconds

Time since reference or first frame: 20.373421000 seconds

Frame Number: 167

Packet Length: 1311 bytes

Capture Length: 1311 bytes

Ethernet II, Src: 00:02:55:af:1c:8d, Dst: 00:09:12:48:3c:02

Destination: 00:09:12:48:3c:02 (Cisco_48:3c:02)

Source: 00:02:55:af:1c:8d (Ibm_af:1c:8d)

Type: IP (0x0800)

Internet Protocol, Src Addr: 9.3.5.173 (9.3.5.173), Dst Addr: 9.3.4.71 (9.3.4.71)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

0000 00.. = Differentiated Services Codepoint: Default (0x00)

.... ..0. = ECN-Capable Transport (ECT): 0

.... ...0 = ECN-CE: 0

Total Length: 1297

Identification: 0x0586 (1414)

Flags: 0x00

```

    0... = Reserved bit: Not set
    .0.. = Don't fragment: Not set
    ..0. = More fragments: Not set
Fragment offset: 0
Time to live: 30
Protocol: UDP (0x11)
Header checksum: 0x765d (correct)
Source: 9.3.5.173 (9.3.5.173)
Destination: 9.3.4.71 (9.3.4.71)
User Datagram Protocol, Src Port: 32925 (32925), Dst Port: kerberos (88)
Source port: 32925 (32925)
Destination port: kerberos (88)
Length: 1277
Checksum: 0xc985 (correct)
Kerberos TGS-REQ
Pvno: 5
MSG Type: TGS-REQ (12)
padata: PA-TGS-REQ
  Type: PA-TGS-REQ (1)
    Value: 6E82045F3082045BA003020105A10302... AP-REQ
      Pvno: 5
      MSG Type: AP-REQ (14)
      Padding: 0
      APOptions: 00000000
        .0.. .... .... .... .... .... .... = Use Session Key:
Do NOT use the session key to encrypt the ticket
        ..0. .... .... .... .... .... .... = Mutual required:
Mutual authentication is NOT required
      Ticket
        Tkt-vno: 5
        Realm: KDC.ITSC.AUSTIN.IBM.COM
        Server Name (Unknown): krbtgt KDC.ITSC.AUSTIN.IBM.COM
          Name-type: Unknown (0)
          Name: krbtgt
          Name: KDC.ITSC.AUSTIN.IBM.COM
        enc-part rc4-hmac
          Encryption type: rc4-hmac (23)
          Kvno: 2
          enc-part: 68A3BA1DDDD3C5E05848A0D3BA2C1F05...
        Authenticator des-cbc-md5
          Encryption type: des-cbc-md5 (3)
          Authenticator data: 7333893FE36467A6E3AB514BE8740E20...
    KDC_REQ_BODY
      Padding: 0
      KDCOptions: 00800000 (Renewable)
        .0.. .... .... .... .... .... .... = Forwardable: Do NOT use
forwardable tickets
        ..0. .... .... .... .... .... .... = Forwarded: This is NOT a
forwarded ticket

```

```

...0 ..... = Proxyable: Do NOT use
proxyable tickets
.... 0.... = Proxy: This ticket has
NOT been proxied
.... .0.. = Allow Postdate: We do NOT
allow the ticket to be postdated
.... ..0. = Postdated: This ticket is
NOT postdated
.... .... 1... = Renewable: This ticket is
RENEWABLE
.... .... ...0 = Opt HW Auth: False
.... .... ....0 = Canonicalize: This is NOT
a canonicalized ticket request
.... .... .... ..0. = Disable Transited Check:
Transited checking is NOT disabled
.... .... .... ...0 = Renewable OK: We do NOT
accept renewed tickets
.... .... .... 0... = Enc-Tkt-in-Skey: Do NOT
encrypt the tkt inside the skey
.... .... .... ..0. = Renew: This is NOT a
request to renew a ticket
.... .... .... ...0 = Validate: This is NOT a
request to validate a postdated ticket
Realm: KDC.ITSC.AUSTIN.IBM.COM
Server Name (Principal): nfs nfs403
Name-type: Principal (1)
Name: nfs
Name: nfs403
till: 2004-08-07 06:31:00 (Z)
Nonce: 1091824267
Encryption Types: des-cbc-md5 des-cbc-crc
Encryption type: des-cbc-md5 (3)
Encryption type: des-cbc-crc (1)

```

No.	Time	Source	Destination	Protocol Info
168	20.375701	9.3.4.71	9.3.5.173	KRB5

TGS-REP

```

Frame 168 (1282 bytes on wire, 1282 bytes captured)
Arrival Time: Aug 6, 2004 15:30:07.130245000
Time delta from previous packet: 0.002280000 seconds
Time since reference or first frame: 20.375701000 seconds
Frame Number: 168
Packet Length: 1282 bytes
Capture Length: 1282 bytes
Ethernet II, Src: 00:09:12:48:3c:02, Dst: 00:02:55:af:1c:8d
Destination: 00:02:55:af:1c:8d (Ibm_af:1c:8d)
Source: 00:09:12:48:3c:02 (Cisco_48:3c:02)
Type: IP (0x0800)

```

```

Internet Protocol, Src Addr: 9.3.4.71 (9.3.4.71), Dst Addr: 9.3.5.173
(9.3.5.173)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    0000 00.. = Differentiated Services Codepoint: Default (0x00)
    .... ..0. = ECN-Capable Transport (ECT): 0
    .... ...0 = ECN-CE: 0
  Total Length: 1268
  Identification: 0xb478 (46200)
  Flags: 0x00
    0... = Reserved bit: Not set
    .0.. = Don't fragment: Not set
    ..0. = More fragments: Not set
  Fragment offset: 0
  Time to live: 127
  Protocol: UDP (0x11)
  Header checksum: 0x6687 (correct)
  Source: 9.3.4.71 (9.3.4.71)
  Destination: 9.3.5.173 (9.3.5.173)
User Datagram Protocol, Src Port: kerberos (88), Dst Port: 32925 (32925)
  Source port: kerberos (88)
  Destination port: 32925 (32925)
  Length: 1248
  Checksum: 0x5cb7 (correct)

```

Kerberos TGS-REP

```

Pvno: 5
MSG Type: TGS-REP (13)
Client Realm: KDC.ITSC.AUSTIN.IBM.COM
Client Name (Principal): nfs nfs403
  Name-type: Principal (1)
  Name: nfs
  Name: nfs403
Ticket
  Tkt-vno: 5
  Realm: KDC.ITSC.AUSTIN.IBM.COM
  Server Name (Principal): nfs nfs403
    Name-type: Principal (1)
    Name: nfs
    Name: nfs403
  enc-part des-cbc-md5
    Encryption type: des-cbc-md5 (3)
    Kvno: 5
    enc-part: 2F97563D681D6BB1DC8B010025F82DB9...
  enc-part des-cbc-md5
    Encryption type: des-cbc-md5 (3)
    enc-part: 76C193F0DB908DB403831A7B45167357...

