

## Contents

- [1. Unknown](#)
- [2. Unknown](#)
- [3. Unknown](#)
- [4. Unknown](#)
- [5. Unknown](#)
- [6. Unknown](#)
- [7. Unknown](#)
- [8. Unknown](#)
- [9. Unknown](#)
- [10. Unknown](#)
- [11. Unknown](#)
- [12. Unknown](#)
- [13. Unknown](#)
- [14. Unknown](#)
- [15. Unknown](#)
- [16. Unknown](#)
- [17. Unknown](#)
- [18. Unknown](#)
- [19. Unknown](#)
- [20. Unknown](#)
- [21. Unknown](#)
- [22. Unknown](#)
- [23. Unknown](#)
- [24. Unknown](#)
- [25. Unknown](#)
- [26. Unknown](#)
- [27. Unknown](#)
- [28. Unknown](#)
- [29. Unknown](#)
- [30. Unknown](#)
- [31. Unknown](#)
- [32. Unknown](#)
- [33. Unknown](#)
- [34. Unknown](#)
- [35. Unknown](#)
- [36. Unknown](#)
- [37. Unknown](#)
- [38. Unknown](#)
- [39. Unknown](#)
- [40. Unknown](#)
- [41. Unknown](#)
- [42. Unknown](#)
- [43. Unknown](#)
- [44. Unknown](#)
- [45. Unknown](#)
- [46. Unknown](#)

- 47. Unknown
- 48. Unknown
- 49. Unknown
- 50. Unknown
- 51. Unknown
- 52. Unknown
- 53. Unknown
- 54. Unknown
- 55. Unknown
- 56. Unknown
- 57. Unknown
- 58. Unknown
- 59. Unknown
- 60. Unknown
- 61. Unknown
- 62. Unknown
- 63. Unknown
- 64. Unknown
- 65. Unknown
- 66. Unknown
- 67. Unknown
- 68. Unknown
- 69. Unknown
- 70. Unknown
- 71. Unknown
- 72. Unknown
- 73. Unknown
- 74. Unknown
- 75. Unknown
- 76. Unknown
- 77. Unknown
- 78. Unknown
- 79. Unknown
- 80. Unknown
- 81. Unknown
- 82. Unknown
- 83. Unknown
- 84. Unknown
- 85. Unknown
- 86. Unknown
- 87. Unknown
- 88. Unknown
- 89. Unknown
- 90. Unknown
- 91. Unknown
- 92. Unknown
- 93. Unknown
- 94. Unknown

[95. Unknown](#)

[96. Unknown](#)

[97. Unknown](#)

[98. Unknown](#)

[99. Unknown](#)

[100. Unknown](#)

[\[ Team LiB \]](#)

NEXT ▶



[Table of Contents](#)

[Index](#)

[Reviews](#)

[Reader Reviews](#)

[Errata](#)

## **Kerberos: The Definitive Guide**

By [Jason Garman](#)

START READING

Publisher: O'Reilly

Pub Date: August 2003

ISBN: 0-596-00403-6

Pages: 272

Single sign-on is the holy grail of network administration, and Kerberos is the only game in town. Microsoft, by integrating Kerberos into Active Directory in Windows 2000 and 2003, has extended the reach of Kerberos to all networks large or small. *Kerberos: The Definitive Guide* shows you how to implement Kerberos on Windows and Unix systems for secure authentication. In addition to covering the basic principles behind cryptographic authentication, it covers everything from basic installation to advanced topics like cross-realm authentication, defending against attacks on Kerberos, and troubleshooting.

[\[ Team LiB \]](#)

NEXT ▶



[Table of Contents](#)

[Index](#)

[Reviews](#)

[Reader Reviews](#)

[Errata](#)

## **Kerberos: The Definitive Guide**

By [Jason Garman](#)

[START READING](#)

Publisher: O'Reilly

Pub Date: August 2003

ISBN: 0-596-00403-6

Pages: 272

[Dedication](#)

[Copyright](#)

[Preface](#)

[Organization of This Book](#)

[Conventions Used in This Book](#)

[Comments and Questions](#)

[Thanks...](#)

### [Chapter 1. Introduction](#)

[Section 1.1. Origins](#)

[Section 1.2. What Is Kerberos?](#)

[Section 1.3. Goals](#)

[Section 1.4. Evolution](#)

[Section 1.5. Other Products](#)

### [Chapter 2. Pieces of the Puzzle](#)

[Section 2.1. The Three As](#)

[Section 2.2. Directories](#)

[Section 2.3. Privacy and Integrity](#)

[Section 2.4. Kerberos Terminology and Concepts](#)

[Section 2.5. Putting the Pieces Together](#)

## Chapter 3. Protocols

Section 3.1. The Needham-Schroeder Protocol

Section 3.2. Kerberos 4

Section 3.3. Kerberos 5

Section 3.4. The Alphabet Soup of Kerberos-Related Protocols

## Chapter 4. Implementation

Section 4.1. The Basic Steps

Section 4.2. Planning Your Installation

Section 4.3. Before You Begin

Section 4.4. KDC Installation

Section 4.5. DNS and Kerberos

Section 4.6. Client and Application Server Installation

## Chapter 5. Troubleshooting

Section 5.1. A Quick Decision Tree

Section 5.2. Debugging Tools

Section 5.3. Errors and Solutions

## Chapter 6. Security

Section 6.1. Kerberos Attacks

Section 6.2. Protocol Security Issues

Section 6.3. Security Solutions

Section 6.4. Protecting Your KDC

Section 6.5. Firewalls, NAT, and Kerberos

Section 6.6. Auditing

## Chapter 7. Applications

Section 7.1. What Does Kerberos Support Mean?

Section 7.2. Services and Keytabs

Section 7.3. Transparent Kerberos Login with PAM

Section 7.4. Mac OS X and the Login Window

Section 7.5. Kerberos and Web-Based Applications

Section 7.6. The Simple Authentication and Security Layer (SASL)

Section 7.7. Kerberos-Enabled Server Packages

Section 7.8. Kerberos-Enabled Client Packages

Section 7.9. More Kerberos-Enabled Packages

## Chapter 8. Advanced Topics

Section 8.1. Cross-Realm Authentication

Section 8.2. Using Kerberos 4 Services with Kerberos 5

Section 8.3. Windows Issues

Section 8.4. Windows and Unix Interoperability

## Chapter 9. Case Study

Section 9.1. The Organization

Section 9.2. Planning

Section 9.3. Implementation

## Chapter 10. Kerberos Futures

Section 10.1. Public Key Extensions

Section 10.2. Smart Cards

[Section 10.3. Better Encryption](#)  
[Section 10.4. Kerberos Referrals](#)  
[Section 10.5. Web Services](#)

[Appendix A. Administration Reference](#)

[Section A.1. MIT](#)  
[Section A.2. Configuration File Format](#)

[Colophon](#)

[Index](#)

[ [Team LiB](#) ]

◀ PREVIOUS    NEXT ▶

[ [Team LiB](#) ]

◀ PREVIOUS    NEXT ▶

## Dedication

*Dedicated in loving memory to my grandfather, Harry Stumpff.*

—Jason Garman

[ [Team LiB](#) ]

◀ PREVIOUS    NEXT ▶

[ [Team LiB](#) ]

◀ PREVIOUS    NEXT ▶

# Copyright

Copyright 2003 O'Reilly & Associates, Inc.

Printed in the United States of America.

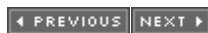
Published by O'Reilly & Associates, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly & Associates books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safari.oreilly.com>). For more information, contact our corporate/institutional sales department: (800) 998-9938 or [corporate@oreilly.com](mailto:corporate@oreilly.com).

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly & Associates, Inc. Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly & Associates, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps. The association between the image of a barred owl and the topic of Kerberos is a trademark of O'Reilly & Associates, Inc.

While every precaution has been taken in the preparation of this book, the publisher and author assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

[\[ Team LiB \]](#)



[\[ Team LiB \]](#)



# Preface

Kerberos is a sophisticated network authentication system—one that has been publicly available since 1989 and provides that eternal holy grail of network administrators, single-sign-on. Yet, in that intervening decade, documentation on Kerberos has been notably lacking. While many large organizations and academic institutions have enjoyed the benefits of using Kerberos in their networks, the deployment of Kerberos in smaller networks has been severely hampered by a lack of documentation.

I decided to write this book precisely because of this lack of useful documentation. My own experiences with Kerberos are those of extreme frustration as I attempted to decipher the documentation. I found that I had to keep copious notes to keep everything straight. Those notes eventually became the outline of this book.

Today, Microsoft, through its adoption of the latest Kerberos protocol as the preferred authentication mechanism in its Active Directory, has single-handedly driven the use of Kerberos into the majority of the operating-system market that it controls. Thanks to the openness of Kerberos, organizations now can establish cross-platform, single sign-on network environments, giving an end-user one set of credentials that will provide him access to all network resources, regardless of platform or operating system. Yet the workings and benefits of Kerberos remain a mystery to most network administrators. This book aims to pull away the curtain and reveal the magician working behind the scenes.

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.

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶



# Organization of This Book

Here's a breakdown of how this book is organized:

## [Chapter 1](#)

Provides a gentle introduction to Kerberos, and provides an overview of its history and features. It provides a gentle prologue by bringing you from the reasons for the development of Kerberos at MIT through to the latest versions of the protocol.

## [Chapter 2](#)

Continues where [Chapter 1](#) left off, presenting an introduction to the concepts and terminology that permeate the use and administration of Kerberos. The knowledge of these concepts is essential to the understanding of how Kerberos works as well as how to use and administer it.

## [Chapter 3](#)

Speaking of how Kerberos works, [Chapter 3](#) reviews the Kerberos protocol via a historical perspective that takes you through the evolution of Kerberos from an academic paper published in 1978 to the modern Kerberos 5 protocol used today. [Chapter 3](#) provides a detailed yet easy-to-follow description of how the Kerberos protocol works and describes the numerous encrypted messages that are sent back and forth.

## [Chapter 4](#)

Takes you from the realm of the theoretical and conceptual into the practical aspects involved in administering a Kerberos system. Here, the Kerberos implementations that will be discussed throughout the book are introduced, and the basics of the installation and administration of a Kerberos authentication system are described.

## [Chapter 5](#)

When things go wrong with your Kerberos implementation, [Chapter 5](#) will come in handy. [Chapter 5](#) provides a methodology for diagnosing Kerberos-related problems and demonstrates some of the more common errors that can occur.

## [Chapter 6](#)

Provides a detailed look at the practical security concerns related to running Kerberos.

## [Chapter 7](#)

Reviews some common software that can be configured to use Kerberos authentication.

## [Chapter 8](#)

Provides information about more advanced topics in running a Kerberos authentication system, including how to interoperate between Unix and Windows Kerberos implementations. This chapter also reviews how multiple Kerberos realms can cooperate and share resources through cross-realm authentication.

## [Chapter 9](#)

Presents a sample case study that demonstrates the implementation tasks presented earlier in a practical example.

## [Chapter 10](#)

Finishes off the book with a description of the future directions Kerberos is taking. We'll examine new protocol enhancements that will enable Kerberos to take advantage of new security and encryption

## Conventions Used in This Book

The following conventions are used in this book.

*Italic*

Used for file and directory names and for URLs. It is also used to emphasize new terms and concepts when they are introduced.

**Constant Width**

Used for code examples, commands, options, variables, and parameters.

*Constant Width Italic*

Indicates a replaceable term in code.



Indicates a tip, suggestion, or general note.



Indicates a warning.

## Comments and Questions

We have tested and verified all of the information in this book to the best of our ability, but you may find that features have changed, that typos have crept in, or that we have made a mistake. Please let us know about what you find, as well as your suggestions for future editions, by contacting:

O'Reilly & Associates, Inc. 1005 Gravenstein Highway North Sebastopol, CA 95472 (800)998-9938  
(in the U.S. or Canada) (707)829-0515 (international/local) (707)829-0104 (fax)

You can also send us messages electronically. To be put on the mailing list or request a catalog, send email to:

[info@oreilly.com](mailto:info@oreilly.com)

To ask technical questions or comment on the book, send email to:

[bookquestions@oreilly.com](mailto:bookquestions@oreilly.com)

We have a web site for the book, where we'll list examples, errata, and any plans for future editions. You can access this page at:

<http://www.oreilly.com/catalog/kerberos/>

For more information about this book and others, see the O'Reilly web site:

<http://www.oreilly.com>

[ [Team LiB](#) ]

◀ PREVIOUS | NEXT ▶

[ [Team LiB](#) ]

◀ PREVIOUS | NEXT ▶

## Thanks...

First, I'd like to thank my editor at O'Reilly, Michael Loukides, without whom this book would not exist. His encouragement and direction (along with his seemingly infinite patience) allowed me to finish this book while sustaining only minor injuries.

There were many people who took the time to review this text and suggest valuable changes. These people, in no particular order, include Mike Lonergan, Ken Hornstein, Frank Balluffi, Robbie Allen, Mohammad Haque, and Marcus Miller. Their constructive criticism of my early drafts helped to make this book as complete and technically accurate as possible.

I'd also like to thank the friends and co-workers who have provided support and entertainment during this process. Brian Dykstra, Brad Johnson, Mark Yu, Nan Ting, Keith Jones, and many others helped me finish this project through their encouragement over this past year.

And last but not least, I'd like to thank my parents, Arthur and Mary Garman, who encouraged me to explore my interest in computers and provided me with the Commodore 64 that sparked my imagination.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

# Chapter 1. Introduction

Who are you? It's a question with an obvious response, at least for people. Humans have the ability to distinguish one another through several senses; most commonly, we use our sense of vision to recognize people we have met before. We also can tell one another apart through other means, such as body language, speech patterns and accents, and shared secrets between people. It has even been shown that newborn babies can discern between their mother and other females solely through their scent. Our ability to recognize patterns in our surroundings provides us with this ability to determine the identity of, or *authenticate*, people we know.

However, when you bring a computer into the picture, the situation changes dramatically. Computers (at least today's computers) don't have eyes, ears, or noses. Even if they did, the current state-of-the-art in pattern recognition is still woefully inaccurate for widespread use. While there is a lot of research in this area, the most common method by far for authenticating people to computers is through passwords. A password, also known as a *shared secret*, is the one critical piece of information that determines whether the person behind the keyboard really is whom they claim to be. While humans sometimes use this shared secret method—for example, a secret handshake, or perhaps the knowledge of obscure trivia—computers almost exclusively use shared secrets to authenticate people.

There are two issues with passwords as used today for authentication. The first is a human problem. We don't like to remember a long, complex string of numbers, letters, and maybe even symbols that make up a secure password. If left to our own devices, we use simple dictionary words or maybe even our spouses' name or birthdate as passwords. Unfortunately, a "shared secret" that really isn't a secret (such as your spouse's name) is easily guessable by an attacker who wishes to impersonate you to the computer. This problem is exacerbated by the fact that, even within a company network, there are literally dozens of machines a person has access to, each of which requires its own password. As a general rule, as the number of passwords goes up, the quality of each password decreases.

The second issue is a technical problem. While the computer gives you the illusion of security by printing stars, or nothing at all, on the screen while you type your password, somehow that information must travel some communications network to a computer on the other end. The most common method that computers use to send passwords over the network is by sending the password in "clear text," that is, unmodified. While this wouldn't be a problem if each computer had a completely separate, dedicated connection to every other computer it wishes to communicate with, in reality, computer networks are a shared resource. Sending passwords over the network in the clear is analogous to standing in a crowded room shouting across the room to a friend standing on the other side.

Kerberos is a network authentication system that can help solve those two issues. It reduces the number of passwords each user has to memorize to use an entire network to one—the Kerberos password. In addition, Kerberos incorporates encryption and message integrity to solve the second issue, ensuring that sensitive authentication data is never sent over the network in the clear. By providing a secure authentication mechanism, Kerberos is an essential part of a total network security plan, providing clear benefits for both end users and administrators.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

# 1.1 Origins

The word Kerberos originates from Greek mythology, which contains the legend of Cerberus. Cerberus guarded the realm of the underworld, ruled by Hades and his wife, Persephone. What Cerberus looked like depends on whom you ask; Hesiod claims that Cerberus has fifty heads, while Apollodorus describes him as a strange mixture of creatures with three dog-shaped heads, a serpent as a tail, and heads of snakes over his back. Cerberus is most often pictured as a creature with three heads. Either way, Cerberus was a vicious creature that few dared to challenge.

The Greeks believed that when a person dies, his soul is sent to Hades to spend eternity. While all souls were sent to Hades, those people who had led a good life would be spared the eternal punishment that those who had not would have to endure. Cerberus, as the gatekeeper to Hades, ensured that only the souls of the dead entered Hades, and he ensured that souls could not escape once inside.

As the gatekeeper to Hades, Cerberus authenticated those who attempted to enter (to determine whether they were dead or alive) and used that authentication to determine whether to allow access or not. Just like the ancient Cerberus, the modern Kerberos authenticates those users who attempt to access network resources.

Like every other great figure in mythology, Cerberus had a fatal flaw that enabled some clever people to pass through Cerberus to Hades. We'll revisit the legend and discuss one such story and its modern counterparts in [Chapter 6](#).

Finally, if the ancient mythological character was named Cerberus, why is the modern authentication system called Kerberos? Simply put, they are just different spellings of the same word. In order to provide a distinction between the ancient mythology and the present-day software system, we will refer to the mythological character as Cerberus and the modern software system as Kerberos.

## 1.1.1 Modern History

The modern-day origins of the Kerberos network authentication system are a bit more mundane than the ancient mythology of Cerberus. Kerberos began as a research project at the Massachusetts Institute for Technology (MIT) in the early 1980s. The MIT faculty at the time recognized that the explosion of widely available, inexpensive computers would transform the computing industry.

### 1.1.1.1 The time-sharing model

Traditionally, computers were a large, expensive, and centralized resource that end users accessed through dumb terminals connected via serial lines. This is called the time-sharing model ([Figure 1-1](#)).

**Figure 1-1. Time-sharing model**  
*Time-sharing model*      Dumb terminal



## 1.2 What Is Kerberos?

The full definition of what Kerberos provides is a secure, single-sign-on, trusted, third-party mutual authentication service. What does that mean? Let's break that definition down into its parts and quickly describe each one.

### *Secure*

Kerberos is *secure* since it never transmits passwords over the network in the clear. Kerberos is unique in its use of *tickets*, time-limited cryptographic messages that prove a user's identity to a given server without sending passwords over the network or caching passwords on the local user's hard disk.

### *Single-sign-on*

*Single-sign-on* means that end users only need to log in once to access all network resources that support Kerberos. Once a user has authenticated to Kerberos at the start of her login session, her credentials are transparently passed to every other resource she accesses during the day.

### *Trusted third-party*

*Trusted third-party* refers to the fact that Kerberos works through a centralized authentication server that all systems in the network inherently trust. All authentication requests are routed through the centralized Kerberos server.

### *Mutual authentication*

*Mutual authentication* ensures that not only is the person behind the keyboard who he claims to be, but also proves that the server he is communicating with is who it claims to be. Mutual authentication protects the confidentiality of sensitive information by ensuring that the service the user is communicating with is genuine.

These three concepts describe the basis of the Kerberos network authentication service. We'll take a closer look at these concepts and the surrounding terminology in the following chapter.



## 1.3 Goals

The Kerberos system has several goals. It strives to improve security and convenience at the same time. First is the goal of centralizing authentication into one server (or set of servers). The Kerberos system operates through a set of centralized *Key Distribution Centers*, or KDCs. Each KDC on your network contains a database of usernames and passwords for both users and Kerberos-enabled services. Centralizing this information eases the burden on administrators, as they now only need to maintain this single username/password database. In addition, it provides an advantage to security administrators, who now only have a small set of machines on which usernames and passwords are stored, and can specially harden and protect these machines accordingly.