```

No.	Time	Source	Destination	Protocol	Info
173	20.385891	9.3.5.173	9.3.5.173	NFS	V4

NULL Call (Reply In 175)

```

Frame 173 (1288 bytes on wire, 1288 bytes captured)
  Arrival Time: Aug 6, 2004 15:30:07.140435000
  Time delta from previous packet: 0.000048000 seconds
  Time since reference or first frame: 20.385891000 seconds
  Frame Number: 173
  Packet Length: 1288 bytes
  Capture Length: 1288 bytes
Raw packet data
  No link information available
Internet Protocol, Src Addr: 9.3.5.173 (9.3.5.173), Dst Addr: 9.3.5.173 (9.3.5.173)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    0000 00.. = Differentiated Services Codepoint: Default (0x00)
      .... ..0. = ECN-Capable Transport (ECT): 0
      .... ...0 = ECN-CE: 0
  Total Length: 1288
  Identification: 0x058c (1420)
  Flags: 0x04 (Don't Fragment)
    0... = Reserved bit: Not set
    .1.. = Don't fragment: Set
    ..0. = More fragments: Not set
  Fragment offset: 0
  Time to live: 60
  Protocol: TCP (0x06)
  Header checksum: 0x0000 (incorrect, should be 0x1705)
  Source: 9.3.5.173 (9.3.5.173)
  Destination: 9.3.5.173 (9.3.5.173)
Transmission Control Protocol, Src Port: 32775 (32775), Dst Port: 2049 (2049),
Seq: 1, Ack: 1, Len: 1236
  Source port: 32775 (32775)
  Destination port: 2049 (2049)
  Sequence number: 1 (relative sequence number)
  Next sequence number: 1237 (relative sequence number)
  Acknowledgement number: 1 (relative ack number)
  Header length: 32 bytes
  Flags: 0x0018 (PSH, ACK)
    0... .... = Congestion Window Reduced (CWR): Not set
    .0.. .... = ECN-Echo: Not set
    ..0. .... = Urgent: Not set
    ...1 .... = Acknowledgment: Set
    .... 1... = Push: Set
    .... .0.. = Reset: Not set
    .... ..0. = Syn: Not set

```

```

.... ...0 = Fin: Not set
Window size: 262140
Checksum: 0xa5d8 (correct)
Options: (12 bytes)
  NOP
  NOP
  Time stamp: tsval 1091824482, tsecr 1091824482
Remote Procedure Call, Type:Call, XID:0x77709946
  Fragment header: Last fragment, 1232 bytes
    1... .... = Last Fragment: Yes
    .000 0000 0000 0000 0000 0100 1101 0000 = Fragment Length: 1232
XID: 0x77709946 (2003867974)
Message Type: Call (0)
RPC Version: 2
Program: NFS (100003)
Program Version: 4
Procedure: NULL (0)
The reply to this request is in frame 175
Credentials
  Flavor: RPCSEC_GSS (6)
  Length: 20
  GSS Version: 1
  GSS Procedure: RPCSEC_GSS_INIT (1)
  GSS Sequence Number: 1
  GSS Service: rpcsec_gss_svc_none (1)
  GSS Context: <EMPTY>
    length: 0
    contents: <EMPTY>
Verifier
  Flavor: AUTH_NULL (0)
  Length: 0
Network File System
  Program Version: 4
  V4 Procedure: NULL (0)
  GSS Token
    GSS Token Length: 1165
    GSS-API
      OID: 1.2.840.113554.1.2.2 (iso.2.840.113554.1.2.2) (KRB5 - Kerberos
5)
      krb5_blob: 01006E82047830820474A003020105A1...
      krb5_tok_id: KRB5_AP_REQ (0x0001)
      Kerberos AP-REQ
        Pvno: 5
        MSG Type: AP-REQ (14)
        Padding: 0
        APOptions: 20000000 (Mutual required)
        .0.. .... = Use Session
Key: Do NOT use the session key to encrypt the ticket

```

```

..1. .... = Mutual
required: MUTUAL authentication is REQUIRED
Ticket
  Tkt-vno: 5
  Realm: KDC.ITSC.AUSTIN.IBM.COM
  Server Name (Principal): nfs nfs403
  Name-type: Principal (1)
  Name: nfs
  Name: nfs403
  enc-part des-cbc-md5
  Encryption type: des-cbc-md5 (3)
  Kvno: 5
  enc-part: 2F97563D681D6BB1DC8B010025F82DB9...
Authenticator des-cbc-md5
  Encryption type: des-cbc-md5 (3)
  Authenticator data: 5B51E74785A4CBB0369C571DED75B7EB...

```

No.	Time	Source	Destination	Protocol	Info
175	20.431180	9.3.5.173	9.3.5.173	NFS	V4

NULL Reply (Call In 173)

```

Frame 175 (276 bytes on wire, 276 bytes captured)
  Arrival Time: Aug 6, 2004 15:30:07.185724000
  Time delta from previous packet: 0.044970000 seconds
  Time since reference or first frame: 20.431180000 seconds
  Frame Number: 175
  Packet Length: 276 bytes
  Capture Length: 276 bytes

```

Raw packet data

No link information available

Internet Protocol, Src Addr: 9.3.5.173 (9.3.5.173), Dst Addr: 9.3.5.173 (9.3.5.173)

```

Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
  0000 00.. = Differentiated Services Codepoint: Default (0x00)
  .... ..0. = ECN-Capable Transport (ECT): 0
  .... ...0 = ECN-CE: 0
Total Length: 276
Identification: 0x0596 (1430)
Flags: 0x04 (Don't Fragment)
  0... = Reserved bit: Not set
  .1.. = Don't fragment: Set
  ..0. = More fragments: Not set

```

```

Fragment offset: 0
Time to live: 60
Protocol: TCP (0x06)
Header checksum: 0x0000 (incorrect, should be 0x1aef)
Source: 9.3.5.173 (9.3.5.173)

```

```

Destination: 9.3.5.173 (9.3.5.173)
Transmission Control Protocol, Src Port: 2049 (2049), Dst Port: 32775 (32775),
Seq: 1, Ack: 1237, Len: 224
Source port: 2049 (2049)
Destination port: 32775 (32775)
Sequence number: 1 (relative sequence number)
Next sequence number: 225 (relative sequence number)
Acknowledgement number: 1237 (relative ack number)
Header length: 32 bytes
Flags: 0x0018 (PSH, ACK)
  0... .... = Congestion Window Reduced (CWR): Not set
  .0.. .... = ECN-Echo: Not set
  ..0. .... = Urgent: Not set
  ...1 .... = Acknowledgment: Set
  .... 1... = Push: Set
  .... .0.. = Reset: Not set
  .... ..0. = Syn: Not set
  .... ...0 = Fin: Not set
Window size: 262140
Checksum: 0x31fd (correct)
Options: (12 bytes)
  NOP
  NOP
  Time stamp: tsval 1091824482, tsecr 1091824482
Remote Procedure Call, Type:Reply XID:0x77709946
Fragment header: Last fragment, 220 bytes
  1... .... = Last Fragment: Yes
  .000 0000 0000 0000 0000 0000 1101 1100 = Fragment Length: 220
XID: 0x77709946 (2003867974)
Message Type: Reply (1)
Program: NFS (100003)
Program Version: 4
Procedure: NULL (0)
Reply State: accepted (0)
This is a reply to a request in frame 173
Time from request: 0.045289000 seconds
Verifier
  Flavor: RPCSEC_GSS (6)
  GSS Token
    GSS Token Length: 37
    GSS-API
      OID: 1.2.840.113554.1.2.2 (iso.2.840.113554.1.2.2) (KRB5 -
Kerberos 5)
      krb5_blob: 01010000FFFFFFFFE975FD4C7682CD7D...
        krb5_tok_id: KRB5_GSS_GetMIC (0x0101)
        krb5_sgn_alg: DES_MAC_MD5 (0x0000)
        krb5_snd_seq: E975FD4C7682CD7D
        krb5_sgn_cksum: 8E63BCC171B105D2
Accept State: RPC executed successfully (0)

```



```

Network File System
  Program Version: 4
  V4 Procedure: NULL (0)
  GSS Context: <DATA>
    length: 4
    contents: <DATA>
  GSS Major Status: 0
  GSS Minor Status: 0
  GSS Sequence Window: 48
  GSS Token
    GSS Token Length: 131
  GSS-API
    OID: 1.2.840.113554.1.2.2 (iso.2.840.113554.1.2.2) (KRB5 - Kerberos
5)
    krb5_blob: 02006F71306FA003020105A10302010F...
    krb5_tok_id: KRB5_AP_REP (0x0002)
    Kerberos AP-REP
      Pvno: 5
      MSG Type: AP-REP (15)
      enc-part des-cbc-md5
        Encryption type: des-cbc-md5 (3)
        enc-part: A949C61E03CE98601CEC6B2A55D24725...

```

No.	Time	Source	Destination	Protocol	Info
176	20.434340	9.3.5.173	9.3.5.173	NFS	V4

COMPOUND Call (Reply In 177)

```

Frame 176 (212 bytes on wire, 212 bytes captured)
  Arrival Time: Aug 6, 2004 15:30:07.188884000
  Time delta from previous packet: 0.003160000 seconds
  Time since reference or first frame: 20.434340000 seconds
  Frame Number: 176
  Packet Length: 212 bytes
  Capture Length: 212 bytes

```

Raw packet data

```

No link information available
Internet Protocol, Src Addr: 9.3.5.173 (9.3.5.173), Dst Addr: 9.3.5.173
(9.3.5.173)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    0000 00.. = Differentiated Services Codepoint: Default (0x00)
    .... ..0. = ECN-Capable Transport (ECT): 0
    .... ...0 = ECN-CE: 0
  Total Length: 212
  Identification: 0x059b (1435)
  Flags: 0x04 (Don't Fragment)
    0... = Reserved bit: Not set
    .1.. = Don't fragment: Set

```

```

    ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 60
    Protocol: TCP (0x06)
    Header checksum: 0x0000 (incorrect, should be 0x1b2a)
    Source: 9.3.5.173 (9.3.5.173)
    Destination: 9.3.5.173 (9.3.5.173)
Transmission Control Protocol, Src Port: 32775 (32775), Dst Port: 2049 (2049),
Seq: 1237, Ack: 225, Len: 160
    Source port: 32775 (32775)
    Destination port: 2049 (2049)
    Sequence number: 1237 (relative sequence number)
    Next sequence number: 1397 (relative sequence number)
    Acknowledgement number: 225 (relative ack number)
    Header length: 32 bytes
    Flags: 0x0018 (PSH, ACK)
        0... .... = Congestion Window Reduced (CWR): Not set
        .0.. .... = ECN-Echo: Not set
        ..0. .... = Urgent: Not set
        ...1 .... = Acknowledgment: Set
        .... 1... = Push: Set
        .... .0.. = Reset: Not set
        .... ..0. = Syn: Not set
        .... ...0 = Fin: Not set
    Window size: 262140
    Checksum: 0x56db (correct)
    Options: (12 bytes)
        NOP
        NOP
        Time stamp: tsval 1091824482, tsecr 1091824482
SEQ/ACK analysis
    This is an ACK to the segment in frame: 175
    The RTT to ACK the segment was: 0.003160000 seconds
Remote Procedure Call, Type:Call, XID:0x7770994d
    Fragment header: Last fragment, 156 bytes
        1... .... .... .... .... .... .... = Last Fragment: Yes
        .000 0000 0000 0000 0000 0000 1001 1100 = Fragment Length: 156
XID: 0x7770994d (2003867981)
Message Type: Call (0)
RPC Version: 2
Program: NFS (100003)
Program Version: 4
Procedure: COMPOUND (1)
The reply to this request is in frame 177
Credentials
    Flavor: RPCSEC_GSS (6)
    Length: 24
    GSS Version: 1
    GSS Procedure: RPCSEC_GSS_DATA (0)

```

```

GSS Sequence Number: 2
GSS Service: rpcsec_gss_svc_none (1)
GSS Context: <DATA>
  length: 4
  contents: <DATA>
Verifier
  Flavor: RPCSEC_GSS (6)
GSS Token
  GSS Token Length: 37
  GSS-API
  OID: 1.2.840.113554.1.2.2 (iso.2.840.113554.1.2.2) (KRB5 -
Kerberos 5)
  krb5_blob: 01010000FFFFFFFF28D134F10157FBBD...
  krb5_tok_id: KRB5_GSS_GetMIC (0x0101)
  krb5_sgn_alg: DES MAC MD5 (0x0000)
  krb5_snd_seq: 28D134F10157FBBD
  krb5_sgn_cksum: 9971CB9E15D6C1A5
Network File System
  Program Version: 4
  V4 Procedure: COMPOUND (1)
  Tag: nfs4pathlookup
  length: 14
  contents: nfs4pathlookup
  fill bytes: opaque data
  minorversion: 0
  Operations (count: 3)
  Opcode: PUTROOTFH (24)
  Opcode: LOOKUP (15)
  Filename: exports
  length: 7
  contents: exports
  fill bytes: opaque data
  Opcode: GETFH (10)

```

Unsuccessful authentication during mount request

Example B-13 shows the iptrace output taken during an unsuccessful authentication that was done while carrying out an NFS V4 mount. We outlined the two major packages, which are:

- ▶ Kerberos ticket request KRB5 TGS-REQ
- ▶ Invalid Kerberos ticket reply KRB5 TGS-REP

Example: B-13 iptrace output showing unsuccessful authentication during mount

No.	Time	Source	Destination	Protocol	Info
115	10.421160	9.3.5.175	9.3.4.71	KRB5	TGS-REQ

```

Frame 115 (1335 bytes on wire, 1335 bytes captured)
  Arrival Time: Aug 6, 2004 15:34:37.198215000
  Time delta from previous packet: 0.000246000 seconds
  Time since reference or first frame: 10.421160000 seconds
  Frame Number: 115
  Packet Length: 1335 bytes
  Capture Length: 1335 bytes
Ethernet II, Src: 00:02:55:af:12:7a, Dst: 00:09:12:48:3c:02
  Destination: 00:09:12:48:3c:02 (Cisco_48:3c:02)
  Source: 00:02:55:af:12:7a (Ibm_af:12:7a)
  Type: IP (0x0800)
Internet Protocol, Src Addr: 9.3.5.175 (9.3.5.175), Dst Addr: 9.3.4.71
(9.3.4.71)
  Version: 4
  Header length: 20 bytes
  Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    0000 00.. = Differentiated Services Codepoint: Default (0x00)
    .... ..0. = ECN-Capable Transport (ECT): 0
    .... ...0 = ECN-CE: 0
  Total Length: 1321
  Identification: 0x8efa (36602)
  Flags: 0x00
    0... = Reserved bit: Not set
    .0.. = Don't fragment: Not set
    ..0. = More fragments: Not set
  Fragment offset: 0
  Time to live: 30
  Protocol: UDP (0x11)
  Header checksum: 0xecce (correct)
  Source: 9.3.5.175 (9.3.5.175)
  Destination: 9.3.4.71 (9.3.4.71)
User Datagram Protocol, Src Port: 33735 (33735), Dst Port: kerberos (88)
  Source port: 33735 (33735)
  Destination port: kerberos (88)
  Length: 1301
  Checksum: 0xe902 (correct)
Kerberos TGS-REQ
  Pvno: 5
  MSG Type: TGS-REQ (12)
  padata: PA-TGS-REQ
    Type: PA-TGS-REQ (1)
      Value: 6E82045F3082045BA003020105A10302... AP-REQ
        Pvno: 5
        MSG Type: AP-REQ (14)
        Padding: 0
        APOptions: 00000000
          .0.. .... .... .... .... .... .... = Use Session Key:
Do NOT use the session key to encrypt the ticket