Kerberos provides a secure means of authentication over insecure networks. Instead of sending plain-text passwords over the network in the clear, Kerberos uses encrypted *tickets* to prove the identity of both end users and network servers. These tickets are generated by the centralized Key Distribution Centers on behalf of users who wish to authenticate to the network. When using Kerberos, user passwords are never sent over the network in the clear.

In addition, implementing the other two elements of the "three A's" (authorization and auditing—authentication, of course, is the third A) are made easier using Kerberos. While Kerberos does not directly provide authorization or auditing services, Kerberos' ability to accurately identify both users and services allows programmers and administrators to provide authorization and auditing to further enhance the security of their network. We'll talk more about what exactly authorization and auditing are in the next chapter.

[\[ Team LiB \]](#)

◀ PREVIOUS   NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS   NEXT ▶

## 1.4 Evolution

The modern Kerberos protocol has gone through several major revisions since it was first conceived as part of Project Athena. During each revision, major improvements have been made in usability, extensibility, and security.

### 1.4.1 Early Kerberos (v1, v2, v3)

The early versions of Kerberos (pre-Version 4) were created and used internally at MIT for testing purposes. These implementations contained significant limitations and were only useful to examine new ideas and observe the practical issues that arose during development and testing.

### 1.4.2 Kerberos 4

The first version of Kerberos distributed outside of MIT was Kerberos 4. First released to the public on January 24, 1989, Kerberos 4 was adopted by several vendors, who included it in their operating systems. In addition, other, large distributed software projects such as the Andrew File System adopted the concepts behind Kerberos 4 for their own authentication mechanisms.

The basics of what was to become the Kerberos 4 protocol are documented in the Athena Technical Plan. Ultimately, the details of the protocol were documented through the source code in the reference implementation published by MIT.

However, due to export control restrictions on encryption software imposed by the U.S. government, Kerberos 4 could not be exported outside of the United States. Since Kerberos 4 uses DES encryption, organizations outside of the U.S. could not legally download the Kerberos 4 software as-is from MIT. In response, the MIT development team stripped all of the encryption code from Kerberos 4 to create a specialized, exportable version. Errol Young, at Bond University of Australia, took this stripped version of Kerberos 4 and added his own implementation of DES to create "eBones." Since eBones contained encryption software developed outside of the United States, it was unencumbered by the U.S. encryption export controls, and could be legally used anywhere in the world.

Today, several implementations of Kerberos 4 still exist. The original MIT Kerberos 4 implementation is now in a maintenance mode and officially considered "dead." The kth-krb distribution, developed in Sweden, is still actively developed but it is highly recommended that new installations use the superior Kerberos 5 instead. In this book, coverage of Kerberos 4 is restricted to a discussion of the protocol in [Chapter 3](#). Most of the book covers the next version of Kerberos, Kerberos 5.

### 1.4.3 Kerberos 5

Kerberos 5 was developed to add features and security enhancements that were not present in Version 4 of the protocol. Kerberos 5 is the latest version of the Kerberos protocol and is documented in RFC 1510.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 1.5 Other Products

Many other products have been developed that either directly implement the Kerberos protocols or borrow concepts from Kerberos to implement similar authentication systems. We'll take a brief look at these alternative systems, and discuss the relationship between these systems and Kerberos.

### 1.5.1 DCE

The Distributed Computing Environment, or DCE, is a set of libraries and services that enable organizations to build cross-platform, integrated computing environments. It includes components that enable applications to communicate across a diverse set of platforms and securely locate and access information, whether it's in the same room on a local network or across the globe over the Internet. DCE provides many services to make this possible, including directory services, remote procedure calls, and time-synchronization. Most notable to our discussion, it provides a security service, which just happens to be based on Kerberos 5.

Work on DCE began in 1989, and was developed through a committee of vendors who have submitted various bits and pieces. The work was coordinated by The Open Group, an organization that is most widely known for the Motif widget set. Unfortunately, while the concepts that underlie DCE were revolutionary and ahead of their time, DCE was difficult to install and administer, and early versions were riddled with bugs. Today, DCE itself is not in wide use, but the concepts behind it have been integrated in most modern operating systems today, including Windows 2000 and above.

In 1997, The Open Group released the source code to the latest version of DCE, 1.2.2, to download for free from their web site. More information on DCE, including information on how to download Free DCE, can be found at <http://www.opengroup.org/dce/>.

### 1.5.2 Globus Security Infrastructure

The Globus Security Infrastructure is part of a larger project, the Globus Toolkit. The goal of the Globus Toolkit is to develop services that enable *grid computing*, also known as High Performance Computing (HPC) or compute clusters. Globus includes services to locate people and resources on the network, as well as submit and control compute jobs running on machines in the network. In order to perform its tasks securely, however, it needed a secure authentication and privacy mechanism. The Globus Security Infrastructure, or GSI, is the Globus Toolkit's implementation of a secure authentication system.

While the GSI operates under different principles than Kerberos, most notably through its use of public key encryption and infrastructure, it provides the same single-sign-on user experience that Kerberos does. In addition, the developers of Globus recognized the need for interoperability with existing Kerberos installations, so the Globus team has developed several tools that allow interoperability between Kerberos tickets and Globus certificates.

More information is available about the Globus Toolkit at <http://www.globus.org/>.

## Chapter 2. Pieces of the Puzzle

In the previous chapter, we examined the ideas and history behind the Kerberos network authentication system. Now we'll begin to discover how Kerberos works. Instead of introducing these concepts as they're needed in the next chapter, I feel that it is easier to understand the nitty-gritty details of Kerberos when you have a working background in the surrounding terminology. To emphasize the importance of a solid understanding in these concepts, I have set aside this chapter to introduce you to the essential concepts and terminology that surround the use and administration of a Kerberos authentication system. While you may be familiar with some of these concepts, we're going to examine each one in turn and describe how it relates to Kerberos.

Kerberos is a complex system, with many parts. It requires the proper functioning of many separate software components, and with each comes a set of terms and concepts that underlie the entire system. A complete introduction to all of these concepts is critical to the understanding of the whole.

After all of these terms have been introduced, we'll finish off by putting all of the pieces together and set the stage for the detailed description of the Kerberos protocols in [Chapter 3](#). For those who simply wish to implement a Kerberos realm and not worry about the low-level details of the protocol, this chapter will prepare you to skip directly to [Chapter 4](#).

## 2.1 The Three As

We'll start out our discussion with a topic that many network professionals deal with on a daily basis, the three As. Authentication, authorization, and auditing are a crucial part of any network security scheme, yet the distinction between them is often unclear. Each one of these components serves a separate, distinct purpose in a network security scheme. In particular, we will focus on authentication and authorization, and how they relate to each other.

### 2.1.1 Authentication

Simply put, *authentication* is the process of verifying the identity of a particular user. To authenticate a user, the user is asked for information that would prove his identity. This information can fall into one or more of three categories: what he knows, what he has, or what he is. These categories are referred to as *factors*.

The first factor, what he knows, is the most common factor used in authentication today. A secret password is generated when the user is granted access to a machine or network. That secret can either be generated by the user himself, by choosing his own password and giving it to the system administrator when he grants the user access, or automatically through some process that generates random passwords.

The second factor, what he has, is a less common but more secure alternative. An example of this type of authentication is the widely deployed RSA SecurID token. The SecurID token is a small electronic device that has an embedded encryption key and an LCD display. Every minute, an algorithm runs inside the device and updates the LCD display with a new six-digit combination. Only the person who possesses the device can tell what the correct password is. Other systems, such as smart card systems, operate on similar principles.

The third factor, what he is, enters into the realm of biometrics. Since all humans have distinguishing characteristics, biometrics measures the physical properties of some portion of our body and uses that information to authenticate users. Current biometric systems include fingerprint scanning, retina scanning, voiceprint recognition, and face recognition. Biometrics does not yet enjoy a wide market for several reasons: products are still immature for widespread use, some are very expensive (such as retina scanning), and, perhaps the most important reason of all, there is currently little software support for these devices.

Of course, an authentication system can combine these factors. For example, the RSA SecurID login process involves not only the SecurID token but also a numeric PIN. Therefore, SecurID combines the first two factors, what you have and what you know. Obviously, a system that combines more than one factor is more secure than a system which depends on only one.

The Kerberos protocol itself does not specify which authentication factors must be used. Although most implementations use a password-based system, there are implementations, such as the one present in Microsoft Windows 2000, which allow Kerberos clients to use any combination of SecurID

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 2.2 Directories

A common misconception surrounding Kerberos and other authentication technologies is that they somehow replace directories, such as the Unix `/etc/passwd` file, NIS, NetInfo, or LDAP. Along the same lines, another common misconception is that directories make good authentication systems by themselves. Therefore, a distinction needs to be made between authentication, authorization, and directories. For a real-life analogy of what roles each of these components play, see the sidebar [Confusing Authentication, Authorization, and Directories](#).

Directories contain data describing resources, such as computers, printers, and user accounts that are contained within a particular network. Directories can be as simple as a text file, such as the `/etc/passwd` and `/etc/group` files on traditional Unix systems, which list the active user accounts and their group permissions. Or a directory can be a complex LDAP directory structure, such as Microsoft's Active Directory.

Directories can contain authentication data. Authenticating "against" a directory takes two forms: a client machine can contact a directory, obtain the hashed version of the user's password, hash the password given by the user, and compare the two. This method is used by NIS, for example. The other form, employed by most LDAP authentication mechanisms, is to attempt to bind to the LDAP directory using the credentials that the user provided. If the user is granted access to the directory, the authentication is successful. The `pam_ldap` PAM module uses this latter method to authenticate against an LDAP directory.

Using Kerberos to handle authentication is superior to these methods for several reasons:

- 
- Using Kerberos tickets, users can be granted single-sign-on access to all network resources without requiring the client machine to cache the user's password. Kerberos tickets are cryptographic messages that are only valid for a relatively short period of time, typically 8-24 hours. The compromise of a user's password, on the other hand, provides an attacker the ability to masquerade as the legitimate user for a much longer period of time—specifically, until the password is changed or expires.
- 
- With Kerberos, the user's password is never sent in the clear over the network during the login process.
- 
- Kerberos defines a widely adopted and standardized protocol that is suited for authentication.

Therefore, while a directory may contain authentication information (for example, Microsoft's Active Directory stores the Kerberos database in its LDAP store), it is preferable to use Kerberos to perform authentication rather than using the directory for authentication directly.



[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 2.3 Privacy and Integrity

Next, we'll review some concepts that are integral to keeping communications on computer networks secure. In particular, we will discuss the roles of encryption and message-integrity algorithms. The distinction between encryption and message-integrity is important, as we'll see later in the discussion of Kerberos encryption types. Those familiar with encryption and message integrity can skip to the next section, which describes the Kerberos-specific terminology.

### 2.3.1 Encryption

The modern word cryptography is derived from two ancient Greek words, *cryptos*, which means hidden or secret, and *graphein*, or writing. Kerberos uses cryptography to provide encryption and decryption of its messages over the network. Therefore, encryption refers to the process of converting a message, or plaintext, into gibberish, which if intercepted, does not reveal the contents of the original message. Governments and corporations have long employed encryption to keep their information secure from prying eyes. The emergence of the Internet, where any network administrator can monitor and read traffic on her network and any traffic passing through her network, has forced software makers to build encryption into every day software programs. Kerberos uses encryption not only to protect the authentication exchanges it sends and receives from snoopers, but also to prevent hackers from creating fake messages.

There are many different ways of encrypting data. These methods are referred to as encryption algorithms, or in Kerberos-speak, encryption types. There are several different encryption types that are supported in Kerberos 5 implementations. The most widely supported encryption type is DES, but work is underway to replace it with Triple DES and the new Advanced Encryption Standard (AES). Another widely used encryption type is the RC4 algorithm, which is used primarily in Microsoft's implementation of Kerberos.

The advantage of moving to stronger encryption algorithms is protection against brute-force cryptanalysis. We'll take a look in more detail about brute-force attacks against the encryption algorithms in Kerberos in [Chapter 6](#).

### 2.3.2 Message Integrity

While encryption provides privacy, message integrity ensures the recipient that the message was not tampered with during transit. While encryption as it is used in Kerberos gives you message integrity for "free," since only the two end points have the required key to encrypt and decrypt messages, there are specialized message-integrity algorithms that can ensure message integrity without the overhead of encryption. You will see message-integrity algorithms referred to as one-way hashes, or just hashes.

Hashes work as mathematical one-way functions. They take an input message that is arbitrarily long, run it through a mathematical algorithm, and output a fixed size (typically 64-256 bits) message that represents the input. The idea behind the hash function is that while it is easy to calculate the hash output for a given input, it is mathematically hard to go the opposite way and derive an input that produces the

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 2.4 Kerberos Terminology and Concepts

Now we'll begin to examine terminology that is specific to the Kerberos authentication system. There are many parts to Kerberos, and each has a name that will be defined here and used throughout the rest of the book. The descriptions that follow suffice for implementing a Kerberos realm, but the details of how these work are covered in the next chapter, where we will examine the protocols in detail.

### 2.4.1 Realms, Principals, and Instances

Every entity contained within a Kerberos installation, including individual users, computers, and services running on servers, has a *principal* associated with it. Each principal is associated with a long-term key. This key can be, for example, a password or passphrase. Principals are globally unique names. To accomplish this, the principal is divided into a hierarchical structure.

Every principal starts with a username or service name. The username or service name is then followed by an optional instance. The instance is used in two situations: for service principals (which we'll discuss later), and in order to create special principals for administrative use. For example, administrators can have two principals: one for day-to-day usage, and another (an "admin" principal) to use only when the administrator needs elevated privileges.

The username and optional instance, taken together, form a unique identity within a given *realm*. Each Kerberos installation defines an administrative realm of control that is distinct from every other Kerberos installation. Kerberos defines this as the realm name. By convention, the Kerberos realm for a given DNS domain is the domain converted to uppercase. So, for example, Wedgie International, which owns the domain name `wedgie.org`, would create a Kerberos realm for its users named `WEDGIE.ORG`.

While it is the convention to make the realm name equivalent to the DNS domain name, it is not necessary to do so. It certainly makes configuration easier, as we'll see later on, but it is perfectly legal to have a realm name of, say, `MYREALM.BOGUS` when your domain name is `wedgie.org`. Also note that realms are case-sensitive (unlike domain names), so the realm `MyRealm.BOGUS` is different from `MYREALM.BOGUS`.

Now let's examine a Kerberos principal that has been assigned to John Doe, who works in the IT department of Wedgie International:

```
jdoue@IT.WEDGIE.ORG
```

This is the simplest form a principal can take, and is a valid principal under both Kerberos 4 and Kerberos 5. This principal represents the username `jdoue`, with no instance, and a realm of `IT.WEDGIE.ORG`.

#### 2.4.1.1 Service and host principals

Users aren't the only ones assigned principals in a Kerberos realm; hosts and servers offering Kerberos

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 2.5 Putting the Pieces Together

Now that we've covered the basic topics that you'll need to understand Kerberos, let's begin to put all of these pieces together by examining the credential cache above.

Inside the credential cache, I have obtained an initial Ticket Granting Ticket through the Authentication Server (this is the first ticket out of three). By logging into this system, the system created this credential cache and obtained a TGT for me. During my log in session, I also logged into a host called `cfs.wedgie.org`, which has a Kerberized telnet daemon running on it. Because I was using Kerberos authentication, I was able to log into `cfs` without typing a password; instead, my telnet client obtained a service principal from the Ticket Granting Server, and used that ticket to contact the Kerberized telnet on `cfs`. Later, I did the same, except this time I logged into `web.wedgie.org`.

During this time, after logging in to three machines (including my initial authentication to Kerberos), I have only typed in my password once. The Kerberos software requested, generated, and sent tickets on my behalf as necessary to transparently authenticate me to the other machines as I accessed them. As a user, all of this happens behind the scenes. Now we'll peel back the curtain, and uncover the magic that occurs behind the scenes.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

# Chapter 3. Protocols

The previous two chapters introduced the major concepts that underlie the Kerberos authentication system, and presented a short, high-level discussion of how Kerberos performs its magic. This chapter continues that discussion by drilling down into the nitty-gritty of the Kerberos protocol and presenting it on a fundamental level.

Creating a protocol that verifies the identity of two endpoints on a network given an underlying network that provides no security is a daunting task. Kerberos was designed under the assumption that attackers can read, copy, and create network traffic at will.

As you now know, there are two versions of Kerberos that are currently in wide usage: Kerberos 4 and Kerberos 5. This chapter covers the protocol details of both. While the concepts and protocol design of both Kerberos 4 and 5 are very similar, there are major differences between their byte-level protocol and implementation.

The original Kerberos 4 protocol was never published apart from the Kerberos 4 source distribution. As such, the Kerberos 4 source code from MIT is the only official documentation of the Kerberos 4 protocol. On the other hand, the newer Kerberos 5 protocol is extensively documented in RFC 1510, and also through a series of documents that are collectively known as the Kerberos Clarifications.

The basic operation of Kerberos is based on a paper published in 1978 by Needham and Schroeder. Since the Needham and Schroeder protocol is the basis upon which Kerberos is built, we will begin our discussion there.

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

## 3.1 The Needham-Schroeder Protocol

Roger Needham and Michael Schroeder of the Xerox Palo Alto Research Center published a paper in December of 1978 describing their framework for designing a secure network authentication system. The paper, entitled "Using Encryption for Authentication in Large Networks of Computers," described two different protocols that could be implemented to provide a reliable, secure authentication service for a distributed network of computers. The first protocol described in the paper uses private key encryption, and it is this protocol that forms the basis of the Kerberos network authentication protocol.

Needham and Schroeder outlined several assumptions around which they designed their protocol. One assumption, the ability for a malicious attacker to capture packets in-transit on the network, modify them, and send packets of his own design, was described by the authors as an "extreme view," yet now is regarded as a routine requirement for any secure network protocol. Designing a protocol that is resistant to these types of attacks is difficult, and I'll point out the specific design decisions that were made to thwart them as I discuss the protocol.

Other assumptions made by the authors, however, did not hold up as well in practice as they did on paper. The assumption that users' secret keys are not readily available through an exhaustive search has not held up in the hostile environments in which Kerberos operates. No matter how much education you provide users, users will continue to choose poor passwords. The Needham and Schroeder protocol, and consequently the basic Kerberos protocol, provides no protection against an offline brute force or dictionary attack against a user's secret key, as we'll see in [Chapter 6](#).

The Needham-Schroeder protocol defines three participants in the protocol exchange: a client machine, a server that the client wishes to access, and an authentication server. The client is any machine that requests authentication; usually, it's a user's personal desktop. The server is any application server, say a mail server, which provides a service the client wishes to contact. Finally, the authentication server is a dedicated server that holds a copy of the encryption keys for all users and servers on the network (the "trusted third-party"). This should sound familiar; these are the same three players involved with the Kerberos protocol.

The concept behind the Needham-Schroeder protocol is not to authenticate the user directly by sending a password or password equivalent (such as a hash of the password) to the authentication server. Instead, the Needham-Schroeder protocol provides a mechanism to securely distribute a short-lived encryption key to two parties (a client and a server) so their communication can be secured with the encryption key. The verification of each endpoint's identity happens to be a side effect of this key exchange process. We'll see what this means as we discuss how the protocol works.

The protocol begins with the client contacting the authentication server. The client sends the authentication server a message containing the its own identity and the identity of the application server that it wishes to contact. In addition, the client includes a nonce, or a random value, with its request. We'll see why this random value is important in a moment. [Figure 3-1](#) illustrates the information sent by the client to the authentication server.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶



## 3.2 Kerberos 4

The Kerberos 4 protocol is largely based on the Needham-Schroeder protocol, with two major changes.

The hosts involved in the Kerberos 4 protocol exchanges map directly to the principals involved in the Needham-Schroeder protocol. The authentication client is a Kerberos 4 user workstation, and the authentication server maps to a Kerberos 4 Key Distribution Center.