```

```

        ..0. .... = Mutual required:
Mutual authentication is NOT required
    Ticket
        Tkt-vno: 5
        Realm: KDC.ITSC.AUSTIN.IBM.COM
        Server Name (Unknown): krbtgt KDC.ITSC.AUSTIN.IBM.COM
        Name-type: Unknown (0)
        Name: krbtgt
        Name: KDC.ITSC.AUSTIN.IBM.COM
        enc-part rc4-hmac
        Encryption type: rc4-hmac (23)
        Kvno: 2
        enc-part: 242AC4DAA22D420C592754948B7C2399...
    Authenticator des-cbc-md5
        Encryption type: des-cbc-md5 (3)
        Authenticator data: 0EDE8A3F1B5C370A818DC7CFEF1BC12B...
KDC_REQ_BODY
    Padding: 0
    KDCOptions: 00800000 (Renewable)
        .0.. .... = Forwardable: Do NOT use
forwardable tickets
        ..0. .... = Forwarded: This is NOT a
forwarded ticket
        ...0 .... = Proxyable: Do NOT use
proxiable tickets
        .... 0... = Proxy: This ticket has
NOT been proxied
        .... .0.. .... = Allow Postdate: We do NOT
allow the ticket to be postdated
        .... ..0. .... = Postdated: This ticket is
NOT postdated
        .... .... 1... = Renewable: This ticket is
RENEWABLE
        .... .... ..0 .... = Opt HW Auth: False
        .... .... .... 0 .... = Canonicalize: This is NOT
a canonicalized ticket request
        .... .... .... ..0. .... = Disable Transited Check:
Transited checking is NOT disabled
        .... .... .... .... 0 .... = Renewable OK: We do NOT
accept renewed tickets
        .... .... .... .... 0... = Enc-Tkt-in-Skey: Do NOT
encrypt the tkt inside the skey
        .... .... .... .... ..0. = Renew: This is NOT a
request to renew a ticket
        .... .... .... .... .... 0 = Validate: This is NOT a
request to validate a postdated ticket
    Realm: KDC.ITSC.AUSTIN.IBM.COM
    Server Name (Principal): nfs nfs403.kdc.itsc.austin.ibm.com
    Name-type: Principal (1)

```

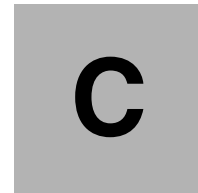
Name: nfs
Name: nfs403.kdc.itsc.austin.ibm.com
till: 2004-08-07 06:35:49 (Z)
Nonce: 1091824487
Encryption Types: des-cbc-md5 des-cbc-crc
Encryption type: des-cbc-md5 (3)
Encryption type: des-cbc-crc (1)

No.	Time	Source	Destination	Protocol	Info
	116 10.422686	9.3.4.71	9.3.5.175	KRB5	KRB

Error: KRB5KDC_ERR_S_PRINCIPAL_UNKNOWN

Frame 116 (164 bytes on wire, 164 bytes captured)
Arrival Time: Aug 6, 2004 15:34:37.199741000
Time delta from previous packet: 0.001526000 seconds
Time since reference or first frame: 10.422686000 seconds
Frame Number: 116
Packet Length: 164 bytes
Capture Length: 164 bytes
Ethernet II, Src: 00:09:12:48:3c:02, Dst: 00:02:55:af:12:7a
Destination: 00:02:55:af:12:7a (Ibm_af:12:7a)
Source: 00:09:12:48:3c:02 (Cisco_48:3c:02)
Type: IP (0x0800)
Internet Protocol, Src Addr: 9.3.4.71 (9.3.4.71), Dst Addr: 9.3.5.175 (9.3.5.175)
Version: 4
Header length: 20 bytes
Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
0000 00.. = Differentiated Services Codepoint: Default (0x00)
.... ..0. = ECN-Capable Transport (ECT): 0
.... ...0 = ECN-CE: 0
Total Length: 150
Identification: 0xb67d (46717)
Flags: 0x00
0... = Reserved bit: Not set
.0.. = Don't fragment: Not set
..0. = More fragments: Not set
Fragment offset: 0
Time to live: 127
Protocol: UDP (0x11)
Header checksum: 0x68de (correct)
Source: 9.3.4.71 (9.3.4.71)
Destination: 9.3.5.175 (9.3.5.175)
User Datagram Protocol, Src Port: kerberos (88), Dst Port: 33735 (33735)
Source port: kerberos (88)
Destination port: 33735 (33735)
Length: 130
Checksum: 0x31df (correct)
Kerberos KRB-ERROR

Pvno: 5
MSG Type: KRB-ERROR (30)
stime: 2004-08-06 20:35:56 (Z)
susec: 211147
error_code: KRB5KDC_ERR_S_PRINCIPAL_UNKNOWN (7)
Realm: KDC.ITSC.AUSTIN.IBM.COM
Server Name (Principal): nfs nfs403.kdc.itsc.austin.ibm.com
Name-type: Principal (1)
Name: nfs
Name: nfs403.kdc.itsc.austin.ibm.com



AIX 5.3 NFS quick reference

The purpose of this appendix is to provide a quick reference sheet for NFS on AIX 5.3. It includes the following sections:

- ▶ NFS configuration files
- ▶ NFS daemons
- ▶ NFS commands
- ▶ Export options
- ▶ mount command options
- ▶ nfsd command options

NFS configuration files

<i>/etc/rc.nfs</i>	Starts NFS on boot.
<i>/etc/filesystems</i>	Contains file systems to be mounted.
<i>/etc/exports</i>	Contains NFS export definitions.
<i>/etc/xtab</i>	Contains names of file systems currently exported.
<i>/etc/rmtab</i>	Contains names of machines and the file systems they have mounted.
<i>/etc/sm</i>	Directory used by rpc.statd
<i>/etc/sm.bak</i>	Directory used by rpc.statd
<i>/etc/state</i>	File used by rpc.statd
<i>/etc/nfs/realm.map</i>	File used by the NFS registry daemon to map incoming Kerberos principals.
<i>/etc/nfs/local_domain</i>	File containing the local NFS domain.
<i>/etc/nfs/hostkey</i>	File used by the NFS server to specify the Kerberos host principal and location of the keytab file.
<i>/etc/nfs/princmap</i>	File maps host names to Kerberos principals when the principal is not the fully qualified domain name of the server.
<i>/etc/nfs/security_default</i>	File containing the list of security flavors that may be used by the NFS client.

/etc/bootparms

File containing list of servers that diskless clients can use to boot from.

/etc/networks

File containing information about networks.

/etc/pcnfsd.conf

File containing options for the rpc.pcnfsd daemon.

/etc/rpc

File containing database information for RPC programs.

/etc/netgroup

File defining network-wide groups; used for checking permissions when doing remote mounts, remote logons, and remote shells.

NFS daemons

/usr/sbin/rpc.lockd

Processes lock requests through the RPC package.

/usr/sbin/rpc.statd

Provides crash-and-recovery functions for the NFS locking services.

/usr/sbin/biod

Sends the client's read and write requests to the server; runs only on the client.

/usr/sbin/rpc.mountd

Answers requests from clients for file system mounts; runs only on the server

/usr/sbin/nfsd

Starts the daemons that handle a client's request for file system operations; runs only on the server.

/usr/sbin/portmap

Maps RPC program numbers to Internet port numbers.

/usr/sbin/rpc.statd

Returns performance statistics obtained from the kernel.

/usr/sbin/pcnfsd

Handles service requests from PC-NFS clients.

/usr/bin/gssd

New daemon for NFS V4 that services kernel requests for GSS operations.

/usr/sbin/nfsrgyd

New daemon for NFS V4 that provides a name translation service for NFS servers and clients.

NFS commands

/usr/sbin/mount

Shows what file systems are mounted on the machine the command is run on, including name of the server, and mount options.

/usr/bin/showmount -e [host]

Shows contents of /etc/xtab file on the [host].

/usr/bin/showmount -a [host]

Shows the contents of the /etc/rmtab on the [host].

/usr/sbin/exportfs -va

Exports all file systems defined in /etc/exports and prints the name of each directory as it is exported.

/usr/sbin/exportfs -vua

Unexports all exported directories and prints the name of each directory as it is unexported.

/usr/sbin/mknfs

Configures a system to run NFS and starts NFS daemons.

/usr/sbin/nfso

Configures and lists NFS network options.

/usr/sbin/automount

Mounts an NFS automatically.

/usr/bin/chnfsexp

Changes the attributes of an NFS exported directory.

/usr/sbin/chnfsmnt

Changes the attributes of an NFS mounted directory.

/usr/sbin/lsnfsexp

Displays the characteristics of directories that are exported with NFS.

/usr/sbin/lsnfsmnt

Displays the characteristics of mounted NFS.

/usr/sbin/mknfsexp

Exports a directory.

/usr/sbin/mknfsmt	Mounts a directory using NFS.
/usr/sbin/rm nfs	Changes the configuration of NFS in system inittab and stops the NFS daemons.
/usr/sbin/rm nfs exp	Removes NFS exported directories from a server's list of exports.
/usr/sbin/chnfsdom	Changes the local NFS domain.
/usr/sbin/nfs4cl	Displays or modifies current NFS V4 statistics and properties.
/usr/sbin/nfshostkey	Configures the host key for an NFS server.
/usr/sbin/chnfsim	Changes the NFS foreign identity mappings.

Export options

-o options

-rw

All clients have read-write permission (default).

-ro

All clients have read-only permission.

-rw=Client[:Client]

Exports the directory with read-write permission to the specified clients. Exports the directory read-only to clients not in the list.

-access=Client[:Client,...]

Gives mount access to each client listed. A client can be either a host name or a net group name.

-root=Client[:Client]

Allows root access from the specified clients.

-vers=version_number[:version_number]

Specifies which versions of NFS are allowed to access the exported directory. Valid versions are 2, 3, and 4.

-exname=external-name

Exports the directory by the specified external name. The external name must begin with the nfsroot name.

-sec=flavor[:flavor...]

Used to specify a list of security methods that may be used to access files under the exported directory. Allowable flavor values are: sys, dh, none, krb5, krb5i and krb5p.

-nfsroot

Sets the nfsroot to a specified directory. For example, **/exports -nfsroot**.

mount command options

-o options

ro

Specifies that the mounted file is read-only.

rw

Specifies that the mounted file is read/write accessible (default).

fg

Attempts mount in foreground if first attempt is unsuccessful (default).

bg

Attempts mount in background if first attempt is unsuccessful.

hard

Retries a request until server responds (default).

soft

Returns an error if the server does not respond.

intr

Allows keyboard interrupts on hard mounts.

nointr

Specifies no keyboard interrupts allowed on hard mounts.

acl

Requests using the Access Control List RPC program for this NFS mount.

sec=[flavor1:...:flavorn]

Specifies a list of security methods that may be used to access files under the mount point. Allowable security flavors are: sys, dh, krb5, krb5i and krb5p

vers=Version

Specifies NFS version. Options are 2, 3, and 4. vers=4 is only applicable to AIX 5.3.

nfs command options

Use the **nfs** command to configure Network File System tuning parameters. The **nfs** command sets or displays current or next boot values for Network File System tuning parameters. This command can also make permanent changes or defer changes until the next reboot. Whether the command sets or displays a parameter is determined by the accompanying flag. The **-o** flag performs both actions. It can either display the value of a parameter or set a new value for a parameter.

Important: Extreme care must be taken before values are changed using the **nfs** command. *An incorrect change can render the system unusable.*

NFS V4 introduces the following new tunable parameters:

utf8

This option enables NFS V4 to perform UTF8 checking.

A value of 1 turns on UTF-8 checking of file names.
A value of 0 turns it off.

utf8_validation

Enables checking of file names for the NFS V 4 client and server to ensure that they conform to the UTF-8 specification.

A value of 1 turns on UTF-8 checking of file names.
A value of 0 turns it off.

nfs_v4_pdt

Sets the number of tables for memory pools used by the biods for NFS V4 mounts.

You should use the **vmstat -v** command to look for non-zero values in the client file system I/Os blocked with no fsbuf field.

Increase the number until the blocked I/O count is no longer incremented during workload. The number might need to be increased in conjunction with **nfs_v4_vm_bufs**.

nfs_v4_vm_bufs

Sets the number of initial free memory buffers used for each NFS V4 paging device table (pdt) created after the first table. The very first pdt has a set value of 256,

512, 640, or 1000, depending on system memory. This initial value is also the default value of each newly created pdt.

You should use the `vmstat -v` command to look for non-zero values in the client file system I/Os blocked with no fsbuf field.

The `nfs_v4_vm_bufs` option must be set prior to `nfs_v4_pdt`s.