The first change to the Needham-Schroeder protocol reduced the amount of network messages sent between the client and the authentication server. The original Needham-Schroeder protocol did not have a dependence on a network time source, but the cost was an extra two message exchanges. The last two message exchanges in the Needham-Schroeder protocol establish that there is no man in the middle posing as the authentication server, and that the session key is not a replay. In the Kerberos 4 protocol, replay is thwarted through an authenticator message that is constructed of the local time of the client encrypted with the newly-negotiated session key of the connection. While this requires time synchronization between all hosts involved, it does reduce the number of network messages required per authentication exchange.

The second, more significant, change to the basic protocol creates the concept of a Ticket Granting Ticket, which allows users to authenticate to multiple application servers while entering their authentication secret only once. If the original Needham-Schroeder protocol were implemented as-is, a user would need to enter her password every time that she wishes to log into an application server. One of the major design goals for Kerberos was to create a single-sign-on system in which users only need to enter their credentials once per day, and all future authentication requests are handled transparently, without user intervention.

As a result, the Kerberos 4 protocol is split into two logical components: the Authentication Server and the Ticket Granting Server. Note that there is an unfortunate clash in terminology here. The Kerberos Authentication Server should not be confused with the Needham-Schroeder authentication server; the former performs a subset of the services of the latter, as we'll see in a bit. To keep the distinction clear, references to the Kerberos Authentication Server and Ticket Granting Server will be capitalized or abbreviated as AS or TGS, respectively. While these components are usually implemented as a single program that runs on the KDC, they are logically separate processes.

Still other changes reflect the realities of the security of today's computer networks. The original Needham-Schroeder protocol assumed that all secrets involved, including the user's long term key, the application server's long term key, and the session key that is randomly generated by the KDC, are always kept secret. In reality, machines are compromised and people give away their passwords. In addition, the single-sign-on capability provided in Kerberos 4 means that users' workstations will have cached credentials that, if left unguarded, can be used by an attacker to impersonate the user. Therefore, Kerberos 4 introduces limited lifetimes for credentials, enforced by the Kerberos KDC and Kerberos libraries. Without ticket expiration, a user could log in once, and never have to log in again (provided that their credentials are never removed from the workstation). Lifetimes ensure that users must verify their identity periodically—say, once a day—by entering their password again. They also close the window of vulnerability in the case of stolen credentials

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 3.3 Kerberos 5

If you look strictly at the feature set, Kerberos 5 is an evolution of Kerberos 4. The Kerberos 5 protocol contains all of the functionality present in the Kerberos 4 protocol, but with many extensions. However, from an implementation perspective, Kerberos 5 is a completely new protocol, and looks nothing like Kerberos 4 on the inside. In this section, we'll examine the new features present in Kerberos 5 as well as the new infrastructure provided by the protocol to make these features work.

The Kerberos 4 protocol had its share of shortcomings: it had a rather obtuse structure (for example, instead of standardizing on one byte order, it had a flag to specify which byte order was used to send a particular message) and it wasn't expandable, since many of its fields had fixed sizes. This limitation led to other problems, most notably the dependence on single-DES encryption keys. At the time that Kerberos 4 was developed, a brute-force attack against DES was still prohibitively expensive in terms of both resources and time. As computer speed continues to grow exponentially, it is now within the realm of well-funded adversaries to mount a brute-force attack against DES. Therefore, a more secure encryption algorithm with a longer encryption key size is needed. Unfortunately, since all of the fields in Kerberos 4 are fixed size, there is no way to retrofit Kerberos 4 with another encryption algorithm.

Another feature that users and administrators alike demanded from a new version of Kerberos was support for credential forwarding and delegation. Credential forwarding enables users to transfer their tickets to a remote server once they are authenticated to it. For example, take a user who has just logged in remotely to an application server via Kerberized telnet. Using Kerberos 4, if the user wishes to authenticate to, say, a file server, from the application server, she's essentially stuck. She must re-login to Kerberos from the remote server through kinit to acquire a new Ticket Granting Ticket on the remote server. This introduces security problems, since in order to re-authenticate to Kerberos, the user must retype her password, which in the case of telnet, is probably being sent in clear text over the network. In addition, the system no longer provides single-sign-on.

Kerberos 5 introduces support for credential forwarding so that in the previous example, when the user logs into the remote application server, her Ticket Granting Ticket is securely transmitted to the remote server and can be used by applications on that remote server to transparently authenticate her to further Kerberos services.

However, with these enhancements and extensibility comes complexity. In order to create an extensible protocol that can be implemented on multiple platforms by multiple vendors, and ensure that all these implementations can interoperate, the Kerberos development team chose to use a technology known as ASN.1 to describe their new Kerberos 5 protocol. ASN.1 allows protocol designers to create protocols with an abstract language, automating implementation details and allowing for future extensions. We'll talk more about ASN.1 and how ASN.1 is used to define the Kerberos 5 protocol messages in a bit.

Kerberos 5 strives to be as compatible with older clients and application servers as possible. In order to ensure compatibility with old Kerberos 4 software, Kerberos 5 provides the Kerberos 5-to-Kerberos 4 ticket translator service (commonly known as krb524). This service takes a valid Kerberos 5 ticket as input (the only requirement is that the ticket and session key encryption types be single-DES, as that is

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 3.4 The Alphabet Soup of Kerberos-Related Protocols

Finally, there are several protocols that, while strictly speaking are not directly related to Kerberos, will be encountered when implementing a Kerberos authentication system.

### 3.4.1 The Generic Security Services API (GSSAPI)

The Generic Security Services API, as the name implies, is not specific to any authentication technique. Therefore, its mention in a book on Kerberos may seem a bit out of place. However, GSSAPI is widely used by protocol implementers as a means to implement Kerberos 5 support in their applications. By using GSSAPI, a protocol gains the ability to use other strong authentication methods "for free," and the GSSAPI layer also shields implementers from the complexities of the raw Kerberos 5 API.

GSSAPI is geared toward developers of client/server applications who wish to add strong authentication support to their protocols. It provides a generic interface and message format that can encapsulate authentication exchanges from any authentication method that has a GSSAPI-compliant library. GSSAPI insulates application programmers from the specific programming interface for particular authentication methods. GSSAPI also provides a standard message format so that protocols can support many different authentication methods without changing the protocol itself. GSSAPI does not define a protocol, authentication, or security mechanism itself; it instead makes it easier for application programmers to support multiple authentication mechanisms by providing a uniform, generic API for security services.

Most Kerberos 5 implementations also include a GSSAPI library. This means that all applications that support GSSAPI also support Kerberos 5. The notable exception is the Windows Kerberos implementation, which does not include GSSAPI support but instead includes a Microsoft-specific API, the Security Support Provider Interface (SSPI). SSPI is not API-compatible with GSSAPI; that is, programs written for GSSAPI will not compile with SSPI. Instead, applications written for SSPI can be made to be wire-compatible with GSSAPI applications. Therefore, an SSPI client can communicate with a GSSAPI server. Microsoft provides some example code that demonstrates how to achieve this network message-level interoperability.

While GSSAPI is mostly standardized, there are still some differences between the C language bindings of the available implementations, particularly the MIT and Heimdal implementations of GSSAPI. During the configuration stage, most open source software will detect which GSSAPI implementation you have and compile the appropriate code to work with it, but some software may only work with one or the other. Work to unify these APIs is ongoing.

The relevant standards documents defining GSSAPI include RFC 2743, which documents the basic GSSAPI message types, RFC 1509, which defines the C language bindings and API, and RFC 1964, which defines the Kerberos 5 GSSAPI mechanism.

### 3.4.2 The Simple and Protected GSSAPI Negotiation Mechanism (SPNEGO)

[\[ Team LiB \]](#)

[← PREVIOUS](#) [NEXT →](#)

[\[ Team LiB \]](#)

[← PREVIOUS](#) [NEXT →](#)

## Chapter 4. Implementation

The previous chapters discussed the concepts and theory that form the basis of the Kerberos authentication system. Now, armed with a solid background, we're ready to tackle the actual implementation of a Kerberos authentication system from start to finish. This chapter prepares you to install the Kerberos KDCs in your network and also the Kerberos libraries on servers and client machines. We will continue the process in [Chapter 7](#) by detailing installation processes for Kerberized application software.

[\[ Team LiB \]](#)

[← PREVIOUS](#) [NEXT →](#)

[\[ Team LiB \]](#)

[← PREVIOUS](#) [NEXT →](#)

## 4.1 The Basic Steps

We'll begin by outlining the steps for establishing a Kerberos realm. During this chapter, we'll follow these steps to create a sample Kerberos realm:

1.
  1. Plan your installation.
  - 2.
  2. Install the KDC software.
  - 3.
  3. Establish the Kerberos realm and create an administrative user principal.
  - 4.
  4. Add user principals to your realm.
  - 5.
  5. Install Kerberized server software, and install service principals for the server software as necessary.

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

## 4.2 Planning Your Installation

Your Kerberos implementation will be an important part of your network. As such, the Kerberos service needs to be always available, responsive, and in the event of failure, easily restored from backup. Therefore, integrating Kerberos authentication into your network calls for some planning.

The first consideration is what exactly you'll be using Kerberos for. The answer to this question depends on whether you'll need compatibility with Kerberos 4 clients/services or not. We'll handle the simple case where you have no need to service Kerberos 4 clients or services first.

In this case, you'll be able to implement a Kerberos 5-based solution with no need for backwards compatibility with Kerberos 4-based systems. All of the KDCs we'll cover here will be able to handle Kerberos 5 clients, and there will be no need to enable any optional Kerberos 4 compatibility.

On the other hand, if you have to support Kerberos 4 services or clients, you'll need to plan a bit more carefully to integrate those legacy components into your Kerberos implementation. Typically, in this situation, you'll want to stick with a Unix-based KDC, since these have built-in support for the older Kerberos 4 protocol.

Your only option when dealing with Kerberos 4 client machines (machines which will be authenticating end users) is to use a KDC with direct support for Kerberos 4. This limits you to Unix-based KDCs. However, if you are supporting a Kerberos 4-based service (such as AFS), you can get away with a mixture of a Windows domain controller (or another KDC that supports only Kerberos 5 directly) and a machine that is running the Kerberos 5-to-4 ticket translator daemon (known as `krb524`) that is included with both MIT and Heimdal. We'll talk about this option in more detail in [Chapter 8](#).

You'll want to determine the number of KDCs you'll deploy in your network. Since authentication requests to the KDC can be easily handled with today's overpowered processors, a single or dual processor machine should suffice for thousands of clients. Note that this applies to Unix-based system running only a KDC; Windows domain controllers function as much more than just a Kerberos KDC and therefore may have a higher server-to-client ratio than a dedicated Unix KDC. I won't go into detail about Active Directory planning here; readers interested in more detailed discussion about Active Directory should refer to *Active Directory* by Robbie Allen and Alistair G. Lowe-Norris (O'Reilly).

You should take into consideration not only how many authentication clients you'll 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 KDCs. Each authentication exchange requires time for at least one full round trip between client and KDC, and if this latency is long—for example, traveling through a satellite uplink or across congested Internet backbones—then users' authentication requests will become noticeably slow. Consequently, you want to position your KDCs so that they are as close to the clients network-wise as possible.



[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 4.3 Before You Begin

Kerberos requires the proper functioning of several external services. Notably, the clocks on all machines participating in your realm must be synchronized to within a few minutes, and a working DNS domain should be established with forward and reverse mappings to at least the Kerberos KDC and application servers you intend to Kerberize.

First, NTP (the Network Time Protocol, official home page at <http://www.ntp.org/>) should be installed on every server and KDC in your network. NTP will synchronize the clocks of these machines to a central source, which can be either a local time source (such as a GPS unit receiving time signals from the GPS satellites, or a Cesium time source, if you happen to have an atomic clock available at your site) or a remote Internet-accessible time server. While it is possible to set up all of your machines to synchronize over the network to an external, publicly available time server, site administrators are strongly encouraged to set up a centralized time source for their network, and set up other machines on the network to synchronize to that server. The time server machine can then synchronize to an accurate time source, such as a public time server.

The details of setting up NTP are beyond the scope of this book, and online references and software are readily available from the above URL. Clients will need to be synchronized also, but hand synchronization is sufficient unless you have enough control over the client systems to install NTP on them as well. While Kerberos does require synchronized clocks, Kerberos implementations typically provide for a plus-or-minus five-minute error when comparing times. Even with this built in time slack, poorly synchronized clocks are the root of many problems that are encountered when using Kerberos. Therefore, investing time up front to ensure accurate clocks on all machines on your network is time well spent.

Windows machines that are part of an Active Directory domain automatically synchronize their clocks to the domain controller's clock through the Windows Time Service. Unix systems typically do not have NTP installed and configured out of the box, so manual configuration is required to keep these systems' clocks in sync.

The proper functioning of your DNS server and client DNS resolver libraries are also important, and some time should be spent ensuring that they are functioning properly before installation. Every Unix system has different semantics for what it thinks its own hostname is: some use the fully qualified domain name (hostname plus domain name); others use just the hostname component. And to make matters worse, some Unix systems map their own hostname to 127.0.0.1 (the loopback IP address). Observe the following guidelines to avoid most DNS-related problems when implementing and maintaining your Kerberos system:

*Ensure valid forward and reverse DNS mappings for machines in your domain*

All machines in your domain that will participate in your Kerberos realm need to have working DNS entries, both forward and reverse. This means that, for every machine, a DNS entry exists that maps the hostname to an IP address, and a reverse entry exists for that IP address mapping it back to the original hostname.

*Set all system hostnames to the fully qualified domain names*

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 4.4 KDC Installation

With the boring planning out of the way, we're ready to get down to the nitty-gritty of installing a KDC. We'll cover each implementation in turn, and I'll take you through building the software, all the way to setting up test users and distributing the Kerberos database among a network of slave KDCs.

This section covers the topics required to set up the KDC software. It does not cover related topics, such as security of the underlying operating system; these security-related topics can be found in [Chapter 6](#).

### 4.4.1 MIT

Since the MIT Kerberos distribution is available as open source, there are two ways to install it: building it from source, or obtaining a binary distribution from your Unix vendor. As for Linux, many of the popular Linux distributions have pre-built binary packages of MIT Kerberos 5.

We'll cover building MIT Kerberos from source for several reasons. One, many people feel more comfortable building security-sensitive applications directly from source. Also, some Unix distributions do not offer a pre-built MIT Kerberos distribution. Finally, by building from scratch, we can establish a common path structure and feature set that's independent of any tweaks individual vendors have decided to include in their pre-built versions. If you prefer to install from pre-built packages, such as RedHat RPMs, feel free to skip over this section and continue on to the next section, [Section 4.4.1.2](#).

The MIT distribution is available for users in the U.S. and Canada at the MIT Kerberos home page, located at <http://web.mit.edu/network/kerberos-form.html>. Users in other countries should visit the Cryptography Publishing Project's MIT Kerberos download page at <http://www.crypto-publish.org/mit-kerberos5/index.html>. The MIT Kerberos 5 source distribution is available at both sites, and the source code tarball available at both are bit-by-bit identical.

The source code distribution is signed with the PGP key of Tom Yu, a member of the MIT Kerberos development team. It is recommended that you verify the distribution you've downloaded against this signature, to ensure that the file has not been tampered with. If you obtained the source distribution from MIT, the signature is located within the tar file. If you obtained the source distribution from the Cryptography Publishing Project, the signature is located on the download web page. To verify the signature using the GNU Privacy Guard (a freely available implementation of PGP), ensure that the signature and the .tar.gz source tarball are in the same directory, and type:

```
% gpg --verify krb5-1.3.tar.gz.asc
```

where 1.3 is replaced with the latest version of the Kerberos distribution. Next, you'll unpack the source distribution tarball:

```
% gzcat krb5-1.3.tar.gz | tar xfv -
```

This creates a krb5-1.3 directory, under which all of the files in the Kerberos source distribution are placed. Inside of the krb5-1.3 directory, there is a src and a doc directory. Since we want to build the

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 4.5 DNS and Kerberos

A properly functioning DNS server for your domain and functioning DNS resolvers on machines participating in your Kerberos realm is essential for the proper operation of your realm.

Traditional Unix Kerberos 5 implementations use the flat file `/etc/krb5.conf` file for hostname-to-Kerberos realm mapping, much like `/etc/hosts` can be used for name-to-IP mapping. The Kerberos configuration file contains two major pieces of information: the DNS domain name to Kerberos realm mappings, and 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 RFC 2052. 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 clients.



Note that while Windows will use DNS to locate KDCs, it will not use DNS to locate any KDCs for any non-Windows Kerberos realms. Configuration information for non-Windows Kerberos realms must be entered manually using the `ksetup` tool. More information on this tool and enabling interoperability between Windows and other Kerberos implementations can be found in [Chapter 8](#).

### 4.5.1 Setting Up KDC Discovery Over DNS

In order to use KDC discovery over DNS, the following records should be placed in the zone file corresponding to the Kerberos realm. In most cases, since the Kerberos realm name is simply an uppercase version of the DNS domain owned by the organization, these DNS entries are placed into the organization's existing DNS zone file. However, if the Kerberos realm and DNS domain differ, then a new zone must be created with the name of the Kerberos realm.

A SRV DNS Resource Record describes the KDCs available for a particular realm and contains the following information:

*Service*

This item represents the service that this SRV record provides location information for. The service name for the Kerberos KDC is always `_kerberos`. Other Kerberos related services include `kerberos-adm` for the `kadmin` server, and `knasswd` for the Kerberos 5 password-changing service.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 4.6 Client and Application Server Installation

Before a client machine or Kerberized application server can use Kerberos authentication, it must have the proper libraries, configuration files, and in some cases, service and host principals added to a local keytab file. We will limit the discussion (for now) to using the same Kerberos implementation both on the KDC and the clients. Most of the setup information below can be used to interface with a different vendor's KDC, but we will take a closer look at interoperability later in [Chapter 8](#).

### 4.6.1 Unix as a Kerberos Client

There are three major steps to setting up a Unix-based Kerberos client or Kerberized application server: compiling the distribution, installing configuration files, and creating host and service principals if necessary. The first step, compiling the distribution, has already been discussed in the "Building the distribution" sections (under the appropriate heading for your chosen Kerberos implementation); follow the directions to build and install the client libraries.

Next, we'll create configuration files on each of the clients. Both MIT and Heimdal use a configuration file located in `/etc/krb5.conf`. This configuration file contains the name and addresses of all KDCs that the client can communicate with. Alternatively, this information can be placed in DNS, as discussed in [Section 4.5](#). Since most Kerberos installations are still using configuration files, we'll discuss them.

We saw a simple `krb5.conf` file earlier, when we set up the MIT KDC above. That template still applies for clients, and in fact, the `/etc/krb5.conf` configuration file can be copied straight from the KDC to all of the clients. If you want to tweak the configuration file anyway, there are three stanzas that are important for client configuration: `libdefaults`, `realms`, and `domain_realm`.

Let's start with a sample configuration file. It should look familiar; it is the same one presented in the KDC installation section ([Section 4.4](#)):

```
[libdefaults]
    default_realm = WEDGIE.ORG

[realms]
    WEDGIE.ORG = {
        kdc = freebsd.wedgie.org:88
        admin_server = freebsd.wedgie.org:749
        default_domain = wedgie.org
    }

[domain_realm]
    wedgie.org = WEDGIE.ORG
    .wedgie.org = WEDGIE.ORG
```

The `libdefaults` stanza contains parameters that apply to all applications using the Kerberos libraries. The only option that you may have to tweak in this stanza is the `default_realm` option. This option specifies the default realm that the Kerberos libraries will use when trying to find a service principal for a given service. Normally, the Kerberos realm and DNS domain name are the same, but they can be different. If your Kerberos realm differs from the DNS domain that this machine resides in, then you should set this parameter to your Kerberos realm.