Useful examples how the `nfso` command can be used:

1. To print, in colon-delimited format, a list of all tunable parameters and their current values, run:

```
nfso -a -c
```

2. To list the current and reboot value, range, unit, type, and dependencies of all tunable parameters managed by the `nfso` command, run:

```
nfso -L
```

3. To list the reboot values for all Network File System tuning parameters, run:

```
nfso -r -a
```

4. To set a tunable parameter, for example `utf8`, to a value of 1, run:

```
nfso -o utf8=1
```

5. To set a tunable parameter, for example `nfs_v4_pdt`s, to its default value of 1 at the next reboot, run:

```
nfso -r -d nfs_v4_pdt
```

The following references are also useful:

“Network File System (NFS) Overview for System Management” and “TCP/IP Overview for System Management,” *AIX 5L Version 5.3 System User’s Guide: Communications and Networks*, SC23-4909

“Monitoring and Tuning NFS Use,” *AIX 5L Version 5.3 Performance Management Guide*, SC23-4905

To view these documents, choose **System management guides** in the left navigation bar at:

<http://publib.boulder.ibm.com/infocenter/pseries/index.jsp?topic=/com.ibm.aix.doc/cmds/aixcmds6/ypserv.htm>

Abbreviations and acronyms

ACE	Access Control Entry	LDAP	Lightweight Directory Access Protocol
ACL	access control list	LDIF	LDAP Data Interchange Format
AFS	Andrew file system	LIPKEY	Low Infrastructure Public Key
AIO	asynchronous input/output	NAS	Network Authentication Service
ASCII	American Standard Code for Information Interchange	NFS	Network File System
DCE	Distributed Computing Environment	NFS V2	NFS Version 2
DES	Data Encryption Standard	NFS V3	NFS Version 3
DFS	Distributed File System	NFS V4	NFS Version 4
DNS	Domain Name Service	NIM	Network Installation Management
EIM	Enterprise Identity Mapping	NIS	Network Information Service
FS	file system	NLM	Network Lock Manager
GID	group ID	NTP	Network Time Protocol
GPFS	General Parallel File System	OSI	open systems interconnection
GSS-API	Generic Security Service Application Programming Interface	POSIX	Portable Operating System Interface
GUI	graphical user interface	RFC	Request For Comments
IBM	International Business Machines Corporation	RFS	(AT&T) Remote File System
IEEE	Institute of Electrical and Electronics Engineers	RPC	remote procedure call
IETF	Internet Engineering Task Force	SMIT	System Management Interface Tool
IP	Internet Protocol	SPKM	Simple Public-Key Mechanism
ISO	International Organization for Standardization	SSL	Secure Sockets Layer
ITSO	International Technical Support Organization	TCP	Transmission Control Protocol
JFS	journaled file system	UDP	User Datagram Protocol
JFS2	enhanced JFS	UID	User ID
KDC	key distribution center	UTF	Unicode Transformation Format
KRB5	Kerberos version 5	XDR	External Data Representation

Glossary

ACE. (Access Control Entry) One of the entries in an Access Control List (ACL).

ACL. (Access Control List) A list of permission entries that control user and group access to an object.

Authentication. Security method used to confirm the identity of a system user, host, or service.

Authorization. Security method used to control what shared information each system user or client machine can access.

GID. (Group Identifier) Number used to identify a UNIX group.

Identification. Security method used to uniquely establish the identity of information system users, hosts, and services.

Integrated login. System login configured to obtain user authentication and identification (optional) from an external source, such as Kerberos/LDAP.

KDC. (Key Distribution Center) The trusted third-party or intermediary in Kerberos that issues all the Kerberos tickets to the clients.

Kerberos realm. Comprises a set of managed hosts that share the same Kerberos database.

NFS client. A host that accesses, via a mount, one or more directories from an NFS server.

NFS domain. A name that identifies an operating context for NFS servers and clients. It is implied that systems that share the same NFS domain also share the same user and group registries.

NFS export. The operation that makes a directory on an NFS server available for access by NFS clients.

GSS-API. Generic Security Services Application Programming Interface. A generic API for doing client-server authentication.

NFS mount. The operation that maps an exported directory from an NFS server into a client's directory structure, making it appear as if the served directory is local on the client.

NFS server. A host that makes available for NFS access one or more of its directories.

NFS. (Network File System) A protocol for sharing files over a computer network.

Opaque. Used in conjunction with bytes, data structures, tokens, and so on, to represent a collection of bytes whose internal structure is unknown at the time but will be interpreted later on in a processing sequence.

Pseudo-file system. The portion of an NFS server's name space that is exported to NFS clients.

Pseudo-root. The top level of an NFS server's pseudo-file system. By default, it corresponds to the root directory in the server's file tree, but it can be specified to be a lower-level directory, such as /exports.

RPC. (Remote Procedure Call) A protocol whereby one computer process (the client process) can direct another process (the server process) to run a procedure, appearing as if the client process had run the procedure in its own address space. The client and server processes are typically on two separate computers, although they can both be on the same computer.

RPCSEC_GSS. A security flavor that provides authentication, integrity, and privacy protection for remote procedure calls.

UID. (User Identifier) Number used to identify a UNIX user.

WSM. (Web-based System Manager) A Web-based interface for administering an AIX system.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 302. Note that some of the documents referenced here may be available in softcopy only.

- ▶ *AIX - Migrating NIS Maps into LDAP, TIPS0123*
- ▶ *AIX 4.3 Elements of Security, SG24-5962*
- ▶ *AIX 5L Version 5.2 Security Supplement, SG24-6066*
- ▶ *AIX 5L Differences Guide Version 5.3 Edition, SG24-7463*
- ▶ *Exploiting RS/6000 SP Security: Keeping It Safe, SG24-5521*
- ▶ *IBM IBM @server Certification Study Guide - AIX 5L Communications, SG24-6186*
- ▶ *IBM IBM @server pSeries Sizing and Capacity Planning: A Practical Guide, SG24-7071*
- ▶ *Introduction to the IBM Problem Determination Tools, SG24-6296*
- ▶ *RS/6000 SP System Management: Easy, Lean and Mean, GG24-2563*
- ▶ *Using LDAP for Directory Integration, SG24-6163*
- ▶ *Windows-based Single Signon and the EIM Framework on the IBM IBM @server iSeries Server, SG24-6975*

Other publications

These publications are also relevant as further information sources:

- ▶ *AIX 5L Version 5.3 Commands Reference, Volume 1, SC23-4888*
- ▶ *AIX 5L Version 5.3 Commands Reference, Volume 2, SC23-4889*
- ▶ *AIX 5L Version 5.3 Commands Reference, Volume 4, SC23-4891*
- ▶ *AIX 5L Version 5.3 Files Reference, SC23-4895*

- ▶ *AIX 5L Version 5.3 Performance Management Guide*, SC23-4905
- ▶ *AIX 5L Version 5.3 Security Guide*, SC23-4907
- ▶ *AIX 5L Version 5.3 System Management Guide: Communications and Networks*, SC23-4909
- ▶ Garman, *Kerberos: The Definitive Guide*, O'Reilly, 2003, ISBN 0596004036
- ▶ Stern, et al., *Managing NFS and NIS, 2nd Edition*, O'Reilly, 2001, ISBN 1565925106

Online resources

These Web sites and URLs are also relevant as further information sources:

- ▶ NFS Version 4 Open Source Reference Implementation
<http://www.citi.umich.edu/projects/nfsv4/linux/>
- ▶ The NFS Version 4 Protocol, Brian Pawlowski, et al
<http://www.nluug.nl/events/sane2000/papers/pawlowski.pdf>
- ▶ NFS Version 3 Design and Implementation, Brian Pawlowski, et al
<http://citeseer.ist.psu.edu/pawlowski94nfs.html>
- ▶ IETF RFC page
<http://www.ietf.org/rfc.html>
- ▶ Other RFC references for NFS version 4
<http://www.nfsv4.org/nfsv4techinfo.html>
- ▶ NFS V4 Working Group
<http://www.nfsv4.org/>
- ▶ Open Systems Interconnection (OSI) Reference Model
<http://ourworld.compuserve.com/homepages/timothydevans/osi.htm>

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications, and additional materials, as well as order hard-copy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

Symbols

- /etc/auto.master 177
- /etc/bootparms 289
- /etc/exports 37
- /etc/exports file 88
- /etc/filesystems 288
- /etc/fstab (Linux) 177
- /etc/group file 49
- /etc/gssapi_mech.conf (Linux) 177
- /etc/ibmslapd.conf 212
- /etc/idmapd.conf (Linux) 177
- /etc/init.d/portmap (Linux) 177
- /etc/init.d/rpcgssd (Linux) 177
- /etc/init.d/rpcidmapd (Linux) 177
- /etc/inittab 137
- /etc/krb5/krb5.conf 268
- /etc/krb5/krb5.keytab 143
- /etc/netgroup 289
- /etc/networks 289
- /etc/nfs.clean 126
- /etc/nfs/hostkey 38, 288
- /etc/nfs/local_domain 38, 288
- /etc/nfs/princmap 288
- /etc/nfs/princmap file 39
- /etc/nfs/realm.map 39, 288
- /etc/nfs/realm.map file 54
- /etc/nfs/security_default 40, 288
- /etc/passwd file 49
- /etc/pcnfsd.conf 289
- /etc/rc.nfs 288
- /etc/rmtab 288
- /etc/rpc 289
- /etc/security/user 173
- /etc/services 158
- /etc/sm 288
- /etc/sm.bak 288
- /etc/state 288
- /etc/syslog.conf 208
- /etc/xtab 38
- /usr/lib/security/methods.cfg 166
- /var/krb5/log/krb5kdc.log 211
- /var/ldap/db2cli.log 212
- /var/ldap/ibmslapd.log 212

A

- aclconvert command 73
- acledit command 32, 73
- aclget command 32, 73, 82
- aclgettypes command 33, 73
- aclput command 32, 73, 82
- ACLs
 - AIXC ACLs
 - AIXC ACL and NFS V4 client 64
 - commands 73
 - NFS V4 ACLs
 - access evaluation 69
 - administration 72
 - chmod command and 77
 - directory structure and 79
 - file system support 65
 - format 65
 - inheritance 68
 - inheritance and move vs. copy 79
 - inheritance and umask 78
 - inheritance, maximizing benefits of 80
 - maintaining existing ACLs 82
 - NFS V3 clients and 87
 - permission bits 67
 - permission restrictions 69
 - permissions scenarios 85
 - special user permissions 69
 - UNIX permissions and 71
 - types
 - AIXC ACL 33, 62–63
 - NFS V4 ACL 32, 65
- add_principal command 139
- addprinc command 139
- administration
 - sample scripts 256
- aio 157
- alias tree
 - configuration 131
 - setting up alias tree extension in NFS V4 131
- attributes
 - mandatory attributes 21
 - named attributes 23
 - recommended attributes 21
- authentication 46

- AUTH_SYS 47, 51, 59
 - example unsuccessful 281
 - host 88
 - Kerberos 53, 59
 - user 59
- authentication grammar 166, 173
- authorization 47
 - access control lists 62
 - exports 88
 - host 88
 - standard UNIX file permissions 63
 - user 62
- automount command 44, 291

B

- bootinfo command 156
- bosboot command 157

C

- chkconfig (Linux) command 181
- chnfs command 43, 127
- chnfsdom command 38, 43, 292
- chnfsexp command 44, 291
- chnfsim command 32, 43, 292
- chnfsmnt command 44, 291
- chnfsrtd command 39, 43, 146
- chnfssec command 40, 43
- chnfssim command 39
- chsec command 174
- chuser command 173
- commands
 - /etc/nfs.clean 126
 - /etc/rc.nfs 127
 - aclconvert 73
 - acledit 32, 73
 - aclget 32, 73, 82
 - aclgettypes 33, 73
 - aclput 32, 73, 82
 - add_principal 139
 - addprinc 139
 - automount 44, 291
 - bootinfo 156
 - bosboot 157
 - chkconfig (Linux) 181
 - chmod 77
 - chnfs 43, 127
 - chnfsdom 38, 43, 292
 - chnfsexp 44, 291

- chnfsim 32, 43, 292
- chnfsmnt 44, 291
- chnfsrtd 39, 43, 146
- chnfssec 40, 43
- chnfssim 39
- chsec 174
- chuser 173
- config.krb5 137
- db2iauto 162
- db2set 162
- errpt 211
- exportfs 44, 88, 291
 - options 293
- find 132
- fsuser 210
- get_principals 140
- getprincs 140
- ibmdirctl 162
- id 154
- installp 135
- iprepor 210
- iptrace 210
- kadmin.local 139
- kdestroy 149
- kinit 138
- klist 139
- ktpass.exe (Windows) 195
- ktutil 143
- ldapcfg 162
- ldapmodify 163
- ldapsearch 164
- lsattr 157
- lsitab 137
- lsnfsexp 44, 291
- lsnfsmnt 44, 291
- lsnf 212
- ls-secdapclntd 167
- lssrc 128
- lsuser 169
- mkggroup 166
- mkkrb5srv 136
- mknfs 43, 291
- mknfsexp 44, 291
- mknfsmnt 44, 292
- mkuser 166
- mount 129, 291
 - options 294
- nfs4cl 43, 132, 211, 292
- nfshostkey 38, 43, 144, 292

- nfshostmap 39, 43
- nfso 44, 291
 - options 294–295
- nfsstat 44, 211
- ps 166
- refresh 209
- rmnfs 44, 292
- rmnfsexp 44, 292
- rmnfsmnt 44
- rpcgen 44
- rpcinfo 44, 210
- setclock 105
- showmount 123, 211, 291
- snap 263
- startsrc 124
- stopsrc 188
- umount 123
- uptime 153
- config.krb5 command 137

D

- db2iauto command 162
- db2set command 162
- diskless client 32
- domain name 124

E

- EIM
 - bos.eim.rte file set 57
- Enterprise Identity Mapping
 - See EIM
- errpt command 211
- Ethereal 208, 210, 212–213
- exportfs command 44, 88, 291
 - options 293
- exports file 88
- External Data Representation
 - See XDR
- external name space (exname) 34

F

- file handle 30
 - persistent 30
 - volatile 30
- file locking
 - clientid 29
 - stateid 29

file sets

- bos.eim.rte 57
- db2_08_01.ldap 158
- krb5.client 145
- ldap.client.adt 158
- ldap.client.rte 158
- ldap.html.en_US.config 158
- ldap.html.en_US.man 158
- ldap.msg.en_US 158
- ldap.server.cfg 158
- ldap.server.com 158
- ldap.server.java 158
- ldap.server.rte 158
- ldap.webdadmin 158
- modcrypt.base 145

files

- /etc/bootparms 43, 289
- /etc/environment 41
- /etc/exports 43, 88, 288
- /etc/filesystems 42, 288
- /etc/fstab (Linux) 177
- /etc/group 49
- /etc/gssapi_mech.conf (Linux) 177
- /etc/ibmslapd.conf 212
- /etc/idmapd.conf (Linux) 177
- /etc/inittab 137
- /etc/krb5/krb5.conf 137, 268–269
- /etc/krb5/krb5.keytab 143
- /etc/krb5/krb5_cfg_type 137
- /etc/netgroup 289
- /etc/networks 43, 289
- /etc/nfs/hostkey 38, 42, 288
- /etc/nfs/local_domain 38, 42, 288
- /etc/nfs/princmap 39, 42, 288
- /etc/nfs/realm.map 39, 42, 54, 288
- /etc/nfs/security_default 40, 42, 288
- /etc/passwd 49
- /etc/pcnfsd.conf 43, 289
- /etc/rc.nfs 288
- /etc/rmtab 288
- /etc/rpc 43, 289
- /etc/security/user 173
- /etc/services 158
- /etc/sm 288
- /etc/sm.bak 288
- /etc/state 288
- /etc/syslog.conf 208
- /etc/xtab 38, 43, 288
- /usr/lib/security/methods.cfg 166