[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

## Chapter 5. Troubleshooting

Working with Kerberos can seem like an exercise in futility. The combination of complex software, interoperation between Kerberos implementations and diverse operating systems, and terse (at best) error messages tend to bring premature balding to the administrators responsible for the smooth operation of their systems. However, working with Kerberos does not have to be this frustrating.

To provide a systematic approach to solving problems in Kerberos, this chapter begins with a discussion of the various debugging tools and techniques, including a sample decision tree to follow when facing a new problem. Becoming familiar with these tools and techniques before disaster strikes will be helpful when the inevitable problem occurs. The second section will use those tools and techniques to diagnose some typical problems that occur in a Kerberos system. It will describe the symptoms of these common problems and then show possible solutions to solve them.

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

## 5.1 A Quick Decision Tree

So you're having a problem with your Kerberos installation. The first step to solving this problem, like debugging any other issue, is to narrow down the root cause. We'll determine if the problem falls into three distinct categories, and continue our analysis from there.

An easy way to categorize an error involving Kerberos authentication is through what tickets the client can acquire for a service. Let's look at the three top-level categories:

- 
- Client can't get an initial Ticket Granting Ticket. This is most likely a client-specific problem, especially if logging in with the user principal and password works on other clients. Of course, it could also mean that the password entered for the user principal is incorrect.
- The most likely culprits include time-synchronization problems and issues reaching the Kerberos server due to misconfiguration of the client. It is also possible that the client does not share a compatible encryption type for the users' secret key with the KDC. This can happen, for example, when attempting to interoperate between a Unix client and a Windows domain controller. By default, Windows domain controllers create user entries with an RC4-HMAC encryption type, which most Unix Kerberos implementations do not understand. Newer versions of Heimdal and MIT Kerberos 5 will support this encryption type.
- 
- Client has valid TGT but gets error before a service ticket is acquired. Once again, this is most likely a problem with the client. The usual suspects in this scenario include Kerberos misconfiguration on the client, which we'll cover in a subsequent section.
- Another possibility is that the service principal that the client requested simply does not exist. Examining the KDC log files is a good way to determine if the client is reaching the KDC, and if so, if it is attempting to acquire tickets for a principal that does not exist, or is in a different realm.
- 
- Client has valid TGT and service ticket, but reports error on connection to Kerberized service. There is most likely a problem with the server. At this point, the client has received a ticket for the service and presented it to the service for authentication. The most common cause for a failure at this point is a mismatch in the encryption types or key version numbers for the Kerberos service between the service's keytab and the KDC. The KDC may have issued a ticket with an encryption type that the service did not understand, or perhaps the service's keytab contains an incorrect encryption key. The service may not be able to read its keytab at all; ensure that the service's keytab is readable by the user the service runs as, and that the appropriate configuration is in place to point the service to the location of the keytab.
- Another possibility is that the server's DNS or Kerberos configuration file is not configured correctly. Make sure that the target server's hostname can be correctly resolved by using diagnostic tools such as ping and nslookup.

General root causes that should be investigated include time synchronization and correct hostname and

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 5.2 Debugging Tools

The MIT Kerberos distribution includes a small sample Kerberized client/server application. These example applications are located in the `src/appl/sample` subdirectory of the MIT Kerberos 5 distribution.

Just like any other Kerberized server, the sample server requires a service principal and access to the secret key associated with that principal through a keytab file. By default, the sample server uses a principal name of "sample," with an instance of the hostname that it is running on. If you're having trouble with a particular service principal, the sample server and client can use any principal name to communicate with each other, given the sample server has read access to the service's keytab file.

The command-line arguments accepted by the sample server are:

```
> ./sserver -h
usage: ./sserver [-p port] [-s service] [-S keytab]
```

The `-p` argument specifies what TCP port that the server will listen on for client requests. If this argument isn't specified, then `sserver` will immediately exit. The `-s` option can be used to specify a particular service principal (instead of the default, "sample"). For example, the host principal can be specified by `-s host`. Finally, the `-S` option specifies a keytab file in which the server can find the secret key for the service principal. By default, `sserver` will use `/etc/krb5.keytab`.

Ensure that a valid keytab entry for the principal you're using to test exists in a keytab file and is readable by the user you're starting `sserver` as. Note that the server won't test for the readability of the keytab until a client connects to it, and your client will report "Permission denied".

The command-line arguments that the client accepts are similar:

```
> ./sclient
usage: ./sclient <hostname> [port] [service]
```

A successful exchange looks like the following, assuming that you have shells open on both of the hosts `freebsd` and `slave`, and their prompts are `freebsd>` and `slave>` respectively:

```
freebsd> ./sserver -p 8888 -s sample -S /tmp/sample.keytab
slave> ./sclient freebsd 8888 sample
sendauth succeeded, reply is:
reply len 27, contents:
You are jgarman@WEDGIE.ORG
```

If you choose a service name other than "sample," specify the service name on the command lines to both the server and the client. Just like with any other Kerberos client/server application, you'll see that you now have Kerberos tickets for the sample service principal:

```
client> klist
Ticket cache: FILE:/tmp/krb5cc_p27758
Default principal: jgarman@WEDGIE.ORG

Valid starting    Expires          Service principal
02/26/03 02:34:47    02/26/03 10:58:19    krbtgt/WEDGIE.ORG@WEDGIE.ORG
02/26/03 02:35:51    02/26/03 10:58:19    sample/freebsd.wedgie.org@WEDGIE.ORG
```

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 5.3 Errors and Solutions

With the debugging tools presented above, we'll run through a few problem scenarios, from the initial symptoms of a problem through to its solution.

### 5.3.1 Errors Obtaining an Initial Ticket

Several errors can occur when attempting to obtain an initial Ticket Granting Ticket from a Kerberos KDC. Since there are many ways to obtain a TGT, such as through integrated login with a PAM Kerberos module, the best way to narrow down problems is by using the Unix kinit program manually. This will work even if your KDC is a Windows domain controller, given that the principal you're testing has been set up for DES encryption (see [Chapter 8](#)).

Let's go through a few examples:

```
> kinit
Password for jgarman@WEDGIE.ORG:
kinit(v5): Preauthentication failed while getting initial credentials
```

If your realm requires pre-authentication (see [Chapter 6](#)), then this message is typically just Kerberos-speak for "incorrect password." Note that Windows domain controllers require pre-authentication by default. Also note that this message can result from a client that does not support the pre-authentication type required by the KDC. However, all of the Kerberos implementations we cover here support the Encrypted Timestamp (PA-ENC-TIMESTAMP) pre-authentication method. Of course, if you are interoperating with a Kerberos implementation that does not support pre-authentication, and your realm requires it, you will have to disable pre-authentication in the KDC policy.

Next, there is a possibility that the KDC could not find an appropriate encryption key with which to encrypt the response. When a Kerberos 5 client contacts a KDC through the AS exchange for an initial Ticket Granting Ticket, the client sends a list of encryption types that it understands. If the KDC cannot find a secret key associated with one of the encryption types included in the request, it will return an error.

Encryption type mismatches can also occur later on in the Kerberos exchange, and we'll cover that in a later section. Errors obtaining an initial ticket can also be caused by hostname/DNS misconfiguration, or a missing or incorrect Kerberos configuration file. These possibilities will also be covered soon in a later section.

Finally, another common error that can cause the pre-authentication failed message is a clock synchronization problem (which can cause all sorts of other strange problems, as well), covered next.

### 5.3.2 Unsynchronized Clocks

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## Chapter 6. Security

Cerberus, the fierce three-headed creature that guarded the entrance to Hades, prevented the living from entering the underworld and devoured the brave souls who attempted to leave. While Cerberus was successful in keeping the living from visiting the netherworld, like all great characters in mythology, he had a fatal flaw. In the Aeneid, when the Trojan hero Aeneas descends to visit his father, he encounters the menacing Cerberus. He tosses Cerberus a spiced cake laced with honey and poppy seeds, and Cerberus promptly devours it and falls unconscious. With hell's keeper fast asleep, Aeneas swiftly crosses into the underworld.

We'd hope that the modern equivalent to the ancient Cerberus would not have such a simple, fatal flaw. While Kerberos is the most popular cross-platform, network-wide authentication system available, it by no means has a perfect security record. It is certainly true that a lot of thought was put into making Kerberos as secure as possible; however, there are still security issues that require careful attention. Thankfully, unlike proprietary security software, Kerberos has been scrutinized for holes both in the basic protocol itself as well as the most common reference implementation from MIT.

It is important to recognize that implementing Kerberos on your network does not guarantee perfect security. While Kerberos is extremely secure in a theoretical sense, there are many practical security issues to be considered. In addition, it is important to remember that Kerberos provides only an authentication service; it does not prevent compromises caused by buggy server software, administrators granting permissions to unauthorized users, or poorly chosen passwords.

While most documentation on the subject of Kerberos security simply says to "secure the KDC," there is much more to the story of Kerberos security than turning off unnecessary services on your KDC machines (although that is certainly good advice!). In this chapter, we will begin with a discussion of potential attacks against your Kerberos authentication system, follow up with steps that should be taken to prevent these attacks, and finally examine Kerberos KDC logs. After reading this chapter, you should understand the security implications that Kerberos presents and how to protect your network from the attack scenarios presented.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶





## 6.1 Kerberos Attacks

While it may not be possible for a hacker to feed your Kerberos KDC a spiced cake to put it to sleep, there are some electronic attacks that can compromise the security of your Kerberos system. Listed below are potential compromise scenarios, and their effect on the security of the Kerberos system.

- 
- Root compromise of a Kerberos KDC machine. A root-level compromise of a KDC machine (master or any of the slaves) gives the attacker full control over the entire Kerberos authentication system. Even though the Kerberos database is encrypted on disk with the Kerberos master key, the master key is also kept on the KDC's disk so no manual intervention is required (to enter in the master password) when the KDC service is started. In addition, since all Kerberos implementations provide fail-safe access to the Kerberos database for the root or Administrator user on the KDC, your entire Kerberos database should be considered compromised in the event of attackers gaining root access to any KDC on your network. See [Section 6.4](#) later in this chapter for tips on preventing a successful attack against your KDC.
- 
- Compromise of a Kerberos administrator's credentials. If an attacker obtains the password of a Kerberos administrative principal, that attacker has complete access to the entire Kerberos database. Most KDC implementations allow administrators to remotely dump the contents of the database for backup purposes, and an attacker can use this functionality to make a complete copy of your authentication database. With full access to the database, the attacker can also create and modify any Kerberos principal. Ensure that only a very small set of users have administrative access, and set policies on those users that enforce strict password checking and at least monthly password changes.
- 
- Root compromise of a server machine. For Kerberos's mutual authentication to work, a service must have access to a service principal. These service principals, as explained in [Chapter 4](#), reside on the server's filesystem, either as part of a keytab typically used by Unix implementations, or the LSA Secrets in Microsoft implementations (see Q184017 on Microsoft's support site). If an attacker obtains root access to a server machine, all Kerberized services running on that machine are compromised. In addition, some services, such as the AFS distributed filesystem, share a single service principal across all servers. In this case, root access to an AFS file server machine would compromise all file and database servers in the AFS cell. Once an attacker has access to a service principal's credentials, the attacker can impersonate that service and also decrypt encrypted traffic sent between clients and the compromised service. The security of Kerberized services running on a server depends on the security of that individual server; therefore, all servers should be secured in proportion to the value of the resources stored on that server.



It is important to note that while the compromise of a server machine does compromise the services running on that machine, it does not compromise individual users' credentials. At no time in the Kerberos protocol exchange does a Kerberos service receive any information that would allow it to reconstruct the authenticating user's password directly. However, it's still

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 6.2 Protocol Security Issues

First, let's revisit the underlying reason why the Kerberos protocol was developed. Kerberos was designed to protect authentication data from passing over a network in the clear. Before Kerberos, when a user wished to log into a remote service, the client software would pass the user's credentials (a password) to the server in clear text. Since networks are broadcast mediums, where every station connected to a network segment can "hear" all traffic on that segment, sending passwords in the clear over a network is extremely insecure. Therefore, Kerberos encrypts all authentication exchanges that occur over the network. Encryption is only part of the solution, however, and the designers of Kerberos have put much thought into ensuring as secure a system as possible. In this section, we'll explore several attacks against the distributed authentication systems, such as Kerberos. We'll also discuss the particular techniques that Kerberos employs to mitigate the threats posed by these attacks.

### 6.2.1 Dictionary and Brute-Force Attacks

In the original Kerberos 4 protocol, the KDC issues an encrypted TGT to any client that requests it. Recall from [Chapter 3](#) that this TGT is encrypted with the user's secret key (derived from her password). The security of the entire system is dependent on not being able to decrypt this message, since if an attacker is able to retrieve the key used to encrypt the message, he now has the user's password and can impersonate that user at will. Therefore, if an attacker wishes to obtain a user's password, he can ask the KDC for a valid TGT for the victim's username. While there are no ways to break the encryption methods used in Kerberos tickets directly, the attacker can then continue to brute-force the decryption of the TGT by launching an *offline dictionary attack*.

During a dictionary attack, an attacker feeds a list of commonly used passwords, or a dictionary, to a cracking program. For each entry in the dictionary, a program attempts to decrypt the message using the password. If a hit is made, the program reports back to the attacker the user's password.

Since the transformation from the user's password to the encryption key is known (the string-to-key transformation covered in [Chapter 3](#)), it is trivial for an attacker to build a program that can translate common passwords into Kerberos encryption keys. Then, the attacker collects a large number of valid TGTs from the KDC and continues the work of cracking the TGTs off-line; that is, for each decryption attempt, he does not have to contact the KDC. Instead, once these TGTs are acquired from the KDC, no further communication is necessary to attack the passwords.

This method is made possible since there is a known plaintext included in the TGT, namely the string "tgt" itself. The Kerberos Ticket Granting Service principal name is always "krbtgt," and that principal name is the signal that indicates a successful decryption. By the time the attacker has successfully determined the password, the now unencrypted ticket has expired; however, the attacker now has a valid username/password combination to the Kerberos server and can obtain new tickets using that valid username and password.

Code is available to perform this attack against Kerberos 4 KDCs via a patch to the password cracker program John the Ripper. The code is available from Dug Song's web site at [http://www.dugsong.com/~dugsong/1614.html](#).

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 6.3 Security Solutions

Now that you have a solid understanding of the security issues and limitations of Kerberos, let's examine how to work around these limitations and ensure that your Kerberos implementation is as secure as possible.

### 6.3.1 Requiring Pre-Authentication

First, we will start with pre-authentication. The Microsoft Windows KDC is the only implementation of those covered in this book that requires clients to pre-authenticate by default. In some implementations, a command-line option or flag can be used to require all clients to use pre-authentication. Other implementations require the administrator to explicitly specify which principals need to pre-authenticate before being granted a TGT.

#### 6.3.1.1 MIT

The MIT KDC allows administrators to require the use of pre-authentication on a per-principal basis. Pre-authentication can be enabled for a principal in the MIT KDC through the following `kadmin` command:

```
kadmin: modify_principal +requires_preauth principal
```

#### 6.3.1.2 Heimdal

The Heimdal KDC also allows administrators to require the use of pre-authentication on a per-principal basis. To require pre-authentication for a principal in the Heimdal KDC database, use the following `kadmin` command:

```
kadmin> modify_principal
Max ticket life [1 day]:
Max renewable life [1 week]:
Principal expiration time [never]:
Password expiration time [never]:
Attributes []: +requires-pre-auth
```

The Heimdal KDC also allows you to turn off pre-authentication on all principals when starting the KDC, for emergency or testing purposes. The `-p` or `--no-require-preauth` switches disable pre-authentication checks for all principals until the KDC restarts.

#### 6.3.1.3 Windows domain controllers

The Windows domain controller KDC service enables pre-authentication for all principals by default. To view the current pre-authentication settings for a principal in the Windows Active Directory, use the following procedure:

- 1.
1. Log into a Windows machine that has the Active Directory administrative snap-ins installed. You must have Domain Administrator privileges to modify these settings.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 6.4 Protecting Your KDC

Since the KDC contains the secret encryption keys for all of the users as well as all of the services in your administrative realm, it is obviously very important that the KDC be well protected. It is both an advantage and a disadvantage of Kerberos that all key information is centralized; on one hand, it is easier to heavily secure one machine than to try to heavily secure a lot of distributed machines, but on the other hand, a compromise of the KDC machine compromises all authentication information in the realm.

Therefore, the machines that run KDC software should be specially prepared and dedicated solely to this purpose. During the operating system install, the machine should be physically separated or firewalled from the network to prevent exposure to the outside world. The machine is most vulnerable to outside attack during the installation of the operating system and KDC software, since the safeguards protecting the machine have not been set up yet. For example, automated worms such as Code Red have exploited unpatched Windows boxes running IIS within less than 10 minutes of exposure to the outside world.

No other server software should be installed on the KDC, especially servers that have high public visibility such as mail, web, and database servers. Remote login, if required, should be limited to a very small subset of administrative users who have local login passwords separate from their Kerberos passwords. Passwords for the administrator or root account on the KDC machines must be tightly controlled and changed periodically to prevent compromise.

Finally, physical security of the KDC machines is paramount. Physical access to any machine implicitly gives an attacker administrator-level access to that machine. Since the KDC contains all of your Kerberos realm's secret keys, physical access to the KDC would compromise all of those keys. Therefore, KDCs should be located in a locked room with limited access, preferably with some type of entry/exit logging. And remember to always log out of the console of the KDC after performing any necessary administrative tasks.

### 6.4.1 Protecting a Unix KDC

First, choose a Unix operating system that you are intimately familiar with. Good selections for a dedicated KDC machine include the free Unix systems, such as FreeBSD, OpenBSD, and Linux. These operating systems can be downloaded for free, include full source code, and are well supported by the online community, which addresses security issues quickly. Other Unix operating systems such as Solaris are also good choices, but more care must be taken in preparing commercial operating systems, as they usually ship with more network services enabled by default.

When installing the operating system, choose the smallest distribution of software possible. Since there will not be any users directly logging into this machine's console, do not install X Window System servers or clients, or desktop environments such as CDE, Gnome, and KDE. The only optional component that should be installed is a C compiler to compile the KDC software, if you are going to use one of the open source Kerberos implementations.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

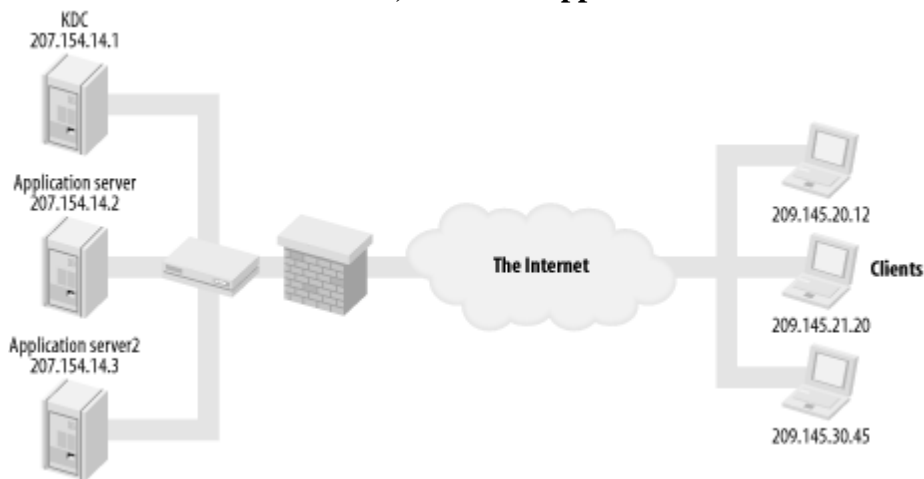


## 6.5 Firewalls, NAT, and Kerberos

Since Kerberos relies heavily on the proper functioning of DNS and some protocol messages include IP addresses in them, firewalls and NAT in particular pose obstacles to the proper functioning of Kerberos. First, let's examine what ports must be opened on a firewall if Kerberos protocol messages need to pass through it, and then look at the thorny issue of using NAT and Kerberos together.

There are several situations to consider from the perspective of a firewall administrator. The most common is a setup where client machines are located outside of a corporate firewall, and the KDCs and application servers are located inside of the firewall. All machines involved have public IP addresses and NAT is not in use. This setup is pictured in [Figure 6-4](#).

**Figure 6-4. Clients outside firewall, KDC and application servers inside firewall**



In order for outside clients to obtain tickets for your Kerberos realm, several ports need to be opened through the firewall to your KDCs. These ports are required in addition to whatever ports are already open to communicate with the application servers. In a scenario like this one, it is very important that no plain text passwords pass through the firewall through your application servers; therefore, it is still recommended that individual applications use a layer of encryption on top of their usual protocol to prevent an accidental password exposure.

### 6.5.1 Kerberos Network Ports

To enable the clients outside of the corporate firewall to communicate with the KDC and Kerberized services inside the firewall, some ports must be opened on the corporate firewall ([Table 6-1](#)).

Table 6-1. Kerberos 5 ports for client-to-KDC communication

Machine	Local port (server)	Remote port (client)	Description
All KDCs	88/udp 88/tcp	Above 1024	Kerberos 5 ticket service

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 6.6 Auditing

Although it is certainly important to ensure that your machines are secure from outside attack, you also need to periodically audit the activity of your KDC to look for any malicious activity. Depending on your KDC vendor, the amount of logging that occurs by default can vary from none (Windows 2000's default configuration) to a lot (Heimdal & MIT). In this section, we will examine the information that KDCs log, how to enable logging on your KDC, and how to read and understand the resulting log files.

The logging facilities built in to these KDC implementations not only serve auditing purposes, but they play a big role in debugging issues that may arise during the operation of your Kerberos system. First, let's take a look back at the Kerberos protocol exchange. At each point where the KDC is contacted, the KDC usually provides an option to log that information to a file.

### 6.6.1 Enabling Logging

Each KDC has different auditing options, and different procedures for enabling auditing.

#### 6.6.1.1 MIT

To enable logging in the MIT KDC, the `krb5.conf` file can contain a `[logging]` stanza with several variables that control where the logging output goes. Here are the variables:

`kdc`

The `kdc` variable controls where the log for the KDC's authentication service and Ticket Granting Service is sent. The logs produced in the file specified in the `KDC` variable contain all of the transactions between users, servers, and the KDC.

`admin_server`

The `admin_server` variable controls where the logs for the `kadmin` server are sent. The logs produced in the file specified in the `admin_server` variable contain all of the transactions between Kerberos administrators and the KDC that are performed through the `kadmin` interface.

Each option can take several different arguments, depending on the type of file, device, or syslog facility you wish the logs to be sent to. If you want logs sent to several destinations, you can list them, one at a time, on separate lines.

`FILE=filename FILE :filename`

These options send the specified logs to a file called *filename*. In the first form with a "=", the file is overwritten each time the KDC starts. The second form, specified with a ":", indicates that the file will be appended to each time the KDC starts.

`STDERR`

This option specifies that the logs should be sent to the standard error output of the KDC.

`CONSOLE`

This option specifies that the logs should be sent to the standard output of the KDC.

[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

## Chapter 7. Applications

Establishing a Kerberos realm and creating KDCs for your realm is only the beginning of creating a Kerberos-based authentication infrastructure. To enjoy the benefits of Kerberos, you, as the network administrator, also have to install Kerberos-enabled services and client software. This chapter illustrates how to enable Kerberos support in several popular server packages and the corresponding client programs.

[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

## 7.1 What Does Kerberos Support Mean?

There are essentially two "types" of Kerberos support that a client/server application can implement. The first, and unfortunately, most common is for the client to send the server the user's Kerberos password in plain text. The server then acquires a TGT on the user's behalf (and hopefully verifies that the TGT is valid by also acquiring a service ticket for itself—see the description of the man-in-the-middle attack in [Chapter 6](#) for details on why this is important). This method has a distinct advantage: most protocols that require authentication only support simple, plain text username and password authentication. Even if the protocol is extensible enough to support stronger authentication methods, these stronger authentication methods are usually not widely supported by the variety of clients in use. This method, of course, has the disadvantage of sending the user's credentials in plain text over the network. Since Kerberos is designed as a single-sign-on solution, exposure of a user's credentials in this way is even more dangerous since the same username and password is accepted for authentication by other Kerberos-enabled services. Finally, this method does not allow for a true single-sign-on solution; instead, it provides users with a single login and password that they have to enter multiple times.

The other method of supporting Kerberos authentication is what I'll call "native" Kerberos authentication support. Native Kerberos authentication support provides a true single-sign-on capability, in which users can login once to their local workstation, and acquire service tickets for Kerberos-enabled servers throughout the day. This requires special support on both the client and server so that the Kerberos tickets are communicated in a secure manner. This method provides for a superior user experience: the user only has to enter her credentials once per login session. Further authentication to Kerberos-enabled services is handled transparently, without user intervention. However, native Kerberos support in client applications is still not widespread, and may require users to change to a client application that does support Kerberos.

Our primary focus in this chapter is to enable native Kerberos support for popular applications. However, there are still some protocols where support for native Kerberos authentication is not currently possible or is not widely available. We will discuss how to use the Kerberos 5 PAM modules to add Kerberos 5 password verification support to those protocols.

[\[ Team LiB \]](#)

◀ PREVIOUS   NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS   NEXT ▶

## 7.2 Services and Keytabs

Remember that Kerberos provides a service that verifies the identity of two connection endpoints, identified by unique names, or principals. It is rather obvious so far that each user is associated with a principal name that is stored in the Kerberos database, since all authentication schemes by their very nature require that all users be uniquely identified with an associated secret. However, the concept that all services that users contact through Kerberos also require a principal and secret key is a new one to most administrators.

On Windows hosts, service keys are automatically created as needed when Kerberized services are installed. Unix-based Kerberos realms require a bit more manual configuration, and this section discusses the issues that Kerberos administrators have to work with when installing Kerberized services.

As we saw in [Chapter 2](#), a service principal has three major components: the service name, the hostname of the machine that provides the service, and the Kerberos realm to which the machine belongs. Here's a sample service principal:

```
imap/freebsd.wedgie.org@WEDGIE.ORG
```

In this example, the service name is "imap", the host that this service is running on is "freebsd.wedgie.org", and the realm that this machine belongs to is "WEDGIE.ORG".

Of course, a secret key is associated with every principal, and so there is an encryption key or keys (possibly more than one, with different key version numbers, encryption types, and salts) associated with a given service principal. On Unix hosts, service keys are stored on the server providing the service, in a special file called the *keytab*. There can be, and for security reasons, should be, one keytab file per service offered.

Since keytab files contain highly sensitive information, notably encryption keys, it is imperative to ensure proper access controls to these files. Each Kerberized service should run as a different, unique username, and the keytab file for that service should be readable only by that username. As discussed in [Chapter 6](#), the compromise of a service's key allows an attacker to masquerade as any authorized principal when communicating to that service, and also allows an attacker to read any conversation between clients and the compromised service.

The default keytab file—for most Unix-based Kerberos implementations—is `/etc/krb5.keytab`. By convention, this keytab contains the encryption keys associated with the host principal. While generating a host keytab for every machine in your realm is optional, it is highly recommended for those machines that run a Kerberized telnet or SSH daemon, or use Kerberos passwords to determine access to the local machine.

The process of exporting encryption keys from a Kerberos KDC to a keytab varies from one implementation to the next. The process for some common implementations is covered in [Chapter 4](#). When extracting keytabs, ensure that they are protected during transport over the network from eavesdroppers. This can be accomplished by either running `kadmin` on the host you're extracting a

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 7.3 Transparent Kerberos Login with PAM

When a user logs into his workstation at the beginning of the day, we want that user to acquire a Kerberos Ticket Granting Ticket when he enters his credentials. We'll call this transparent Kerberos login. Windows 2000, XP, and 2003 automatically acquire tickets upon login when the user is part of a Windows domain. However, for other systems, we have to configure this step manually. In Unix, the simplest and most portable way to get initial credentials for a user upon login is through the Pluggable Authentication Modules (PAM), which is available on most operating systems. Using PAM, you can acquire Kerberos tickets for logins that occur on the system's console (and any other network-based protocol, but we want to avoid sending passwords over the network).

Historically, applications such as the console login program and the X Windows System login program (xdm) all had to be modified to support new authentication methods. This introduces a maintenance and security nightmare, as locally-maintained patches must be made to system software to enable authentication methods other than the standard Unix password file. Worse yet, if the operating system comes without source, you may not even be able to replace the program with one that performs the necessary authentication method.

PAM solves this problem by providing a standard plug-in interface that both application developers and authentication method developers can write to. A mapping file is created that maps applications' authentication requests to the appropriate authentication methods, so that authentication modules can be added and removed on the fly, without recompiling the application. Linux, FreeBSD, Solaris, and HP-UX all include PAM support, and more operating systems are adding support.

However, PAM is not a panacea. It only supports traditional username and password authentication, so PAM works best when authenticating local (ie., on the console) login requests. Network-based services should use native Kerberos authentication to take advantage of the single-sign-on capabilities of Kerberos and to avoid sending plain text passwords across the network. PAM cannot provide native Kerberos authentication through the Kerberos ticket exchange. In addition, PAM implementations differ slightly from vendor to vendor, so PAM modules that may work on one vendor's OS may not work on another vendor's OS. The differences are usually small enough that they can be easily worked around, but it is something to be aware of when using PAM.

### More Information About PAM

The following Unix operating system vendors have web sites that describe their support of the PAM framework:

- 
- HP-UX: <http://www.hp.com/products1/unix/operating/security>
-



[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 7.4 Mac OS X and the Login Window

The initial console login window presented to Mac OS X users is called, appropriately enough, the loginwindow. Unfortunately, loginwindow's PAM support is incomplete. But luckily for Kerberos 5 users, Apple has provided special support in the loginwindow contained in Mac OS X 10.2 and above to provide users with Kerberos tickets when logging into their OS X system.

The procedure for enabling Kerberos support in the Mac OS X loginwindow application is documented in the AppleCare document #107154, "Mac OS X 10.2: How to Enable Kerberos Authentication for Login Window." Note that the method to enable this facility is subject to change in future OS X revisions.

Just like PAM, there are two basic options available when enabling Kerberos login support in OS X. You can either require valid Kerberos credentials for successful local login, or simply acquire Kerberos tickets if the local password is the same as the Kerberos password.

The Mac OS X Security and Authorization Services use the `/etc/authorization` file, and this is the file that we'll use to enable Kerberos authentication in loginwindow. First, to require valid Kerberos credentials for login to the local system, Mac OS X can either require a valid host keytab or operate without a host keytab. Note that as we discussed in [Chapter 6](#), a host keytab is required to defend against man-in-the-middle attacks against the Kerberos system.



Since Mac OS X does not include the `kadmin` utility, the best way to get a host key onto the Macintosh host is to create and extract the host key for the OS X host on the KDC and use Secure Shell (installed by default on OS X) to copy the key securely to the Mac.

In order to require Kerberos credentials when a host keytab is present, search for the `system.login.console` key in the `/etc/authorization` file and replace it with the following:

```
<key>system.login.console</key>
  <dict>
    <key>eval</key>
<string>loginwindow_builtin:login,krb5auth:authenticate,loginwindow_builtin:su
ccess
</string>
  </dict>
```

If, on the other hand, you still wish to require Kerberos credentials even though a valid keytab is not present, you can replace the `system.login.console` key with the following text:

```
<key>system.login.console</key>
  <dict>
    <key>eval</key>
<string>loginwindow_builtin:login,krb5auth:authnoverify,loginwindow_builtin:su
ccess
</string>
  </dict>
```

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 7.5 Kerberos and Web-Based Applications

Web-based authentication is an important issue for many organizations that want to extend their single-sign-on infrastructure to the web, for both internal intranet applications as well as external internet applications. Authentication can either be handled by the web application itself, by providing the user an HTML page with form entries for a username and password, or by the web server, through the HTTP protocol. This section discusses an Apache module that provides administrators the ability to verify Kerberos passwords through an Apache module.

The web server and browser perform HTTP authentication, with the resulting verified username returned by the web server to the web application. When an end user requests a resource on a web server for which the server is configured to require authentication, the web server returns an error 401 (Not Authorized) to the client. This error message includes an HTTP header, WWW-Authenticate, that provides the client a challenge. Based on the response that the client provides the server, the server may choose to provide the client access to the requested resource, or continue to return 401 errors to the client if the response returned by the client is unsatisfactory. With this generic method, any challenge-response security protocol can be used for HTTP authentication.

The HTTP specification defines two authentication methods based on the above challenge-response architecture: Basic and Digest authentication (defined in RFC 2617). The most widely implemented of the two is the Basic method, in which the challenge consists simply of a realm name (not to be confused with a Kerberos realm) returned by the server to the client defining the resource for which the client is requesting access. Upon receipt of a 401 error with a request for Basic authentication, the browser displays a dialog box with prompts for username and password, and a label containing the name of the realm to which the user is requested to authenticate herself. The username and password is sent back to the server encoded in Base-64 (essentially in the clear, as no encryption is performed, Base-64 is just a transformation to eliminate ambiguity when decoding the response). Digest authentication uses a true challenge-response architecture but is not widely implemented since it requires the server to have the plain text of all users' passwords in order to verify authentication responses.

In addition to Basic and Digest, Microsoft IIS and Internet Explorer also support NTLM authentication, which uses the challenge-response Microsoft NTLM protocol to perform authentication between Microsoft-based browsers and servers. With the introduction of Windows 2000, Microsoft added Kerberos authentication to both IIS and Internet Explorer. Microsoft chose to implement Kerberos authentication through the use of a new HTTP authentication mechanism, Negotiate, layered with the Simple and Protected GSSAPI Protection Mechanism (SPNEGO, defined in RFC 2478). Through this mechanism, Internet Explorer can use native Kerberos authentication to access protected resources on an IIS server and provide single-sign-on for these resources. More information on the Microsoft implementation of Negotiate and SPNEGO as a web authentication method can be found at <http://msdn.microsoft.com/library/en-us/dnsecure/html/http-ss0-1.asp>.

An implementation of SPNEGO and the Negotiate HTTP mechanism that interoperates with Microsoft Internet Explorer is under development, and the code is available at <http://modgssapache.sourceforge.net>. As this code is still under development, installation and configuration is still rough and subject to change. Installation instructions for the current version are available at the above web site, and I'll document my own adventures with it on the O'Reilly Network

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 7.6 The Simple Authentication and Security Layer (SASL)

The Cyrus SASL project forms the basis for several other products' authentication and session encryption support, most notably the Cyrus IMAP mail server and the OpenLDAP directory server. The Cyrus Simple Authentication and Security Layer (SASL) project provides an extensible framework for network protocol authentication. It is more generic than PAM in that SASL supports more complex authentication exchanges, such as Kerberos mutual authentication, and also supports the negotiation of a security layer (encryption) for later protocol exchanges once authentication is complete. SASL is documented as Internet RFC 2222.

SASL supports native Kerberos 5 authentication through the GSSAPI interface. Other authentication methods that SASL provides to applications include Kerberos 4 and standard `/etc/password` or `/etc/shadow` authentication (optionally through a privileged daemon process for services that don't have the necessary privileges to read the system password database). In addition, SASL supports several database-backed authentication methods, including the `sasldb`, which uses a lightweight database such as Berkeley DB or GDBM to store username/password pairs, and a `mysql` driver that uses the MySQL database to store authentication secrets.

SASL also includes a daemon process, `ssaslauthd`, which can provide password-based Kerberos 5 support to SASL-based applications similar to that of PAM. We'll cover how to build and enable this password verification method as well.

The Cyrus SASL home page is located at <http://asg.web.cmu.edu/sasl>, and the latest version of the Cyrus SASL distribution available at the time this was written is 2.1.10. We'll step through the process of building the Cyrus SASL library with Kerberos 5 support through the GSSAPI.

The first step, of course, is to acquire the distribution and unpack it. The latest version of Cyrus SASL is available from <ftp://ftp.andrew.cmu.edu/pub/cyrus-mail>. Download the distribution file (`cyrus-sasl-2.1.12.tar.gz` at the time of this writing), uncompress, and untar it.

### 7.6.1 Building the Distribution

Once the distribution is unpacked, we're ready to configure it for GSSAPI support. The only option required to the configure script to enable GSSAPI support is the `enable-gssapi` option, which takes one argument: the root directory of your installed Kerberos 5 installation. Of course, additional configure options can be appended for other authentication services that SASL supports.



Note that Cyrus SASL has several external dependencies, notably a recent vintage database library such as the Berkeley DB or GNU DBM. During build testing of Cyrus SASL on a FreeBSD host, the configure process claimed to find a compatible DB engine, yet the build failed until GDBM was installed. If you encounter build failures, ensure that you have a compatible DB library

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 7.7 Kerberos-Enabled Server Packages

While PAM is a great solution for local login on the system console, the real advantages to using Kerberos are only realized if client/server applications that users interact with are configured for native Kerberos support.

Our users now have Kerberos tickets upon login. The next step is to start adding Kerberos support to the application servers that users access. We want users to enjoy the benefits of a fully-Kerberized environment as much as possible, so I'll focus on enabling native Kerberos support in as many packages that support it, but fall back to the single-login capability provided by other packages that do not have built-in Kerberos support.

We already saw an example of a network protocol with native Kerberos support back in [Chapter 4](#), when we configured the Kerberos telnet server to test our new Kerberos implementation. We're going to take that a step further in this section and examine how to add Kerberos support to other popular network protocols.

### 7.7.1 Electronic Mail (Cyrus IMAP)

Cyrus IMAP is a part of Project Cyrus, a project developed at Carnegie Mellon University to provide a reliable, scalable electronic mail system for the campus. The Cyrus mail server had, in its original design goals from 1994, many of the same goals of administrators today: the mail service had to scale to thousands of simultaneous readers, it had to support many different clients on different hardware and operating-system platforms, and it had to integrate with the campus-wide authentication system, which happens to be based on Kerberos. Today, Cyrus supports the two major mail access protocols: the Internet Mail Access Protocol (IMAP) and Post Office Protocol (POP). A separate program, a Mail Transfer Agent, handles the task of transferring mail from system to system through the Simple Mail Transfer Protocol (SMTP). Newer mail clients support SMTP authentication, and we'll discuss Kerberos support for MTAs in the next section.

Cyrus IMAP is available from Carnegie Mellon at <http://asg.web.cmu.edu/cyrus/imapd>, and the latest stable version available at the time of this writing is 2.1.12. Cyrus IMAP uses the Cyrus SASL library to handle authentication and session encryption tasks. Therefore, before building Cyrus IMAP, you'll need a working installation of Cyrus SASL.

#### 7.7.1.1 Building and configuring the distribution

Cyrus IMAP is a complex package and most of the build and configuration options relate to how it handles mail, and not its authentication mechanism. Therefore, we're going to focus on the particular options necessary to enable GSSAPI support in Cyrus IMAP.

After acquiring the source distribution, untar it and the following configure line will configure Cyrus IMAP for GSSAPI and SASL support:



[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 7.8 Kerberos-Enabled Client Packages

To truly use Kerberos as a cross-platform single-sign-on system, Kerberized client software has to be installed as well. A complimentary pair of client and server Kerberized applications must be matched to perform native Kerberos authentication. Applications that use server-side Kerberos password verification will work with unmodified clients, but their use is discouraged as it negates the single-sign-on benefits provided through native Kerberos authentication. This section describes some of the software packages available that provide client-side native Kerberos functionality.

### 7.8.1 Kerberized Secure Shell Clients

In a previous section, we built OpenSSH with GSSAPI support. This OpenSSH with GSSAPI patches works on many platforms, including all of the common Unix variants, and Mac OS X. However, OpenSSH operates only on the command line, and compiling OpenSSH on Windows can be difficult. A popular, free, and graphical Secure Shell client for Windows is PuTTY, and a company named Certified Security Solutions has developed patches to PuTTY to incorporate GSSAPI authentication support, and provides binaries that are free for noncommercial and internal commercial use.

The modified PuTTY client is available at <http://www.certifiedsecuritysolutions.com/downloads.html>. Separate distributions are available for Windows 2000 and older Windows operating systems. The distribution for Windows 2000/XP/2003 includes support for the Windows SSPI that can communicate with the GSSAPI-enabled OpenSSH without requiring Kerberos for Windows to be installed on the Windows host.

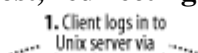
### 7.8.2 Reflection X

Reflection X is an X11 server package published by WRQ, Inc. Reflection X allows users on Windows platforms to access X11 applications on Unix hosts. While there are many such packages available, Reflection X has decent Kerberos functionality provided as part of the Security Components. By using the Reflection Security Components, you can set up a cross-platform single-sign-on infrastructure between Windows clients, Windows servers, and Unix servers.

The latest version is WRQ Reflection X 10, and includes support for the latest industry standard X11R6.6 protocol as well as traditional, character-based terminal emulation protocols. For more information on the X server support and terminal emulation features provided by the Reflection X package, visit the WRQ home page at <http://www.wrq.com/>.

To start X11 applications on a Unix server, the X server software running on the Windows host must have a remote login client built in to log into the Unix host, redirect the X11 display to the Windows machine's IP, and then start the X11 application. A diagram of this process is shown in [Figure 7-2](#).

**Figure 7-2. Logging into Unix host, redirecting display, and starting application**



[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

## 7.9 More Kerberos-Enabled Packages

While I've tried to present a sample of some of Kerberos-enabled packages, there are still many more applications that support Kerberos authentication. Many applications, like databases, file servers, and print spool software include Kerberos authentication. With the background presented in this chapter, you will be able to enable and configure this support in those products as well.

[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

## Chapter 8. Advanced Topics

So far, we have covered enough of the Kerberos authentication system to establish useful Kerberos realms and enable Kerberos support in applications to take advantage of a single-sign-on environment. This chapter will prepare you to create networks with multiple Kerberos realms and interoperate between different Kerberos implementations. It also discusses some issues to be aware of when working with multiple Kerberos implementations.

[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

[\[ Team LiB \]](#)

[◀ PREVIOUS](#) [NEXT ▶](#)

## 8.1 Cross-Realm Authentication

All of the Kerberos discussion so far has assumed that all users and resources on your network are located in a single Kerberos realm. However, what if there are several departments, locations, or other divisions that are under different administrative control, each with their own Kerberos realm? These users want to access not only resources in their local Kerberos realm, but also resources in the other realms as well, with a minimum of hassle. Kerberos cross-realm authentication can solve this problem.

In Kerberos, cross-realm is implemented by sharing an encryption key between two realms. The key that is shared is the Ticket Granting Service principal's key. A typical Ticket Granting Service principal for a single realm looks like:

```
krbtgt/WEDGIE.ORG@WEDGIE.ORG
```

Note that the instance is the same as the realm name. In cross-realm, two principals are created on each participating realm. For two realms, ONE.COM and TWO.COM, these principals would be:

```
krbtgt/TWO.COM@ONE.COM
```

```
krbtgt/ONE.COM@TWO.COM
```

These principals have to be created on both realms, and are known as remote Ticket Granting Server principals. The Kerberos trust can be one way or both ways; since there are two separate, shared keys involved, one realm can choose to trust the other realm's tickets, but not the other way around.

When a user who is in the ONE.COM realm wishes to communicate with a Kerberized service in TWO.COM, the client program first requests a ticket for the remote realm's Ticket Granting Server, the `krbtgt/TWO.COM@ONE.COM` principal above. Using that intermediate Ticket Granting Ticket, the client is then able to acquire a service ticket directly for the requested service in the TWO.COM realm. This is called *direct cross-realm trust*, and is the only type of cross-realm trust supported in the older Kerberos 4 protocol. In direct cross-realm trust, every two realms that wish to communicate must share a separate set of keys. Of course, this can get rather unwieldy as the number of shared keys grows exponentially with the increasing number of realms. This can be managed somewhat by building a *certification path* between several realms, a feature introduced with the Kerberos 5 protocol. A certification path defines realms that may be used as intermediaries when acquiring service tickets in foreign realms. Direct cross-realm requires every foreign realm be directly connected, through a shared key, to the local realm, creating a full mesh configuration between the realms. Certification paths allow multiple realms to use another realm as an intermediary, creating hub-and-spoke systems in which multiple realms share a key with a single intermediary realm.

Let's take an example. There are several universities collaborating on a project to produce workable cold fusion together with a fictitious government agency, the National Energy Research Directive (NERD). Obviously, these organizations want to utilize each other's resources, and researchers don't like to memorize numerous logins and passwords, so cross-realm Kerberos is proposed as a solution to the authentication needs of all the organizations involved. However, direct cross-realm between all of these organizations would require approximately  $n^2$  different keys, where  $n$  is the number of participating Kerberos realms. This is a management nightmare, especially if, in this example, more universities are added to the project as time goes on. A diagram of these cross-realm relationships in a full mesh configuration is shown in [Figure 8-1](#).

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 8.2 Using Kerberos 4 Services with Kerberos 5

Those who have Kerberos 4 services that need to be integrated into a Kerberos 5 realm need to implement the Kerberos 5-to-4 ticket translator daemon. Both MIT and Heimdal include support for this protocol, the krb524 protocol. As discussed in [Chapter 3](#), the only limit on where the krb524 daemon can run is that the daemon must have access to the service keys for the Kerberos 4-based services for which it translates tickets.

The MIT Kerberos 5 distribution includes a separate krb524 daemon, krb524d. There are two different modes of operation that krb524d supports: master and keytab. The master mode is meant to be run on a KDC in the Kerberos realm, and reads the necessary service keys directly from the Kerberos database. If it is not possible to run the krb524d directly on the KDC, then the second mode of operation can be used: keytab. Keytab mode requires that a Kerberos keytab be installed on the machine running krb524d that includes the service keys for all of the Kerberos 4 services in the realm.

The command-line arguments to krb524d are summarized below:

```
# krb524d
Usage: krb524d [-k[eytab]] [-m[aster] [-r realm]] [-nofork]
```

Either the -k or the -m options are required. The -m option enables the master mode, as described above, where krb524d reads the necessary service keys directly from the Kerberos database on the local disk. The -k option requires an argument, namely, the keytab where the keys are stored for the Kerberos 4 services located in the Kerberos 5 realm.

As an example, let's create a service principal for a popular Kerberos 4-based service, the AFS network filesystem. We first create a service principal for AFS, ensuring that the only encryption type associated with the new principal is single DES. With MIT Kerberos, the kadmin commands to create this principal would be similar to the following:

```
> kadmin
Authenticating as principal jgarman/admin@UNIX.SAMPLE.COM with password.
Enter password:
kadmin: addprinc -randkey -e des-cbc-crc:v4
afs/unix.sample.com@UNIX.SAMPLE.COM
WARNING: no policy specified for afs/unix.sample.com@UNIX.SAMPLE.COM;
defaulting to
no policy
Principal "afs/unix.sample.com@UNIX.SAMPLE.COM" created.
```

After the principal has been created, the keytab can be extracted to a file, which can be placed on the machine running the krb524d daemon.

```
kadmin: ktadd -k /tmp/afs.keytab -e des-cbc-crc:v4
afs/unix.sample.com@UNIX.SAMPLE.COM
Entry for principal afs/unix.sample.com@UNIX.SAMPLE.COM with kvno 3,
encryption type
DES cbc mode with CRC-32 added to keytab WRFILE:/tmp/afs.keytab.
```

Heimdal Kerberos also includes the Kerberos 5-to-4 ticket translator daemon, but it is integrated with the rest of the KDC and does not require running a separate daemon.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 8.3 Windows Issues

While the Windows implementation of Kerberos is compatible with the specifications in RFC 1510, the Microsoft implementation of Kerberos varies significantly enough from the MIT and Heimdal implementations to warrant its own explanation. In order to provide the additional functionality required for the Windows Active Directory, as well as backwards compatibility with older Windows NT workstations, the Windows Kerberos environment differs in several important areas from its Unix counterpart.

### 8.3.1 Encryption Algorithm Support

The primary encryption type used in Windows is based on the RC4 stream cipher, with an MD5-HMAC algorithm used for the checksum field. This encryption type is referred to as RC4-HMAC, and has a variable key length to support both weaker, "export" quality key lengths, as well as stronger 128-bit key lengths.

The reasoning behind this decision by Microsoft is two-fold: first, for compatibility with older Windows NT domains; and second, for political reasons. During the initial design of Windows 2000, neither DES nor triple DES were approved for export from the United States. Microsoft wanted to encourage deployment of Windows 2000; therefore, the RC4-HMAC cipher was chosen as the default Kerberos encryption type since it is the same cipher used to generate the older NT4 password hashes. This way, when an older NT4 domain is migrated to an Active Directory domain, the users' passwords continue to work without manual intervention.

Microsoft did add DES support to Windows 2000 before its release, and users created in a Windows Active Directory have both RC4 and DES encryption keys associated with their account. However, there are two situations when a DES key is not available for an account in the Active Directory. The first situation is the one discussed above, in which an NT4 domain is converted into a Windows Active Directory domain. Since the hashing algorithm only works one way, there is no way for Windows to convert the existing users' RC4 encryption keys into DES keys. The second special-case situation is when a new Windows 2000 domain is created. As part of the domain creation procedure, an Administrator account is created as the new Domain Administrator. This account only has an RC4 key when it is initially created.

In order to add DES keys to users' accounts in both of the above situations, simply change the user's password. When a user's password is changed, the KDC will generate both RC4 and DES encryption keys for that user.

Note that even without the "Use DES encryption types for this account" checkbox checked for a user, the DES keys do exist in the Active Directory database (subject to the limitations of the previous two scenarios), but are not used by the KDC when responding to ticket requests unless the checkbox is activated.



[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 8.4 Windows and Unix Interoperability

In the previous chapters, we focused mostly on the design and implementation of a homogenous Kerberos network. However, the true allure of moving to a Kerberos-based authentication scheme network-wide is to enable centralized authentication, and more importantly, single-sign-on across all platforms. Cross-platform single-sign-on is considered to be a panacea of network authentication, and even with Kerberos, can be very difficult to achieve because of the wide variation between Kerberos implementations. The end objective is for users to have only one set of credentials, a username/password pair that will enable them to access all network resources regardless of the platforms these services may reside on.

These interoperability scenarios are also addressed in a Microsoft document, the Step-by-Step Guide to Kerberos 5 Interoperability, available at <http://www.microsoft.com/windows2000/techinfo/planning/security/kerbsteps.asp>.

### 8.4.1 Using a Windows Domain Controller as a KDC for Unix Clients

Using a Windows domain controller as a KDC for non-Microsoft platforms is trivial to set up; as long as the users have DES keys enabled in Active Directory, they will be able to login to the Windows domain controller without a problem. The only difference is in administration; you'll be using MMC to create and modify Kerberos users in this case. Since Microsoft does not implement a kadmin interface similar to MIT or Heimdal's, creating keytabs for Unix services when using a Windows domain controller is a bit different than the process of generating keytabs from Unix KDCs.

#### 8.4.1.1 Creating Unix keytabs from a Windows domain controller

When using a Windows domain controller as the KDC for a mixed platform Kerberos environment, a method is required to extract keys from the Windows KDC into keytab entries for Unix hosts and services. The kadmin programs that are included with the MIT and Heimdal Kerberos distributions do not work with Windows domain controllers since Microsoft uses its own administration protocol for communication with the KDC. Instead, Microsoft includes a program to create a keytab file with a specified password (run through the Kerberos 5 string2key function to create the appropriate DES key).

This program, ktpass, is not installed by default. If your domain controller does not have a ktpass program installed, it can be found in the support/tool subdirectory of the Windows 2000 Server installation CD.

First, a user account for the service must be created in the Active Directory. Since Active Directory does not handle Kerberos-style username and instance principal formats, this username cannot be the desired principal name (as Windows does not allow the "/" character in usernames, along with most other special characters). Instead, this name can be any valid Windows username, and will be mapped to the principal name later. It is recommended that these accounts be placed in a separate OU to distinguish them from other user accounts in the domain.

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

## Chapter 9. Case Study

In the previous eight chapters, we examined the technical details behind the Kerberos system, and how to implement Kerberos in your network. Now, in this chapter, we will take a step back and examine a hypothetical organization that wants to implement a network-wide single-sign-on solution. This organization has chosen to use Kerberos. We describe the decision process as the necessary Kerberos realms are created and implemented. The example includes many of the decision processes that apply to organizations implementing Kerberos in their own networks.

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

## 9.1 The Organization

The fictitious organization that we'll use for our example is the Sample Internet Service Provider, a prominent provider of local dial-up, T1, and DSL service in the Anytown area. The Sample ISP has an internal network with two major divisions in its IT organization. One division of the IT organization provides end user support and services to the Windows desktops and servers. This department administers the company email server, which runs Microsoft Exchange, and has a Windows 2000 Active Directory system already in place to handle user logins on the Windows network.

The second IT department administers the backend Unix systems, most notably a large bank of web-hosting machines running Linux and Apache. In addition, the Sample ISP has a small testing and staging laboratory where new software is tested before deployment. The Unix systems currently do not have a centralized authentication system in place; there is a mishmash of `/etc/passwd` files, `htpasswd` files, and password hashes stored in a MySQL database that handle the current authentication needs.

The current setup has some serious problems from a manageability standpoint. Adding or removing users on the Unix machines is a tedious process that involves logging into each machine separately and adding or removing an entry from the local machine's `/etc/passwd` file. In addition, the lack of synchronization between the Unix machines means that users have separate passwords for each machine they have access to. As a result, the Sample ISP has many stale `passwd` files on its machines, some containing entries for users who should no longer have access.

To solve the authentication problems, an infrastructure should be established that centralizes the administration of the user authentication information. In addition to centralizing the authentication information for the Unix systems, management has decided to establish a cross-platform single-sign-on system so that staff can login once via their desktop Windows systems and then be able to transparently authenticate to any other system, whether Windows- or Unix-based. Of course, Kerberos is chosen to provide this capability. More specifically, Kerberos v5, as it is the latest revision of the Kerberos protocol and provides compatibility with the existing Windows 2000 Active Directory setup.

Right now, the only applications that the Sample ISP is planning to kerberize are remote login to the Unix machines as well as some X-Windows applications that the support and network operations staff run on a regular basis.

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

## 9.2 Planning

The first step is a planning stage. Here we evaluate the current setup and the requirements that the new Kerberos realms need to fulfill, and balance those against the cost constraints involved with the project. During this planning stage, we will sketch out the new Kerberos realm structure, define what set of users each Kerberos realm will contain, and finally, prepare the necessary systems to install the Kerberos KDC software.

### 9.2.1 Planning the Kerberos Realms

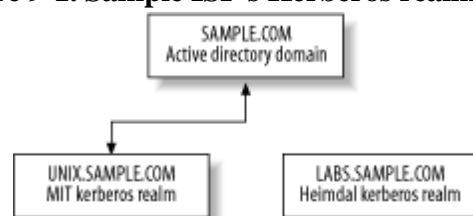
The first decision to make when implementing Kerberos is whether there will be multiple Kerberos realms, and if so, what their relationship to each other will be. We've decided to split the organization into three realms to enforce the separation between the three functions of the ISP, namely, the production/business operations, the Unix servers involved in the customer support and hosting functions, and the lab, which is isolated from everything else.

In this case, one realm is already established: the Windows Active Directory domain. This domain was established as `SAMPLE.COM`, which is also the ISP's DNS domain name. There are two more realms that we will establish as part of this example, named `UNIX.SAMPLE.COM` and `LABS.SAMPLE.COM`. We will create them as subdomains of the existing `SAMPLE.COM` realm to make the cross-realm relationships easier—the hierarchical realm structure creates an implicit certification path for cross-realm authentication, as we saw in [Chapter 8](#).

With the realm names out of the way, we need to establish trust relationships between the realms, if any. Remember that a trust relationship between realms does not automatically provide access to resources in one realm from the trusted realm. However, with that limitation in mind, it is still important to create a layered approach to security, and we want to restrict the trust relationships of the Kerberos realms as much as possible. While this does force users who wish to use resources in both realms to login to both realms, it enforces the administrative and security separation between the Kerberos realms.

Considering the above, we'll separate the `LABS.SAMPLE.COM` realm from the production `SAMPLE.COM` and `UNIX.SAMPLE.COM` realms, to enforce the separation of the testing environment in the labs realm from the production realms. A two-way cross-realm trust is established between the two production realms, `SAMPLE.COM` and `UNIX.SAMPLE.COM`, in order to enable sharing of resources between the two realms with one set of credentials. [Figure 9-1](#) depicts the Kerberos realms that are involved, and the trust relationships between them.

**Figure 9-1. Sample ISP's Kerberos realm layout**



[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 9.3 Implementation

Now we can start creating the two new Kerberos realms. Before we begin, we must establish the prerequisites that must be satisfied before implementation of the new realms can start. The first prerequisite is a DNS server with functioning forward and reverse DNS zones for the sample.com DNS domain. In our example, the service is hosted through the existing Active Directory domain, and appropriate DNS records have been added to the zone files already for all of the machines in our sample network.

The second prerequisite is that all machines have NTP installed and configured. The Windows domain will perform time synchronization against the domain controllers, but NTP must be manually installed and configured on the Unix machines. Before the Kerberos realms is implemented as described below, these two services must be functioning correctly.

### 9.3.1 Implementing UNIX.SAMPLE.COM

We'll start with UNIX.SAMPLE.COM. Both KDCs, unixkdc1.sample.com and unixkdc2.sample.com, have a fresh installation of the latest FreeBSD distribution. Two 18GB hot-swappable SCSI disks have been installed into each machine, and each box has a hardware RAID card set up to do mirroring across the installed drives. The partition layout looks like [Table 9-1](#).

Table 9-1. Partition table for Unix hosts

Mount point	Description	Size
/	Root drive, includes all operating system programs and data	5GB
/krb5	Kerberos database and software	4GB
/var	Log files	5GB
/tmp	Temporary files	2GB
swap	Kernel swap space	1GB

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

## Chapter 10. Kerberos Futures

Kerberos is constantly evolving to integrate new technologies and thwart new threats. As a result, the Kerberos working group has developed several extensions to the base Kerberos 5 protocol to provide the necessary capabilities to continue using Kerberos in the future. These protocol extensions are currently available as Internet Drafts from the IETF. The principal draft is the Kerberos Clarifications, which will replace the current RFC 1510 as the authoritative document for Kerberos protocol implementers. While the Kerberos Clarifications is true to its name and, for the most part, simply provides a more concise and clear description of the current protocol, it also contains new recommendations and small protocol changes that result from years of practical implementation experience and security reviews. Other related draft documents describe more dramatic protocol extensions that are optional.

The current home page of the Kerberos Clarifications is the Kerberos page at the USC Center for Computer Systems Security, located at <http://www.kerberos.isi.edu>. Additionally, current Internet Drafts can be downloaded from the IETF home page at <http://www.ietf.org>. The index to the current Internet Drafts issued by the Kerberos working group is located at <http://www.ietf.org/ids.by.wg/krb-wg.html>. Readers interested in a more technical discussion of these proposals are encouraged to read the Internet Drafts published at the IETF and USC Kerberos sites.

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶



## 10.1 Public Key Extensions

Kerberos was designed when public key encryption was still in its infancy. Public key algorithms were not widely in use at the time, the technology was heavily patented by RSA Security, Inc., and the computing power commonly available at the time was not sufficient to sustain a large public key infrastructure. Today, these factors have changed: the patents on the popular RSA public key algorithm have expired, opening the door to royalty-free implementations of RSA, and Moore's law of exponential growth in computing speed has provided more than enough computing power to handle the large calculations involved in public key cryptography. The development and wide deployment of the Secure Sockets Layer (SSL) and its successor, Transport Layer Security (TLS), which use public key cryptography primarily for securing communication with web sites, have proven that public key cryptography is useful and can be trusted for secure systems.

The question is, then, what benefits can the introduction of public key cryptography bring to Kerberos? To answer this question, we need a clear understanding of what makes public key cryptography different from traditional, private key cryptography.

### 10.1.1 Public Key Cryptography

Traditional cryptographic systems require the sender and recipient to share a common secret key to communicate securely. The sender uses the secret key to encrypt messages to the recipient, and the recipient uses the same secret key to decrypt the incoming message. This is known as *symmetric*, or *private key cryptography*, and is the type of cryptography used by the Kerberos protocol. The problem, of course, is how the two parties agree on and communicate the secret key. This problem is generally known as *key management*, and is a crucial factor in the practical operation of any encryption scheme. Key management affects all encryption algorithms, regardless of their strength, and a compromised key in a relatively weak cryptosystem has exactly the same consequences as a compromised key in a relatively strong cryptosystem—namely, that all communications using that key are now easily decrypted by an attacker. Implementers of private key cryptosystems typically address the key distribution issue by using some form of personal contact between sender and recipient prior to the first message exchange to establish a secret key; for example, users requesting access to a Kerberos-protected network may have to apply for an account and receive an initial password from a system administrator in person. This process becomes unwieldy in a large environment; however, a different type of cryptographic algorithm provides another approach to the problem of key management.

*Public key*, or *asymmetric key* cryptography splits the encryption key into two parts. Each user in a public key system generates a key pair, consisting of a private key and public key. These two keys are mathematically linked so that messages are encrypted with one key and decrypted with the other. Furthermore, deriving one half of the key given the other half is mathematically very difficult, and the difficulty of performing this operation determines the relative strength of public key encryption algorithms.

The two parts of the encryption key are termed the *private key* and the *public key*. The private key is kept secret, and the public key is widely distributed to everyone that you wish to communicate with. When someone wants to send you an encrypted message, he encrypts the message with your public key. Once encrypted, the only person who can decrypt the message is the person who holds the other half of the key—the private key, which is kept secret. This is the principle of public key cryptography.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 10.2 Smart Cards

Traditionally, Kerberos has relied solely on one of the three factors of authentication, namely, something you know. As discussed early on in [Chapter 2](#), the security of authentication systems can be greatly enhanced by requiring more than one factor to grant authentication. Smart cards provide another factor (what you have), and some Kerberos implementations support the use of smart cards for initial authentication.

The use of smart cards solves one of the most problematic issues with Kerberos; namely its dependence on users to choose (and remember) good passwords. Traditionally, the user's long-term key is a password, which is something the user must choose and memorize. The human brain is notoriously poor at producing and consequently remembering random sequences, so passwords are typically something easily remembered by the user. As a consequence, passwords have low entropy, and most fall to dictionary attacks. The use of pre-authentication in the initial Authentication Server exchange mitigates this risk somewhat, but a determined attacker who can sniff Kerberos protocol exchanges over the network can still obtain encrypted material on which to perform a dictionary attack.

In addition, smart cards limit the exposure of the sensitive cryptographic keys used throughout the Kerberos protocol. Secret keys stored on machine hard disks, such as keytab files, are vulnerable to attack. Even though filesystem protection is designed to prevent unauthorized users from reading sensitive files, software bugs persist that, when exploited, provide attackers with administrative access to the entire computer, including any encryption keys stored within.

Smart cards solve this problem by storing the key material internally on the smart card itself, and never allowing the key material to leave the smart card. Instead, the smart card has enough processing power to perform the cryptographic functions necessary to generate and respond to Kerberos authentication messages. Storing the key material on the smart card and securing the smart card from unauthorized access means that an attacker who has control over the user's workstation can never retrieve the encryption keys stored inside of the smartcard. This also mitigates Trojan horse techniques, where a program masquerading as the Kerberos login program acquires unwitting users' passwords.

Since a smart card is a physical device, it needs an interface to the host computer—the smart card reader. Smart card readers can connect to the host computer through several physical means, including serial, USB, and for laptops, PCMCIA slots. Because of the requirement for specialized hardware connected to the host machine, smart cards are currently only practically deployable in an organization's network.

Attacks on smart cards are difficult, as they are small physical devices designed to resist attack. It requires a determined and well-funded adversary to carry out an attack on a smart card. Analyses of the smart card's power usage and timing have been developed that greatly reduce the search space of possible encryption keys during a brute-force attack on a key stored inside of a smart card. Since the amount of calculations needed to perform encryption algorithms depends on the size and content of the encryption key, these attacks analyze the minute differences in power and time as the smart card performs these operations on various data. Determined attackers can narrow down the possible encryption keys based on this analysis and on detailed knowledge of the algorithms involved

## 10.3 Better Encryption

The art and algorithms of cryptography are always evolving, driven by the explosive growth in computer power and cryptographic theory. Increasing computer power provides a dual driving force for emerging cryptographic algorithms: first, it obsoletes older algorithms and short key lengths as they fall to practical brute-force attacks. A 56-bit single DES key can be brute forced by a network of commodity computers in less than a week, and that time is decreasing rapidly. Conversely, the increase in computing power makes possible the complex calculations of even more sophisticated algorithms and longer key lengths necessary to secure information from prying eyes. Theory drives the development of cryptographic algorithms as well, providing new ways to protect data as well as techniques to crack codes.

Because Kerberos is a system that depends heavily on cryptography, it is crucial that these new encryption methods are implemented in the Kerberos protocol. The Kerberos 5 protocol was designed to be extendable and support multiple encryption types; however, currently the only interoperable encryption type available across Kerberos implementations is single DES. Thankfully, the upcoming release of MIT Kerberos 1.3 will provide wider support for the RC4-HMAC encryption type first introduced by Microsoft for use in Windows 2000's Kerberos service.

For further growth, there are proposed Internet Drafts that specify more, stronger encryption options for future implementations of the Kerberos protocol. The new NIST encryption standard, the Advanced Encryption Standard or AES, is one of the encryption algorithms that is proposed for future implementations of the Kerberos protocol. AES will replace the decades-old DES encryption algorithm as the federal standard for encrypting sensitive but unclassified information. The algorithm for AES, Rijndael, was chosen in 2000 among a field of algorithms submitted by civilian cryptographers from around the world. Rijndael is a block cipher that boasts a variable key size, providing protection against brute force attacks in the foreseeable future.

The latest Kerberos Clarifications require that new Kerberos implementations support AES encryption types, greatly increasing the cryptographic security of future Kerberos implementations. The Kerberos Clarifications have demoted the current single DES encryption type to optional ("SHOULD support") status, due to its small fixed key size. The use of stronger cryptographic algorithms in the future will continue to protect Kerberos from brute-force attacks.

[\[ Team LiB \]](#)

◀ PREVIOUS

NEXT ▶

## 10.4 Kerberos Referrals

As originally implemented in MIT Kerberos 5, each Kerberos client requires detailed configuration information about all realms the client participates in. With Unix clients, the information is coded in the `/etc/krb5.conf` file. This file must be kept up to date and distributed to all clients, which, in large and complex network environments, can quickly become an unwieldy and unmanageable task. Furthermore, machines that are not centrally managed or mobile machines such as laptops are even more problematic, as distributing changes to the Kerberos configuration files to these machines is nearly impossible.

Microsoft recognized the need for a new method for handling this configuration information in a centralized place when it implemented Kerberos in its Windows 2000 operating system, and created a system by which the KDC can provide clients correct replies, even when queries are misdirected or malformed. Through this mechanism, clients only require minimal configuration, enough to find their local Kerberos realm, and all queries are directed to the local KDC, even cross-realm queries destined for a foreign Kerberos realm. The Kerberos support in Microsoft's Windows 2000 and later operating systems includes support for—and, indeed, depends on—the functioning of Kerberos referrals for Windows domain operations.

There are three classes of information that the Microsoft implementation of Kerberos referrals handles for Kerberos clients: user and service principal name canonicalization through the AS and TGS exchanges respectively, and cross-realm trust relationships. Extensions to the Kerberos protocol are used to provide this capability and retain backwards compatibility with implementations that do not understand Kerberos referrals.

But first, what is name canonicalization? Both people and hosts can be known by several names, such as in the case of hostname aliases or multihomed hosts. Each user can have several names associated with him as well, such as his email address or his Kerberos principal name. Since Kerberos associates a single principal with a single key, it is required that these names be changed into a normalized format and name, so that there is a single Kerberos principal that refers uniquely to a given user or host. For example, a host with several IP addresses and several hostnames must be assigned a single "official" hostname. This hostname is referred to as the host's *canonical* hostname. Kerberos referrals move the burden of name canonicalization from the client and DNS to the KDC.

### 10.4.1 User Principal Canonicalization

KDC-side user principal canonicalization provides for both user and administrator convenience. Similar to passwords, users want to remember only one identifying principal. For example, a user's email address may be different than his Kerberos principal name. Furthermore, the Kerberos principal name associated with the user may change if he moves between departments, while his email address stays the same.

Therefore, Kerberos referrals provide a new field in the AS exchange for clients to send an alternative name when acquiring the initial TGT. When the KDC receives an AS request with this field filled in, the KDC will search a directory to map the user name given to a principal that exists in the local Kerberos realm. If the user does not exist in the local realm, the KDC will attempt to find the user in a foreign realm.

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## 10.5 Web Services

Web services are an important new technology, and are used extensively in new frameworks such as Microsoft's .Net. Web services facilitate the transfer of structured data across networks by defining a standardized transport mechanism (for example, SOAP over HTTP). While the explosion of the World Wide Web was due to the large amount of human-readable content available through HTML pages, the development of more complex systems requires a standard by which applications can communicate directly with one another over the web. Web services seek to use the power of the web to provide language- and platform-neutral communication methods that can link applications across many different organizations.

However, current web services typically do not provide secure authentication and encryption support. Many web services that require access control use the authentication and security mechanisms of the underlying protocol (HTTP)—for example, by using Basic Authentication for access control and SSL-encrypted HTTP (HTTPS) for transport. This solution does not scale well, and if the HTTP server is decoupled from the web service, it presents a problem where authentication information for the web service must be kept synchronized with the HTTP server.

To address these shortcomings, the WS-Security specification is under development by IBM, Microsoft, and VeriSign. The WS-Security specification defines a set of SOAP extensions that can be used to provide confidentiality and integrity services to web services. WS-Security defines a set of SOAP messages that can encapsulate generic security token objects and associate these security tokens with specific SOAP messages. Therefore, WS-Security is not tied to one particular algorithm or security system, and indeed can be used with security and authentication protocols such as SSL, X.509 (for public key certificates), and Kerberos.

While WS-Security specifies a standardized format for the transmission and encoding of security tokens and encrypted messages, it does not cover the messages needed to perform the actual authentication or key exchange required to establish secure communications. Instead, the WS-Security proposal delegates this task to the individual security mechanism used in the communications. In the case of Kerberos, the traditional Kerberos protocol is used to establish identity and generate a session key that then can be used by applications that use WS-Security to protect SOAP message exchanges.

Work on WS-Security is ongoing, and general information on WS-Security can be found on Microsoft's MSDN site at

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwssecur/html/securitywhitepaper.asp>.

The full WS-Security specification can be found at IBM's DeveloperWorks web site at

<http://www-106.ibm.com/developerworks/webservices/library/ws-secure/>.



[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

# Appendix A. Administration Reference

Each of the KDC implementations covered in this book has different administrative interfaces. We've already seen the basics of each administrative interface when we set up the KDC, but this section provides an in-depth reference on the various commands available to Kerberos administrators.

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS    NEXT ▶

## A.1 MIT

In MIT Kerberos 5, Kerberos database tasks are performed by the `kadmind` daemon. Normally, this daemon is run on KDC startup when the main Kerberos daemon, `krb5kdc`, is started. The `kadmind` daemon listens for client requests on TCP port 749. The client, `kadmin`, can be run on any machine that is able to communicate with the KDC. It is recommended that a firewall be used to limit network access to port 749 to restrict unauthorized users from connecting to the administrative daemon.

The `kadmin` client uses configuration from `/etc/krb5.conf` to locate the master KDC that runs the `kadmind` server. It will use the value of the `admin_server` parameter located in the realm that the client is a member of. If you compiled with DNS support (the default), it will also attempt to use DNS to locate the admin server service. If these methods fail, `kadmin` will give up attempting to look for a server, and exit with an error message. You can manually specify a realm name and server address with the `-r` and `-s` options, respectively.

After a connection has been established between the `kadmin` client and the `kadmind` server, the client performs mutual authentication with the administration server, using a temporary credential cache to acquire tickets to authenticate with the server, for security reasons.

Note that a MIT `kadmin` client is required to communicate with an MIT `kadmind` server. You cannot use Heimdal `kadmin` to administer an MIT KDC.

There is also a fail-safe copy of `kadmin` named `kadmin.local` that accesses the Kerberos database directly on the KDC. To run `kadmin.local`, you must be root on the master KDC. The commands available in `kadmin.local` are the same as the commands available in `kadmin`.

### A.1.1 Connecting to kadmin

You do not need any special privileges on the client machine to use `kadmin`; all you need is the password to a principal that has privileges enabled in the `kadmin` ACL file.

```
% /usr/local/sbin/kadmin
Authenticating as principal jgarman/admin@WEDGIE.ORG with password.
Enter password:
kadmin:
```

Inside of `kadmin`, a list of available commands is always available through the `"?"` command.

The fail-safe copy of `kadmin`, `kadmin.local`, requires root privileges to run. You have to be root on the master KDC to run `kadmin.local` because it modifies the Kerberos database directly. As such, running `kadmin.local` does not require any Kerberos daemons to be running, and in fact it is most commonly used to set up the initial Kerberos principals when establishing a new realm. Starting `kadmin.local` is very similar to starting `kadmin`:

```
# /usr/local/sbin/kadmin.local
Authenticating as principal jgarman/admin@WEDGIE.ORG with password.
```

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

[\[ Team LiB \]](#)

◀ PREVIOUS | NEXT ▶

## A.2 Configuration File Format

Both MIT and Heimdal use the same basic format for their text configuration file, `krb5.conf`. This file contains all of the information needed for the Kerberos libraries that are linked into Kerberos clients, servers, administrative utilities, and the KDC itself. Since this file is rather standardized between the major Kerberos implementations on Unix, a `krb5.conf` file generated for one can easily be used on another implementation, usually with no changes required.

While normally this configuration file is located in `/etc`, an alternate location can be defined by setting the `KRB5_CONFIG` environment variable. Both MIT and Heimdal honor this environment variable. For example, in a Bourne shell, the following command would instruct further Kerberos applications to use the `/etc/krb5.conf.backup` file as the Kerberos configuration file instead:

```
% export KRB5_CONFIG=/etc/krb5.conf.backup
```

The `krb5.conf` file is comprised of a number of key-value pairs, organized into groups, referred to as *stanzas*. Stanza names are enclosed in opening and closing brackets, and each key/value pair must belong to one stanza. Key/value pairs are separated by an equals sign, with the key name on the left and its associated value on the right of the equals sign. The value in a key/value pair can either be a single value, or it can be another set of key/value pairs, enclosed by braces. The most common example of this is in the `realms` stanza, where a key is defined for each realm, whose value is another set of key/value pairs defining the KDC and other important servers in that realm. That is, each key/value pair can take on of the two following forms:

```
key = value
```

```
keyWithSubkeys = {  
    subkey1 = value  
    subkey2 = value  
};
```

A sample `krb5.conf` file is shown below:

```
[libdefaults]  
    default_realm = SAMPLE.COM  
  
[realms]  
    SAMPLE.COM = {  
        kdc = kerberos.sample.com:88  
        kdc = kerberos-2.sample.com:88  
        admin_server = kerberos.sample.com  
    };  
  
    W2K.SAMPLE.COM = {  
        kdc = windows.sample.com:88  
    };  
  
[domain_realm]  
    .sample.com = SAMPLE.COM  
    sample.com = SAMPLE.COM  
    testbox.sample.com = W2K.SAMPLE.COM  
    windows.sample.com = W2K.SAMPLE.COM
```

This `krb5.conf` file defines three stanzas: `libdefaults`, `realms`, and `domain_realm`. These are the most common stanzas that are present in `krb5.conf` files and they represent the basic data that every Kerberos client and service needs to have in order to participate in the Kerberos protocol. The following six

[\[ Team LiB \]](#)

[← PREVIOUS](#) [NEXT →](#)

[\[ Team LiB \]](#)

[← PREVIOUS](#)

## Colophon

Our look is the result of reader comments, our own experimentation, and feedback from distribution channels. Distinctive covers complement our distinctive approach to technical topics, breathing personality and life into potentially dry subjects.

The animal on the cover of *Kerberos: The Definitive Guide* is a barred owl (*Strix varia*). It is distinguished by the brown bar markings on its chest and the distinctive dark rings around its eyes. The barred owl generally resides in the woodlands of North America, making its nest in the cavity of trees. The female owl lays between two to four eggs, and the parents often remain with the young for more than four months, making them an exception among other types of owls. Barred owls stay with their chosen mate for life and tend to live about 20 years.

The barred owl eats mainly small mammals, such as mice, shrews, and squirrels. The barred owl will also eat birds, fish, frogs, and insects. These owls can grow to approximately 17 inches, with a wingspan of 44 inches. They are territorial in spring and fall, hooting at other owls to warn them against intruding. The barred owl's only natural predator is the Great Horned owl, but many deaths are attributed to human influence, such as shooting, car accident, or loss of habitat.

Colleen Gorman was the production editor and the copyeditor for *Kerberos: The Definitive Guide*. Mary Brady, Marlowe Shaeffer, Jane Ellin, and Mary Anne Weeks Mayo provided quality control. John Bickelhaupt wrote the index.

Ellie Volckhausen designed the cover of this book, based on a series design by Edie Freedman. The cover image is an engraving from Heck's Pictorial Archive of Nature and Science. Jessamyn Read produced the cover layout with QuarkXPress 4.1 using Adobe's ITC Garamond font.

David Futato designed the interior layout. This book was converted by Julie Hawks to FrameMaker 5.5.6 with a format conversion tool created by Erik Ray, Jason McIntosh, Neil Walls, and Mike Sierra that uses Perl and XML technologies. The text font is Linotype Birka; the heading font is Adobe Myriad Condensed; and the code font is LucasFont's TheSans Mono Condensed. The illustrations that appear in the book were produced by Robert Romano and Jessamyn Read using Macromedia FreeHand 9 and Adobe Photoshop 6. The tip and warning icons were drawn by Christopher Bing. This colophon was written by Colleen Gorman.

The online edition of this book was created by the Safari production group (John Chodacki, Becki Maisch, and Madeleine Newell) using a set of Frame-to-XML conversion and cleanup tools written and maintained by Erik Ray, Benn Salter, John Chodacki, and Jeff Liggett.

[\[ Team LiB \]](#)

◀ PREVIOUS

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

\* ([glob character](#))  
[56-bit keys, insecurity of](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[abstract syntax](#)

[Abstract Syntax Notation One](#) [See ASN.1]

[account logon auditing](#)

[account management \(PAM\)](#)

[ACL files](#)

[active attacks](#) [2nd](#)

[Active Directory domain, establishing](#)

[Active Port tool](#)

[add command \(Heimdal kadmin\)](#)

[add command \(Heimdal ktutil\)](#)

[addent command \(MIT ktutil\)](#)

[addprinc command \(MIT kadmin\)](#)

[addressless tickets](#)

[admin\\_server variable](#)

[admind4 daemon](#)

[administration](#)

[Aeneid](#)

[AES \(Advanced Encryption Standard\)](#)

[AFS network filesystem Kerberos 4-based service](#)

[Andrew File System](#)

[Apache mod\\_auth\\_kerb module](#) [See mod\_auth\_kerb module]

[appdefaults stanza](#)

[Apple Mail.app](#)

[application servers](#)

[authentication of](#)

[Kerberos software, installing on](#)

[applications](#)

[configuring](#)

[Kerberos, types of support](#)

[web-based, Kerberos and](#)

[apxs2 program](#)

[AS\\_REP \(Authentication Server Reply\) messages](#)

[AS\\_REQ \(Authentication Server Request\) messages](#)

[ASN.1 \(Abstract Syntax Notation One\)](#)

[ASs \(Authentication Servers\)](#) [2nd](#)

[initial authentication with PKINIT](#)

[asymmetric key cryptography](#) [See public key cryptography]

[Athena Technical Plan](#) [2nd](#)

[attacks](#)

[active attacks](#) [2nd](#)

[brute-force attacks](#)

[clients, root compromise of](#)

[denial of service](#)

[dictionary attacks](#)

[exploitation of software flaws](#)

[insider attacks](#)

[man-in-the-middle attacks](#)

[replay attacks](#)

[servers, root compromise of](#)



[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[Basic authentication](#)

[Bastille Linux](#)

[biometrics](#)

[BNF \(Backus-Naur Form\)](#)

[brute-force attacks](#) [2nd](#)

[Bugtraq](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)]  
] [[X](#)] [[Y](#)]

canonicalization

[DNS dependent method](#)

[name canonicalization](#)

[service name canonicalization](#)

[user principal canonicalization](#)

[capaths stanza](#)

[case study](#)

[applications, configuring](#)

[disk layout and backup](#)

[Heimdal Kerberos 2nd](#)

[KDCs, planning](#)

[Kerberos realms](#)

[implementation](#)

[Kerberos realms, planning](#)

[MIT Kerberos 2nd](#)

[partitioning](#)

[pre-existing network structure 2nd](#)

[Cerberus 2nd](#)

[certificates](#)

[Certified Security Solutions](#)

[character classes](#)

[clear command \(MIT ktutil\)](#)

[clear text](#)

[client servers](#)

[Kerberos software, installing on](#)

[client software](#) [See applications]

[client-server model](#)

[clients](#)

[Kerberos-enabled packages](#)

[electronic mail](#)

[PuTTY](#)

[Reflection X](#)

[clock synchronization](#)

[unsynchronized clocks, problems arising from](#)

[collisions](#)

[compute clusters](#)

[configuration file format, MIT and Heimdal Kerberos](#)

[cpw command \(Heimdal kadmin\)](#)

[cpw command \(MIT kadmin\) 2nd](#)

[cracklib](#)

[create command, kdb5\\_util program](#)

[credential forwarding](#)

[credentials cache](#)

[credentials lifetimes](#)

[cross-realm authentication](#)

[PKCROSS, using](#)

[cross-realm referral](#)

[cross-realm relationships, establishing](#)

[cross-realm trust 2nd](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[DCE \(Distributed Computing Environment\)](#)

[dcpromo command](#)

[debugging tools](#)

[decision tree for troubleshooting](#)

[delent command \(MIT ktutil\)](#)

[delete command \(Heimdal kadmin\)](#)

[delprinc command \(MIT kadmin\)](#)

[denial of service attacks](#)

[DER \(Distinguished Encoding Rule\)](#)

[DES \(Data Encryption Standard\), insecurity of 2nd](#)

[dictionary attacks](#)

[Digest authentication](#)

[directories](#)

[directory services using OpenLDAP](#)

[Distributed Computing Environment \(DCE\)](#)

[DNS \(Domain Name Service\)](#)

[canonicalization method using](#)

[domain name-to-realm mapping](#)

[forward and reverse mappings](#)

[KDC discovery](#)

[and Kerberos 2nd](#)

[realms, creating](#)

[Kerberos Clarifications call for reduced dependency on](#)

[domain\\_realm stanza](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[eBones](#)

[email](#)

[Cyrus IMAP, using](#)

[Eudora client](#)

[encrypted timestamp pre-authentication \(PA-ENC-TIMESTAMP\)](#)

[encryption](#) [2nd](#) [\[See also cryptography\]](#)

[Kerberos 5, types used in](#)

[type mismatches, troubleshooting](#)

[errors](#)

[add\\_principal: Operation requires "add" privilege...](#)

[Cannot contact any KDC for requested realm](#)

[Cannot set GSS-API authentication names](#)

[incorrect net address](#)

[Preauthentication failed](#)

[Eudora mail client](#)

[event 675](#)

[ext\\_keytab command \(Heimdal kadmin\)](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[factors](#)

[smart cards](#)

[forwardable tickets](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[Generic Security Services API](#) [See GSSAPI]

[get command \(Heimdal kadmin\)](#)

[get command \(Heimdal ktutil\)](#)

[getprinc command \(MIT kadmin\)](#) [2nd](#)

[Globus Security Infrastructure](#)

[grid computing](#)

[GSSAPI \(Generic Security Services API\)](#)

[gssapi\\_client/gssapi\\_server applications](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[hashes](#)

[Heimdal Kerberos 2nd](#)

[adding an admin user](#)

[adding slave KDCs](#)

[administrative commands](#)

[add \(kadmin\)](#)

[add \(ktutil\)](#)

[cpw \(kadmin\)](#)

[delete \(kadmin\)](#)

[ext\\_keytab \(kadmin\)](#)

[get \(kadmin\)](#)

[get \(ktutil\)](#)

[list \(kadmin\)](#)

[list \(ktutil\)](#)

[modify \(kadmin\)](#)

[remove \(ktutil\)](#)

[building from source](#)

[case study 2nd](#)

[configuration file format](#)

[configuration options](#)

[kadmin](#)

[kadmin program, incompatibility with Windows domain controllers](#)

[kstash command](#)

[kth-krb dependency for Kerberos 4 compatibility](#)

[ktutil](#)

[logging](#)

[interpretation of data](#)

[Master KDC daemons 2nd](#)

[password lifetimes, setting](#)

[password strength-checking](#)

[pre-authentication](#)

[realm configuration files](#)

[realms, creating](#)

[sample client/server applications](#)

[servers, starting](#)

[software build and installation](#)

[software, installation on client and application servers](#)

[testing](#)

[troubleshooting the krb5.conf file](#)

[whitespace and the kadmind.acl file](#)

[heimdal.db](#)

[High Performance Computing \(HPC\)](#)

[Horowitz password-changing protocol](#)

[host principals](#)

[creating](#)

[HPC \(High Performance Computing\)](#)

[hprop command](#)

[hpropd daemon](#)

[htaccess files](#)

[kth-krb5.conf](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[IIS \(Internet Information Services\), security concerns](#)  
[instances](#)

[Internet Engineering Task Force's \(IETF\) Kerberos working group](#)

[Internet Information Services \(IIS\), security concerns](#)

[interoperability, Unix and Windows](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[John the Ripper password cracker](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)] [[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)]  
] [[X](#)] [[Y](#)]

[kadmin](#)

[kadmin commands](#)

[logging](#)

[password expiration, setting](#)

[kadmin.local](#)

[kadmin.local program](#)

[kadmin daemon](#) [2nd](#) [3rd](#)

[kadmin variable](#)

[kdb5\\_util program](#)

[create command](#)

[kdc daemon](#)

[kdc variable](#)

[kdc.conf file](#) [2nd](#)

[KDCs \(Key Distribution Centers\)](#) [2nd](#) [3rd](#)

[cross-realm communication using PKCROSS](#)

[database replication](#)

[discovery over DNS](#)

[logging](#)

[messages to and from](#)

[MIT Kerberos 5, adding slaves](#)

[MIT version, installing from source](#)

[security](#)

[continual maintenance](#)

[security concerns](#)

[Unix](#)

[server-client ratio, Windows vs. Unix](#)

[servers, compromise of](#)

[software build and install](#)

[using Windows domain controllers for Unix clients](#)

[kerberized services](#)

[kerberized telnet, debugging with](#)

[Kerberos](#) [2nd](#) [3rd](#)

[application server packages](#)

[authentication, advantages for](#)

[case study](#) [See case study]

[client packages](#)

[electronic mail](#)

[PuTTY](#)

[Reflection X](#)

[client software](#) [See applications]

[clock synchronization, need for](#)

[and DNS](#)

[and DNS \(Domain Name Service\)](#)

[evolution of the protocol](#)

[future developments](#) [2nd](#)

[cryptography, improvements](#)

[web services, use in](#)

[Heimdal](#) [See Heimdal Kerberos]

[implementation](#)



[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[LDAP \(Lightweight Directory Access Protocol\)](#)

[libdefaults stanza](#)

[Lightweight Directory Access Protocol \(LDAP\)](#)

[limited delegation](#)

[list command \(Heimdal kadmin\)](#)

[list command \(Heimdal ktutil\)](#)

[list command \(MIT ktutil\)](#)

[listprincs command \(MIT kadmin\)](#) [2nd](#)

[logging](#) [\[See also auditing\]](#)

[Heimdal Kerberos](#) [2nd](#)

[interpretation of data](#)

[of KDC contacts](#)

[MIT Kerberos](#) [2nd](#)

[Pre-authentication Failed messages](#)

[Windows domain controllers](#) [2nd](#)

[logging stanza](#)

[loginwindow \(Mac OS X\)](#)

[logon auditing](#)

[lsof tool](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[m-key file](#)

[Mac OS X](#)

[and host keytabs](#)

[loginwindow](#)

[Mail.app, kerberos-supporting mail client](#)

[Mail.app](#)

[man-in-the-middle attacks](#)

[Massachusetts Institute for Technology \(MIT\)](#)

[master keys](#)

[message integrity](#)

[algorithms](#)

[Microsoft ksetup tool](#)

[Microsoft Management Console \(MMC\)](#)

[Microsoft Solutions for Securing Windows 2000 Server](#)

[Microsoft Windows](#) [\[See Windows\]](#)

[MIT \(Massachusetts Institute for Technology\)](#)

[MIT Kerberos](#) [2nd](#)

[adding slave KDCs](#)

[administrative commands](#)

[addent \(ktutil\)](#)

[addprinc \(kadmin\)](#)

[clear \(ktutil\)](#)

[cpw \(kadmin\)](#)

[delent \(ktutil\)](#)

[delprinc \(kadmin\)](#)

[getprinc \(kadmin\)](#)

[ktadd \(kadmin\)](#)

[list \(ktutil\)](#)

[listprincs \(kadmin\)](#)

[modprinc \(kadmin\)](#)

[rkt \(ktutil\)](#)

[wkt \(ktutil\)](#)

[case study](#) [2nd](#)

[configuration file format](#)

[daemons](#)

[installation on servers](#)

[kadmin](#)

[connecting to](#)

[kadmin program, incompatibility with Windows domain controllers](#)

[KDC, installation from source](#)

[krb5.conf file, troubleshooting](#)

[krb524d daemon](#)

[ktutil](#)

[logging](#)

[interpretation of data](#)

[password lifetimes, setting](#)

[password strength-checking](#)

[pre-authentication](#)

[principals, configuration of administrative privileges](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[name canonicalization](#)

[SASL](#)

[NAT \(Network Address Translation\) firewalls and Kerberos](#)

[native Kerberos authentication](#)

[Needham, Roger](#)

[Needham-Schroeder protocol](#)

[Kerberos 4, compared to](#)

[Kerberos, compared to](#)

[Negotiate authentication](#)

[NetBIOS name and Windows Kerberos realms](#)

[netstat command](#)

[network "sniffers"](#)

[network authentication systems](#)

[network services, determining what's installed](#)

[network-wide single-sign-on example](#) [See case study]

[NFS protocol](#)

[NTLM authentication](#)

[NTP \(Network Time Protocol\)](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[offline dictionary attacks](#)

[one-way hashes](#)

[OpenLDAP](#)

[OpenSSH 2nd](#)

[operating systems, installation and security](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[PA-ENC-TIMESTAMP \(encrypted timestamp pre-authentication\)](#)

[PAC \(Privilege Access Certificate\)](#)

[PAM \(Pluggable Authentication Modules\)](#)

[configuring](#)

[control field](#)

[limitations](#)

[module name field](#)

[stacking](#)

[tasks](#)

[web sites](#)

[pam\\_krb5 module](#)

[options](#)

[partitioning of Unix hosts](#)

[password histories](#)

[password verifiers](#)

[passwords](#) [2nd](#)

[changing](#)

[Heimdal Kerberos](#)

[MIT Kerberos](#)

[Windows domain controllers](#)

[encryption keys, conversion to](#)

[Heimdal strength-checking](#)

[John the Ripper cracker program](#)

[Kerberos 5, changing in](#)

[MIT strength-checking](#)

[policies, enforcing](#)

[security](#)

[enforcing](#)

[Windows strength-checking](#)

[PGP \(Pretty Good Privacy\)](#)

[PKCROSS](#)

[PKINIT](#)

[ports needed by KDCs](#)

[postdated tickets](#)

[pre-authentication](#) [2nd](#) [3rd](#)

[principals](#)

[administration](#)

[Heimdal Kerberos](#)

[MIT Kerberos](#)

[Windows domain controllers](#)

[ASN.1 definitions](#)

[configuring administrative privileges of \(MIT Kerberos\)](#)

[creating](#)

[Kerberos 5](#)

[service and host](#)

[target principals](#)

[private key cryptography](#)

[private keys](#)

[Project Athena](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[Qualcomm](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[RC4-HMAC encryption type](#)

[improving support for](#)

[realm names](#)

[realms 2nd](#)

[ASN.1 definitions](#)

[configuration files 2nd](#)

[considerations in planning](#)

[creating 2nd](#)

[cross-realm authentication](#)

[cross-realm relationships, setting up](#)

[DNS mappings](#)

[Heimdal Kerberos, creating in](#)

[MIT Kerberos 5, creation using](#)

[TGSs \(Ticket Granting Servers\)](#)

[Windows domain controllers, creation using](#)

[realms stanza](#)

[Reflection X](#)

[configuring for Kerberos authentication](#)

[remote login server package \(OpenSSH\)](#)

[remove command \(Heimdal ktutil\)](#)

[renewable tickets](#)

[replay attacks 2nd](#)

[replay caches](#)

[rkt command \(MIT ktutil\)](#)

[RSA Laboratories](#)

[RSA public key algorithm](#)

[RSA SecurID token](#)

[\[ Team LiB \]](#)

[ Team LiB ]

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[salt](#) [2nd](#)

[SASL \(Simple Authentication and Security Layer\)](#)

[building](#)

[configuring](#)

[dependencies](#)

[name canonicalization](#)

[sasauthd](#)

[configuring](#)

[testing](#)

[web sites](#)

[sasldb](#)

[Schroeder, Michael](#)

[Secure European System for Applications in a Multivendor Environment \(SESAME\)](#)

[SecurID token](#)

[security](#)

[auditing](#) [\[See also logging\]](#) [2nd](#)

[KDCs, protecting](#)

[Kerberos, protocol issues](#)

[and key length](#)

[operating system installation](#)

[passwords](#)

[enforcing security standards](#)

[policies, enforcing](#)

[pre-authentication](#)

[root-level attacks](#)

[smart cards and](#)

[Unix patches, sources](#)

[server hostname misconfiguration, troubleshooting](#)

[servers](#)

[Kerberos-enabled packages](#)

[Cyrus IMAP](#)

[OpenLDAP](#)

[OpenSSH](#)

[service keys](#)

[service name canonicalization](#)

[service principals](#) [2nd](#)

[Kerberos](#) [4](#)

[service tickets](#)

[SESAME \(Secure European System for Applications in a Multivendor Environment\)](#)

[session keys](#) [2nd](#) [3rd](#)

[encryption type \(Kerberos 5\)](#)

[shared secrets](#)

[shell clients, kerberized](#)

[Simple and Protected GSSAPI Negotiation Mechanism](#) [\[See SPNEGO\]](#)

[Simple Authentication and Security Layer](#) [\[See SASL\]](#)

[single-sign-on systems](#)

[TGTs and](#)

[slave replication, setting up](#)

[smart cards](#) [2nd](#)

[Kerberos](#) [1](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

[target principals](#)

[tcp\\_client/tcp\\_server applications](#)

[TGSs \(Ticket Granting Servers\)](#) [2nd](#) [3rd](#)

[TGTs \(Ticket Granting Tickets\)](#) [2nd](#)

[errors obtaining](#)

[plaintext attacks against](#)

[three A's](#)

[Ticket Granting Servers](#) [\[See TGSs\]](#)

[Ticket Granting Tickets](#) [\[See TGTs\]](#)

[tickets](#) [2nd](#) [3rd](#) [4th](#)

[addressless tickets](#)

[cache](#)

[encryption type \(Kerberos 5\)](#)

[Kerberos 5 options](#)

[Kerberos 5-to4 translation](#)

[lifetime of](#) [2nd](#)

[service tickets](#)

[time-sharing model](#)

[TLS \(Transport Layer Security\)](#)

[TLV rule](#)

[transfer syntax](#)

[transparent Kerberos logins](#)

[troubleshooting](#)

[client problems](#)

[debugging tools](#)

[encryption type mismatches](#)

[kerberized telnet, debugging with](#)

[Kerberos configuration, missing or incorrect](#)

[krb5.conf file](#)

[mismatched domain name-to-realm mapping](#)

[problems, categorizing](#)

[server hostname misconfiguration](#)

[TGTs, errors obtaining](#)

[unsynchronized clocks](#)

[trusted third-parties](#)

[type](#)

[\[ Team LiB \]](#)



[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

Unix

[KDCs, securing on](#)

[keytabs, creating from Windows domain controllers](#)

[partition table for hosts](#)

[security enhancement tools](#)

[security patches, sources of](#)

[Windows interoperability](#)

[Unix clock synchronization](#)

[user principal canonicalization](#)

[usernames](#)

[UTC \(Universal Coordinated Time\)](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[v5passwd daemon](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Y\]](#)

Web Security, Privacy & Commerce

[book web site](#)

[Web services](#)

[web-based authentication](#)

Windows

[Active Directory authorization field](#)

[cached login credentials](#)

[disabling](#)

[client authentication against non-Microsoft KDCs](#)

[standalone clients](#)

[clock synchronization](#)

[cross-realm trust](#)

[encryption algorithm support](#)

Kerberos

[implementation issues](#)

[incompatibility, pre-Windows 2000 systems](#)

[support in](#)

[Kerberos implementations](#)

[ktpass program](#)

[supported salts](#)

[Unix interoperability](#)

[X11 Unix applications, accessing from](#)

[Windows 2000 Hardening Guide](#)

[Windows 2000, rotating logfile size, setting](#)

[Windows domain controllers 2nd](#)

[administration](#)

[adding keys into keytabs](#)

[adding principals](#)

[changing passwords](#)

[deleting principals](#)

[listing principals](#)

[modifying principal attributes](#)

[as KDCs for Unix clients](#)

[DNS service and](#)

[logging](#)

[interpretation of data](#)

[password lifetimes, setting](#)

[password strength-checking](#)

[pre-authentication](#)

[realms, creating](#)

[restricting login privileges](#)

[security concerns](#)

[Unix keytabs, creating from](#)

[wkt command \(MIT ktutil\)](#)

[WRQ, Inc.](#)

[WS-Security specification](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[Xerox Palo Alto Research Center](#)

[\[ Team LiB \]](#)

[\[ Team LiB \]](#)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#) [\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#)  
[\[X\]](#) [\[Y\]](#)

[Young, Errol](#)

[Yu, Tom](#)

[\[ Team LiB \]](#)