L

- LDAP user registry 49
- ldapcfg command 162
- ldapmodify command 163
- ldapsearch command 164
- LDIF
 - sample file 270
- Linux client
 - pseudo-file system in NFS V4 187
 - Read/Write NFS V4 mounts 185
 - Read-only NFS V4 mount 182
- lsattr command 157
- lsitab command 137
- lsnfsexp command 44, 291
- lsnfmnt command 44, 291
- ls of command 212

M

- mknfs command 43, 291
- mknfsexp command 44, 291
- mknfmnt command 44, 292
- mount command 291
 - options 294
- multi-homed server 105

N

- namespace 25
 - pseudo-file system 27
- NAS
 - configuration 134
 - installing IBM NAS file sets 135
 - path variable 121
 - set up with legacy database 134
- Network File System
 - See NFS
- Network Installation Management
 - See NIM
- Network Time Protocol
 - See NTP
- NFS 4
 - definition 3
 - NFS V3 (NFS Version 3) 17
- NFS daemons 14
 - /usr/bin/gssd 290
 - /usr/sbin/biod 290
 - /usr/sbin/nfsd 290
 - /usr/sbin/nfsrgyd 290
 - /usr/sbin/pcnfsd 290

- /usr/sbin/portmap 290
- /usr/sbin/rpc.lockd 290
- /usr/sbin/rpc.mountd 290
- /usr/sbin/rpc.statd 290
- biod 17, 42
- gssd 41, 61
- NFS Lock Manager (NLM) 19
- nfsd 16
- nfsrgyd 41
- portmap 15, 42
- rpc.lockd 16, 41
- rpc.mountd 16, 42
- rpc.pcnfsd 42
- rpc.rstatd 42
- rpc.statd 16, 41
- NFS domain
 - choosing your NFS domain 98
 - configuration 146
 - multiple NFS domains 57
 - single NFS domain 51
- NFS domain name 124
- NFS registry daemon 143
- NFS V4
 - ACLs 65
 - AIXC ACL and NFS V4 client 64
 - alias tree extension 131
 - authentication methods 102
 - client with integrated login services 170
 - exports options 89
 - full client 108
 - general deployment strategy 95
 - host authentication 88
 - host authorization 88
 - host identification 87
 - identity mapping 24
 - implementation 119
 - integrating with Linux client 176
 - Linux client 170
 - mandatory attributes 21
 - named attributes 23
 - nobody, mapping unknown users to 57
 - pseudo root FS 124
 - recommended attributes 21
 - security context 60
 - setting up client with NAS 145
 - slim client 108
 - slim client versus full client 108
 - syslogd settings 121
 - unknown user 57

- user authentication 59
- user authorization 62
- user/group identification 50
 - with Windows KDC 190
- NFS V4 (NFS version 4) 20
- NFS_NOBODY 41
- NFS_PORT_RANGE 41
- nfs4cl command 43, 211, 292
- nfshostkey command 38, 43, 292
- nfshostmap command 39, 43
- nfso command 44, 291
 - options 294–295
- nfsstat command 44, 211
- NIM 32
- NIS user registry 49
- NTP

P

- passwd file 49
- princmap file 39
- pseudo-file system 27
 - advantages 128
 - configuration 125
 - pseudo root FS 124

R

- realm.map file 54
- Redbooks Web site 302
 - contact us xv
- refresh command 209
- Remote Procedure Call
 - See RPC)
- RFC
 - RFC2078 7
 - RFC2307 7, 170
 - RFC3010 5
 - RFC3530 5
- rmnfs command 44, 292
- rmnfsexp command 44, 292
- RPC
 - security flavors 47
- rpcgen command 44
- rpcinfo command 44, 210
- RPCSEC_GSS 47
 - authentication flow 61
 - configuration 154
 - configuring on clients 154
 - Kerberos 47

- LIPKEY 47
- protection levels
 - authentication 47
 - integrity 47
 - privacy 47
- security context 60

S

- scripts
 - change pseudo-root FS 256
 - copy ACLs 260
 - create full client (LDAP) 259
 - create full client (legacy db) 258
 - create KDC server 256
 - nfs_pd script 263
- sec= exports option 89
- security
 - AUTH_SYS 51
 - flavors 107
 - Kerberos 28, 53
 - krb5 107
 - krb5i 107
 - krb5p 107
 - LIPKEY 28
 - RPC flavors 47
- security categories 46
- security components
 - authentication 46
 - authorization 47
 - identification 46
- service ticket 59
- showmount command 211, 291
- slim client 150, 152
- slim client for cloning 150
- slim client installation steps 150
- slim client verification 153
- snap command 263
- stateful 13
- stateless 13
- Symbols
 - /etc/exports 288
 - /etc/krb5/krb5.conf 269
 - /etc/syslog.conf 122
 - /etc/xtab 288
 - /var/krb5/log/kadmin.log 211
 - NFS_NOBODY 41
 - NFS_PORT_RANGE 41
- syslogd 121

T

- ticket-granting ticket 59
- time synchronization 104
 - NTP 104
 - setclock 105
 - timed 104
- troubleshooting 207
 - EIM not configured 213
 - exporting file systems 215
 - GSS-API error codes 232
 - IBM Tivoli Directory Server 212
 - Kerberos status codes 234
 - mount problems 218
 - NAS 211
 - realm is already mapped to domain 214
 - tools 208
 - using errpt command 211
 - using Ethereal 212
 - using fuser command 210
 - using iptrace and ipreport 210
 - using lsof command 212
 - using nfs4cl command 211
 - using nfsstat command 211
 - using rpcinfo command 210
 - using showmount command 210
 - using syslogd 208

U

- UID 48
- UNIX user registry 49
- unmounting exported NFS V4 file system 123
- user registries
 - NIS 49
- UTF-8 30–31

V

- vers=exports option 89

W

- Web-based Systems Manager 73
 - ACL administration 73
- Windows KDC 190

X

- XDR



Securing NFS in AIX An Introduction to NFS V4 in AIX 5L

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



Securing NFS in AIX

An Introduction to NFS V4 in AIX 5L Version 5.3



NFS Version 4 (NFS V4) is the latest defined client-to-server protocol for NFS. A significant upgrade from NFS V3, it was defined under the IETF framework by many contributors. NFS V4 introduces a major changes to the way NFS has been implemented and used up until now, including stronger security, wide area network sharing, and broader platform adaptability.

This IBM Redbook is intended to provide a broad understanding of NFS V4 and specific AIX NFS V4 implementation details. It discusses considerations for deployment of NFS V4, with a focus on exploiting the stronger security features of the new protocol.

In the initial implementation of NFS V4 in AIX 5.3, the most important functional differences are related to security. Chapter 3 and parts of the planning and implementation chapters in Part 2 cover this topic in detail.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks