Managing Kerberos and Other Authentication Services in Oracle® Solaris 11.3

Part No: E54787
November 2016

ORACLE®

Managing Kerberos and Other Authentication Services in Oracle Solaris 11.3

Part No: E54787

Référence: E54787

Accessibilité de la documentation

Pour plus d'informations sur l'engagement d'Oracle pour l'accessibilité à la documentation, visitez le site Web Oracle Accessibility Program, à l'adresse `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Accès aux services de support Oracle

Les clients Oracle qui ont souscrit un contrat de support ont accès au support électronique via My Oracle Support. Pour plus d'informations, visitez le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` ou le site `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` si vous êtes malentendant.

Contents

Tables

Examples

Using This Documentation

- **Overview** – Describes how to administer secure authentication on one or more Oracle Solaris systems. The guide covers Pluggable Authentication Modules (PAM), Kerberos, the Simple Authentication and Security Layer (SASL), and Secure RPC for NFS and NIS. An appendix describes DTrace probes for Kerberos with examples of their use.
- **Audience** – System, security, and network security administrators.
- **Required knowledge** – Access requirements and network security requirements.

Product Documentation Library

Documentation and resources for this product and related products are available at http://www.oracle.com/pls/topic/lookup?ctx=E53394.

Feedback

Provide feedback about this documentation at http://www.oracle.com/goto/docfeedback.

♦ ♦ ♦ CHAPTER 1

Using Pluggable Authentication Modules

This chapter covers the pluggable authentication module (PAM). PAM provides a framework to plug in checks for users of applications on the Oracle Solaris OS. PAM provides a central framework for managing the use of applications, including authenticating the user, managing password changes, closing and opening the user's session, and tracking account limitations, such as time of day. PAM is extensible to third-party applications, and therefore can provide seamless management of access to services on a system.

This chapter covers the following topics:

About PAM

PAM provides a framework for applications to perform various authentication functions. It provides central authentication, session management, password management, and account limitations for users of applications, including system programs. PAM enables these programs, such as `login`, `su`, and `ssh`, to remain unchanged when user management details change. Site applications can use PAM to manage their own account, credential, session, and password requirements. PAM is "plugged in" to these applications.

Introduction to the PAM Framework

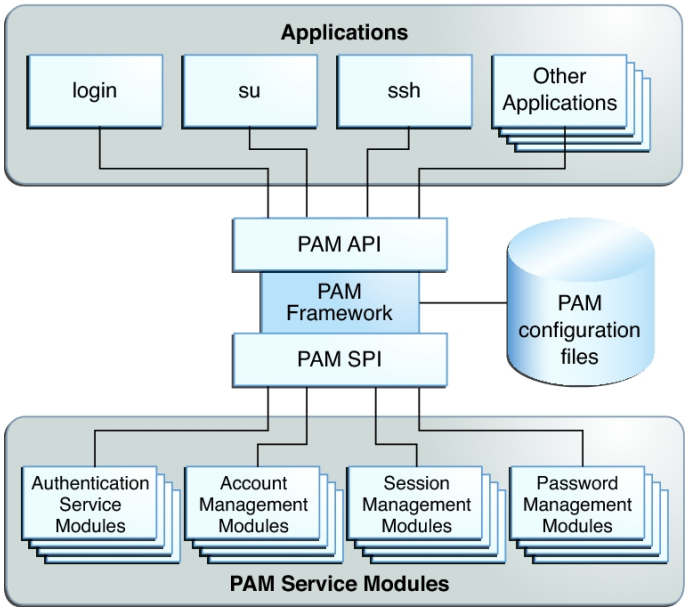
The PAM framework consists of four parts:

- Applications that use PAM
- PAM framework
- PAM service modules
- PAM configuration, including choice of modules and user assignment

The framework provides a uniform way for authentication-related activities to take place. This approach enables application developers to use PAM services without having to know the semantics of the authentication policy. With PAM, administrators can tailor the authentication process to the needs of a particular system without having to change any applications. Rather, administrators adjust the PAM configuration.

The following figure illustrates the PAM architecture.

FIGURE 1 PAM Architecture



The architecture works as follows:

- Applications communicate with the PAM framework through the PAM application programming interface (API).

  For information about using the API, see the `pam`(3PAM) man page and Chapter 3, "Writing PAM Applications and Services" in *Developer's Guide to Oracle Solaris 11 Security*.
- PAM service modules communicate with the PAM framework through the PAM service provider interface (SPI). For more information, see the `pam_sm`(3PAM) man page.

  For a brief description of selected service modules, see "PAM Service Modules" on page 37 and the `pam.conf`(4) and `pam_user_policy`(5) man pages.

Administrators can configure one or more series of modules to manage site requirements. This series of modules is called a PAM *stack*. The stack is evaluated in order. If an application

requires more than one PAM stack, the application developer must create more than one *service name*. For example, the `sshd` daemon provides and requires several service names for PAM. For the list of PAM service names for the `sshd` daemon, search for the word `PAM` in the `sshd(1M)` man page. For details of the PAM stack, see "PAM Stacking" on page 32. "PAM Stacking Example" on page 36 steps through a PAM authentication stack.

Benefits of Using PAM

The PAM framework enables you to configure the requirements that users must satisfy to use an application. The following are some of the benefits that PAM provides:

- Flexible PAM configuration policy
  - Per-service name authentication policy
  - Site-wide PAM policy and per-user PAM policy
  - Administrative choice of a default authentication policy
  - Enforcement of multiple user requirements on high-security systems
- Ease of use for the end user
  - No retyping of identical passwords for different authentication services
  - User prompting by multiple authentication services rather than requiring a user to type multiple commands
- Ease of configuration for the administrator
  - The ability to pass options to the PAM service module
  - The ability to implement a site-specific security policy without having to change the applications

Planning a Site-Specific PAM Configuration

As delivered, the PAM configuration implements a standard security policy that covers system services that require authentication, such as `login` and `ssh`. If you need to implement a different security policy for some system services or create a policy for third-party applications, consider the following issues:

- Determine that the provided configuration files do not satisfy your requirements.

  Test the default configuration. Test the per-user files in the `/etc/security/pam_policy` directory. Test whether the default service name `other` handles your requirements. "PAM Stacking Example" on page 36 steps you through the `other` stack.
- Identify any service names whose stack needs modification. For an example of modifying a service name's PAM stack, see "How to Create a Site-Specific PAM Configuration File" on page 21.
- For any third-party application that is coded to use the PAM framework, determine the PAM service names that the application uses.

- For each service name, determine which PAM modules to use.

  Review the section 5 man pages for the PAM modules. These man pages describe how each module functions, what options are available, and the interactions between stacked modules. For a brief summary of selected modules, see "PAM Service Modules" on page 37. PAM modules are also available from outside sources.
- Per service name, decide the order in which to run the modules.
- Select the control flag for each module. For more information about control flags, see "PAM Stacking" on page 32. Note that the control flags can have security implications.

  For a visual representation, see Figure 2, "PAM Stacking: Effect of Control Flags," on page 35 and Figure 3, "PAM Stacking: How Integrated Value Is Determined," on page 36.
- Choose the options that are necessary for each module. The man page for each module lists the options that are available for that module.
- Test the use of the application with the PAM configuration. Test as the `root` role, other roles, privileged users, and regular users. If some users are not permitted to use the application, test those users.

Assigning a Per-User PAM Policy

The `pam_user_policy` PAM module enables system administrators to assign specific PAM configurations on a per-user basis. When this module is the first module in a PAM stack, and the `pam_policy` security attribute for the user specifies a PAM configuration file, that file specifies the PAM policy for the user.

For more information, see the following:

- `pam_user_policy(5)` man page
- "PAM Stacking" on page 32
- "How to Create a Site-Specific PAM Configuration File" on page 21
- Example 4, "Setting Per-User PAM Policy by Using a Rights Profile," on page 25

Configuring PAM

You can use PAM as is. This section gives examples of PAM configurations that are not in effect by default.

TABLE 1 PAM Task Map

Task	Description	For Instructions
Plan for your PAM installation.	Covers how to plan customizing PAM for your site.	"Planning a Site-Specific PAM Configuration" on page 19

Task	Description	For Instructions
Assign a new PAM policy to a user.	Customizes per-user authentication requirements for multiple services.	"How to Create a Site-Specific PAM Configuration File" on page 21
Create users with encrypted home directories.	Modifies a PAM stack to enable the creation of encrypted home directories.	Example 1, "Using a Modified PAM Stack to Create an Encrypted Home Directory," on page 22
Add new PAM modules.	Explains how to install and test customized PAM modules.	"How to Add a PAM Module" on page 23
Assign a non-default PAM policy to users.	Shows how to add a PAM policy to a rights profile for assignment to a range of users at sites that use Kerberos, LDAP, or a combination of logins.	"How to Assign a Modified PAM Policy" on page 25
Assign a non-default PAM policy to users.	Distributes customized PAM stacks to all systems.	"How to Assign a Modified PAM Policy" on page 25
Initiate error logging.	Logs PAM error messages through `syslog`.	"How to Log PAM Error Reports" on page 28
Troubleshoot PAM errors.	Provides steps to locate, solve, and test PAM misconfigurations.	"How to Troubleshoot PAM Configuration Errors" on page 29

▼ How to Create a Site-Specific PAM Configuration File

In the default configuration, the `ssh` and `telnet` entry services are covered by the `other` service name. The PAM configuration file in this procedure changes the requirements for `ssh` and `telnet`.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. Create a new PAM policy configuration file.

Use the `pfedit` command to create the file. Place the file in a site configuration directory such as `/opt`. You can also place it in the `/etc/security/pam_policy` directory.

Note - Do not modify existing files in the `/etc/security/pam_policy` directory.

Include explanatory comments in the file.

```
# pfedit /opt/local_pam/ssh-telnet-conf
#
# PAM configuration which uses UNIX authentication for console logins,
# (see pam.d/login), and LDAP for SSH keyboard-interactive logins
# This stack explicitly denies telnet logins.
#
sshd-kbdint  auth requisite          pam_authtok_get.so.1
```

```
sshd-kbdint  auth binding            pam_unix_auth.so.1 server_policy
sshd-kbdint  auth required           pam_unix_cred.so.1
sshd-kbdint  auth required           pam_ldap.so.1
#
telnet auth     requisite    pam_deny.so.1
telnet account  requisite    pam_deny.so.1
telnet session  requisite    pam_deny.so.1
telnet password requisite    pam_deny.so.1
```

2. **Protect the file.**

   Protect the file with `root` ownership and `444` permissions.

```
# ls -l /opt/local_pam

total 5
-r--r--r--   1 root         4570 Jun 21 12:08 ssh-telnet-conf
```

3. **Assign the policy.**

   See "How to Assign a Modified PAM Policy" on page 25.

Example 1 Using a Modified PAM Stack to Create an Encrypted Home Directory

By default, the `zfs_pam_key` module is not in the `/etc/security/pam_policy/unix` file. In this example, the administrator creates a version of the `unix` PAM per-user policy, then uses the new version to create users whose home directories are encrypted.

```
# cp /etc/security/pam_policy/unix /opt/local_pam/unix-encrypt
# pfedit /opt/local_pam/unix-encrypt.conf
...
other   auth required           pam_unix_auth.so.1
other   auth required           pam_unix_cred.so.1
## pam_zfs_key auto-creates an encrypted home directory
##
other auth required           pam_zfs_key.so.1 create
```

The administrator uses this policy file when adding users. Note that encryption cannot be added to a filesystem. The filesystem must be created with encryption turned on. For more information, see the `zfs_encrypt(1M)`.

The administrator creates a user and assigns a password.

```
# useradd -K pam_policy=/opt/local_pam/unix-encrypt.conf jill
# passwd jill
New Password: xxxxxxxx
Re-enter new Password: xxxxxxxx
passwd: password successfully changed for jill
```

Then, the administrator creates the encrypted home directory by logging in as the user.

```
# su - jill
Password: xxxxxxxx
Creating home directory with encryption=on.
Your login password will be used as the wrapping key.
Oracle Corporation      SunOS 5.11      11.3    October 2014

# logout
```

For the options to the ZFS service module, see the pam_zfs_key(5) man page.

Finally, the administrator verifies that the new home directory is an encrypted filesystem.

```
# mount -p | grep ~jill
rpool/export/home/jill - /export/home/jill zfs - no
rw,devices,setuid,nonbmand,exec,rstchown,xattr,atime
# zfs get encryption,keysource rpool/export/home/jill
NAME                   PROPERTY    VALUE              SOURCE
rpool/export/home/jill  encryption  on                 local
rpool/export/home/jill  keysource   passphrase,prompt  local
```

Example 2 Preventing Users From Seeing Error Messages at Login

In this example, the administrator prevents the display of a login error message by using the nowarn option.

```
# pfedit /opt/local_pam/unix_ldap.conf
...
##
## turn off login error message`
sshd-kbdint  auth required          pam_unix_cred.so.1    nowarn
sshd-kbdint  auth required          pam_ldap.so.1    nowarn
```

▼ How to Add a PAM Module

This procedure shows how to add, protect, and test a new PAM module. New modules might be required for site-specific security policies or to support third-party applications. To create a PAM module, see Chapter 3, “Writing PAM Applications and Services” in *Developer’s Guide to Oracle Solaris 11 Security*.

Note - You must install a 32-bit version and a 64-bit version of the PAM service module.

Before You Begin Complete “Planning a Site-Specific PAM Configuration” on page 19.

You must assume the root role. For more information, see “Using Your Assigned Administrative Rights” in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Install then protect both versions of the PAM service module on disk.**
   Ensure that the ownership and permissions protect the module files with `root` ownership and `444` permissions.

```
# cd /opt/pam_modules
# ls -lR
.:
total 4
-r--r--r--   1 root     root        4570 Nov 27 12:34 pam_app1.so.1
drwxrwxrwx   2 root     root           3 Nov 27 12:38 sparcv9

./64:
total 1
-r--r--r--   1 root     root        4862 Nov 27 12:38 pam_app1.so.1
```

   The 32-bit module is in the `/opt/pam_modules` directory and the 64-bit module is in the `64` subdirectory.

2. **Add the module to the appropriate PAM configuration file.**
   In the following example, the module is for a new application, `app1`. Its service name is the same as the application name. Create an `app1` service-name file in the `/etc/pam.d` directory. The first entry in the file enables the `app1` service to be assigned to individual users.

```
# cd /etc/pam.d
# pfedit app1
...
# PAM configuration
#
# app1 service
#
auth definitive         pam_user_policy.so.1
auth required           /opt/pam_modules/$ISA/pam.app1.so.1  debug
```

   The `$ISA` token in the module path directs the PAM framework to the appropriate 32-bit or 64-bit architecture version of the service module for the calling application. For 32-bit applications, */a/b/*$ISA/*module*.so becomes */a/b/module*.so. and for 64-bit applications it becomes */a/b/*64/*module*.so. In this example, you installed the 32-bit `pam.app1.so.1` service module in the `/opt/pam_modules` directory and the 64-bit module in the `/opt/pam_modules/64` directory.
   For more information, see the `pfedit`(1M) and `pam.conf`(4) man pages.
   To limit the `app1` PAM policy to selected users, see Example 3, “Adding a New Module to a Per-User PAM Policy File,” on page 25.

3. **Test your new service.**
   Log in directly by using `login` or `ssh`. Then, run the commands that are affected by the new module. Test users who are allowed and who are denied use of the affected commands. For troubleshooting assistance, see “How to Troubleshoot PAM Configuration Errors” on page 29.

4. **Assign the policy.**

   See "How to Assign a Modified PAM Policy" on page 25.

Example 3 Adding a New Module to a Per-User PAM Policy File

In this example, the `app1` service is not used by all users, so the administrator adds the service as a per-user policy.

```
# cd /etc/pam.d
# cp app1 /opt/local_pam/app1-conf
# pfedit /opt/local_pam/app1-conf

## app1 service
##
app1 auth definitive         pam_user_policy.so.1
app1 auth required           /opt/pam_modules/$ISA/pam_app1.so.1  debug
```

The administrator deletes the `app1` file from the `pam.d` directory.

```
# rm /etc/pam.d/app1
```

Then, the administrator adds the `app1-conf` policy to the system administrator's PAM policy.

```
# rolemod -K pam_policy=/opt/local_pam/app1-conf sysadmin
```

Example 4 Setting Per-User PAM Policy by Using a Rights Profile

This example uses the `pam_policy` security attribute to enable users from different naming services to be authenticated. The `any` PAM policy file is provided in the `/etc/security/pam_policy` directory. The comments in the file describe this policy.

Do not modify files in this directory.

```
# profiles -p "PAM Per-User Policy of Any" \
'set desc="Profile which sets pam_policy=any";
set pam_policy=any; exit;'
```

To assign this rights profile, see "How to Assign a Modified PAM Policy" on page 25.

▼ How to Assign a Modified PAM Policy

In this procedure, you configure a non-default PAM policy on all system images. After all files are copied, you can assign the new or modified PAM policy to individual users or to all users.

Before You Begin You have modified and tested the PAM configuration files that implement the new policy.

You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Add the non-default PAM files to all systems.**

   You must add all new PAM modules and new and modified PAM configuration files to all systems.

   a. **First, add any new PAM modules to every system.**

      i. **Add the 32-bit PAM module to the architecture-appropriate directory.**

      ii. **Add the 64-bit PAM module to the architecture-appropriate directory.**

      For an example of directory setup, see Step 1 in "How to Add a PAM Module" on page 23.

   b. **Next, add any new PAM configuration files to every system.**

      For example, add the `/opt/local_pam/ssh-telnet-conf` file to every system.

   c. **Then, copy any modified PAM configuration files to every system.**

      For example, copy a modified `/etc/pam.conf` file and any modified `/etc/pam.d/`*service-name-files* to every system.

2. **Assign a non-default PAM policy to all users.**

   a. **Modify the `policy.conf` file in one of the following ways:**

      - **Add a PAM configuration file to the `PAM_POLICY` keyword in the `policy.conf` file.**

        ```
        # pfedit /etc/security/policy.conf
        ...
        # PAM_POLICY=
        PAM_POLICY=/opt/local_pam/ssh-telnet-conf
        ...
        ```

      - **Add a rights profile to the `PROFS_GRANTED` keyword in the `policy.conf` file.**

        For example, assign the PAM Per-User Policy of Any rights profile from Example 4, "Setting Per-User PAM Policy by Using a Rights Profile," on page 25.

        ```
        # pfedit /etc/security/policy.conf
        ...
        AUTHS_GRANTED=
        # PROFS_GRANTED=Basic Solaris User
        ```

```
PROFS_GRANTED=PAM Per-User Policy of Any,Basic Solaris User
...
```

b. Copy the modified `policy.conf` file to every system.

3. To assign a non-default PAM policy to individual users, you can assign the policy directly to a user or add the policy to a rights profile that is assigned to users.

- **Assign the PAM policy directly to individual users.**

```
# usermod -K pam_policy="/opt/local_pam/ssh-telnet-conf" jill
```

- **Include the PAM policy in a rights profile and assign the profile to individual users.**

  This example uses the `ldap` PAM policy.

```
# profiles -p "PAM Per-User Policy of LDAP" \
'set desc="Profile which sets pam_policy=ldap";
set pam_policy=ldap; exit;'
```

  Then assign the rights profile to a user.

```
# usermod -P +"PAM Per-User Policy of LDAP" jill
```

Example 5 Limiting the `ktelnet` PAM Stack to Selected Users

The administrator wants to allow a limited number of users the ability to use `telnet` in a Kerberos realm. So, before the `telnet` service is enabled, the administrator changes the default `ktelnet` configuration file, and places the default `ktelnet` file in the `pam_policy` directory.

First, the administrator configures a per-user `ktelnet` file.

```
# cp /etc/pam.d/ktelnet /etc/security/pam_policy/ktelnet-conf
# pfedit /etc/security/pam_policy/ktelnet-conf
...
# Kerberized telnet service
#
ktelnet  auth required          pam_unix_cred.so.1
ktelnet  auth required          pam_krb5.so.1
```

The administrator protects the file with `444` permissions.

```
# chmod 444 /etc/security/pam_policy/ktelnet-conf
# ls -l /etc/security/pam_policy/ktelnet-conf
-r--r--r--   1 root     root         228 Nov 27 15:04 ktelnet-conf
```

Then, the administrator modifies the `ktelnet` file in the `pam.d` directory.

- The first entry enables per-user assignment.

- The second entry denies the use of `ktelnet` unless you are assigned `pam_policy=ktelnet` by the administrator.

```
# cp /etc/pam.d/ktelnet /etc/pam.d/ktelnet.orig
               # pfedit /etc/pam.d/ktelnet
...
# Denied Kerberized telnet service
#
auth definitive         pam_user_policy.so.1
auth required           pam_deny.so.1
```

The administrator tests the configuration with a privileged user, a regular user, and the root role. When the configuration passes, the administrator enables the `telnet` service and assigns the per-user policy to the Kerberos administrators.

```
# svcadm enable telnet
# rolemod -S ldap -K pam_policy=ktelnet-conf kerbadmin
```

The administrator copies the modified files to all Kerberos servers, and enables `telnet` on those servers.

How to Log PAM Error Reports

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. Determine which `system-log` service instance is online.

```
# svcs system-log
STATE          STIME    FMRI
disabled       13:11:55 svc:/system/system-log:rsyslog
online         13:13:27 svc:/system/system-log:default
```

2. Configure the `syslog.conf` file for the level of logging that you need.

See the `DESCRIPTION` section of the `syslog.conf(4)` man page for information about the logging levels. Most PAM error reporting is done through the `LOG_AUTH` facility.

For example, create a file for debug output.

```
# touch /var/adm/pam_debuglog
```

Then, add the `syslog.conf` entry to send debug output to that file.

Note - If the `rsyslog` service instance is online, modify the `rsyslog.conf` file.

```
# pfedit /etc/syslog.conf
```

```
...
*.debug         /var/adm/pam_debuglog
...
```

3. **Refresh the configuration information for the `system-log` service.**

```
# svcadm refresh system-log:default
```

Note - Refresh the `system-log:rsyslog` service instance if the `rsyslog` service is online.

How to Troubleshoot PAM Configuration Errors

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **For each PAM entry that you are troubleshooting, add the `debug` option.**

   For example, the following entries in the `/etc/pam.d/cron` file create debug output for the service.

```
account definitive      pam_user_policy.so.1     debug
account required        pam_unix_account.so.1    debug
```

2. **Log PAM errors at the appropriate level and refresh the `syslog` daemon.**

   For details, see "How to Log PAM Error Reports" on page 28.

3. **If the problem is a corrupt PAM configuration, do the following:**

   a. **Run the application from one terminal window and modify the PAM configuration file in another window.**

   b. **Verify that the errors are corrected by testing the changes in the application window.**

4. **If the problem is a corrupt PAM configuration that prevents login, boot into single-user mode, then correct the file, reboot, and test.**

   - **To boot a SPARC system, type the following command at the PROM prompt:**

```
ok > boot -s
```

   - **To boot an x86 system, add the `-s` option to the kernel options line in the GRUB menu.**

   For more information, see the `boot`(1M) and `grub`(5) man pages.

5. **Verify that the errors are corrected.**
   Log in directly by using `login` or `ssh`. Test that regular users, privileged users, and roles can use the affected commands.

PAM Configuration Reference

This section provides additional details about PAM, including PAM stacking.

PAM Configuration Files

System applications, such as `login` and `ssh`, that use the PAM framework are configured in the PAM configuration files in the `/etc/pam.d` directory. The `/etc/pam.conf` file can also be used. Changes to these files affect all users on the system.

Additionally, the `/etc/security/pam_policy` directory holds PAM configuration files. These files cover multiple services and are designed for per-user assignment. Files in this directory must not be modified.

- `/etc/pam.d` **directory –** Contains service-specific PAM configuration files, including the wildcard file, `other`. To add a service for an application, add a single *service-name* file that is the service name used by the application. If appropriate, your application can use the PAM stack in the `other` file.

  The service files in the `/etc/pam.d` directory provide the default configuration in most PAM implementations. They are self-assembled by using the IPS mechanism as described in the `pkg`(5) man page. This default simplifies interoperability with other cross-platform PAM applications. For more information, see the `pam.conf`(4) man page.
- `/etc/pam.conf` **file –** The legacy PAM configuration and policy file. This file is delivered empty. The preferred mechanism for configuring PAM is to use the files in the `/etc/pam.d` directory. For more information, see the `pam.conf`(4) man page.
- `/etc/security/pam_policy` **directory –** Contains PAM policy files that contain policies for multiple services. These files can be assigned to an individual, to a group of individuals, or to all users, as needed. Such an assignment overrides the system PAM configuration files in `pam.conf` or the `/etc/pam.d` directory. Do not modify these files. To add a per-user file, see "How to Create a Site-Specific PAM Configuration File" on page 21. For information about per-user files, see the `pam_user_policy`(5) man page.

The security administrator manages all PAM configuration files. An incorrect order of entries, that is, an incorrect PAM stack, can cause unforeseen side effects. For example, a badly configured file might lock out users so that single-user mode becomes necessary for repair. For assistance, see "PAM Stacking" on page 32 and "How to Troubleshoot PAM Configuration Errors" on page 29.

PAM Configuration Search Order

Application calls to the PAM framework search for the configured PAM services in the following order:

1. The service name is looked up in the `/etc/pam.conf` file.
2. Specific services are used in the `/etc/pam.d/`*service-name* file.
3. The service name `other` is checked in the `/etc/pam.conf` file.
4. The `/etc/pam.d/other` file is used.

This order enables an existing `/etc/pam.conf` file to work with the per-service PAM configuration files in the `/etc/pam.d` directory.

PAM Configuration File Syntax

The `pam.conf` file and the PAM per-user files use a syntax that is different from the service-specific files in the `pam.d` directory.

- The entries in the `/etc/pam.conf` file and the `/etc/security/pam_policy` files are in one of two formats:

  service-name module-type control-flag module-path module-options

  service-name module-type `include` *path-to-included-PAM-configuration*

- The entries in the *service-name* files in the `/etc/pam.d` directory omit the service name. The name of the file provides the service name.

  module-type control-flag module-path module-options

  module-type `include` *path-to-included-PAM-configuration*

The PAM configuration file syntax items are as follows:

service-name

The case-insensitive name of the service, for example, `login` or `ssh`. An application can use different service names for the services that the application provides. For example, search for the word `PAM` in the `sshd`(1M) man page for the service names for the different services that the `sshd` daemon provides.

The predefined service name "`other`" is the default service name if no specific service configuration is provided.

module-type

Indicates the type of service, that is, `auth`, `account`, `session`, or `password`.

control-flag

Indicates the role of the module in determining the success or failure value for the service. Valid control flags are described in "PAM Stacking" on page 32.

module-path

The path to the module that implements the module type. If the pathname is not absolute, it is assumed to be relative to the path `/usr/lib/security/$ISA/`. The `$ISA` macro or token directs the PAM framework to look in the module path's architecture-specific directory.

module-options

Options such as `nowarn` and `debug` that can be passed to the service modules. A module's man page describes the options for that module.

path-to-included-PAM-configuration

Specifies the full path to a PAM configuration file or a file name that is relative to the `/usr/lib/security` directory.

PAM Stacking

When an application calls one of the following functions, the PAM framework reads the PAM configuration files to determine which modules implement this application's PAM service name:

- `pam_authenticate`(3PAM)
- `pam_acct_mgmt`(3PAM)
- `pam_setcred`(3PAM)
- `pam_open_session`(3PAM)
- `pam_close_session`(3PAM)
- `pam_chauthtok`(3PAM)

If the configuration files contain only one module, the result of that module determines the outcome of the operation. For example, the default authentication operation for the `passwd` application contains one module, `pam_passwd_auth.so.1`, in the `/etc/pam.d/passwd` file.

```
auth required           pam_passwd_auth.so.1
```

If, on the other hand, multiple modules implement a service, those modules are said to be *stacked*, that is, a PAM stack exists for that service name. For example, consider the entries in a sample `/etc/pam.d/login` service:

```
auth definitive         pam_user_policy.so.1
auth requisite          pam_authtok_get.so.1
auth required           pam_unix_auth.so.1
auth required           pam_dhkeys.so.1
auth required           pam_unix_cred.so.1
auth required           pam_dial_auth.so.1
```

These entries create an `auth` stack for the `login` service name. To determine the outcome of this stack, the result codes of the individual modules require an *integration process*.

In the integration process, the modules are executed in their order in the file. Each success or failure code is integrated in to the overall result according to the module's control flag. The control flag can cause early termination of the stack. For example, the failure of a `requisite` or `definitive` module terminates the stack. If there are no previous failures, the success of a `sufficient`, `definitive`, or `binding` module also terminates the stack. After the stack is processed, the individual results are combined into a single, overall result that is delivered to the application. For a graphic view of the flow, see Figure 2, “PAM Stacking: Effect of Control Flags,” on page 35 and Figure 3, “PAM Stacking: How Integrated Value Is Determined,” on page 36.

The control flag indicates the role that a PAM module plays in determining success or failure. The control flags and their effects are:

- **Binding** – Success in meeting a binding module's requirements returns success immediately to the application if no previous failures have been recorded. If these conditions are met, then no further execution of modules occurs.

  Failure causes a required failure to be recorded and the processing of modules to be continued.
- **Definitive** – Success in meeting a definitive module's requirements returns success immediately to the application if no previous failures have been recorded.

  If a previous failure has been recorded, that failure is immediately returned to the application with no further execution of modules. Failure results in an immediate error return with no further execution of modules.
- **Include** – Adds lines from a separate PAM configuration file to be used at this point in the PAM stack. This flag does not control success or failure behaviors. When a new file is read, the PAM `include` stack is incremented. When the stack check in the new file finishes, the `include` stack value is decremented. When the end of a file is reached and the PAM `include` stack is `0`, then the stack processing ends. The maximum number for the PAM `include` stack is `32`.
- **Optional** – Success in meeting an optional module's requirements is not necessary for using the service.

  Failure causes an optional failure to be recorded.
- **Required** – Success in meeting a required module's requirements is necessary for the stack to succeed. Final success for the stack is returned only if no binding or required modules have reported failures.

  Failure results in an error return after the remaining modules for this service have been executed.
- **Requisite** – Success in meeting a requisite module's requirements is necessary for the stack to succeed. All requisite modules in the stack must return success for the stack to be able to return success to the application.

  Failure results in an immediate error return with no further execution of modules.

- **Sufficient** – If no previous required failures have been recorded, success in a sufficient module returns success immediately with no further execution of modules.

  Failure causes an optional failure to be recorded.

The following two connected diagrams show how a result is determined in the integration process.

- The first diagram shows how success or failure is recorded for each type of control flag. The results are shown in the second diagram.
- The second diagram shows how the integrated value is determined. Optional failure and required failure return failure, and success returns success. The application determines how to handle these return codes.

FIGURE 2 PAM Stacking: Effect of Control Flags

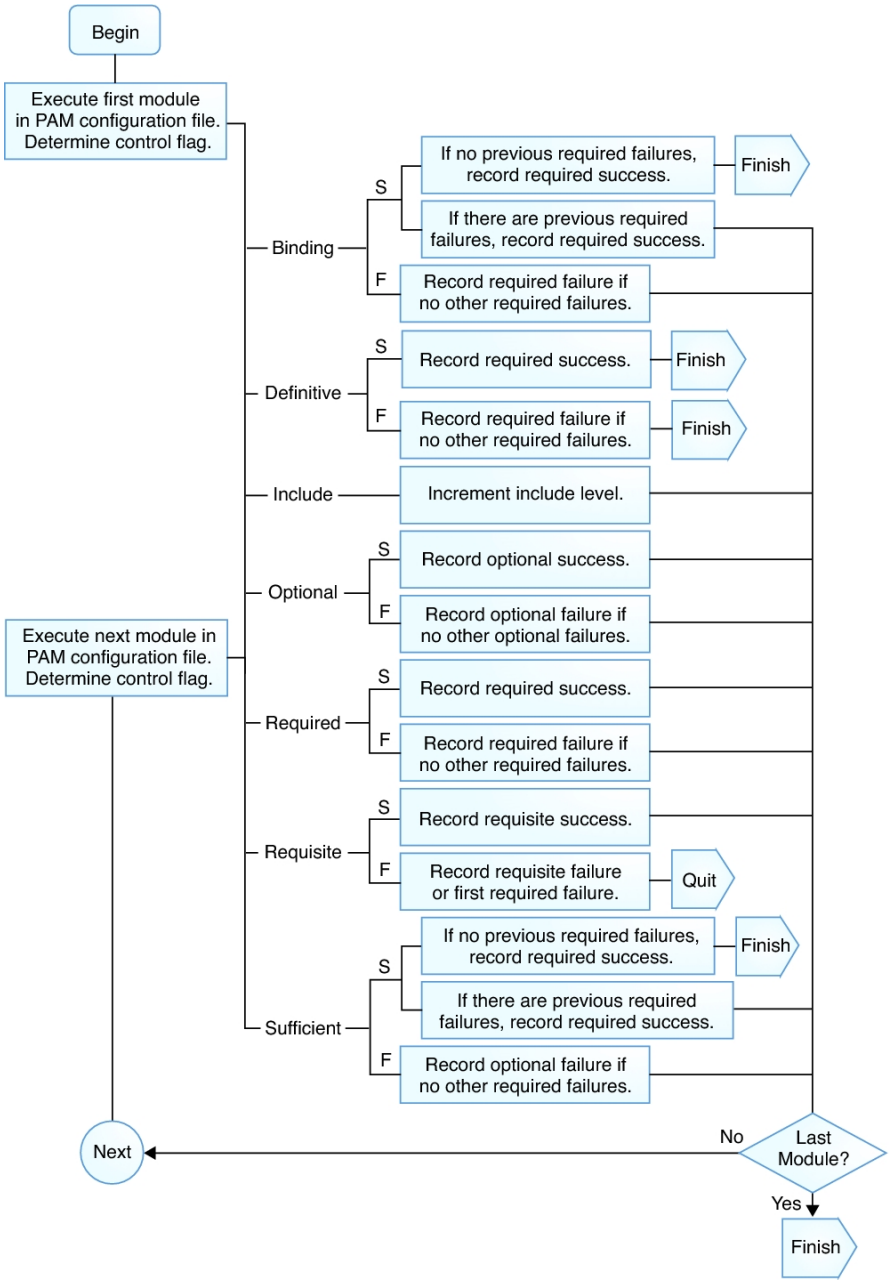
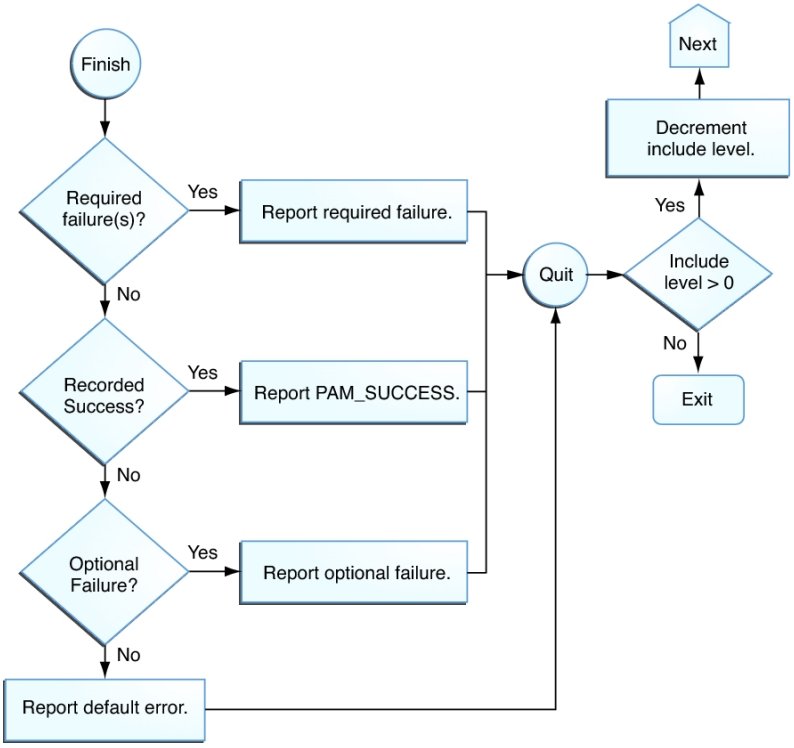


FIGURE 3 PAM Stacking: How Integrated Value Is Determined



PAM Stacking Example

The following example shows the default definitions for authentication management in a sample `/etc/pam.d/other` file. These definitions are used for authentication if no service-specific authentication definitions have been configured.

```
##
# Default definitions for Authentication management
# Used when service name is not explicitly mentioned for authentication
#
auth definitive         pam_user_policy.so.1
auth requisite          pam_authtok_get.so.1
auth required           pam_dhkeys.so.1
auth required           pam_unix_auth.so.1
auth required           pam_unix_cred.so.1
```

First, the PAM policy for the user is checked by using the `pam_user_policy.so` module. The `definitive` control flag dictates that if the evaluation of the configured PAM stack succeeds, success is returned to the application, because no other modules have been checked at this point. If the evaluation of the configured PAM stack fails, then a failure code is returned to the application and no further checking is done. If no per-user PAM policy is assigned to this user, then the next module is executed.

If a per-user PAM policy is not assigned to this user, then the `pam_authtok_get` module is executed. The control flag for this module is set to `requisite`. If `pam_authtok_get` fails, then the authentication process ends and the failure is returned to the application.

If `pam_authtok_get` does not fail, then the next three modules are executed. These modules are configured with the `required` control flag so that the integration process continues regardless of whether an individual failure is returned. After `pam_unix_cred` is executed, no modules remain. At this point, if all the modules succeeded, success is returned to the application. If any of `pam_dhkeys`, `pam_unix_auth`, or `pam_unix_cred` has returned a failure, failure is returned to the application.

PAM Service Modules

This section lists selected PAM service modules. The modules are listed by their man page followed by a brief description of where and when they are used. For more information, read the man page.

For a list of all PAM service modules that Oracle Solaris provides, see section 5 of the man pages. New modules are added on a regular basis. For example, in this release, a number of modules are added for authentication with Windows systems. Your site might also add PAM modules from third parties.

`pam_allow(5)`	Returns `PAM_SUCCESS` for all calls. See also the `pam_deny(5)` man page.
`pam_authtok_check(5)`	Validates the password token for password change.
`pam_authtok_get(5)`	Provides password prompting functionality to the PAM stack.
`pam_authtok_store(5)`	Updates the password token for `PAM_USER`.
`pam_deny(5)`	Returns the module type default failure return code for all calls. See also the `pam_allow(5)` man page.
`pam_dhkeys(5)`	Provides functionality to two PAM services: Secure RPC authentication and Secure RPC authentication token management.
`pam_krb5(5)`	Provides functions to verify the identity of a Kerberos user and to manage the Kerberos credentials cache.

pam_krb5_migrate(5)	Helps to migrate PAM_USER to the client's local Kerberos realm.
pam_ldap(5)	Provides functionality for the PAM authentication and account management stacks by the configured LDAP directory server.
pam_list(5)	Provides functions to validate the user's account on this host. The validation is based on a list of users and netgroups on the host.
pam_passwd_auth(5)	Provides authentication functionality to the password stack.
pam_pkcs11(5)	Enables a user to log in to a system by using an X.509 certificate and its dedicated private key that is stored in a PKCS#11 token.
pam_roles(5)	Verifies that a user is authorized to assume a role and prevents direct login by a role.
pam_smb_passwd(5)	Supports the changing or adding of SMB passwords for local Oracle Solaris users. See also the smb(4) man page.
pam_smbfs_login(5)	Synchronizes passwords between Oracle Solaris clients and their CIFS/SMB servers.
pam_tsol_account(5)	Verifies Trusted Extensions account limitations that are related to labels.
pam_tty_tickets(5)	Provides a mechanism for checking a ticket that was created by a prior successful authentication.
pam_unix_account(5)	Provides functions to validate that the user's account is not locked or expired and that the user's password does not need to be changed. Includes checks of access_times and access_tz.
pam_unix_auth(5)	Provides functions to verify that the password is the correct password for PAM_USER.
pam_unix_cred(5)	Provides functions that establish user credential information. It enables the authentication functionality to be replaced independently from the credential functionality.
pam_unix_session(5)	Opens and closes a session, and also updates the /var/adm/lastlog file.
pam_user_policy(5)	Calls a user-specific PAM configuration.
pam_zfs_key(5)	Provides functions to load and change the ZFS encryption passphrase for a user's encrypted home directory.

♦ ♦ ♦ CHAPTER 2

About the Kerberos Service

This chapter introduces the Kerberos service. This chapter contains the following information:

What Is the Kerberos Service?

The *Kerberos service* is a client-server architecture that provides secure transactions over networks. The service offers strong user authentication, as well as integrity and privacy. *Authentication* guarantees that the identities of both the sender and the recipient of a network transaction are true. The service can also verify the validity of data being passed back and forth (*integrity*) and encrypt the data during transmission (*privacy*). Using the Kerberos service, you can log in to other systems, execute commands, exchange data, and transfer files securely. Additionally, the service provides *authorization* services, which enables administrators to restrict access to services and systems. Moreover, as a Kerberos user, you can regulate other people's access to your account.

The Kerberos service is a *single sign-on* system, which means that you only need to authenticate yourself to the service once per session. All subsequent transactions during the session are automatically secured. After the service has authenticated you, you do not need to authenticate yourself every time you use a Kerberos-based command such as `ftp` or `ssh`, or a command to access data on an NFS file system. Thus, you do not have to send your password over the network, where it can be intercepted, each time you use these services.

The Kerberos service in Oracle Solaris is based on the Kerberos V5 network authentication protocol that was developed at the Massachusetts Institute of Technology (MIT). If you have used the Kerberos V5 product, you will therefore find the Oracle Solaris version very familiar. Because the Kerberos V5 protocol is a *de facto* industry standard for network security, the

Oracle Solaris version enables secure transactions over heterogeneous networks. Moreover, the service provides authentication and security both between domains and within a single domain.

The Kerberos service provides flexibility in running Oracle Solaris applications. You can configure the service to enable both Kerberos-based and non-Kerberos-based requests for network services such as the NFS service and `ftp`. As a result, current applications still work even if they are running on systems on which the Kerberos service is not enabled. Of course, you can also configure the Kerberos service to enable only Kerberos-based network requests.

The Kerberos service security mechanism allows the use of Kerberos for authentication, integrity, and privacy when using applications that use the Generic Security Service Application Programming Interface (GSS-API). However, applications do not have to remain committed to the Kerberos service if other security mechanisms are developed. Because the service is designed to integrate modularly into the GSS-API, applications that use the GSS-API can choose the security mechanism that best suits their needs.

How the Kerberos Service Works

This section provides an overview of the Kerberos authentication process. For a more detailed description, see “How Kerberos Credentials Provide Access to Services” on page 52.

From the user's standpoint, the Kerberos service is mostly invisible after the Kerberos session has been started. Commands such as `ssh` or `ftp` work about the same. Initializing a Kerberos session often involves no more than logging in and providing a Kerberos password.

The Kerberos system revolves around the concept of a *ticket*. A ticket is a set of electronic information that identifies a user or a service such as the NFS service. Just as your driver's license identifies you and indicates what driving privileges you have, so a ticket identifies you and your network access privileges. When you perform a Kerberos-based transaction (for example, if you request an NFS-mounted file), you transparently send a request for a ticket to a *Key Distribution Center*, or KDC. The KDC accesses a database to authenticate your identity and returns a ticket that grants you permission to access the NFS server. “Transparently” means that you do not need to explicitly request a ticket. The request happens when you attempt to access the server. Because only authenticated clients can get a ticket for a specific service, another client cannot access the NFS server under an assumed identity.

Tickets have certain attributes associated with them. For example, a ticket can be *forwardable*, which means that it can be used on another system without a new authentication process. A ticket can also be *postdated*, which means that it is not valid until a specified time. How tickets can be used is set by *policies*, for example, to specify which users are allowed to obtain which types of ticket. Policies are determined when the Kerberos service is installed or administered.

Note - You will frequently see the terms *credential* and *ticket*. In the greater Kerberos world, they are often used interchangeably. Technically, however, a credential is a ticket plus the *session key* for that session. This difference is explained in more detail in "How Kerberos Credentials Provide Access to Services" on page 52.

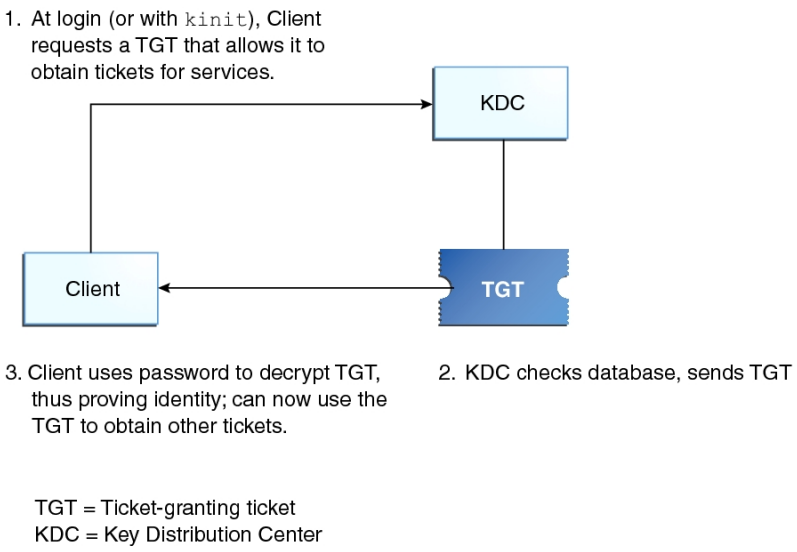
The following sections further explain the Kerberos authentication process.

Initial Authentication: the Ticket-Granting Ticket

Kerberos authentication has two phases: an initial authentication that enables all subsequent authentications, and the subsequent authentications themselves.

The following figure shows how the initial authentication takes place.

FIGURE 4 Initial Authentication for a Kerberos Session



1. A client (a user, or a service such as NFS) begins a Kerberos session by requesting a *ticket-granting ticket* (TGT) from the Key Distribution Center (KDC). This request is often done automatically at login.

   A ticket-granting ticket is needed to obtain other tickets for specific services. Think of the ticket-granting ticket as similar to a passport. Like a passport, the ticket-granting

ticket identifies you and allows you to obtain numerous “visas” (tickets), which instead of granting access to foreign countries enable you to access remote systems or network services. Like passports and visas, the ticket-granting ticket and the other various tickets have limited lifetimes. The difference is that “Kerberized” commands notice that you have a passport and obtain the visas for you. You don't have to perform the transactions yourself.

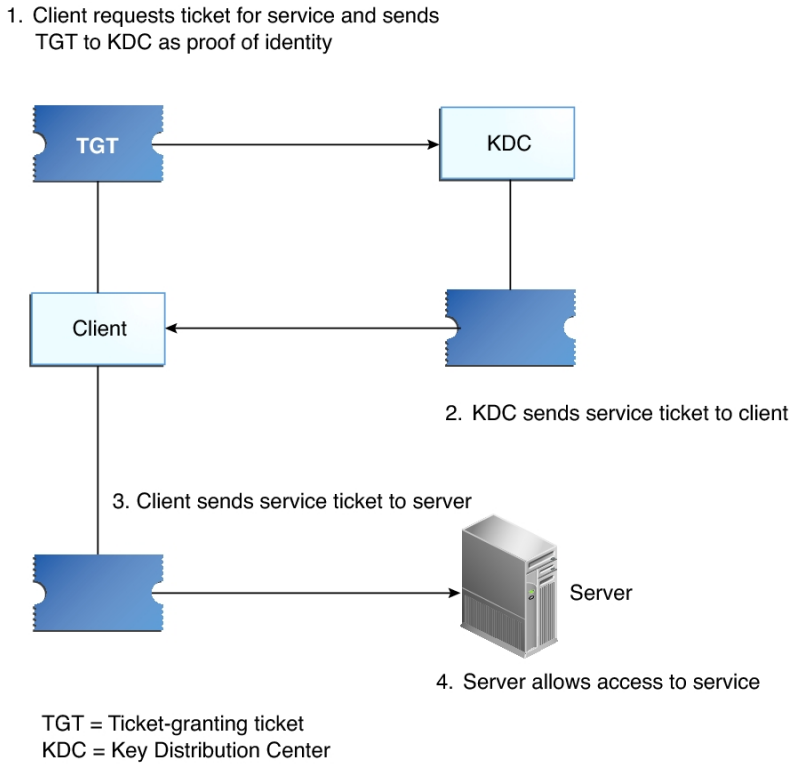
Another analogy for the ticket-granting ticket is that of a three-day ski pass that is good at four different ski resorts. You show the pass at whichever resort you decide to go to and you receive a lift ticket for that resort as long as the pass has not expired. Once you have the lift ticket, you can ski all you want at that resort. If you go to another resort the next day, you once again show your pass and you get an additional lift ticket for the new resort. The difference is that the Kerberos-based commands notice that you have the weekend ski pass and get the lift ticket for you so you don't have to perform the transactions yourself.

2. The KDC creates a ticket-granting ticket and sends it back, in encrypted form, to the client. The client decrypts the ticket-granting ticket by using the client's password.
3. Now in possession of a valid ticket-granting ticket, the client can request tickets for all sorts of network operations, such as `nfs` or `ssh`, for as long as the ticket-granting ticket lasts. This ticket usually lasts for a few hours. Each time the client performs a unique network operation, it requests a ticket for that operation from the KDC.

Subsequent Kerberos Authentications

After the client has received the initial authentication, each subsequent authentication follows the pattern that is shown in the following figure.

FIGURE 5 Obtaining Access to a Service Using Kerberos Authentication



1. The client requests a ticket for a particular service, for example, to log in remotely to another system, from the KDC by sending the KDC its ticket-granting ticket as proof of identity.
2. The KDC sends the ticket for the specific service to the client.

   Suppose user `jdoe` wants to access an NFS file system that has been shared with `krb5` authentication required. Because `jdoe` is already authenticated (that is, `jdoe` already has a ticket-granting ticket), as `jdoe` attempts to access the files, the NFS client system automatically and transparently obtains a ticket from the KDC for the NFS service. To use a different Kerberized service, `jdoe` obtains another ticket, as in Step 1.
3. The client sends the ticket to the server.

   When using the NFS service, the NFS client automatically and transparently sends the ticket for the NFS service to the NFS server.
4. The server allows the client access.

Although these steps imply that the server never communicates with the KDC, the server does register itself with the KDC, just as the first client does. For simplicity's sake, that section has been omitted.

Kerberos Authentication of Batch Jobs

Batch jobs, such as cron, at, and batch, are delayed execution processes. In a Kerberos environment, all processes including delayed execution processes require credentials. However, users' credentials are relatively short-lived. By default, user credentials are valid for 8 hours and renewable for as long as a week. These times are designed to limit the exposure of sensitive keys to malicious users, but can prevent the execution of jobs at arbitrary times.

In Oracle Solaris, batch jobs that access Kerberos services can run without exposing the user's longterm key. The solution involves storing credentials that include the Kerberos service, the user name, and the client host name in a per-session user credential cache. A PAM module is used to authenticate the batch job. Which services a host can obtain tickets for can be centrally stored in the LDAP directory server.

For more information, see the `pam_krb5_keytab`(5) and `pam_gss_s4u`(5) man pages and “Configuring Delayed Execution for Access to Kerberos Services” on page 134.

Kerberos, DNS, and the Naming Service

The Kerberos service is compiled to use DNS to resolve host names. The `nsswitch` service is not checked at all to resolve host names.

Kerberos Components

Kerberos includes realms, network programs, principals, servers, and other components. For a list of commands and modules, see “Kerberos Utilities” on page 48.

Kerberos Network Programs

Note - In this Oracle Solaris release, all remote login commands except `ssh` are deprecated. If you want to use one of the following deprecated commands to reach an older system, you can do so. If you want to use a deprecated command on this system because the command is used in a legacy script, you must enable the SMF service for that command, as in `svcadm enable login:rlogin`. Alternatively, you can alter the scripts to use the `ssh` command. Similarly, for an older system to reach this system with a deprecated command, the service for the command must be enabled on this system.

Deprecated commands such as `telnet` can require weak encryption keys. These keys are not allowed by default in the `krb5.conf` file. For more information, see "Supported Encryption Types in Kerberos" on page 60 and the `krb5.conf(4)` man page.

For more information, see "Controlling Access to System Resources" in *Securing Systems and Attached Devices in Oracle Solaris 11.3*. See also the man pages that are listed in "Kerberos User Commands" on page 181.

The Kerberos-based ("Kerberized") commands available to users are the following:

- `ftp`
- `rcp`, `rlogin`, `rsh`
- `ssh`, `scp`, `sftp`
- `telnet`

These applications are the same as the Oracle Solaris application of the same name. However, they have been extended to use Kerberos principals to authenticate transactions, thereby providing Kerberos-based security. For information about principals, see "Kerberos Principals" on page 45.

These commands are discussed further in "Kerberos User Commands" on page 181.

Kerberos Principals

A client in the Kerberos service is identified by its *principal*. A principal is a unique identity to which the KDC can assign tickets. A principal can be a user, such as `jdoe`, or a service, such as `nfs`.

By convention, a principal name is divided into three components: the *primary*, the *instance*, and the *realm*. A typical Kerberos principal would be, for example, `jdoe/admin@CORP.EXAMPLE.COM`. In this example:

- `jdoe` is the primary. The primary can be a user name, as shown here, or a service, such as `nfs`. The primary can also be the word `host`, which signifies that this principal is a service principal that is set up to provide various network services, `ftp`, `scp`, `ssh`, and so on.
- `admin` is the instance. An instance is optional in the case of user principals but is required for service principals. For example, if the user `jdoe` sometimes acts as a system administrator, the principal `jdoe/admin` can distinguish the administrator from the user identity. Likewise, if `jdoe` has accounts on two different hosts, the accounts can use two principal names with different instances, for example, `jdoe/denver.example.com` and `jdoe/boston.example.com`. Notice that the Kerberos service treats `jdoe` and `jdoe/admin` as two completely different principals.

  In the case of a service principal, the instance is the fully qualified host name. `bighop.corp.example.com` is an example of such an instance. The primary/instance for this example might be `ftp/bighop.corp.example.com` or `host/bighop.corp.example.com`.
- `CORP.EXAMPLE.COM` is the Kerberos realm. Realms are discussed in "Kerberos Realms" on page 46.

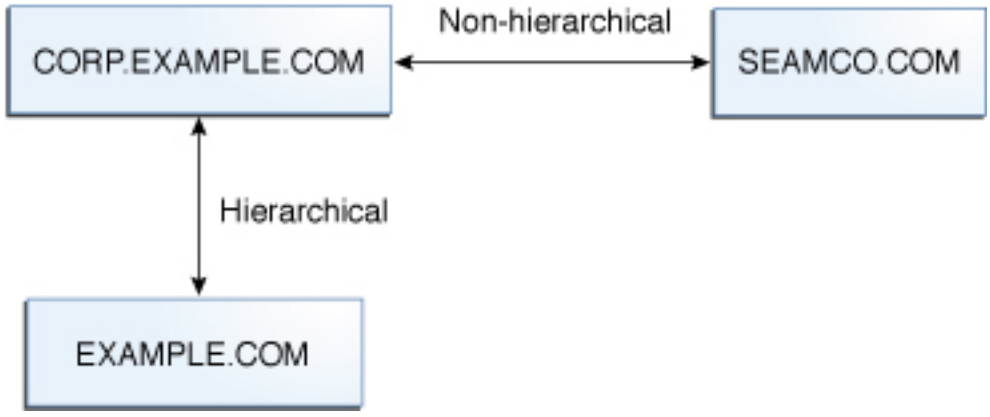
The following are all valid principal names:

- `jdoe`
- `jdoe/admin`
- `jdoe/admin@CORP.EXAMPLE.COM`
- `nfs/host.corp.example.com@CORP.EXAMPLE.COM`
- `host/corp.example.com@CORP.EXAMPLE.COM`

Kerberos Realms

A *realm* is a logical network, similar to a domain, that defines a group of systems under the same *master KDC*. Figure 6, "Kerberos Realms," on page 47 shows how realms can relate to one another. Some realms are hierarchical, where one realm is a superset of the other realm. Otherwise, the realms are nonhierarchical (or "direct") and the mapping between the two realms must be defined. Kerberos *cross-realm authentication* enables authentication across realms. Each realm only needs to have a principal entry for the other realm in its KDC.

FIGURE 6 Kerberos Realms



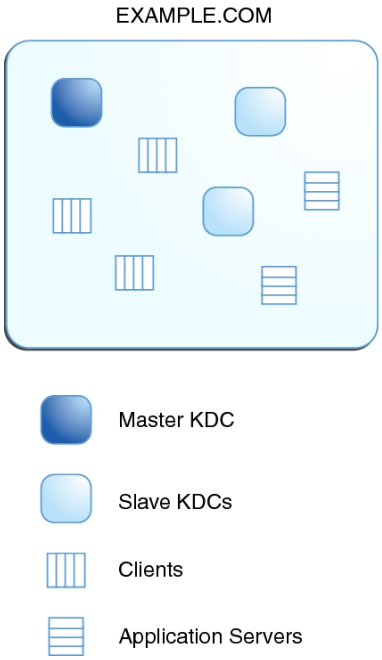
Kerberos Servers

Each realm must include a server that maintains the master copy of the principal database. This server is called the *master KDC server*. Additionally, each realm should contain secondary servers. A secondary server can be a *slave KDC server*, which contains a duplicate copy of the principal database which is refreshed or *propagated* from the master KDC. A secondary server can also be another master KDC to form a *multi-master* configuration. Both a master KDC server and a slave KDC server create tickets that are used to establish authentication.

The realm can also include a Kerberos *application server*. This server provides access to Kerberized services such as `ftp`, `ssh`, and NFS.

The following figure shows what a hypothetical realm might contain.

FIGURE 7 A Typical Kerberos Realm



Kerberos Utilities

Similar to the MIT distribution of the Kerberos V5 product, the Kerberos service in the Oracle Solaris release includes the following:

- Key Distribution Center (KDC):
  - Kerberos database administration daemon – `kadmind`.
  - Kerberos ticket processing daemon – `krb5kdc`.
  - Database administration programs – `kadmin` (master only), `kadmin.local` and `kdb5_util`.
  - Database propagation software – `kprop` (slave only) and `kpropd`.
- User programs for managing credentials – `kinit`, `klist`, and `kdestroy`.
- User program for changing your Kerberos password – `kpasswd`.
- Network applications – `ftp`, `rcp`, `rlogin`, `rsh`, `scp`, `sftp`, `ssh`, and `telnet`.
- Remote application daemons – `ftpd`, `rlogind`, `rshd`, `sshd`, and `telnetd`.

- Keytab administration utility – `ktutil`.
- Generic Security Service Application Programming Interface (GSS-API) – Enables applications to use multiple security mechanisms without requiring you to recompile the application every time a new mechanism is added. The GSS-API uses standard interfaces that enable applications to be portable to many operating systems. GSS-API provides applications with the ability to include the integrity and privacy security services as well as authentication. Both `ftp` and `ssh` use the GSS-API.
- RPCSEC_GSS Application Programming Interface (API) – Enables NFS services to use Kerberos authentication. The RPCSEC_GSS API provides security services that are independent of the mechanisms being used. RPCSEC_GSS sits on top of the GSS-API layer. Any pluggable GSS_API-based security mechanism can be used by applications that use RPCSEC_GSS.

In addition, the Kerberos service in Oracle Solaris includes the following:

- Kerberos V5 service modules for PAM – Provides authentication, account management, session management and password management for the Kerberos service. The modules make Kerberos authentication transparent to the user.
- Kerberos V5 per-user PAM stacks – Provides PAM configuration files for different scenarios in the `/etc/security/pam_policy` directory.
- Kernel modules – Provides kernel-based implementations of the Kerberos service for use by the NFS service, which greatly improves performance.
- Kerberos Administration GUI (`gkadmin`) – Enables you to administer the principals and principal policies in a Java™ technology-based GUI as an alternative to the `kadmin` command.

For more information, see Chapter 7, “Kerberos Service Reference”.

Kerberos Security Services

In addition to providing secure authentication of users, the Kerberos service provides two security services:

- **Integrity –** Just as authentication ensures that clients on a network are who they claim to be, integrity ensures that the data they send is valid and has not been tampered with during transit. Integrity is done through cryptographic checksumming of the data. Integrity also includes user authentication.
- **Privacy –** Privacy takes security a step further. Privacy not only includes verifying the integrity of transmitted data, but it encrypts the data before transmission, protecting it from eavesdroppers. Privacy authenticates users, as well.

Kerberos Encryption Types

Encryption types identify which cryptographic algorithms and mode to use when cryptographic operations are performed. For a list of supported encryption types, see the krb5.conf(4) and kdb5_util(1M) man pages.

When a client requests a ticket from the KDC, the KDC must use keys whose encryption type is compatible with both the client and the server. The Kerberos protocol allows the client to request that the KDC use particular encryption types for the client's part of the ticket reply. The protocol does not allow the server to specify encryption types to the KDC.

Consider the following issues before you change the encryption types:

- The KDC assumes that the first key/encryption-type associated with the server principal entry in the principal database is supported by the server.
- On the KDC, you must ensure that the keys that are generated for the principal are compatible with the systems that will authenticate the principal. By default, the `kadmin` command creates keys for all supported encryption types. If the systems that the principal is used on do not support this default set of encryption types, then you should restrict the encryption types when creating a principal. The two recommended methods of restricting the encryption types are by using of the `-e` flag in `kadmin addprinc` or by setting the `supported_enctypes` parameter in the `kdc.conf` file to this subset. Use the `supported_enctypes` parameter when most of the systems in a Kerberos realm support a subset of the default set of encryption types. Setting `supported_enctypes` specifies the default set of encryption types `kadmin addprinc` uses when it creates a principal for a particular realm.
- When determining the encryption types that a system supports, consider both the version of Kerberos that is running on the system as well as the cryptographic algorithms that are supported by the server application for which a server principal is being created. For example, when creating an `nfs/hostname` service principal, you should restrict the encryption types to the types that the NFS server on that host supports.
- The `master_key_enctype` parameter in the `kdc.conf` file can be used to control the encryption type of the master key that encrypts the entries in the principal database. Do not use this parameter if the KDC principal database has already been created. The `master_key_enctype` parameter can be used at database creation time to change the default master key encryption type from `aes256-cts-hmac-sha1-96` to a different encryption type. Make sure that all slave KDCs support the chosen encryption type and that they have an identical `master_key_enctype` entry in their `kdc.conf` file when configuring the slave KDCs. Also, make sure that the `master_key_enctype` is set to one of the encryption types in `supported_enctypes`, if `supported_enctypes` is set in `kdc.conf`. If either of these issues are not handled properly, then the master KDC might not be able to work with the slave KDCs.
- On the client, you can control its encryption type requests from the KDC through parameters in the `krb5.conf` file. The `default_tkt_enctypes` parameter specifies the

encryption types that the client is willing to use when the client requests a ticket-granting ticket (TGT) from the KDC. The TGT is used by the client to acquire other server tickets in a more efficient manner. The effect of setting `default_tkt_enctypes` is to give the client some control over the encryption types used to protect the communication between the client and KDC when the client requests a server ticket using the TGT (this is called a TGS request). Note that the encryption types that are specified in `default_tkt_enctypes` must match at least one of the principal key encryption types in the principal database stored on the KDC. Otherwise, the TGT request will fail. In most situations, you should not set `default_tkt_enctypes` because this parameter can be a source of interoperability problems. By default, the client code requests that all supported encryption types and the KDC choose the encryption types based on the keys that the KDC finds in the principal database.

- The `default_tgs_enctypes` parameter restricts the encryption types that the client requests in its TGS requests, which are used to acquire server tickets. This parameter also restricts the encryption types the KDC uses when creating the session key that the client and server share. For example, if a client wants to only use 3DES encryption when doing secure NFS, you should set `default_tgs_enctypes` = `des3-cbc-sha1`. Make sure that the client and server principals have a `des-3-cbc-sha1` key in the principal database. As with `default_tkt_enctype`, you should in most cases not set this parameter because it can cause interoperability problems if the credentials are not set up properly both on the KDC and the server.
- On the server, you can control the encryption types that it accepts with the `permitted_enctypes` parameter in the `kdc.conf` file. In addition, you can specify the encryption types that it uses when creating keytab entries. Try to avoid using either of these methods to control encryption types and instead let the KDC determine the encryption types to use because the KDC does not communicate with the server application to determine which key or encryption type to use.

FIPS 140-2 Algorithms and Kerberos Encryption Types

You can configure Kerberos to run in FIPS 140-2 mode in Oracle Solaris. If your realm contains legacy applications or systems that are not FIPS 140-2 compliant, then the realm cannot run in FIPS 140-2 mode.

When running in FIPS 140-2 mode, Kerberos is said to be a *consumer* of the FIPS 140-2 *provider* The provider in Oracle Solaris is the Cryptographic Framework. The only Kerberos encryption type that is FIPS 140-2 validated for the Cryptographic Framework is `des3-cbc-sha1`. It is not the default. For instructions, see "How to Configure Kerberos to Run in FIPS 140-2 Mode" on page 71.

Note - If you have a strict requirement to use only FIPS 140-2 validated cryptography, you must be running the Oracle Solaris 11.3 SRU 5.6 release. Oracle completed a FIPS 140-2 validation against the Cryptographic Framework in this specific release. The current Oracle Solaris release builds on this validated foundation and includes software improvements that address performance, functionality, and reliability. Whenever possible, you should configure Oracle Solaris 11.3 in FIPS 140-2 mode to take advantage of these improvements.

How Kerberos Credentials Provide Access to Services

Remote services allow you access if you can provide a ticket that proves your identity, and a matching session key. The session key contains information that is specific to the user and the service that is being accessed. A ticket and session key are created by the KDC for all users when they first log in. The ticket and the matching session key form a credential. While using multiple networking services, a user can gather many credentials. The user needs to have a credential for each service that runs on a particular server. For example, access to the `nfs` service on a server named `boston` requires one credential. Access to the `nfs` service on another server requires a separate credential.

To access a specific service on a specific server, the user must obtain two credentials. The first credential is for the ticket-granting ticket (TGT). Once the ticket-granting service has decrypted this credential, the service creates a second credential for the server that the user is requesting access to. This second credential can then be used to request access to the service on the server. After the server has successfully decrypted the second credential, then the user is given access.

The process of creating and storing the credentials is transparent. Credentials are created by the KDC that sends the credential to the requester. When received, the credential is stored in a credential cache.

Obtaining a Credential for the Ticket-Granting Service

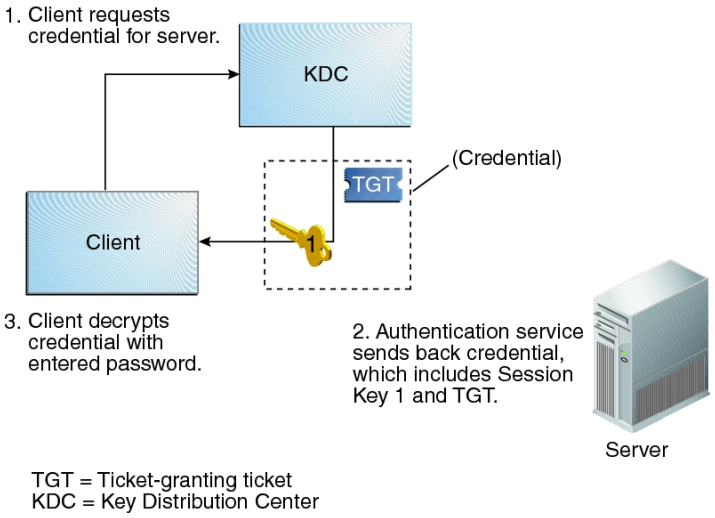
1. To start the authentication process, the client sends a request to the authentication server for a specific user principal. This request is sent without encryption. No secure information is included in the request, so using encryption is not necessary.
2. When the request is received by the authentication service, the principal name of the user is looked up in the KDC database. If a principal matches the entry in the database, the authentication service obtains the private key for that principal. The authentication service then generates a session key to be used by the client and the ticket-granting service (call it Session Key 1) and a ticket for the ticket-granting service (Ticket 1). This ticket is also

known as the *ticket-granting ticket* (TGT). Both the session key and the ticket are encrypted by using the user's private key, and the information is sent back to the client.

3. The client uses this information to decrypt Session Key 1 and Ticket 1 by using the private key for the user principal. Because the private key should be known only by the user and the KDC database, the information in the packet should be safe. The client stores the information in the credentials cache.

During this process, a user is normally prompted for a password. If the password the user specifies is the same as the password that was used to build the private key stored in the KDC database, then the client can successfully decrypt the information that is sent by the authentication service. The client then has a credential to be used with the ticket-granting service and is ready to request a credential for a server.

FIGURE 8 Obtaining a Credential for the Ticket-Granting Service

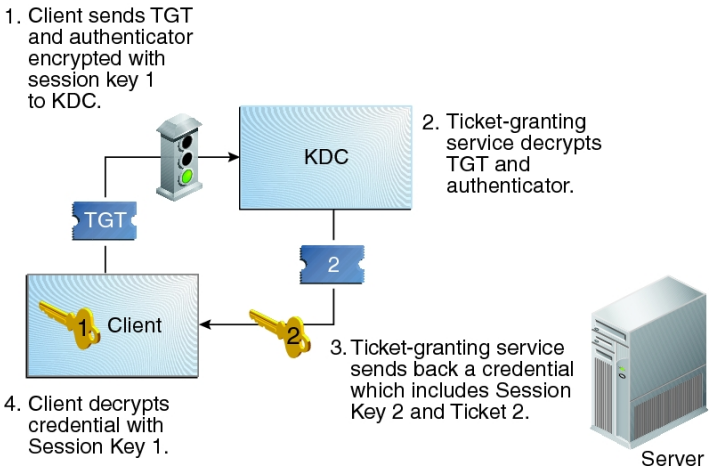


Obtaining a Credential for a Kerberized Server

1. To request access to a specific server, a client must first have obtained a credential for that server from the authentication service. See "Obtaining a Credential for the Ticket-Granting Service" on page 52. The client then sends a request to the ticket-granting service, which includes the service principal name, Ticket 1, and an authenticator that was encrypted with Session Key 1. Ticket 1 was originally encrypted by the authentication service by using the service key of the ticket-granting service.

2. Because the service key of the ticket-granting service is known to the ticket-granting service, Ticket 1 can be decrypted. The information in Ticket 1 includes Session Key 1, so the ticket-granting service can decrypt the authenticator. At this point, the user principal is authenticated with the ticket-granting service.
3. Once the authentication is successful, the ticket-granting service generates a session key for the user principal and the server (Session Key 2) and a ticket for the server (Ticket 2). Session Key 2 and Ticket 2 are then encrypted by using Session Key 1. Because Session Key 1 is known only to the client and the ticket-granting service, this information is secure and can be safely sent over the network.
4. When the client receives this information packet, the client decrypts the information by using Session Key 1, which it had stored in the credential cache. The client has obtained a credential to be used with the server. Now the client is ready to request access to a particular service on that server.

FIGURE 9 Obtaining a Credential for a Server



TGT = Ticket-granting ticket
KDC = Key Distribution Center

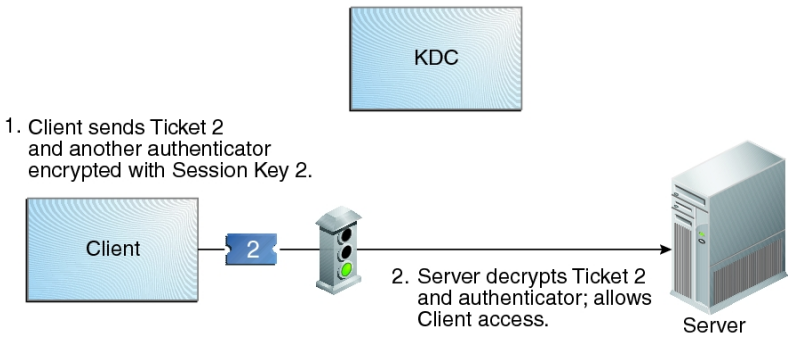
Obtaining Access to a Specific Kerberos Service

1. To request access to a specific service, the client must first have obtained a credential for the ticket-granting service from the authentication server, and a server credential

from the ticket-granting service. See "Obtaining a Credential for the Ticket-Granting Service" on page 52 and "Obtaining a Credential for a Kerberized Server" on page 53. The client can then send a request to the server including Ticket 2 and another authenticator. The authenticator is encrypted by using Session Key 2.

2. Ticket 2 was encrypted by the ticket-granting service with the service key for the service. Because the service key is known by the service principal, the service can decrypt Ticket 2 and get Session Key 2. Session Key 2 can then be used to decrypt the authenticator. If the authenticator is successfully decrypted, the client is given access to the service.

FIGURE 10 Obtaining Access to a Specific Service



Notable Differences Between Oracle Solaris Kerberos and MIT Kerberos

The enhancements that are included in Oracle Solaris but are not included in MIT Kerberos are:

- Kerberos support of Oracle Solaris remote applications
- Incremental propagation of the KDC database
- Client configuration script
- Localized error messages
- Oracle Solaris audit record support
- Thread-safe use of Kerberos by using GSS-API
- Use of the Cryptographic Framework for cryptography

♦ ♦ ♦ CHAPTER 3

Planning for the Kerberos Service

This chapter covers several installation and configuration options that administrators must resolve before they configure or deploy the Kerberos service:

Planning a Kerberos Deployment

Before you install the Kerberos service, you must resolve several configuration issues. Although you can change the configuration after the initial install, some changes can be difficult to implement. In addition, some changes require that the KDC be rebuilt, so consider long-term goals before you deploy Kerberos.

Deploying a Kerberos infrastructure involves such tasks as installing KDCs, creating keys for your hosts, and migrating users. Reconfiguring a Kerberos deployment can be as hard as performing an initial deployment, so plan a deployment carefully to avoid having to reconfigure.

Planning Kerberos Realms

A *realm* is logical network, similar to a domain, that defines a group of systems that are under the same master KDC. As when establishing a DNS domain name, you must resolve issues such as the realm name, the number and size of each realm, and the relationship of a realm to other realms for cross-realm authentication before you configure the Kerberos service.

Kerberos Realm Names

Realm names can consist of any ASCII string. Usually, the realm name is the same as your DNS domain name except that the realm name is in uppercase. This convention helps differentiate problems with the Kerberos service from problems with the DNS namespace, while keeping a name that is familiar. You can use any string, but configuration and maintenance might then require more work. Use realm names that follow the standard Internet naming structure.

Number of Kerberos Realms

The number of realms that your installation requires depends on several factors:

- The number of clients to be supported. Too many clients in one realm makes administration more difficult and eventually requires that you split the realm. The primary factors that determine the number of clients that can be supported are as follows:
  - The amount of Kerberos traffic that each client generates
  - The bandwidth of the physical network
  - The speed of the hosts

  Because each installation will have different limitations, no rule exists for determining the maximum number of clients.
- How far apart the clients are. Setting up several small realms might make sense if the clients are in different geographic regions.
- The number of hosts that are available to be installed as KDCs. Plan to create at least two KDC servers per realm, one master server and at least one slave server.

Alignment of Kerberos realms with administrative domains is recommended. Note that a Kerberos V realm can span multiple subdomains of the DNS domain to which the realm corresponds.

Kerberos Realm Hierarchy

When you are configuring multiple realms for cross-realm authentication, you need to decide how to tie the realms together. You can establish a hierarchical relationship among the realms, which provides automatic paths to the related domains. When all realms in the hierarchical chain are configured properly, these automatic paths can ease the administration burden. However, if there are many levels of domains, you might not want to use the automatic path because it requires too many transactions.

You can also choose to establish the trust relationship directly. A direct trust relationship is useful when too many levels exist between two hierarchical realms or when no hierarchal relationship exists. The connection must be defined in the `/etc/krb5/krb5.conf` file on all hosts that use the connection, so some additional work is required. The direct trust relationship is also referred to as a transitive relationship. For an illustration, see "Kerberos Realms" on page 46. For the configuration procedures, see "Configuring Cross-Realm Authentication" on page 136.

Mapping Host Names to Kerberos Realms

The mapping of host names to realm names is defined in the `domain_realm` section of the `krb5.conf` file. These mappings can be defined for a whole domain and for individual hosts, depending on the requirements.

You can also use DNS to look up information about the KDCs. Using DNS makes changing the information easier because you do not need to edit the `krb5.conf` file on all Kerberos clients for each change.

Note - The procedures in this guide assume the use of DNS.

For more information, see the `krb5.conf`(4) man page.

Kerberos clients in Oracle Solaris interoperate well with Active Directory servers. The Active Directory servers can be configured to provide the realm to host mapping.

Kerberos Client and Service Principal Names

Kerberos in Oracle Solaris does not use the `name-service/switch` service. Rather, the Kerberos service uses DNS to resolve host names. Therefore, DNS must be enabled on all hosts. With DNS, the principal must contain the fully qualified domain name (FQDN) of each host. For example, if the host name is `boston`, the DNS domain name is `example.com`, and the realm name is `EXAMPLE.COM`, then the principal name for the host would be `host/boston.example.com@EXAMPLE.COM`. The examples in this guide require that DNS is configured and that the FQDN is used for each host.

The Kerberos service canonicalizes host alias names through DNS, and uses the canonicalized form (cname) when constructing the service principal for the associated service. Therefore, when creating a service principal, the host name component of service principal names is the canonical form of the host name of the system that provides the service.

The following example shows how the Kerberos service canonicalizes host names. If a user runs the command `ssh alpha.example.com` where `alpha.example.com` is a DNS host alias

for the cname `beta.example.com`, the Kerberos service canonicalizes `alpha.example.com` to `beta.example.com`. The KDC processes the ticket as a request for the service principal *host*`/beta.example.com`.

For the principal names that include the FQDN of a host, be sure to match the string that describes the DNS domain name in the `/etc/resolv.conf` file. The Kerberos service requires that the DNS domain name be in lowercase letters when you are specifying the FQDN for a principal. The DNS domain name can include uppercase and lowercase letters, but only use lowercase letters when you are creating a host principal. For example, the DNS domain name can be `example.com`, `Example.COM`, or any other variation. The principal name for the host would still be `host/boston.example.com@EXAMPLE.COM`.

In addition, the Service Management Facility (SMF) has been configured so that many of the daemons or commands do not start if the DNS client service is not running. The `kdb5_util`, `kadmind`, and `kpropd` daemons, as well as the `kprop` command are configured to depend on the DNS service. To fully use the features that are available from the Kerberos service and SMF, you must enable the DNS client service on all hosts.

Clock Synchronization Within a Kerberos Realm

The internal clocks of all hosts that participate in the Kerberos authentication system must be synchronized within a specified maximum amount of time. Known as *clock skew*, this feature provides another Kerberos security check. If the clock skew is exceeded between any of the participating hosts, requests are rejected.

One way to synchronize all the clocks is to use the Network Time Protocol (NTP) software. For more information, see “Synchronizing Clocks Between KDCs and Kerberos Clients” on page 139. Other ways of synchronizing the clocks can be used; however, some form of synchronization is required.

Supported Encryption Types in Kerberos

An *encryption type* is an identifier that specifies the encryption algorithm, encryption mode, and hash algorithms used in the Kerberos service. The keys in the Kerberos service have an associated encryption type that specifies the cryptographic algorithm and mode to be used when the service performs cryptographic operations with the key. For a list of supported encryption types, see the `krb5.conf`(4) and `kdb5_util`(1M) man pages.

If you want to change the encryption type, do so when you create a new principal database. Because of the interaction between the KDC, the server, and the client, changing the encryption

type on an existing database is difficult. For more information, see “Kerberos Encryption Types” on page 50.

Weak encryption types, such as `des`, are disallowed by default. If you need to use weak encryption types for backward compatibility or interoperability, set the `allow_weak_crypto` entry in the `libdefaults` section of the `/etc/krb5/krb5.conf` file to `true`.

Planning KDCs

KDCs use specific ports, require additional servers to handle larger ticket loads, and then require propagation techniques to keep the servers synchronized. Additionally, encryption types are centrally managed. You have several options for initially configuring your KDCs.

Ports for the KDC and Admin Services

By default, port `88` and port `750` are used for the KDC, and port `749` is used for the KDC administration daemon. You can use different port numbers. However, if you change the port numbers, then you must change the `/etc/services` and `/etc/krb5/krb5.conf` files on every client. In addition, you must update the `/etc/krb5/kdc.conf` file on each KDC.

Number of Slave KDCs

Slave KDCs generate credentials for clients just as the master KDC does. Slave KDCs provide backup if the master becomes unavailable. Plan to create at least one slave KDC per realm.

Additional slave KDCs might be required depending on the following factors:

- The number of physical segments in the realm. Normally, the network should be set up so that each segment can function, at least minimally, without the rest of the realm. To do so, a KDC must be accessible from each segment. The KDC in this case could be either a master or a slave.
- The number of clients in the realm. By adding more slave KDC servers, you can reduce the load on the current servers.

Avoid adding too many slave KDCs. Because the KDC database must be propagated to each server, the more KDC servers that are installed, the longer it can take to get the data updated throughout the realm. Also, because each slave retains a copy of the KDC database, more slaves increase the risk of a security breach.

Configure one or more slave KDCs to be swapped with the master KDC. The advantage of configuring at least one slave KDC in this way is that if the master KDC fails for any

reason, you will have a system preconfigured to become the master KDC. For instructions, see "Swapping a Master KDC and a Slave KDC" on page 140.

Kerberos Database Propagation

The database that is stored on the master KDC must be regularly propagated to the slave KDCs. You can configure the propagation of the database to be incremental. The incremental process propagates only updated information to the slave KDCs rather than the entire database. For information about database propagation, see "Administering the Kerberos Database" on page 145.

If you do not use incremental propagation, one of the first issues to resolve is how often to update the slave KDCs. The need to have up-to-date information that is available to all clients must be weighed against the time required to complete the update.

In large installations with many KDCs in one realm, one or more slaves can propagate the data so that the process is done in parallel. This strategy reduces the amount of time that the update takes, but it also increases administrative complexity. For more information, see "Setting Up Parallel Propagation for Kerberos" on page 154.

KDC Configuration Options

There are several ways to configure a KDC. The simplest ways use the `kdcmgr` utility to configure the KDC automatically or interactively. The automatic version requires that you use command-line options to define the configuration parameters. This method is especially useful for scripts. The interactive version prompts you for all information that is needed. For pointers to the instructions for using this command, see "Configuring KDC Servers" on page 69.

You can also use LDAP to manage the database files for Kerberos. For instructions, see "Configuring KDC Servers on LDAP Directory Servers" on page 88. LDAP simplifies administration at sites that require coordination between the Kerberos databases and their existing directory server setup.

Planning for Kerberos Clients

Kerberos clients can be installed automatically, installed from a script, or manually configured by editing configuration files. For protected network logins, the PAM framework provides the `pam_ldap` module. See the `pam_ldap(5)` man page. Also, when the client requests a service, that service can be granted by a server other than the master KDC.

Planning for Automatic Installation of Kerberos Clients

Kerberos clients can be configured quickly and easily by using the Oracle Solaris Automated Installer (AI) feature. AI server administrators create and assign Kerberos configuration profiles to AI clients. Additionally, the AI server delivers the client keys. Therefore, at installation, the Kerberos client is a fully provisioned Kerberos system, capable of hosting secure services. Using the Automated Installer can lower system administration and maintenance costs.

You can use the `kclient` command to create automated installation for clients of any type of KDC.

- You can use AI for clients that are not clients of an Oracle Solaris KDC. For the list of KDC vendors, see the `kclient`(1M) man page.

  For all KDC types, pre-generated keytab transfer is supported. Oracle Solaris KDC and MS AD also support auto-registering.
- You run the `kclient` command to create Kerberos configuration profiles for AI. For more information, see the `kclient`(1M) man page and "Kerberos Client Configuration Options" on page 63. For instructions to configure Kerberos clients through the AI, see "How to Configure Kerberos Clients Using AI" in *Installing Oracle Solaris 11.3 Systems*.

Kerberos Client Configuration Options

You can configure Kerberos clients by using the `kclient` configuration utility or by manually editing files. The utility runs in interactive mode and noninteractive mode. In interactive mode, you are prompted for Kerberos-specific parameter values, so you can make changes when configuring the client. In noninteractive mode, you supply a file with parameter values. Also, you can add command-line options in the noninteractive mode. Because the interactive and noninteractive modes require fewer steps than manual configuration, they are quicker and less prone to error.

If the following setup is in effect, then no explicit configuration of your Kerberos client is necessary:

- DNS is configured to return SRV records for KDCs.
- The realm name matches the DNS domain name, or the KDC supports referrals.
- The Kerberos client does not require keys that are different from the KDC server's keys.

You still might want to explicitly configure the Kerberos client for the following reasons:

- The zero-configuration process performs more DNS lookups than a directly configured client, and therefore is less efficient than direct configuration.
- If referrals are not used, the zero-configuration logic depends on the DNS domain name of the host to determine the realm. This configuration introduces a small security risk, but the risk is much smaller than enabling `dns_lookup_realm`.

- The `pam_krb5` module relies on a host key entry in the keytab file. Although this requirement can be disabled in the `krb5.conf` file, doing so is not recommended for security reasons. For more information, see "Kerberos Client Login Security" on page 64 and the `krb5.conf(4)` man page.

For a full description of client configuration, see "Configuring Kerberos Clients" on page 107.

Kerberos Client Login Security

At login, a client uses the `pam_krb5` module to verify that the KDC that issued the latest TGT is the same KDC that issued the client host principal that is stored in the `/etc/krb5/krb5.keytab` file. The `pam_krb5` module verifies the KDC when the module is configured in the authentication stack. For some configurations, such as DHCP clients that do not store a client host principal, this check needs to be disabled. To turn off this check, you must set the `verify_ap_req_nofail` option in the `krb5.conf` file to `false`. For more information, see "Disabling Verification of the Ticket-Granting Ticket" on page 119.

Trusted Delegated Services in Kerberos

For some applications, a client might need to delegate authority to a server to act on its behalf in contacting other services. The client must forward credentials to an intermediate server. The client's ability to obtain a service ticket to a server conveys no information to the client about whether the server can be trusted to accept delegated credentials. The `ok_to_auth_as_delegate` option to the `kadmin` command provides a way for a KDC to communicate the local realm policy to a client regarding whether an intermediate server is trusted to accept such credentials.

The encrypted part of the KDC reply to the client can include a copy of the credential ticket flags with the `ok_to_auth_as_delegate` option set. A client can use this setting to determine whether to delegate credentials (by granting either a proxy or a forwarded TGT) to this server. When setting this option, consider the security and placement of the server on which the service runs, as well as whether the service requires the use of delegated credentials.

Planning Kerberos Use of UNIX Names and Credentials

The Kerberos service provides a default mapping of GSS credential names to UNIX user IDs (UIDs) for GSS applications that require this mapping, such as NFS. GSS credential names are equivalent to Kerberos principal names when using the Kerberos service. Also, UNIX users who do not have valid user accounts in the default Kerberos realm can be automatically migrated by using the PAM framework.

Map GSS Credentials to UNIX Credentials

The default mapping algorithm uses the primary name of the Kerberos principal to look up the UID. The lookup occurs in the default realm or any realm that is allowed by the `auth_to_local_realm` parameter in the `/etc/krb5/krb5.conf` file. For example, the user principal name `jdoe@EXAMPLE.COM` is mapped to the UID of the UNIX user named `jdoe` by using the password table. The user principal name `jdoe/admin@EXAMPLE.COM` is not mapped because the principal name includes the `admin` instance component.

If the default mappings for the user credentials are sufficient, the GSS credential table does not need to be populated. If the default mapping is not sufficient, as when you want to map a principal name that contains an instance component, then other methods are required. For more information, see the following:

- "How to Create and Modify a Credential Table" on page 131
- "How to Provide Credential Mapping Between Realms" on page 132
- "Observing Mapping From GSS Credentials to UNIX Credentials" on page 206

`gsscred` Table

The `gsscred` table is used by an NFS server when the server is trying to identify a Kerberos user, if the default mappings are not sufficient. The NFS service uses UNIX UIDs to identify users. These IDs are not part of a user principal or a credential. The `gsscred` table provides additional mapping from GSS credentials to UNIX UIDs from the password file. The table must be created and administered after the KDC database is populated.

When a client request comes in, the NFS service tries to map the credential name to a UNIX UID. If the mapping fails, the `gsscred` table is checked.

Automatic User Migration to a Kerberos Realm

UNIX users who do not have valid user accounts in the default Kerberos realm can be automatically migrated by using the PAM framework. Specifically, you add the `pam_krb5_migrate.so` module to the authentication stack of the PAM service. Services are then configured so that whenever a user who does not have a Kerberos principal performs a successful password login to a system, a Kerberos principal would be automatically created for that user. The new principal password is then the same as the UNIX password. For instructions about using the `pam_krb5_migrate.so` module, see "How to Configure Automatic Migration of Users in a Kerberos Realm" on page 121.

♦ ♦ ♦ CHAPTER 4

Configuring the Kerberos Service

This chapter provides configuration procedures for KDC servers, network application servers, NFS servers, and Kerberos clients. Many of these procedures require `root` access, so they should be performed by system administrators or advanced users. Cross-realm configuration procedures and other topics related to KDC servers are also covered.

This chapter covers the following topics:

Configuring the Kerberos Service

Because some procedures in the configuration process depend on other procedures, they must be done in a specific order. These procedures often establish services that are required to use the Kerberos service. Other procedures are not dependent on any order, and can be done when appropriate. The following task map shows a suggested order for a Kerberos installation.

Note - The examples in these sections use default encryption types, which are not FIPS 140-2 validated for Oracle Solaris. To run in FIPS 140-2 mode, you must limit the encryption types to the `des3-cbc-sha1` encryption type for the database, servers, and client communications. Before creating the KDC, edit the files in "How to Configure Kerberos to Run in FIPS 140-2 Mode" on page 71.

TABLE 2 Task Map: Configuring the Kerberos Service

Task	Description	For Instructions
1. Plan your Kerberos installation.	Resolves configuration issues before you start the software configuration process. Planning ahead saves you time and other resources later.	Chapter 3, "Planning for the Kerberos Service"
2. Configure the KDC servers.	Configures and builds the master KDC and the slave KDC servers and KDC database for a realm.	"Configuring KDC Servers" on page 69
2a. (Optional) Configure Kerberos to run in FIPS 140-2 mode.	Enables the use of FIPS 140-2 validated algorithms only.	"How to Configure Kerberos to Run in FIPS 140-2 Mode" on page 71
2b. (Optional) Configure Kerberos to run on LDAP.	Configures the KDC to use an LDAP Directory Server.	"Configuring KDC Servers on LDAP Directory Servers" on page 88
3. Install clock synchronization software.	Creates a central clock that provides the time for all systems on the network.	"Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139
4. (Optional) Configure swappable KDC servers.	Makes the task of swapping the master KDC and a slave KDC easier.	"How to Configure a Swappable Slave KDC" on page 141
4. (Optional) Increase security on the KDC servers.	Prevents security breaches on the KDC servers.	"Restricting Access to KDC Servers" on page 158

Configuring Additional Kerberos Services

Once the required steps have been completed, perform the following procedures when appropriate.

TABLE 3 Task Map: Configuring Additional Kerberos Services

Task	Description	For Instructions
Configure cross-realm authentication.	Enables communications from one realm to another realm.	"Configuring Cross-Realm Authentication" on page 136
Configure Kerberos application servers.	Enables a server to support services such as `ftp` by using Kerberos authentication.	"Configuring Kerberos Network Application Servers" on page 126
Configure delayed execution services.	Enables a `cron` host to execute tasks at arbitrary times.	"Configuring Delayed Execution for Access to Kerberos Services" on page 134
Configure Kerberos NFS server.	Enables a server to share a file system that requires Kerberos authentication.	"Configuring Kerberos NFS Servers" on page 129
Configure Kerberos clients.	Enables a client to use Kerberos services.	"Configuring Kerberos Clients" on page 107

Task	Description	For Instructions
Synchronize clocks to assist authentication.	Configures a clock synchronization protocol.	“Synchronizing Clocks Between KDCs and Kerberos Clients” on page 139
Manage the Kerberos database.	Maintains the Kerberos database.	“Administering the Kerberos Database” on page 145
Manage the stash file of the Kerberos database.	Handles the keys for the Kerberos database.	“Administering the Stash File for the Kerberos Database” on page 155

Configuring KDC Servers

After you install the Kerberos software, you must configure the Key Distribution Center (KDC) servers. Configuring the master KDC and at least one slave KDC provides the service that issues credentials. These credentials are the basis for the Kerberos service, so the KDCs must be configured before you attempt other tasks.

The most significant difference between a master KDC and a slave KDC is that only the master KDC can handle database administration requests. For example, changing a password or adding a new principal must be done on the master KDC. These changes can then be propagated to the slave KDCs. Both the slave KDC and master KDC generate credentials. The slave KDCs provide redundancy when the master KDC cannot respond.

You can choose to configure and build the master KDC server, the database, and additional servers in various ways:

- Automatic – Recommended for scripts
- Interactive – Sufficient for most installations
- Manual – Necessary for more complex installations
- Manual with LDAP – Necessary when using LDAP with the KDC

TABLE 4 Task Map: Configuring KDC Servers

Task	Description	For Instructions
Install the KDC package.	Required package for creating KDCs.	“How to Install the KDC Package” on page 70
(Optional) Configure Kerberos to run in FIPS 140-2 mode.	Enables the use of FIPS 140-2 validated algorithms only.	“How to Configure Kerberos to Run in FIPS 140-2 Mode” on page 71
Use a script to configure the master KDC.	Simplifies initial configuration.	“How to Use `kdcmgr` to Configure the Master KDC” on page 71 Example 6, “Running the `kdcmgr` Command Without Arguments,” on page 73
Use a script to configure a slave KDC server.	Simplifies initial configuration.	“How to Use `kdcmgr` to Configure a Slave KDC” on page 74

Task	Description	For Instructions
Manually configure the master KDC server.	Provides control over every entry in the KDC configuration files during initial installation.	"How to Manually Configure a Master KDC" on page 75
Manually configure a slave KDC server.	Provides control over every entry in the KDC configuration files during initial installation.	"How to Manually Configure a Slave KDC" on page 80
Manually configure a second master KDC server.	Creates a multi-master configuration with an Oracle Directory Server Enterprise Edition (DSEE) LDAP server.	"How to Manually Configure a Second Master KDC" on page 85
Replace principal keys on a KDC server.	Updates the session key on a legacy KDC server to use stronger encryption types.	"Replacing the Ticket-Granting Service Keys on a Master Server" on page 87
Manually configure the master KDC to use LDAP.	Configures the master KDC server and Kerberos clients to use LDAP.	"Configuring KDC Servers on LDAP Directory Servers" on page 88

How to Install the KDC Package

By default, Kerberos client software is installed on your system. To install a Key Distribution Center (KDC), you must add the KDC package.

Before You Begin You must be assigned the Software Installation rights profile to add packages to the system. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. Install the KDC package.

```
$ pkg install system/security/kerberos-5
```

For more information, see the pkg(1) man page.

2. (Optional) List the Kerberos services.

With the addition of the server package, your system has two additional Kerberos services. Like Kerberos client software, these services are disabled by default. You configure Kerberos before enabling these services.

```
$ svcs -a krb5
STATE          STIME    FMRI
disabled       Sep_10   svc:/network/security/krb5kdc:default
disabled       Sep_10   svc:/network/security/krb5_prop:default
$ svcs -a | grep kerb
STATE          STIME    FMRI
disabled       Sep_07   svc:/system/kerberos/install:default
```

How to Configure Kerberos to Run in FIPS 140-2 Mode

Perform these steps before creating keys for they master KDC and Kerberos principals. You should perform these steps as you manually configure the master KDC, as described in "How to Manually Configure a Master KDC" on page 75.

Before You Begin In order for Kerberos to run in FIPS 140-2 mode, you must enable FIPS 140-2 mode on your system. See "Create a Boot Environment with FIPS 140 Enabled" in *Managing Encryption and Certificates in Oracle Solaris 11.3*.

1. **On the master KDC, edit the encryption types for the KDC.**

   In the `[realms]` section of the `kdc.conf` file, set the master key type for the KDC database:

   ```
   # pfedit  /etc/krb5/kdc.conf
   ...
   master_key_type = des3-cbc-sha1-kd
   ```

2. **In the same file, explicitly forbid other encryption types.**

   Because you can also set encryption by running a command, the configuration files should prevent the use of a non-FIPS 140-2 algorithm argument to a command.

   ```
           supported_enctypes = des3-cbc-sha1-kd:normal
   ```

3. **Edit the encryption types for transactions in the `[libdefaults]` section of the `krb5.conf` file.**

   These parameters limit the encryption types for the Kerberos servers, services, and clients.

   ```
   # pfedit /etc/krb5/krb5.conf
           default_tgs_enctypes = des3-cbc-sha1-kd
           default_tkt_enctypes = des3-cbc-sha1-kd
           permitted_enctypes = des3-cbc-sha1-kd
   ```

4. **In the same file, explicitly forbid weak encryption types.**

   ```
           allow_weak_enctypes = false
   ```

Troubleshooting See "Kerberos Encryption Types" on page 50.

How to Use `kdcmgr` to Configure the Master KDC

The `kdcmgr` script provides a command-line interface to install the master and slave KDCs. For the master, you must create a password for the Kerberos database and a password for

the administrator. On the slave KDCs, you must supply these passwords to complete the installation. For information about these passwords, see the kdcmgr(1M) man page.

Before You Begin You must assume the root role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. Create the master KDC.

On the command line, run the kdcmgr command and name the administrator and the realm.

You are prompted for the Kerberos database password, called the *master key* and the password for the administrative principal. The script prompts for the passwords.

```
kdc1# kdcmgr -a kws/admin -r EXAMPLE.COM create master

Starting server setup
---------------------------------------

Setting up /etc/krb5/kdc.conf

Setting up /etc/krb5/krb5.conf

Initializing database '/var/krb5/principal' for realm 'EXAMPLE.COM',
master key name 'K/M@EXAMPLE.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:     /** Type strong password **/
Re-enter KDC database master key to verify: xxxxxxxx

Authenticating as principal root/admin@EXAMPLE.COM with password.
WARNING: no policy specified for kws/admin@EXAMPLE.COM; defaulting to no policy
Enter password for principal "kws/admin@EXAMPLE.COM":     /** Type strong password **/
Re-enter password for principal "kws/admin@EXAMPLE.COM": xxxxxxxx
Principal "kws/admin@EXAMPLE.COM" created.

Setting up /etc/krb5/kadm5.acl.

-------------------------------------------------
Setup COMPLETE.

kdc1#
```

Note - Save and store these passwords in a safe location.

2. (Optional) Display the status of the master KDC.

```
# kdcmgr status
```

3. Synchronize this system's clock with other clocks in the realm.

For authentication to succeed, every clock must be within the default time that is defined in the libdefaults section of the krb5.conf file. For more information, see the krb5.conf(4) man page and "Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139.

Note - A master KDC cannot be the clock synchronization server.

- **If you do not already have a clock synchronization server, perform this step after you configure one.**
- **If you have a clock synchronization server, make this system a client of the server by following the PTP or NTP instructions.**
  - **To configure a PTP client, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
  - **To configure an NTP client, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**

Example 6 Running the kdcmgr Command Without Arguments

In this example, the administrator supplies the realm name and admin principal when prompted by the script.

```
kdc1# kdcmgr create master

Starting server setup
--------------------------------------

Enter the Kerberos realm: EXAMPLE.COM

Setting up /etc/krb5/kdc.conf

Setting up /etc/krb5/krb5.conf

Initializing database '/var/krb5/principal' for realm 'EXAMPLE.COM',
master key name 'K/M@EXAMPLE.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:     /** Type strong password **/
Re-enter KDC database master key to verify: xxxxxxxx

Enter the krb5 administrative principal to be created: kws/admin

Authenticating as principal root/admin@EXAMPLE.COM with password.
```

```
WARNING: no policy specified for kws/admin@EXAMPLE.COM; defaulting to no policy
Enter password for principal "kws/admin@EXAMPLE.COM":     /** Type strong password **/
Re-enter password for principal "kws/admin@EXAMPLE.COM": xxxxxxxx
Principal "kws/admin@EXAMPLE.COM" created.

Setting up /etc/krb5/kadm5.acl.

--------------------------------------------------
Setup COMPLETE.

kdc1#
```

▼ How to Use kdcmgr to Configure a Slave KDC

Before You Begin The master KDC server is configured.

You must assume the root role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. Create a slave KDC.

On the command line, run the kdcmgr command and name the administrator, the realm, and the master KDC.

The script prompts for the two passwords that you created in "How to Use kdcmgr to Configure the Master KDC" on page 71, one for the administrative principal and one for the KDC database.

```
kdc2# kdcmgr -a kws/admin -r EXAMPLE.COM create -m kdc1 slave

Starting server setup
--------------------------------------

Setting up /etc/krb5/kdc.conf

Setting up /etc/krb5/krb5.conf
Obtaining TGT for kws/admin ...
Password for kws/admin@EXAMPLE.COM: xxxxxxxx

Setting up /etc/krb5/kadm5.acl.

Setting up /etc/krb5/kpropd.acl.

Waiting for database from master...
Waiting for database from master...
Waiting for database from master...
kdb5_util: Cannot find/read stored master key while reading master key
kdb5_util: Warning: proceeding without master key
```

```
Enter KDC database master key: xxxxxxxx

------------------------------------------------
Setup COMPLETE.

kdc2#
```

2. **(Optional) Display the status of the KDC.**

```
# kdcmgr status
```

3. **Synchronize this system's clock with other clocks in the realm.**

   For authentication to succeed, every clock must be within the default time that is defined in the `libdefaults` section of the `krb5.conf` file. For more information, see the `krb5.conf`(4) man page and "Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139.

   - **If you do not already have a clock synchronization server, configure PTP or NTP on this server.**
     - **To configure PTP, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
     - **To configure NTP, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
   - **If you have a clock synchronization server, make this system a client of that server by following the PTP or NTP instructions.**

4. **Return to the master KDC to make it a client of the clock synchronization server.**

How to Manually Configure a Master KDC

In this procedure, incremental propagation is configured. This procedure uses the following configuration parameters:

- Realm name = `EXAMPLE.COM`
- DNS domain name = `example.com`
- Master KDC = `kdc1.example.com`
- `admin` principal = `kws/admin`
- Online help URL = `http://docs.oracle.com/cd/E23824_01/html/821-1456/aadmin-23.html`

Note - Adjust the URL to point to the location of the online help, as described in "gkadmin GUI" on page 162.

Before You Begin The host is configured to use DNS. For specific naming instructions if this master is to be swappable, see "Swapping a Master KDC and a Slave KDC" on page 140.

You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Install the KDC package.**

   Follow the instructions in "How to Install the KDC Package" on page 70.

2. **(Optional) Configure Kerberos to run in FIPS 140-2 mode.**

   If you plan to run the realm in FIPS 140-2 mode, specify the encryption types as you configure the master KDC server. See "How to Configure Kerberos to Run in FIPS 140-2 Mode" on page 71.

3. **Edit the Kerberos configuration file, `krb5.conf`.**

   For a description of this file, see the `krb5.conf`(4) man page.

   In this example, the administrator changes the lines for `default_realm`, `kdc`, `admin_server`, and all `domain_realm` entries, and edits the `help_url` entry.

```
kdc1# pfedit /etc/krb5/krb5.conf
...
[libdefaults]
     default_realm = EXAMPLE.COM

[realms]
     EXAMPLE.COM = {
        kdc = kdc1.example.com
        admin_server = kdc1.example.com
     }

[domain_realm]
     .example.com = EXAMPLE.COM
#
# if the domain name and realm name are equivalent,
# this entry is not needed
#
[logging]
     default = FILE:/var/krb5/kdc.log
     kdc = FILE:/var/krb5/kdc.log

[appdefaults]
```

```
    gkadmin = {
    help_url = http://docs.oracle.com/cd/E23824_01/html/821-1456/aadmin-23.html
  }
```

Note - If you must communicate with an older Kerberos system, you might need to restrict the encryption types. For a description of the issues involved with restricting the encryption types, see "Kerberos Encryption Types" on page 50.

4. Name the realm in the KDC configuration file, `kdc.conf`.

For a description of this file, see the `kdc.conf`(4) man page.

In this example, in addition to the realm name definition, the administrator changes incremental propagation and logging defaults.

```
kdc1# pfedit /etc/krb5/kdc.conf
[kdcdefaults]
     kdc_ports = 88,750

[realms]
     EXAMPLE.COM = {
        profile = /etc/krb5/krb5.conf
        database_name = /var/krb5/principal
        acl_file = /etc/krb5/kadm5.acl
        kadmind_port = 749
        max_life = 8h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        sunw_dbprop_enable = true
        sunw_dbprop_master_ulogsize = 1000
     }
```

Note - If you must communicate with an older Kerberos system, you might need to restrict the encryption types. For a description of the issues involved with restricting the encryption types, see "Kerberos Encryption Types" on page 50.

5. Create the KDC database by using the `kdb5_util` command.

The `kdb5_util` command creates the KDC database. Also, when used with the `-s` option, this command creates a stash file that is used to authenticate the KDC to itself before the `kadmind` and `krb5kdc` daemons are started. For more information, see the `kdb5_util`(1M), `kadmind`(1M), and `krb5kdc`(1M) man pages.

```
kdc1# /usr/sbin/kdb5_util create -s
Initializing database '/var/krb5/principal' for realm 'EXAMPLE.COM'
master key name 'K/M@EXAMPLE.COM'
You will be prompted for the database Master Password.
It is important that you NOT FORGET this password.
Enter KDC database master key:     /** Type strong password **/
Re-enter KDC database master key to verify: xxxxxxxx
```

Tip - If this step fails, verify that the KDC principal is identified by its FQDN.

```
# getent hosts IP-address-of-KDC
IP-address-of-KDC kdc      /** This entry does not include FQDN **/
```

Then, add the FQDN as the first KDC entry in your `/etc/hosts` file, for example:

```
10.1.2.3 kdc1.example.com kdc
```

6. Add administration principals to the database.

You can add as many `admin` principals as you need. You must add at least one `admin` principal to complete the KDC configuration process. For this example, a `kws/admin` principal is added. You can substitute an appropriate principal name instead of "`kws`".

```
kadmin.local: addprinc kws/admin
Enter password
for principal kws/admin@EXAMPLE.COM:     /** Type strong password **/
Re-enter password
for principal kws/admin@EXAMPLE.COM: xxxxxxxx
Principal "kws/admin@EXAMPLE.COM" created.
kadmin.local:
```

For more information, see the `kadmin`(1M) man page.

7. Edit the Kerberos access control list file, `kadm5.acl`.

Once populated, the `/etc/krb5/kadm5.acl` file must contain all principal names that are allowed to administer the KDC.

```
kws/admin@EXAMPLE.COM   *
```

The preceding entry gives the `kws/admin` principal in the `EXAMPLE.COM` realm the ability to modify principals and policies in the KDC. The default principal entry is an asterisk (*), which matches all `admin` principals. This entry can be a security risk. Modify the file to explicitly list every `admin` principal and their rights. For more information, see the `kadm5.acl`(4) man page.

8. Enable the KDC and `kadmin` services.

```
kdc1# svcadm enable -r network/security/krb5kdc
kdc1# svcadm enable -r network/security/kadmin
```

9. Add more principals.

```
kdc1# /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin:
```

a. Create the master KDC `host` principal and add it to the keytab file.

The host principal is used by Kerberized applications, such as `kprop`, to propagate changes to the slave KDCs. This principal is also used to provide secure remote access to the KDC server by using network applications, such as `ssh`. Note that when the principal instance is a host name, the FQDN must be specified in lowercase letters regardless of the case of the domain name in the name service.

```
kadmin: addprinc -randkey host/kdc1.example.com
Principal "host/kdc1.example.com@EXAMPLE.COM" created.
kadmin:
```

Adding the host principal to the keytab file enables this principal to be used by application servers, like `sshd`, automatically.

```
kadmin: ktadd host/kdc1.example.com
Entry for principal host/kdc1.example.com with kvno 3, encryption type AES-256 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/kdc1.example.com with kvno 3, encryption type AES-128 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/kdc1.example.com with kvno 3, encryption type Triple DES
 cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin:
```

b. (Optional) Create the `clntconfig` principal and give it enough privileges to perform `kclient` tasks.

This principal is used by the `kclient` utility during the installation of a Kerberos client. If you do not plan on using this utility, then you do not need to add the principal. The users of the `kclient` utility need to use this password.

```
kadmin: addprinc clntconfig/admin
Enter password for principal clntconfig/admin@EXAMPLE.COM:     /** Type strong  password
 **/
Re-enter password for principal clntconfig/admin@EXAMPLE.COM: xxxxxxxx
Principal "clntconfig/admin@EXAMPLE.COM" created.
kadmin: quit
```

Note - Save and store this password in a safe location.

```
# pfedit /etc/krb5/kadm5.acl
...
clntconfig/admin@EXAMPLE.COM  acdilm
```

For more information, see the `kclient`(1M) man page.

10. **Synchronize this system's clock with other clocks in the realm.**

    For authentication to succeed, every clock must be within the default time that is defined in the `libdefaults` section of the `krb5.conf` file. For more information, see the `krb5.conf(4)` man page and "Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139.

    Note - A master KDC cannot be the clock synchronization server.

    - **If you do not already have a clock synchronization server, perform this step after you configure one.**
    - **If you have a clock synchronization server, make this system a client of the server by following the PTP or NTP instructions.**
        - **To configure a PTP client, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
        - **To configure an NTP client, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**

11. **Configure additional KDCs.**

    To provide redundancy, install at least one other KDC. The secondary KDC can be a slave KDC or an additional master KDC in a multi-master configuration.

    - For instructions to create a slave KDC, see "How to Use `kdcmgr` to Configure a Slave KDC" on page 74 or "How to Manually Configure a Slave KDC" on page 80.
    - For instructions to create a second master KDC, see "How to Manually Configure a Second Master KDC" on page 85.
    - For instructions to create a second master KDC in an LDAP configuration, see "How to Configure a Master KDC on an Oracle DSEE LDAP Directory Server for OpenLDAP Clients" on page 99.

▼ How to Manually Configure a Slave KDC

In this procedure, you configure a new slave KDC named `kdc2`. You also configure incremental propagation. This procedure uses the following configuration parameters:

- Realm name = `EXAMPLE.COM`
- DNS domain name = `example.com`
- Master KDC = `kdc1.example.com`

- Slave KDC = `kdc2.example.com`
- `admin` principal = `kws/admin`

Before You Begin Make sure the master KDC is configured. If this slave is to be swappable, follow the instructions in "How to Swap a Master KDC and a Slave KDC" on page 141.

You must assume the `root` role on the KDC server. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Add slave host principals to the database and create the principal for incremental propagation.**

   a. **Log in to the master KDC by using the `kadmin` command.**

   Use one of the `admin` principal names that you created when you configured the master KDC.

   ```
   kdc1# /usr/sbin/kadmin -p kws/admin
   Enter password: xxxxxxxx
   kadmin:
   ```

   For more information, see the `kadmin`(1M) man page.

   b. **Add slave host principals to the database, if not already done.**

   Slaves must have a host principal. Note that when the principal instance is a host name, the FQDN must be specified in lowercase letters regardless of the case of the domain name in the name service.

   ```
   kadmin: addprinc -randkey host/kdc2.example.com
   Principal "host/kdc2.example.com@EXAMPLE.COM" created.
   kadmin:
   ```

   c. **Create the `kiprop` principal to authorize incremental propagation from the master KDC.**

   ```
   kadmin: addprinc -randkey kiprop/kdc2.example.com
   Principal "kiprop/kdc2.example.com@EXAMPLE.COM" created.
   kadmin:
   ```

   d. **Quit `kadmin`.**

   ```
   kadmin: quit
   ```

2. **On the master KDC, add an entry for each slave to the `krb5.conf` Kerberos configuration file.**

   For a description of this file, see the `krb5.conf`(4) man page.

   ```
   kdc1# pfedit /etc/krb5/krb5.conf
   ```

```
.
.
[realms]
EXAMPLE.COM = {
kdc = kdc1.example.com
kdc = kdc2.example.com
admin_server = kdc1.example.com
}
```

3. **On the master KDC, add a `kiprop` entry to the `kadm5.acl` file to enable the master KDC to receive requests for incremental propagation.**

```
kdc1# pfedit /etc/krb5/kadm5.acl
*/admin@EXAMPLE.COM *
kiprop/kdc2.example.com@EXAMPLE.COM p
```

4. **Restart the `kadmin` service to use the new entry.**

```
kdc1# svcadm restart network/security/kadmin
```

5. **Copy the KDC administration files from the master KDC server to all slave KDCs.**

   Each slave KDC must have up-to-date information about the master KDC server. You can use `sftp` or a similar transfer mechanism to get copies of the following files from the master KDC:

   - `/etc/krb5/krb5.conf`
   - `/etc/krb5/kdc.conf`

6. **On all slave KDCs, add an entry for the master KDC and each slave KDC into the `kpropd.acl` database propagation configuration file.**

```
kdc2# pfedit /etc/krb5/kpropd.acl
host/kdc1.example.com@EXAMPLE.COM
host/kdc2.example.com@EXAMPLE.COM
```

7. **On all slave KDCs, make sure that the `kadm5.acl` Kerberos access control list file is not populated.**

   An unmodified `kadm5.acl` file would look like the following example:

```
kdc2# pfedit /etc/krb5/kadm5.acl
*/admin@___default_realm___ *
```

   If the file has `kiprop` entries, remove them.

8. **On all slave KDCs, define the polling interval in the `kdc.conf` file.**

   Replace the `sunw_dbprop_master_ulogsize` entry with an entry that defines the slave's polling interval. The following entry sets the poll time to two minutes:

```
kdc2# pfedit /etc/krb5/kdc.conf
```

```
[kdcdefaults]
      kdc_ports = 88,750

[realms]
      EXAMPLE.COM= {
        profile = /etc/krb5/krb5.conf
        database_name = /var/krb5/principal
        acl_file = /etc/krb5/kadm5.acl
        kadmind_port = 749
        max_life = 8h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        sunw_dbprop_enable = true
        sunw_dbprop_slave_poll = 2m
      }
```

9. Configure the slave's KDC keytab file.

a. Log in to the slave KDC by using the `kadmin` command.

Log in with one of the `admin` principal names that you created when you configured the master KDC.

```
kdc2# /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin:
```

b. Add the slave's host principal to the slave's KDC keytab file.

This entry enables the `kprop` command and other Kerberized applications to function. Note that when the principal instance is a host name, the FQDN must be specified in lowercase letters regardless of the case of the domain name in the name service. For more information, see the `kprop`(1M) man page.

```
kadmin: ktadd host/kdc2.example.com
Entry for principal host/kdc2.example.com with kvno 3, encryption type AES-256 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/kdc2.example.com with kvno 3, encryption type AES-128 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/kdc2.example.com with kvno 3, encryption type Triple DES
 cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin:
```

c. Add the `kiprop` principal to the slave KDC's keytab file to enable the `kpropd` command to authenticate itself when incremental propagation is started.

```
kadmin: ktadd kiprop/kdc2.example.com
Entry for principal kiprop/kdc2.example.com with kvno 3, encryption type AES-256 CTS
 mode
```

```
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal kiprop/kdc2.example.com with kvno 3, encryption type AES-128 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal kiprop/kdc2.example.com with kvno 3, encryption type Triple DES
 cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
```

10. **On the slave KDC, enable the Kerberos propagation daemon.**

```
kdc2# svcadm enable network/security/krb5_prop
```

11. **On the slave KDC, create a stash file.**

```
kdc2# /usr/sbin/kdb5_util stash
kdb5_util: Cannot find/read stored master key while reading master key
kdb5_util: Warning: proceeding without master key

Enter KDC database master key: xxxxxxxx
```

For more information, see the kdb5_util(1M) man page.

12. **Synchronize this system's clock with other clocks in the realm.**

For authentication to succeed, every clock must be within the default time that is defined in the libdefaults section of the krb5.conf file. For more information, see the krb5.conf(4) man page and "Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139.

- **If you do not already have a clock synchronization server, configure PTP or NTP on this server.**
  - **To configure PTP, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
  - **To configure NTP, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
- **If you have a clock synchronization server, make this system a client of that server by following the PTP or NTP instructions.**

13. **Return to the master KDC to make it a client of the clock synchronization server.**

14. **On the slave KDC, enable the KDC daemon.**

```
kdc2# svcadm enable network/security/krb5kdc
```

How to Manually Configure a Second Master KDC

In this procedure, you configure a second master KDC named `kdc2` with the following parameters:

- Realm name = `EXAMPLE.COM`
- DNS domain name = `example.com`
- Master KDC = `kdc1.example.com`
- `admin` principal for `kdc1` = `kws/admin`
- LDAP server for `kdc1` = `dsserver.example.com`
- Second master KDC and LDAP server = `kdc2.example.com`
- `admin` principal for `kdc2` = `tester/admin`

You run all commands in this procedure on the new second master KDC.

Before You Begin Make sure that the master KDC is configured.

You must assume the `root` role on the KDC server. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Log in to the second master KDC and configure the LDAP TLS bindings.**

   For examples of configuring TLS keys, see:

   - Step 1 in "How to Configure a Master KDC on an Oracle DSEE LDAP Directory Server for OpenLDAP Clients" on page 99
   - Step 10 in "How to Configure a Master KDC on an Oracle Unified Directory LDAP Directory Server" on page 94.

2. **Add the first master KDC and the LDAP servers to this master KDC's Kerberos configuration file.**

   For a description of this file, see the `krb5.conf(4)` man page. The following example assumes that you are configuring an Oracle Directory Server Enterprise Edition server.

```
kdc2# pfedit /etc/krb5/krb5.conf
.
.
[realms]
     EXAMPLE.COM = {
        kdc = kdc1.example.com
        kdc = kdc2.example.com
        admin_server = kdc2.example.com
        admin_server = kdc1.example.com
        database_module = LDAP
     }
[dbmodules]
```

```
    LDAP = {
       db_library =kldap
       ldap_kerberos_container_dn = "cn=krbcontainer,dc=example,dc=com"
       ldap_kdc_dn = "cn=kdc service,ou=profile,dc=example,dc=com"
       ldap_kadmind_dn = "cn=kadmin service,ou=profile,dc=example,dc=com"
       ldap_cert_path = /var/ldap
       ldap_servers = ldaps://kdc2.example.com ldaps://dsserver.example.com
    }
...
```

3. Create a stash file for LDAP binding to the KDC and `kadmin` services.

On a second master KDC, the `kadmin` service must also have a stash file for LDAP binding.

```
kdc2# kdb5_ldap_util stashsrvpw "cn=kdc service,ou=profile,dc=example,dc=com"
kdc2# kdb5_ldap_util stashsrvpw "cn=kadmin service,ou=profile,dc=example,dc=com"
```

4. Create the master KDC's stash file.

```
kdc2# kdb5_util stash
kdb5_util: Cannot find/read stored master key while reading master key
kdb5_util: Warning: proceeding without master key

Enter KDC database master key: xxxxxxxx
```

For more information, see the `kdb5_util(1M)` man page.

5. Add the location of the Kerberos configuration file to the KDC configuration file.

```
# pfedit /etc/krb5/kdc.conf
 ...
 [realms]
         EXAMPLE.COM = {
                 profile = /etc/krb5/krb5.conf
 ...
```

6. Create principals for the second master KDC.

The `kdb5_ldap_util` command usually creates the administrative principals for a KDC. However, most of these principals were created when the first master KDC was created. The `kadmin` and `changepw` principals in this step are specifically for the second master KDC.

```
# kadmin -p tester/admin
kadmin: addprinc -randkey kadmin/kdc2.example.com
kadmin: addprinc -randkey changepw/kdc2.example.com
```

7. Synchronize this system's clock with other clocks in the realm.

For authentication to succeed, every clock must be within the default time that is defined in the `libdefaults` section of the `krb5.conf` file. For more information, see the `krb5.conf(4)` man page and “Synchronizing Clocks Between KDCs and Kerberos Clients” on page 139.

Note - A master KDC cannot be the clock synchronization server.

- **If you do not already have a clock synchronization server, perform this step after you configure one.**
- **If you have a clock synchronization server, make this system a client of the server by following the PTP or NTP instructions.**
  - **To configure a PTP client, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
  - **To configure an NTP client, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**

8. **Enable the KDC and `kadmin` services.**

```
kdc1# svcadm enable -r network/security/krb5kdc
kdc1# svcadm enable -r network/security/kadmin
```

Replacing the Ticket-Granting Service Keys on a Master Server

When the ticket-granting service (TGS) principal only has a DES key, the key restricts the encryption type of the ticket-granting ticket (TGT) session key to DES. When the KDC is updated to a release that supports stronger encryption types, you must replace the DES key of the TGS principal so that the principal can generate stronger encryption for all session keys.

You can replace the key remotely or on the master server. You must be an `admin` principal who is assigned the `changepw` privilege.

- To replace the TGS service principal key from any Kerberos system, use the `kadmin` command.

```
kdc1 % /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin: cpw -randkey krbtgt/EXAMPLE.COM@EXAMPLE.COM
Enter TGS key: xxxxxxxx
Enter new TGS key:      /** Type strong password **/
Re-enter TGS key to verify: xxxxxxxx
```

`cpw` is an alias for the `change_password` command. The `-randkey` option prompts you for the new password.

- If you are logged on to the KDC master as `root`, you can use the `kadmin.local` command. You are prompted for the new database password.

```
kdc1# kadmin.local -q 'cpw -randkey krbtgt/EXAMPLE.COM@EXAMPLE.COM'
```

Note - Save and store this password in a safe location.

Configuring KDC Servers on LDAP Directory Servers

In order to configure and build a master KDC server and secondary servers on LDAP, you have to create the LDAP back end and the Kerberos KDC, and then configure Kerberos and LDAP to recognize each other. This section shows how to configure three different LDAP back ends, OpenLDAP, Oracle Unified Directory (OUD), and Oracle Directory Server Enterprise Edition and connect them to the KDC. It also describes how to add Kerberos attributes to the LDAP people object class and how to destroy a Kerberos realm on an LDAP directory server.

TABLE 5 Task Map: Configuring Kerberos to Use LDAP

Task	Description	For Instructions
Manually configure a master KDC to use OpenLDAP.	Configures a KDC to use an OpenLDAP server as the LDAP back end.	"Configuring a Master KDC on an OpenLDAP Directory Server" on page 88
Manually configure a master KDC to use the Oracle Unified Directory (OUD).	Configures a KDC to use an OUD server as the LDAP back end.	"Configuring a Master KDC on an Oracle Unified Directory Server" on page 93
Manually configure the master KDC and Kerberos clients to use the Oracle Directory Server Enterprise Edition (DSEE).	Configures a KDC and a second master KDC to use an Oracle DSEE server as the LDAP back end.	"Configuring a Master KDC on an Oracle DSEE LDAP Directory Server" on page 99
Add Kerberos principals to an LDAP directory server.	Associates Kerberos attributes with the people object class in LDAP.	"How to Mix Kerberos Principal Attributes in a Non-Kerberos Object Class Type on an Oracle DSEE Server" on page 106
Destroy a realm on an LDAP directory server.	Removes all of the data associated with a realm.	"How to Destroy a Kerberos Realm on an LDAP Directory Server" on page 107

Configuring a Master KDC on an OpenLDAP Directory Server

By installing the KDC and OpenLDAP on the same server you get better performance.

The main steps involved in configuring the KDC and OpenLDAP on the same server are:

1. Installing the OpenLDAP package

2. Placing the OpenLDAP utilities in your path
3. Configuring the OpenLDAP server and protecting its directories
4. Starting the OpenLDAP daemon
5. Adding organizational entries to the OpenLDAP server
6. Adding the OpenLDAP server to the KDC configuration file
7. Creating the Kerberos database
8. Adding KDC roles and `kadmin` roles to the OpenLDAP server
9. Creating the Kerberos database keys
10. Synchronizing the master KDC's clock with the clock synchronization server
11. Enabling the KDC and `kadmin` services

▼ How to Configure a Master KDC on an OpenLDAP Directory Server

This procedure configures a KDC master and an OpenLDAP server on the same system. The KDC uses the OpenLDAP client library, as will the Kerberos clients that you configure later.

Before You Begin Make sure the system is configured to use DNS. This procedure uses the OpenLDAP for LDAP. For more information, see OpenLDAP Home Page.

You are in the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Install the `openldap` package.**

```
# pkg install library/openldap
```

2. **Add the OpenLDAP utilities to your path.**

```
# export PATH="/usr/lib/openldap/bin:$PATH"
```

3. **Start configuring the OpenLDAP server by editing its configuration file, `slapd.conf`.**

Change the domain and add the schemas, access, and index information.

```
# pfedit /etc/openldap/slapd.conf
include         /etc/openldap/schema/core.schema
include         /etc/openldap/schema/cosine.schema
include         /etc/openldap/schema/inetorgperson.schema
include         /etc/openldap/schema/nis.schema
include         /etc/openldap/schema/kerberos.schema

...
```

```
access to dn.base=""
    by * read

access to dn.base="cn=Subschema"
    by * read

access to attrs=userPassword,userPKCS12
    by self write
    by * auth

access to attrs=shadowLastChange
    by self write
    by * read

# Providing access to realm container
access to dn.subtree="cn=EXAMPLE.COM,cn=krbcontainer,dc=example,dc=com"
    by dn.exact="cn=kdc service,dc=example,dc=com" read
    by dn.exact="cn=kadmin service,dc=example,dc=com" write
    by * none

# Providing access to principals, if not underneath realm container
access to dn.subtree="ou=users,dc=example,dc=com"
    by dn.exact="cn=kdc service,dc=example,dc=com" read
    by dn.exact="cn=kadmin service,dc=example,dc=com" write
    by * none

access to *
    by * read

...

index   krbPrincipalName

...
```

4. **Change permissions on OpenLDAP directories.**

```
# chown -R openldap:openldap /var/openldap
# chown openldap:openldap /etc/openldap/slapd.conf
# mkdir /var/openldap/openldap-data
# chown openldap:openldap /var/openldap/openldap-data
```

5. **Remove `dn` and `changetype`, and add extra lines to the `kerberos.ldif` file, then copy it to the `kerberos.schema` file.**

```
# pfedit /usr/share/lib/ldif/kerberos.ldif
# cp /usr/share/lib/ldif/kerberos.ldif /etc/openldap/schema/kerberos.schema
```

6. **Start the OpenLDAP daemon with specific arguments to support domain sockets.**

```
# /usr/lib/slapd -u openldap -g openldap -f /etc/openldap/slapd.conf \
   -h "ldapi:/// ldaps:///"
```

The OpenLDAP daemon listens to port 389. By default, the domain socket location is `/var/openldap/run/ldapi`.

7. Specify the domain socket location for the client in the `/etc/openldap/ldap.conf` file.

```
# pfedit /etc/openldap/ldap.conf
URI     ldapi:/var/openldap/run/ldapi
```

8. Verify that the `slapd` daemon is using IPC to listen to domain sockets.

```
# /usr/lib/openldap/bin/ldapsearch -Y EXTERNAL -D "cn=directory manager" -b "" \
   -s base '(objectclass=*)' namingContexts
```

9. Add organizational entries to the OpenLDAP server.

```
# pfedit entries.ldif
dn: dc=example,dc=com
objectClass: dcObject
objectClass: organization
o: example.com
dc: example

dn: ou=groups,dc=example,dc=com
objectClass: top
objectClass: organizationalunit
ou: groups

dn: ou=users,dc=example,dc=com
objectClass: top
objectClass: organizationalunit
ou: users

# ldapadd -D "cn=directory manager,dc=example,dc=com" -W -f entries.ldif
```

10. Add the OpenLDAP server to the Kerberos configuration file.

```
# pfedit /etc/krb5/krb5.conf
 [realms]
        EXAMPLE.COM = {
                kdc = krb1.example.com
                admin_server = krb1.example.com
                database_module = LDAP
        }

 [dbmodules]
        LDAP = {
```

```
                db_library = kldap
                ldap_kerberos_container_dn = "cn=krbcontainer,dc=example,dc=com"
                ldap_kdc_dn = "cn=kdc service,ou=profile,dc=example,dc=com"
                ldap_kadmind_dn = "cn=kadmin service,ou=profile,dc=example,dc=com"
                ldap_cert_path = /var/ldap
                ldap_servers = ldapi:///
        }
  ...
```

11. Create the LDAP entries in the Kerberos database.

```
# kdb5_ldap_util -D "cn=directory manager,dc=example,dc=com" create \
   -subtrees ou=users,dc=example,dc=com -r EXAMPLE.COM -s
```

For more information, see the kdb5_ldap_util(1M) man page.

12. Add Kerberos entries to the OpenLDAP configuration.

a. Create and add the initial profile.

```
# pfedit profile.ldif
dn: ou=profile,dc=example,dc=com
ou: profile
objectclass: top
objectclass: organizationalUnit

# ldapadd -D "cn=directory manager,dc=example,dc=com" -W -f profile.ldif
```

b. Create and add KDC roles and `kadmin` roles.

```
# pfedit kdc_roles.ldif
 dn: cn=kdc service,ou=profile,dc=example,dc=com
 cn: kdc service
 sn: kdc service
 objectclass: top
 objectclass: person
 userpassword: nnnnnnnn

 dn: cn=kadmin service,ou=profile,dc=example,dc=com
 cn: kadmin service
 sn: kadmin service
 objectclass: top
 objectclass: person
 userpassword: nnnnnnnn

# ldapadd -D "cn=directory manager,dc=example,dc=com" -W -f kdc_roles.ldif
```

13. Create stash files for LDAP binding to the KDC and `kadmin` services.

```
# kdb5_ldap_util -D "cn=directory manager,dc=example,dc=com" stashsrvpw \
   cn="kdc service,ou=profile,dc=example,dc=com"
```

```
# kdb5_ldap_util -D "cn=directory manager,dc=example,dc=com" stashsrvpw \
   cn="kadmin service,ou=profile,dc=example,dc=com"
```

14. **Synchronize this system's clock with other clocks in the realm.**

    For authentication to succeed, every clock must be within the default time that is defined in the `libdefaults` section of the `krb5.conf` file. For more information, see the `krb5.conf`(4) man page and "Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139.

    Note - A master KDC cannot be the clock synchronization server.

    - **If you do not already have a clock synchronization server, perform this step after you configure one.**
    - **If you have a clock synchronization server, make this system a client of the server by following the PTP or NTP instructions.**
        - **To configure a PTP client, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
        - **To configure an NTP client, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**

15. **Enable the KDC and `kadmin` services.**

```
# svcadm enable krb5kdc
# svcadm enable kadmin
```

Configuring a Master KDC on an Oracle Unified Directory Server

By installing the KDC and LDAP on the same server you get better performance.

The main steps are:

1. Installing the OUD package
2. Configuring the OUD server
3. Adding the OUD server to the Kerberos configuration file
4. Creating keys for the KDC and specifying a privileged port for the OUD servers
5. Configuring KDC roles and services on the OUD server
6. Creating and installing a certificate and keys for the OUD server
7. Testing

8. Synchronizing the master KDC's clock with the clock synchronization server

▼ How to Configure a Master KDC on an Oracle Unified Directory LDAP Directory Server

This procedure configures a KDC master and an Oracle Unified Directory (OUD) server on the same system. The KDC uses the OpenLDAP client library, as will the Kerberos clients that you configure later.

Before You Begin Make sure the system is configured to use DNS. This procedure uses Oracle Unified Directory (OUD) for LDAP. For more information, see the Oracle Fusion Middleware Installation Guide for Oracle Unified Directory 11g Release 2 (11.1.2) .

You are in the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Download the OUD package.**
   Follow the directions on the Oracle Unified Directory web site.

2. **Configure the OUD LDAP server.**
   This configuration uses the following parameters:

   - Listener port: `1389`
   - TLS port: `1636` (privileged port)
   - Administrator connector port: `4444`
   - Password: *nnnnnnnn*
   - Certificates: StartTLS and TLS
   - Process: `java -server -Dorg.opends.server.scriptName=sta...`

   ```
   # cd Oracle/Middleware/Oracle_OUD1
   # export JAVA_HOME=/usr/jdk/instances/jdk1.6.0
   # ./oud-setup
   ```

3. **Verify that the LDAP server is listening.**

   ```
   # ldapsearch -p 1389 -D "cn=directory manager" -h $HOSTNAME -b "" -s base
    objectclass='*'
   ```

4. **Add the initial profile entries to the OUD configuration.**

   ```
   # pfedit profile.ldif
     dn: ou=profile,dc=example,dc=com
     ou: profile
     objectclass: top
     objectclass: organizationalUnit
   # ldapmodify -a -h $HOSTNAME -D "cn=directory manager" -f profile.ldif
   ```

5. **Remove the newlines from all the attribute types in the `kerberos.ldif` file, then add the file to the OUD configuration.**

```
# pfedit /usr/share/lib/ldif/kerberos.ldif
# ldapmodify -p 1389 -a -h $HOSTNAME -D "cn=directory manager" \
   -f /usr/share/lib/ldif/kerberos.ldif
```

6. **Add the OUD server to the Kerberos configuration file.**

```
# pfedit /etc/krb5/krb5.conf
 [realms]
        EXAMPLE.COM = {
                kdc = krb1.example.com
                admin_server = krb1.example.com
                database_module = LDAP
        }

 [dbmodules]
        LDAP = {
                db_library = kldap
                ldap_kerberos_container_dn = "cn=krbcontainer,dc=example,dc=com"
                ldap_kdc_dn = "cn=kdc service,ou=profile,dc=example,dc=com"
                ldap_kadmind_dn = "cn=kadmin service,ou=profile,dc=example,dc=com"
                ldap_cert_path = /var/ldap
                ldap_servers = ldap://krb1:1389
        }
  ...
```

7. **Create the keys and stash files for LDAP binding to the KDC and `kadmin` services.**

```
# kdb5_ldap_util -D "cn=directory manager" create -P nnnnnnnn -r EXAMPLE.COM -s
# kdb5_ldap_util stashsrvpw "cn=kdc service,ou=profile,dc=example,dc=com"
# kdb5_ldap_util stashsrvpw "cn=kadmin service,ou=profile,dc=example,dc=com"
```

8. **Modify the `ldap_servers` entry in the Kerberos configuration file to use a privileged port.**

```
# pfedit /etc/krb5/krb5.conf
       ldap_servers = ldaps://krb1:1636
```

9. **Add Kerberos entries to the OUD server.**

   a. **Create and add KDC roles.**

```
# pfedit kdc_roles.ldif
 dn: cn=kdc service,ou=profile,dc=example,dc=com
 cn: kdc service
 sn: kdc service
 objectclass: top
 objectclass: person
```

```
userpassword: nnnnnnnn

dn: cn=kadmin service,ou=profile,dc=example,dc=com
cn: kadmin service
sn: kadmin service
objectclass: top
objectclass: person
userpassword: nnnnnnnn
```

```
# ldapmodify -p 1389 -a -h $HOSTNAME -D "cn=directory manager" -f kdc_roles.ldif
```

b. Create and add administrative users.

```
# pfedit example.ldif
dn: dc=example,dc=com
changetype: modify
replace: aci
aci: (target ="ldap:///dc=example,dc=com")(targetattr !=
 "userPassword")(version 3.0;acl "Anonymous read-search access";
 allow (read, search, compare)(userdn = "ldap:///anyone");)
aci: (target="ldap:///dc=example,dc=com") (targetattr =
  "*")(version 3.0; acl "allow all Admin group"; allow(all) groupdn =
  "ldap:///cn=Directory Administrators,ou=Groups,dc=example,dc=com";)

dn: ou=Groups, dc=example,dc=com
objectclass: top
objectclass: organizationalunit
ou: Groups

dn: cn=Directory Administrators, ou=Groups, dc=example,dc=com
cn: Directory Administrators
objectclass: top
objectclass: groupofuniquenames
ou: Groups
uniquemember: uid=kvaughan, ou=People, dc=example,dc=com
uniquemember: uid=rdaugherty, ou=People, dc=example,dc=com
uniquemember: uid=hmiller, ou=People, dc=example,dc=com

dn: ou=People, dc=example,dc=com
objectclass: top
objectclass: organizationalunit
ou: People
aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr =
 "userpassword || telephonenumber || facsimiletelephonenumber")(version 3.0;
 acl "Allow self entry modification";allow (write)(userdn = "ldap:///self");)
aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr !=
 "cn || sn || uid")(targetfilter ="(ou=Accounting)")(version 3.0;
 acl "Accounting Managers Group Permissions";allow (write) (groupdn =
  "ldap:///cn=Accounting Managers,ou=groups,dc=example,dc=com");)
aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr !=
 "cn || sn || uid")(targetfilter ="(ou=Human Resources)")(version 3.0;
```

```
 acl "HR Group Permissions";allow (write)(groupdn = "ldap:///cn=HR Managers,
 ou=groups,dc=example,dc=com
 ");)
aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr !=
 "cn ||sn || uid")(targetfilter ="(ou=Product Testing)")(version 3.0;
 acl "QA Group Permissions";allow (write)(groupdn = "ldap:///cn=QA Managers,
 ou=groups,dc=example,dc=com");)
aci: (target ="ldap:///ou=People,dc=example,dc=com")(targetattr !=
 "cn || sn || uid")(targetfilter ="(ou=Product Development)")(version 3.0;
 acl "Engineering Group Permissions";allow (write)(groupdn = "ldap:///
 cn=PD Managers,ou=groups,dc=example,dc=com");)

dn: ou=Special Users,dc=example,dc=com
objectclass: top
objectclass: organizationalUnit
ou: Special Users
description: Special Administrative Accounts

# ldapmodify -p 1389 -a -h $HOSTNAME -D "cn=directory manager" -f example.ldif
```

c. Create and add ACLs for LDAP entries.

```
# pfedit kadmin.aci
## Set kadmin ACL for everything under krbcontainer.
dn: cn=krbcontainer,dc=example,dc=com
changetype: modify
replace: aci
aci: (target="ldap:///cn=krbcontainer,dc=example,dc=com") (targetattr="*")
 (version 3.0; acl "kadmin_ACL"; allow (all)
 userdn="ldap:///cn=kadmin service,ou=profile,dc=example,dc=com";)

## Set kadmin ACL for everything under the people subtree if there are
## mix-in entries for krb princs:
dn: ou=people,dc=example,dc=com
changetype: modify
replace: aci
aci: (target="ldap:///ou=people,dc=example,dc=com") (targetattr="*")
 (version 3.0; acl "kadmin_ACL"; allow (all)
 userdn="ldap:///cn=kadmin service,ou=profile,dc=example,dc=com";)

# ldapmodify -h $HOSTNAME -D "cn=directory manager" -f kadmin.aci
```

10. Generate and store the TLS certificate for the OUD server.

This set of commands also creates the key manager provider, trust manager provider, and connection handler.

```
# export LDAPHOME=~/Oracle/Middleware/asinst_8/OUD
# export LDAPHOME=$PWD
# export LDAP_SERVER_DN=krb1.example.com
# export STORE_PASSWD=xxxxxxxx
```

```
# export LDAP_BINDPWF=$LDAPHOME/config/keystore.pin
# export LDAP_ADMIN_PORT=4444
# export LDAP_BINDDN="cn=directory manager"
# export LDAP_SERVER=krb1.example.com
# rm $LDAPHOME/config/keystore
# rm $LDAPHOME/config/truststore
# echo $STORE_PASSWD > LDAP_BINDPWF
# keytool -genkeypair -alias server-cert -keyalg rsa \
   -dname "CN=$LDAP_SERVER_DN" -keystore $LDAPHOME/config/keystore \
   -storepass $STORE_PASSWD -keypass $STORE_PASSWD
# keytool -selfcert -alias server-cert -validity 1825 \
   -keystore $LDAPHOME/config/keystore -storetype JKS -storepass $STORE_PASSWD
# keytool -list -alias server-cert -keystore $LDAPHOME/config/keystore \
   -storepass $STORE_PASSWD
# keytool -exportcert -alias server-cert -file $LDAPHOME/config/server-cert.txt \
   -rfc -keystore $LDAPHOME/config/keystore -storepass $STORE_PASSWD
# cp $LDAPHOME/config/server-cert.txt /var/ldap/certdb.pem
```

11. **Enable the key manager provider, trust manager provider, and connection handler.**

```
# dsconfig -X -n -h $LDAP_SERVER -p $LDAP_ADMIN_PORT -D "$LDAP_BINDDN" \
  -j $LDAP_BINDPWF set-connection-handler-prop --handler-name "LDAPS Connection Handler"
 \
  --set key-manager-provider:JKS --set trust-manager-provider:JKS \
  --set listen-port:1636 --set enabled:true
# bin/stop-ds
```

12. **(Optional) Verify the configuration with an SSL LDAP query.**

```
# /usr/lib/openldap/bin/ldapsearch -v -x -D "$LDAP_BINDDN" -w $LDAP_BINDPW \
   -H ldaps://$LDAP_SERVER_DN:1636 -b "" -s base objectclass='*'
```

13. **Synchronize this system's clock with other clocks in the realm.**

    For authentication to succeed, every clock must be within the default time that is defined in the `libdefaults` section of the `krb5.conf` file. For more information, see the `krb5.conf`(4) man page and "Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139.

    Note - A master KDC cannot be the clock synchronization server.

    - **If you do not already have a clock synchronization server, perform this step after you configure one.**
    - **If you have a clock synchronization server, make this system a client of the server by following the PTP or NTP instructions.**

- **To configure a PTP client, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
- **To configure an NTP client, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**

Configuring a Master KDC on an Oracle DSEE LDAP Directory Server

By installing the KDC and LDAP on the same server you get better performance. In this section you create two master KDCs, called a *multi-master* configuration. Each master KDC is on an LDAP server.

The main steps are:

1. Configuring the first master Oracle DSEE server, including its certificates, profiles, and Kerberos schema
2. Configuring the first master KDC
3. Configuring the second Oracle DSEE server
4. Configuring the second master KDC
5. Synchronizing the master KDC's clock with the clock synchronization server
6. Enabling the Kerberos services

▼ How to Configure a Master KDC on an Oracle DSEE LDAP Directory Server for OpenLDAP Clients

This procedure configures two KDC masters and two LDAP servers. The first LDAP server and the first KDC master are on the same system, and the second LDAP server and the second KDC master are on a different system.

This procedure uses the following configuration parameters:

- LDAP instance = `/local/dsInst`
- LDAP port = `389 (636: secured)`
- KDC/LDAP first master server = `kdc1.example.com`
- KDC/LDAP second master server = `kdc2.example.com`
- Kerberos realm name = `EXAMPLE.COM`
- DNS domain name = `example.com`
- `admin` principal on first master server= `dav/admin`

- `admin` principal on second master server= `tester/admin`

Before You Begin The systems are configured to use DNS. By installing LDAP and KDC on the same server you get better performance. This procedure uses the Oracle Directory Server Enterprise Edition (DSEE) for LDAP. For more information, see Oracle Fusion Middleware Installation Guide for Oracle Directory Server Enterprise Edition 11g Release 1 (11.1.1.7.0) .

You are in the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. Configure the first Oracle DSEE LDAP server.

The LDAP server uses TLS certificates.

a. Create the server instance and create and add its certificate.

```
# dsadm create -p 389 -P 636 /local/dsInst
# dsadm start /local/dsInst
# dsconf create-suffix -p 389 -e dc=example,dc=com
# dsconf import -p 389 -e /opt/dsee7/resources/ldif/Example.ldif dc=example,dc=com
# ldapsearch -b "dc=example,dc=com" -s base '(objectclass=*)'
# dsadm show-cert -F der /local/dsInst defaultCert > /tmp/defaultCert.cert.der
# pktool setpin keystore=nss dir=/var/ldap
# chmod a+r /var/ldap/*.db
# pktool import keystore=nss objtype=cert trust="CT" \
   infile=/tmp/defaultCert.cert.der label=defaultCert dir=/var/ldap
# certutil -L -d /var/ldap -n defaultCert -a > /var/ldap/certdb.pem
```

b. Add organizational information to the `ldap.conf` file.

```
# pfedit /etc/openldap/ldap.conf
  ...
  #BASE   dc=example,dc=com
  #URI    ldap://ldap.example.com ldap://ldap-master.example.com:666

  #SIZELIMIT      12
  #TIMELIMIT      15
  #DEREF          never

  TLS_CACERT      /var/ldap/certdb.pem
  ...
```

c. Verify that the configured Oracle DSEE server works.

```
# ldapsearch -Z -P /var/ldap -D "cn=directory manager" \
   -h $HOSTNAME -b "" -s base objectclass='*'
```

d. Add the initial profile.

```
# pfedit profile.ldif
```

```
  dn: ou=profile,dc=example,dc=com
  ou: profile
  objectclass: top
  objectclass: organizationalUnit
# ldapmodify -a -h $HOSTNAME -D "cn=directory manager" -f profile.ldif
```

e. Add the `kerberos.ldif` file to the DSEE server.

```
# ldapmodify -h $HOSTNAME -D "cn=directory manager" \
   -f /usr/share/lib/ldif/kerberos.ldif
```

2. Configure the first master KDC and the Oracle DSEE server.

a. Add the Oracle DSEE server to the Kerberos configuration file.

```
# pfedit /etc/krb5/krb5.conf
 [realms]
        EXAMPLE.COM = {
                kdc = kdc1.example.com
                admin_server = kdc1.example.com
                database_module = LDAP
        }

 [dbmodules]
        LDAP = {
                db_library = kldap
                ldap_kerberos_container_dn = "cn=krbcontainer,dc=example,dc=com"
                ldap_kdc_dn = "cn=kdc service,ou=profile,dc=example,dc=com"
                ldap_kadmind_dn = "cn=kadmin service,ou=profile,dc=example,dc=com"
                ldap_cert_path = /var/ldap
                ldap_servers = ldaps://kdc1
        }
  ...
```

b. Create the master KDC and its keys.

```
# kdb5_ldap_util -D "cn=directory manager" create -P nnnnnnnn -r EXAMPLE.COM -s
# kdb5_ldap_util stashsrvpw "cn=kdc service,ou=profile,dc=example,dc=com"
# kdb5_ldap_util stashsrvpw "cn=kadmin service,ou=profile,dc=example,dc=com"
```

c. Create and add the KDC and `kadmin` roles to the Oracle DSEE server.

```
# pfedit kdc_roles.ldif
 dn: cn=kdc service,ou=profile,dc=example,dc=com
 cn: kdc service
 sn: kdc service
 objectclass: top
 objectclass: person
 userpassword: nnnnnnnn
```

```
 dn: cn=kadmin service,ou=profile,dc=example,dc=com
 cn: kadmin service
 sn: kadmin service
 objectclass: top
 objectclass: person
 userpassword: nnnnnnnn

# ldapmodify -a -h $HOSTNAME -D "cn=directory manager" -f kdc_roles.ldif
```

d. Create and add ACLs for Kerberos administrators to the Oracle DSEE server.

```
# pfedit kadmin.aci
 ## Set kadmin ACL for everything under krbcontainer.
 dn: cn=krbcontainer,dc=example,dc=com
 changetype: modify
 replace: aci
 aci: (target="ldap:///cn=krbcontainer,dc=example,dc=com")
  (targetattr="krb*")(version 3.0; acl kadmin_ACL; allow (all)
  userdn = "ldap:///cn=kadmin service,ou=profile,dc=example,dc=com";)

 ## Set kadmin ACL for everything under the people subtree if there are
 ## mix-in entries for krb princs:
 dn: ou=people,dc=example,dc=com
 changetype: modify
 replace: aci
 aci: (target="ldap:///ou=people,dc=example,dc=com")(targetattr="*")
  (version 3.0; acl kadmin_ACL; allow (all)
  userdn = "ldap:///cn=kadmin service,ou=profile,dc=example,dc=com";)

# ldapmodify -h $HOSTNAME -D "cn=directory manager" -f kadmin.aci
```

e. Add the location of the Kerberos configuration file to the KDC configuration file.

```
# pfedit /etc/krb5/kdc.conf
 ...
 [realms]
        EXAMPLE.COM = {
                profile = /etc/krb5/krb5.conf
 ...
```

f. Synchronize this system's clock with other clocks in the realm.

For authentication to succeed, every clock must be within the default time that is defined in the `libdefaults` section of the `krb5.conf` file. For more information, see the `krb5.conf`(4) man page and “Synchronizing Clocks Between KDCs and Kerberos Clients” on page 139.

Note - A master KDC cannot be the clock synchronization server.

- **If you do not already have a clock synchronization server, perform this step after you configure one.**
- **If you have a clock synchronization server, make this system a client of the server by following the PTP or NTP instructions.**
  - **To configure a PTP client, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
  - **To configure an NTP client, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**

g. Enable the KDC and `kadmin` services.

```
# svcadm enable krb5kdc
# svcadm enable kadmin
```

3. Configure the second LDAP server.

You update the first LDAP server, `kdc1`, while configuring the second server, `kdc2`.

a. Create a replica of the first LDAP server.

```
kdc1# dsconf enable-repl -h $HOSTNAME -p 389 -d 1 master dc=example,dc=com
```

b. Configure and enable the replica.

```
kdc2# mkdir -p /local
kdc2# dsadm create -p 389 -P 636 /local/dsInst
kdc2# dsadm start /local/dsInst
kdc2# dsconf create-suffix -p 389 -e dc=example,dc=com
kdc2# dsconf enable-repl -h $HOSTNAME -p 389 -d 2 master dc=example,dc=com
```

c. Enable the LDAP servers to recognize each other.

```
kdc1# dsconf create-repl-agmt -h $HOSTNAME -p 389 dc=example,dc=com kdc2:389
kdc2# dsconf create-repl-agmt -h $HOSTNAME -p 389 dc=example,dc=com kdc1:389
```

4. Verify that the first LDAP server cannot modify the second LDAP server.

On a properly configured LDAP server, the second server queries should report a status of `NOT OK`.

```
kdc1# dsconf show-repl-agmt-status -h $HOSTNAME -p 389 dc=example,dc=com kdc2:389
kdc1# dsconf accord-repl-agmt -h $HOSTNAME -p 389 dc=example,dc=com kdc2:389
kdc1# dsconf init-repl-dest -h $HOSTNAME -p 389 dc=example,dc=com kdc2:389
```

5. Create a stash file for LDAP binding and create the master KDC's stash file.

```
kdc1# kdb5_ldap_util stashsrvpw "cn=kdc service,ou=profile,dc=example,dc=com"
kdc1# kdb5_util stash
```

6. Configure the second KDC master.

The administrative principal is `tester/admin`. The other parameters remain the same. After configuring the LDAP TLS bindings on the second KDC, update the `/etc/krb5/krb5.conf` file.

Note - If the KDC and LDAP servers are on different systems, then the `ldap_servers` keyword in the `krb5.conf` file must list the LDAP server names.

a. Edit the Kerberos configuration file on the second KDC master.

```
kdc2# pfedit /etc/krb5/krb5.conf
 [realms]
        EXAMPLE.COM = {
                kdc = kdc2.example.com
                kdc = kdc1.example.com
                admin_server = kdc1.example.com
                admin_server = kdc2.example.com
                database_module = LDAP
        }

 [dbmodules]
        LDAP = {
                db_library = kldap
                ldap_kerberos_container_dn = "cn=krbcontainer,dc=example,dc=com"
                ldap_kdc_dn = "cn=kdc service,ou=profile,dc=us,dc=example,dc=com"
                ldap_kadmind_dn = "cn=kadmin service,ou=profile,dc=example,dc=com"
                ldap_cert_path = /var/ldap
                ldap_servers = ldaps://kdc2 ldaps://kdc1
        }
 ...
```

b. Create a stash file for LDAP binding to the KDC and the `kadmin` services, and create the master KDC's stash file.

```
kdc2# kdb5_ldap_util stashsrvpw "cn=kdc service,ou=profile,dc=example,dc=com"
kdc2# kdb5_ldap_util stashsrvpw "cn=kadmin service,ou=profile,dc=us,dc=oracle,
dc=com"
kdc2# kdb5_util stash
```

c. Add the location of the Kerberos configuration file to the KDC configuration file.

```
kdc2# pfedit /etc/krb5/kdc.conf
```

```
...
[realms]
        EXAMPLE.COM = {
                profile = /etc/krb5/krb5.conf
...
```

d. Create keys for the principals of the second KDC master.

```
kdc2# kadmin -p tester/admin
kadmin: addprinc -randkey -allow_tgs_req kadmin/kdc2.example.com
kadmin: modprinc -requires_preauth kadmin/kdc2.example.com
kadmin: addprinc -randkey -allow_tgs_req +password_changing_service changepw/kdc2.
example.com
kadmin: modprinc -requires_preauth changepw/kdc2.example.com
```

7. Synchronize this system's clock with other clocks in the realm.

For authentication to succeed, every clock must be within the default time that is defined in the `libdefaults` section of the `krb5.conf` file. For more information, see the `krb5.conf`(4) man page and "Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139.

Note - A master KDC cannot be the clock synchronization server.

- **If you do not already have a clock synchronization server, perform this step after you configure one.**
- **If you have a clock synchronization server, make this system a client of the server by following the PTP or NTP instructions.**
  - **To configure a PTP client, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
  - **To configure an NTP client, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**

8. Enable the KDC and `kadmin` services.

```
kdc2# svcadm enable krb5kdc
kdc2# svcadm enable kadmin
```

How to Mix Kerberos Principal Attributes in a Non-Kerberos Object Class Type on an Oracle DSEE Server

In this procedure, the `krbprincipalaux`, and `krbTicketPolicyAux` and `krbPrincipalName` attributes are associated with the people object class.

This procedure uses the following configuration parameters:

- Oracle DSEE Server = `dsserver.example.com`
- User principal = `mre@EXAMPLE.COM`

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Prepare each entry in the people object class.**

   On the Oracle DSEE server, repeat this step for each entry.

   ```
   cat << EOF | ldapmodify -h dsserver.example.com -D "cn=directory manager"
   dn: uid=mre,ou=people,dc=example,dc=com
   changetype: modify
   objectClass: krbprincipalaux
   objectClass: krbTicketPolicyAux
   krbPrincipalName: mre@EXAMPLE.COM
   EOF
   ```

2. **Add a subtree attribute to the realm container.**

   This example enables searching principal entries in the `ou=people,dc=example,dc=com` container, as well as in the default `EXAMPLE.COM` container.

   ```
   # kdb5_ldap_util -D "cn=directory manager" modify \
      -subtrees 'ou=people,dc=example,dc=com' -r EXAMPLE.COM
   ```

3. **(Optional) If the KDC records are stored in DB2, migrate the DB2 entries.**

   a. **Dump the DB2 entries.**

      ```
      # kdb5_util dump > dumpfile
      ```

   b. **Load the database into the LDAP server.**

      ```
      # kdb5_util load -update dumpfile
      ```

4. **(Optional) Add the principal attributes to the KDC.**

   ```
   # kadmin.local -q 'addprinc mre'
   ```

▼ How to Destroy a Kerberos Realm on an LDAP Directory Server

Use this procedure if a different LDAP Directory Server is handling a realm.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

- **Destroy the realm.**

```
# kdb5_ldap_util -D "cn=directory manager" destroy
```

Configuring Kerberos Clients

Kerberos clients include any host on the network that is not a KDC server and that needs to use Kerberos services. This section provides procedures for installing a Kerberos client, as well as information about using `root` authentication to mount NFS file systems.

Client configuration options are similar to the server options, with the addition of the Automated Installer (AI):

- AI – Recommended for quick and easy installation of multiple Kerberos clients
- Automatic – Recommended for scripts
- Interactive – Sufficient for most installations
- Manual – Necessary for more complex installations

The following task map describes the tasks that are covered in this section.

TABLE 6 Task Map: Configuring Kerberos Clients

Task	Description	For Instructions
Install clients by using the Automated Installer (AI).	Appropriate when you want the Kerberos client to be configured during system installation.	"How to Configure Kerberos Clients Using AI" in *Installing Oracle Solaris 11.3 Systems*
Create an installation profile for similar Kerberos clients.	Creates a reusable client installation profile.	"How to Create a Kerberos Client Installation Profile" on page 108
Install clients with a script.	Appropriate when the installation parameters for each client are the same.	"How to Automatically Configure a Kerberos Client" on page 108
Install clients by answering prompts.	Appropriate when only a few of the installation parameters need to change.	"How to Interactively Configure a Kerberos Client" on page 110
Install clients manually.	Appropriate when each client installation requires unique installation parameters.	"How to Manually Configure a Kerberos Client" on page 114
Join a Kerberos client to an Active Directory Server.	Automatically installs a Kerberos client of an Active Directory server.	"How to Join a Kerberos Client to an Active Directory Server" on page 113

Task	Description	For Instructions
Disable verification of the KDC that issued a client Ticket Granting Ticket (TGT).	Streamlines KDC verification when Kerberos clients do not have a host principal stored in the local keytab file.	"Disabling Verification of the Ticket-Granting Ticket" on page 119
Enable a client to access an NFS file system as the `root` user	Enables the client to mount an NFS file system with `root` access. Also, enables the client to access the NFS file system so that cron jobs can run.	"How to Access a Kerberos Protected NFS File System as the `root` User" on page 120

How to Create a Kerberos Client Installation Profile

This procedure creates a kclient profile that can be used when you install a Kerberos client. By using the profile, you reduce the likelihood of typing errors. Also, using the profile reduces user intervention as compared to the interactive process.

Note - To create systems that initially boot as fully configured Kerberos clients, see "How to Configure Kerberos Clients Using AI" in *Installing Oracle Solaris 11.3 Systems*.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Create a `kclient` installation profile.**

   The following is a sample `kclient` profile:

```
client# pfedit kcprofile
REALM EXAMPLE.COM
KDC kdc1.example.com
ADMIN clntconfig
FILEPATH /net/denver.example.com/export/install/krb5.conf
NFS 1
DNSLOOKUP none
```

2. **Protect the file and store it for use by other clients.**

```
client# cp kcprofile /net/denver.example.com/export/install
denver# chown root kcprofile; chmod 644 kcprofile
```

How to Automatically Configure a Kerberos Client

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Create a kclient profile.**

Use the installation profile from "How to Create a Kerberos Client Installation Profile" on page 108.

2. **Run the `kclient` command with a profile argument.**

   You must provide the password for the `clntconfig` principal to complete the process. You created this password when you configured the master KDC in "Configuring KDC Servers" on page 69. For more information, see the `kclient`(1M) man page.

```
client# /usr/sbin/kclient -p /net/denver.example.com/export/install/kcprofile

Starting client setup
-----------------------------------------------------

kdc1.example.com

Setting up /etc/krb5/krb5.conf.

Obtaining TGT for clntconfig/admin ...
Password for clntconfig/admin@EXAMPLE.COM: xxxxxxxx

nfs/client.example.com entry ADDED to KDC database.
nfs/client.example.com entry ADDED to keytab.

host/client.example.com entry ADDED to KDC database.
host/client.example.com entry ADDED to keytab.

Copied /net/denver.example.com/export/install/krb5.conf.

-----------------------------------------------------
Setup COMPLETE.

client#
```

3. **Synchronize this system's clock with other clocks in the realm.**

   For authentication to succeed, every clock must be within the default time that is defined in the `libdefaults` section of the `krb5.conf` file.

   - **To configure PTP, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
   - **To configure NTP, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**

Example 7 Using an Installation Profile to Configure a Kerberos Client

The following example uses the `kcprofile` client profile and two command-line overrides to configure the client.

```
# /usr/sbin/kclient -p /net/denver.example.com/export/install/kcprofile \
-d dns_fallback -k kdc2.example.com

Starting client setup
-------------------------------------------------

kdc1.example.com

Setting up /etc/krb5/krb5.conf.

Obtaining TGT for clntconfig/admin ...
Password for clntconfig/admin@EXAMPLE.COM:  xxxxxxxx

nfs/client.example.com entry ADDED to KDC database.
nfs/client.example.com entry ADDED to keytab.

host/client.example.com entry ADDED to KDC database.
host/client.example.com entry ADDED to keytab.

Copied /net/denver.example.com/export/install/krb5.conf.

-------------------------------------------------
Setup COMPLETE.

client#
```

▼ How to Interactively Configure a Kerberos Client

This procedure uses the `kclient` installation utility without an installation profile. If the client is to join an Active Directory server, go to "How to Join a Kerberos Client to an Active Directory Server" on page 113.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. Run the `kclient` command with no arguments.

```
client# /usr/sbin/kclient
```

The script prompts you for the following information:

- Kerberos realm name
- KDC master host name

- KDC slave host names
- Domains to map to the local realm
- PAM service names and options to use for Kerberos authentication

For more information, see the `kclient(1M)` man page.

2. **If the KDC server is not running an Oracle Solaris release, answer `y` and define the type of server that is running the KDC.**

   For the list of available servers, see the `-T` option in the `kclient(1M)` man page.

3. **If DNS should be used for Kerberos lookups, answer `y` and indicate the DNS lookup option to use.**

   Valid options are `dns_lookup_kdc`, `dns_lookup_realm`, and `dns_fallback`. For more information about these values, see the `krb5.conf(4)` man page.

4. **Define the name of the Kerberos realm and the master KDC host name.**

   This information is added to the `/etc/krb5/krb5.conf` configuration file.

5. **If slave KDCs are in the realm, answer `y` and provide the slave KDC host names.**

   This information is used to create additional KDC entries in the client's configuration file.

6. **If service or host keys are required, answer `y`.**

   Typically, service or host keys are not required unless the client system is hosting Kerberized services.

7. **If the client is a member of a cluster, answer `y` and provide the logical name of the cluster.**

   The logical host name is used when creating service keys, which is required when hosting Kerberos services from clusters.

8. **Identify any domains or hosts to map to the current realm.**

   This mapping allows other domains to belong in the default realm of the client.

9. **Specify whether the client will use Kerberized NFS.**

   NFS service keys need to be created if the client will host NFS services using Kerberos.

10. **Indicate whether a new PAM policy needs to be created.**

    To set which PAM services use Kerberos for authentication, you provide the service name and a flag that indicates how Kerberos authentication is to be used. The valid flag options are:

    - `first` – Use Kerberos authentication first, and only use UNIX if Kerberos authentication fails

- `only` – Use Kerberos authentication only
- `optional` – Use Kerberos authentication optionally

For information about provided PAM services for Kerberos, review the files in `/etc/security/pam_policy`.

11. Specify whether the master `/etc/krb5/krb5.conf` file should be copied.

This option enables specific configuration information to be used when the arguments to `kclient` are not sufficient.

Example 8 Sample Run of the `kclient` Script

```
...
Starting client setup
--------------------------------------------------

Is this a client of a non-Solaris KDC ? [y/n]: n
No action performed.
Do you want to use DNS for kerberos lookups ? [y/n]: n
No action performed.
Enter the Kerberos realm: EXAMPLE.COM
Specify the KDC host name for the above realm: kdc1.example.com

Note, this system and the KDC's time must be within 5 minutes of each other for
Kerberos to function. Both systems should run some form of time synchronization
system like Network Time Protocol (NTP).
Do you have any slave KDC(s) ? [y/n]: y
Enter a comma-separated list of slave KDC host names: kdc2.example.com

Will this client need service keys ? [y/n]: n
No action performed.
Is this client a member of a cluster that uses a logical host name ? [y/n]: n
No action performed.
Do you have multiple domains/hosts to map to realm ? [y/n]: y
Enter a comma-separated list of domain/hosts to map to the default realm: corphdqtrs.
example.com, \
example.com

Setting up /etc/krb5/krb5.conf.

Do you plan on doing Kerberized nfs ? [y/n]: y
Do you want to update /etc/pam.conf ? [y/n]: y
Enter a comma-separated list of PAM service names in the following format:
service:{first|only|optional}: xscreensaver:first
Configuring /etc/pam.conf.

Do you want to copy over the master krb5.conf file ? [y/n]: n
No action performed.
```

```
-------------------------------------------------
Setup COMPLETE.
```

How to Join a Kerberos Client to an Active Directory Server

This procedure uses the `kclient` command without an installation profile.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. (Optional) Enable DNS resource record creation for the client.

```
client# sharectl set -p ddns_enable=true smb
```

2. Run the `kclient` command.

The following output shows sample output from running the `kclient` command to join the client to the AD domain, `EXAMPLE.COM`.

The `-T` option selects a KDC server type, in this case, a Microsoft Active Directory (AD) server type. By default, you must provide the password for the Administrator principal of the AD server.

```
client# /usr/sbin/kclient -T ms_ad
Starting client setup
-------------------------------------------------

Attempting to join 'CLIENT' to the 'EXAMPLE.COM' domain.
Password for Administrator@EXAMPLE.COM: xxxxxxxx
Forest name found: example.com
Looking for local KDCs, DCs and global catalog servers (SVR RRs).

Setting up /etc/krb5/krb5.conf

Creating the machine account in AD via LDAP.
-------------------------------------------------
Setup COMPLETE.
#
```

For more information, see the `kclient`(1M) man page.

Note - To automatically detect the client's Sites and Services in Active Directory (AD), you must configure an AD server on the same local subnet as the Kerberos client.

How to Manually Configure a Kerberos Client

This procedure uses the following configuration parameters:

- Realm name = `EXAMPLE.COM`
- DNS domain name = `example.com`
- Master KDC = `kdc1.example.com`
- Slave KDC = `kdc2.example.com`
- NFS server = `denver.example.com`
- Client = `client.example.com`
- `admin` principal = `kws/admin`
- User principal = `mre`
- Online help URL = `http://docs.oracle.com/cd/E23824_01/html/821-1456/aadmin-23.html`

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Edit the Kerberos configuration file, `krb5.conf`.**

   Change the realm names and the server names in the Kerberos configuration file. You can also specify the path to the help files for `gkadmin`.

```
kdc1# pfedit /etc/krb5/krb5.conf
[libdefaults]
default_realm = EXAMPLE.COM

[realms]
     EXAMPLE.COM = {
        kdc = kdc1.example.com
        kdc = kdc2.example.com
        admin_server = kdc1.example.com
     }

[domain_realm]
     .example.com = EXAMPLE.COM
     #
     # if the domain name and realm name are equivalent,
     # this entry is not needed
     #
[logging]
     default = FILE:/var/krb5/kdc.log
     kdc = FILE:/var/krb5/kdc.log

[appdefaults]
     gkadmin = {
     help_url = http://www.example.com/doclib/OSMKA/aadmin-23.html
```

Note - If you must communicate with an older Kerberos system, you might need to restrict the encryption types. For a description of the issues involved with restricting the encryption types, see "Kerberos Encryption Types" on page 50.

2. **(Optional) Change the process that is used to locate the KDCs.**

   By default, the Kerberos realm to KDC mapping is determined in the following order:

   - The definition in the `realms` section in `krb5.conf`
   - Looking up SRV records in DNS

   You can change this behavior by adding `dns_lookup_kdc` or `dns_fallback` to the `libdefaults` section of the `krb5.conf` file. For more information, see the `krb5.conf`(4). Note that referrals are always tried first.

3. **(Optional) Change the process used to determine the realm for a host.**

   By default the host to realm mapping is determined in the following order:

   - If the KDC supports referrals, then the KDC can inform the client which realm the host belongs to.
   - The definition of `domain_realm` in the `krb5.conf` file.
   - The DNS domain name of the host.
   - The default realm.

   You can change this behavior by adding `dns_lookup_kdc` or `dns_fallback` to the `libdefaults` section of the `krb5.conf`file. For more information, see the `krb5.conf`(4) man page. Note that referrals are always tried first.

4. **Synchronize this system's clock with other clocks in the realm.**

   For authentication to succeed, every clock must be within the default time that is defined in the `libdefaults` section of the `krb5.conf` file.

   - **To configure PTP, follow the instructions in "Managing the Precision Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**
   - **To configure NTP, follow the instructions in "Managing the Network Time Protocol" in *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.**

5. **Create Kerberos principals.**

```
denver # /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin:
```

For more information, see the kadmin(1M) man page.

a. (Optional) Create a user principal if a user principal does not already exist.

Create a user principal only if the user associated with this host does not already have a principal assigned.

```
kadmin: addprinc mre
Enter password for principal mre@EXAMPLE.COM:     /** Type strong password **/
Re-enter password for principal mre@EXAMPLE.COM: xxxxxxxx
kadmin:
```

b. (Optional) Create a `root` principal and add the principal to the server's keytab file.

Note - If the client does not require `root` access to a remote NFS-mounted file system, then you can skip this step.

If non-interactive `root` access is needed, such as running cron jobs as `root`, then perform this step.

The `root` principal should be a two-component principal. The second component should be the host name of the Kerberos client system to avoid the creation of a realm-wide `root` principal. When the principal instance is a host name, the FQDN must be specified in lowercase letters regardless of the case of the domain name in the name service.

```
kadmin: addprinc -randkey root/client.example.com
Principal "root/client.example.com" created.
kadmin: ktadd root/client.example.com
Entry for principal root/client.example.com with kvno 3, encryption type AES-256 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal root/client.example.com with kvno 3, encryption type AES-128 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal root/client.example.com with kvno 3, encryption type Triple DES
 cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin:
```

c. Create a `host` principal and add the principal to the server's keytab file.

The `host` principal is used by remote access services to provide authentication. The principal enables `root` to acquire a credential, if a credential is not already in the keytab file.

```
kadmin: addprinc -randkey host/denver.example.com
Principal "host/denver.example.com@EXAMPLE.COM" created.
```

```
kadmin: ktadd host/denver.example.com
Entry for principal host/denver.example.com with kvno 3, encryption type AES-256 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/denver.example.com with kvno 3, encryption type AES-128 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/denver.example.com with kvno 3, encryption type Triple DES
 cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin:
```

d. (Optional) Add the server's NFS service principal to the server's keytab file and quit `kadmin`.

This step is required only if the client needs to access NFS file systems using Kerberos authentication.

```
kadmin: ktadd nfs/denver.example.com
Entry for principal nfs/denver.example.com with kvno 3, encryption type AES-256 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal nfs/denver.example.com with kvno 3, encryption type AES-128 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal nfs/denver.example.com with kvno 3, encryption type Triple DES
 cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
```

6. (Optional) Enable Kerberos with NFS.

a. Enable Kerberos security modes in the `/etc/nfssec.conf` file.

In the `/etc/nfssec.conf` file, remove the "#" that comments out the Kerberos security modes.

```
# pfedit /etc/nfssec.conf
.
.
#
# Uncomment the following lines to use Kerberos V5 with NFS
#
krb5            390003  kerberos_v5     default -               # RPCSEC_GSS
krb5i           390004  kerberos_v5     default integrity       # RPCSEC_GSS
krb5p           390005  kerberos_v5     default privacy         # RPCSEC_GSS
```

b. Enable DNS.

If the `svc:/network/dns/client:default` service is not enabled, enable it. For more information, see the `resolv.conf(4)` man page.

```
# svcadm enable network/dns/client:default
```

c. **Restart the `gss` service.**

```
# svcadm restart network/rpc/gss
```

7. **(Optional) For the client to automatically renew the TGT or to warn users about Kerberos ticket expiration, create an entry in the `/etc/krb5/warn.conf` file.**

   For more information, see the `warn.conf(4)` man page and "Automatically Renewing All Ticket-Granting Tickets" on page 125.

Example 9 Configuring an Oracle Solaris Client to Work With a Multiple-Master KDC

The Microsoft Active Directory (AD) Kerberos service provides a KDC that runs on multiple master servers. For an Oracle Solaris client to update information, either the `admin_server` or the `kpasswd_server` declaration in the `/etc/krb5/krb5.conf` file must list all the servers. This example shows how to enable the client to update information about the KDC that `kdc1` and `kdc2` share.

```
[realms]
EXAMPLE.COM = {
kdc = kdc1.example.com
kdc = kdc2.example.com
admin_server = kdc1.example.com
admin_server = kdc2.example.com
}
```

Example 10 Configuring a Kerberos Client for a Non-Oracle Solaris KDC

A Kerberos client can be set up to work with a non-Oracle Solaris KDC, by adding a line to the `/etc/krb5/krb5.conf` file in the `realms` section. This line changes the protocol that is used when the client is communicating with the Kerberos password-changing server. The following excerpt shows the format of this line.

```
[realms]
EXAMPLE.COM = {
kdc = kdc1.example.com
kdc = kdc2.example.com
admin_server = kdc1.example.com
kpasswd_protocol = SET_CHANGE
}
```

Example 11 DNS TXT Records for the Mapping of Host and Domain Name to a Kerberos Realm

```
@ IN SOA kdc1.example.com root.kdc1.example.com (
1989020501   ;serial
```

```
10800        ;refresh
3600         ;retry
3600000      ;expire
86400 )      ;minimum

IN      NS      kdc1.example.com.
kdc1                    IN      A       192.146.86.20
kdc2                    IN      A       192.146.86.21

_kerberos.example.com.             IN      TXT     "EXAMPLE.COM"
_kerberos.kdc1.example.com.        IN      TXT     "EXAMPLE.COM"
_kerberos.kdc2.example.com.        IN      TXT     "EXAMPLE.COM"
```

Example 12 DNS SRV Records for Kerberos Server Locations

This example defines the records for the location of the KDCs, the `admin` server, and the `kpasswd` server, respectively.

```
@ IN SOA kdc1.example.com root.kdc1.example.com (
1989020501   ;serial
10800        ;refresh
3600         ;retry
3600000      ;expire
86400 )      ;minimum

IN      NS      kdc1.example.com.
kdc1                               IN      A       192.146.86.20
kdc2                               IN      A       192.146.86.21

_kerberos._udp.EXAMPLE.COM         IN      SRV 0 0 88  kdc2.example.com
_kerberos._tcp.EXAMPLE.COM         IN      SRV 0 0 88  kdc2.example.com
_kerberos._udp.EXAMPLE.COM         IN      SRV 1 0 88  kdc1.example.com
_kerberos._tcp.EXAMPLE.COM         IN      SRV 1 0 88  kdc1.example.com
_kerberos-adm._tcp.EXAMPLE.COM     IN      SRV 0 0 464 kdc1.example.com
_kpasswd._udp.EXAMPLE.COM          IN      SRV 0 0 464 kdc1.example.com
```

Disabling Verification of the Ticket-Granting Ticket

By default, Kerberos checks that the KDC of the host principal that is stored in the local `/etc/krb5/krb5.keytab` file is the same KDC that issued the ticket-granting ticket (TGT). This check, `verify_ap_req_nofail`, prevents DNS spoofing attacks.

However, this check must be disabled for client configurations where the host principal is unavailable. The following configurations require this check to be disabled:

- The client IP address is dynamically assigned, for example, a DHCP client.
- The client is not configured to host any services, so no host principal was created.
- The host key is not stored on the client.

To disable TGT verification, set the verify_ap_req_nofail option to false in the krb5.conf file. The verify_ap_req_nofail option can be entered in either the [libdefaults] or the [realms] section of the krb5.conf file. In the [libdefaults] section, the setting is used for all realms:

```
client # pfedit /etc/krb5/krb5.conf
[libdefaults]
default_realm = EXAMPLE.COM
verify_ap_req_nofail = false
...
```

If the option is in the [realms] section, the setting applies only to the defined realm. For more information about this option, see the krb5.conf(4) man page.

How to Access a Kerberos Protected NFS File System as the root User

This procedure allows a client to access an NFS file system that requires Kerberos authentication with the root ID privilege. In particular, when the NFS file system is shared with options like: -o sec=krb5,root=client1.example.com.

1. **Run the kadmin command.**

```
denver # /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin:
```

2. **Create a root principal for the NFS client.**

This principal is used to provide root equivalent access to NFS mounted file systems that require Kerberos authentication. The root principal should be a two-component principal. The second component should be the host name of the Kerberos client system to avoid the creation of a realm-wide root principal. Note that when the principal instance is a host name, the FQDN must be specified in lowercase letters regardless of the case of the domain name in the name service.

```
kadmin: addprinc -randkey root/client.example.com
Principal "root/client.example.com" created.
kadmin:
```

3. **Add the root principal to the server's keytab file and quit kadmin.**

This step is required for the client to have root access to file systems mounted using the NFS service. This step is also required for non-interactive root access, such as running cron jobs as root.

```
kadmin: ktadd root/client.example.com
Entry for principal root/client.example.com with kvno 3, encryption type AES-256 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal root/client.example.com with kvno 3, encryption type AES-128 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal root/client.example.com with kvno 3, encryption type Triple DES cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
```

How to Configure Automatic Migration of Users in a Kerberos Realm

Users who do not have a Kerberos principal can be automatically migrated to an existing Kerberos realm by using PAM. You customize per-system PAM configuration files on the migration server and the master server to handle the recognition of UNIX credentials and the re-authentication in the Kerberos realm.

For information about PAM, see Chapter 1, “Using Pluggable Authentication Modules” and the `pam.conf(4)` man page.

In this procedure, the login service names are configured to use automatic migration. This example uses the following configuration parameters:

- Realm name = `EXAMPLE.COM`
- Master KDC = `kdc1.example.com`
- Machine hosting the migration service = `server1.example.com`
- Migration service principal = `host/server1.example.com`

Before You Begin You must assume the `root` role. For more information, see “Using Your Assigned Administrative Rights” in *Securing Users and Processes in Oracle Solaris 11.3*.

1. Ensure that a host service principal for `server1` exists.

The host service principal in the `keytab` file of `server1` is used to authenticate the server to the master KDC.

```
server1 # klist -k
Keytab name: FILE:/etc/krb5/krb5.keytab
KVNO Principal
---- -------------------------------------------------
3 host/server1.example.com@EXAMPLE.COM
...
```

For information about the options to the command, see the `klist(1)` man page.

2. **If `server1` is not listed, configure it as a Kerberos client of the realm `EXAMPLE.COM`.**

   For the steps, see the examples in "Configuring Kerberos Clients" on page 107.

3. **Modify the PAM policy for `server1`.**

   For more information, see "Assigning a Per-User PAM Policy" on page 20.

   a. **Determine which Kerberos policy is in use on `server1`.**

   ```
   % grep PAM_POLICY /etc/security/policy.conf
   # PAM_POLICY specifies the system-wide PAM policy (see pam_user_policy(5))
   ...
   PAM_POLICY=krb5_first
   ```

   b. **Copy that PAM policy file, then modify the new policy file to append the `pam_krb5_migrate.so.1` module to each authentication stack.**

   ```
   server1 # cd /etc/security/pam_policy/; cp krb5_first krb5_firstmigrate
   server1 # pfedit /etc/security/pam_policy/krb5_firstmigrate.
   # login service (explicit because of pam_dial_auth)
   #
   login auth requisite    pam_authtok_get.so.1
   ...
   login auth required  pam_unix_auth.so.1
   login    auth optional  pam_krb5_migrate.so.1
   #
   # rlogin service (explicit because of pam_rhost_auth)
   #
   rlogin auth sufficient  pam_rhosts_auth.so.1
   ...
   rlogin auth required    pam_unix_auth.so.1
   rlogin   auth optional        pam_krb5_migrate.so.1
   #
   # Kerberized rlogin service
   #
   krlogin auth required  pam_unix_cred.so.1
   krlogin auth required  pam_krb5.so.1
   krlogin auth optional  pam_krb5_migrate.so.1
   #
   # rsh service (explicit because of pam_rhost_auth)
   #
   rsh auth sufficient  pam_rhosts_auth.so.1
   rsh auth required    pam_unix_cred.so.1
   rsh auth optional    pam_krb5_migrate.so.1
   #
   # Kerberized rsh service
   #
   krsh auth required  pam_unix_cred.so.1
   krsh auth required  pam_krb5.so.1
   krsh auth optional  pam_krb5_migrate.so.1
   ```

```
#
# Kerberized telnet service
#
ktelnet auth required  pam_unix_cred.so.1
ktelnet auth required  pam_krb5.so.1
ktelnet auth optional  pam_krb5_migrate.so.1
#
# PPP service (explicit because of pam_dial_auth)
#
ppp auth requisite    pam_authtok_get.so.1
...
ppp auth required  pam_unix_auth.so.1
ppp auth optional  pam_krb5_migrate.so.1
#
# GDM Autologin (explicit because of pam_allow).  These need to be
# here as there is no mechanism for packages to amend pam.conf as
# they are installed.
#
gdm-autologin auth  required    pam_unix_cred.so.1
gdm-autologin auth  sufficient  pam_allow.so.1
gdm-autologin auth optional    pam_krb5_migrate.so.1
#
# Default definitions for Authentication management
# Used when service name is not explicitly mentioned for authentication
#
OTHER auth requisite    pam_authtok_get.so.1
...
OTHER auth required  pam_unix_auth.so.1
OTHER auth optional  pam_krb5_migrate.so.1
#
# passwd command (explicit because of a different authentication module)
#
passwd auth required  pam_passwd_auth.so.1
#
# cron service (explicit because of non-usage of pam_roles.so.1)
#
cron account required pam_unix_account.so.1
#
# cups service (explicit because of non-usage of pam_roles.so.1)
#
cups account required pam_unix_account.so.1
#
# GDM Autologin (explicit because of pam_allow) This needs to be here
# as there is no mechanism for packages to amend pam.conf as they are
# installed.
#modified
gdm-autologin account  sufficient  pam_allow.so.1
#
.
.
.
```

c. (Optional) Force an immediate password change.

For the newly created Kerberos accounts, set the password expiration time to the current time by adding the `expire_pw` option to the `pam_krb5_migrate` entries. For more information, see the `pam_krb5_migrate(5)` man page.

```
service-name auth optional     pam_krb5_migrate.so.1 expire_pw
```

d. In this configuration file, modify the `OTHER` account stack to block access if the Kerberos password has expired.

```
# Definition for Account management
# Used when service name is not explicitly mentioned for account management
# Re-ordered pam_krb5 causes password expiration in Kerberos to block access
#
OTHER account requisite   pam_roles.so.1
OTHER account required pam_krb5.so.1
OTHER account required pam_unix_account.so.1
OTHER account required pam_tsol_account.so.1
# OTHER account required pam_krb5.so.1
#
.
.
.
```

e. Change the `PAM_POLICY` entry in the `policy.conf` file to use the modified configuration file.

```
server1 # pfedit /etc/security/policy.conf
...
# PAM_POLICY=krb5_first
PAM_POLICY=krb5_firstmigrate
```

For more information, read the `policy.conf` file.

4. On the master KDC, update the `kadm5.acl` access control file.

The following entries grant migrate and inquire privileges to the `host/server1.example.com` service principal for all users except the `root` user. Use the `U` privilege to list users who must not be migrated. These entries must precede the permit all or `ui` entry. For more information. see the `kadm5.acl(4)` man page.

```
kdc1# pfedit /etc/krb5/kadm5.acl
host/server1.example.com@EXAMPLE.COM U root
host/server1.example.com@EXAMPLE.COM ui *
*/admin@EXAMPLE.COM *
```

5. On the master KDC, enable the `kadmind` daemon to use the `k5migrate` PAM service.

If a `k5migrate` service file is not in the `/etc/pam.d` directory, add the service file to the directory. For more information, see the `pam.d(4)` man page.

This modification enables the validation of UNIX user passwords for accounts that require migration.

```
kdc1# pfedit /etc/pam.d/k5migrate
...
# Permits validation of migrated UNIX accounts
auth    required        pam_unix_auth.so.1
account required        pam_unix_account.so.1
```

Note - `k5migrate` is the name of a PAM service. The file must be named `k5migrate`.

6. **Test your configuration before putting it in production.**

   - **As a regular user, test each modified PAM service.**
   - **As `root`, test each modified PAM service.**
   - **Force a password change, then test the modified PAM services.**

Automatically Renewing All Ticket-Granting Tickets

For ease of administration, you can configure ticket renewal and warning messages about Ticket Granting Ticket (TGT) expiration. Administrators can set warnings for all users, and users can customize their own warnings. For more information, see the `warn.conf(4)` and `ktkt_warnd(1M)` man pages.

Note - The `ktkt_warn` service is disabled by default. To enable the service on existing Kerberos clients, run the `svcadm enable ktkt_warn` command.

EXAMPLE 13 Configuring TGT Expiration Messages for All Users

This example shows several ways to configure the renewal and message system for TGTs.

```
# pfedit /etc/krb5/warn.conf
##
## renew the TGT 30 minutes before expiration and send message to users terminal
##
mre@EXAMPLE.COM renew:log terminal 30m
##
```

```
## send a warning message to a specific email address 20 minutes before TGT expiration
##
mre@EXAMPLE.COM mail 20m mre@example2.com
##
# renew the TGT 20 minutes before expiration and send an email message on failure
##
bricker@EXAMPLE.COM renew:log-failure mail 20m -
##
## catch-all: any principal not matched above will get an email warning
* mail 20m -
```

After configuring the messages, run the `kclient` command on new clients.

```
client# /usr/sbin/kclient -p /net/denver.example.com/export/install/kcprofile
```

On existing clients, enable the service.

```
# svcadm enable network/security/ktkt_warn
```

EXAMPLE 14 Configuring TGT Expiration Messages for a User

Each user can configure an individual `warnd` configuration file, which is named `/var/user/`*`$USER`*`/krb-warn.conf`. The existence of this file prevents the administrator file from being read.

```
% pfedit /var/user/mre/krb-warn.conf
mre@EXAMPLE.COM renew:log mail 25m &
```

The TGT is renewed 25 minutes before expiration, the renewal is logged, and the Kerberos user `mre` is sent mail at that time.

Configuring Kerberos Network Application Servers

Network application servers are hosts that provide access using one or more of the following network applications: `ftp`, `rcp`, `rlogin`, `rsh`, `ssh`, and `telnet`. Only a few steps are required to enable the Kerberos version of these applications on a server.

▼ How to Configure a Kerberos Network Application Server

This procedure uses the following configuration parameters:

- Application server = `boston`

- admin principal = kws/admin
- DNS domain name = example.com
- Realm name = EXAMPLE.COM

Before You Begin Make sure the master KDC is configured and the clocks are synchronized as described in "Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139. To fully test the process, you need several clients.

You must assume the root role on the application server. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Determine if a host principal exists for the new server.**

   The following command reports the existence of the host principal:

```
boston # klist -k | grep host
4 host/boston.example.com@EXAMPLE.COM
4 host/boston.example.com@EXAMPLE.COM
4 host/boston.example.com@EXAMPLE.COM
4 host/boston.example.com@EXAMPLE.COM
```

   If the command does returns a principal, you are done. If it does not return a principal, then create new principals by using the following steps.

2. **Log in to the server with one of the admin principal names that you created when configuring the master KDC.**

```
boston # /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin:
```

3. **Create the server's host principal.**

```
kadmin: addprinc -randkey host/boston.example.com
Principal "host/boston.example.com" created.
kadmin:
```

   The host principal is used in the following ways:

   - To authenticate traffic when using remote commands such as rsh and ssh.
   - By pam_krb5 to prevent KDC spoofing attacks by using the host principal to verify that a user's Kerberos credential was obtained from a trusted KDC.
   - To allow the root user to automatically acquire a Kerberos credential without requiring that a root principal exist. This capability can be useful when doing a manual NFS mount where the share requires a Kerberos credential.

   This principal is required if traffic using the remote application is to be authenticated using the Kerberos service. If the server has multiple host names associated with it, then create a principal for each host name using the FQDN form of the host name.

4. **Add the server's `host` principal to the server's keytab file and quit `kadmin`.**

   If the `kadmin` command is not running, restart it with a command similar to the following: `/usr/sbin/kadmin -p` *kws*`/admin`

   If the server has multiple host names associated with it, then add a principal to the keytab for each host name.

```
kadmin: ktadd host/boston.example.com
Entry for principal host/boston.example.com with kvno 3, encryption type AES-256 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/boston.example.com with kvno 3, encryption type AES-128 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/boston.example.com with kvno 3, encryption type Triple DES cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
```

How to Use the Generic Security Service With Kerberos When Running FTP

The generic security service (GSS) can be used by Kerberos network applications for authentication, integrity, and privacy. The following steps show how to enable the GSS service for ProFTPD.

Before You Begin You must assume the `root` role on the FTP server. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Add principals for the FTP server and create the FTP server's keytab file.**

   These steps might not be needed if the changes were made earlier.

   a. **Start the `kadmin` command.**

```
ftpserver1 # /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin:
```

   b. **Add the `ftp` service principal for the FTP server.**

```
kadmin: addprinc -randkey ftp/ftpserver1.example.com
```

   c. **Add the `ftp` service principal to a new keytab file.**

      A new keytab file allows this information to be available to the ftp service without exposing all of the information in the server's keytab file.

```
kadmin: ktadd -k /etc/krb5/ftp.keytab ftp/ftpserver1.example.com
```

For more information, see the `ktadd` command in the `kadmin`(1M) man page.

2. **Change ownership of the new keytab file.**

```
ftpserver1 # chown ftp:ftp /etc/krb5/ftp.keytab
```

3. **Enable GSS for the FTP server.**

Make the following changes to the `/etc/proftpd.conf` file.

```
# pfedit /etc/proftpd.conf
LoadModule     mod_gss.c

GSSEngine      on
GSSKeytab      /etc/krb5/ftp.keytab
```

4. **Restart the FTP server.**

```
# svcadm restart network/ftp
```

Configuring Kerberos NFS Servers

NFS services use UNIX user IDs (UIDs) to identify a user and cannot directly use GSS credentials. To translate the credential to a UID, you might need to create a credential table that maps user credentials to UNIX UIDs. For information about the default credential mapping, see "Map GSS Credentials to UNIX Credentials" on page 65. The procedures in this section focus on the tasks that are necessary to configure a Kerberos NFS server, to administer the credential table, and to initiate Kerberos security modes for NFS-mounted file systems. The following task map describes the tasks that are covered in this section.

TABLE 7 Task Map: Configuring Kerberos NFS Servers

Task	Description	For Instructions
Configure a Kerberos NFS server.	Enables a server to share a file system that requires Kerberos authentication.	"How to Configure Kerberos NFS Servers" on page 130
Create a credential table and modify it.	Creates a credential table for mapping GSS credentials to UNIX UIDs when the default mapping is not sufficient, then adds an entry.	"How to Create and Modify a Credential Table" on page 131
Map user credentials from another realm to UNIX UIDs.	Updates information in the credential table.	Example 15, "Adding a Principal in a Different Domain to the Kerberos Credential Table," on page 132
Create credential mappings between two like realms.	Maps UIDs from one realm to another where the realms share a password file.	"How to Provide Credential Mapping Between Realms" on page 132

Task	Description	For Instructions
Share a file system with Kerberos authentication.	Shares a file system with security modes so that Kerberos authentication is required.	"How to Set Up a Secure NFS Environment With Multiple Kerberos Security Modes" on page 133

How to Configure Kerberos NFS Servers

This procedure uses the following configuration parameters:

- Realm name = `EXAMPLE.COM`
- DNS domain name = `example.com`
- NFS server = `denver.example.com`
- `admin` principal = `kws/admin`

Before You Begin You must assume the `root` role on the NFS server. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

Make sure the master KDC is configured and the clocks are synchronized as described in "Synchronizing Clocks Between KDCs and Kerberos Clients" on page 139. To fully test the process, you need several clients.

1. **Configure the NFS server as a Kerberos client.**

   Follow the instructions in "Configuring Kerberos Clients" on page 107.

2. **Add the NFS service principal.**

   Use the `kadmin` command.

```
denver # /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin:
```

   a. **Create the NFS service principal.**

      Note that when the principal instance is a host name, the FQDN must be specified in lowercase letters regardless of the case of the domain name in the name service.

      Repeat this step for each unique interface on the system that might be used to access NFS data. If a host has multiple interfaces with unique names, each unique name must have its own NFS service principal.

```
kadmin: addprinc -randkey nfs/denver.example.com
Principal "nfs/denver.example.com" created.
kadmin:
```

b. **Add the server's NFS service principal to the server's keytab file and quit `kadmin`.**
Repeat this step for each unique service principal that you created in Step 2a.

```
kadmin: ktadd nfs/denver.example.com
Entry for principal nfs/denver.example.com with kvno 3, encryption type AES-256 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal nfs/denver.example.com with kvno 3, encryption type AES-128 CTS
 mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal nfs/denver.example.com with kvno 3, encryption type Triple DES
 cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
```

3. **Create special GSS credential maps, if needed.**
Normally, the Kerberos service generates appropriate maps between the GSS credentials and the UNIX UIDs. The default mapping is described in "Map GSS Credentials to UNIX Credentials" on page 65. If the default mapping is not sufficient, see "How to Create and Modify a Credential Table" on page 131 for more information.

4. **Share the NFS file system with Kerberos security modes.**
For more information, see "How to Set Up a Secure NFS Environment With Multiple Kerberos Security Modes" on page 133.

▼ How to Create and Modify a Credential Table

The `gsscred` credential table is used by an NFS server to map Kerberos credentials to a UNIX UID. By default, the primary part of the principal name is matched to a UNIX login name. You create this table if the default mapping is not sufficient.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Ensure that the security mechanism indicated in `/etc/gss/gsscred.conf` is `files`.**

```
# cat /etc/gss/gsscred.conf
...
#
files
#
#
# Syslog (auth.debug) a message for GSS cred to Unix cred mapping
#SYSLOG_UID_MAPPING=yes
```

2. **Create the credential table by using the `gsscred` command.**

```
# gsscred -m kerberos_v5 -a
```

The `gsscred` command gathers information from all sources that are listed with the `passwd` entry in the `svc:/system/name-service/switch:default` service. If you do not want the local password entries included in the credential table, you can temporarily remove the `files` entry. For more information, see the `gsscred`(1M) man page.

3. **(Optional) Add an entry to the credential table.**

For example, as the `root` role on the NFS server, add an entry to map the principal `sandy/admin` to UID `3736`. The `-a` option adds the entry to the credential table.

```
# gsscred -m kerberos_v5 -n sandy/admin -u 3736 -a
```

Example 15 Adding a Principal in a Different Domain to the Kerberos Credential Table

In this example, you use a fully-qualified domain name (FQDN) to specify a principal in a different domain.

```
# gsscred -m kerberos_v5 -n sandy/admin@EXAMPLE.COM -u 3736 -a
```

How to Provide Credential Mapping Between Realms

This procedure provides appropriate credential mapping between realms that use the same password file. In this example, the realms `CORP.EXAMPLE.COM` and `SALES.EXAMPLE.COM` use the same password file. The credentials for *username*`@CORP.EXAMPLE.COM` and *username*`@SALES.EXAMPLE.COM` are mapped to the same UID.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

- **On the client system, add `default_realm` and `auth_to_local_realm` entries to the `krb5.conf` file.**

```
# pfedit /etc/krb5/krb5.conf
[libdefaults]
default_realm = CORP.EXAMPLE.COM
.
[realms]
CORP.EXAMPLE.COM = {
.
auth_to_local_realm = SALES.EXAMPLE.COM
.
}
```

Troubleshooting For help with troubleshooting credential mapping problems, see "Observing Mapping From GSS Credentials to UNIX Credentials" on page 206.

How to Set Up a Secure NFS Environment With Multiple Kerberos Security Modes

This procedure enables an NFS server to provide secure NFS access by using several security modes. When a client negotiates a security mode with the NFS server, the client uses the first mode that is offered by the server. This mode is used for all subsequent client requests of the file system shared by that server.

Before You Begin You must assume the `root` role on the NFS server. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Verify that there is an NFS service principal in the keytab file.**

   The `klist` command reports if there is a keytab file and displays the principals. If the results show that no keytab file exists or that no NFS service principal exists, you need to verify the completion of all the steps in "How to Configure Kerberos NFS Servers" on page 130.

   ```
   # klist -k
   Keytab name: FILE:/etc/krb5/krb5.keytab
   KVNO Principal
   ---- --------------------------------------------------------
   3 nfs/denver.example.com@EXAMPLE.COM
   3 nfs/denver.example.com@EXAMPLE.COM
   3 nfs/denver.example.com@EXAMPLE.COM
   3 nfs/denver.example.com@EXAMPLE.COM
   ```

   For more information, see the `klist`(1) man page.

2. **Enable Kerberos security modes in the `/etc/nfssec.conf` file.**

   In the `/etc/nfssec.conf` file, remove the "#" that comments out the Kerberos security modes.

   ```
   # pfedit /etc/nfssec.conf
   .
   .
   #
   # Uncomment the following lines to use Kerberos V5 with NFS
   #
   krb5            390003  kerberos_v5     default -               # RPCSEC_GSS
   krb5i           390004  kerberos_v5     default integrity       # RPCSEC_GSS
   krb5p           390005  kerberos_v5     default privacy         # RPCSEC_GSS
   ```

3. **Share the file systems with the appropriate security modes.**

   `share -F nfs -o sec=`*mode file-system*

mode Specifies the security modes to be used when sharing the file system. When using multiple security modes, the first mode in the list is used as the default.

file-system Defines the path to the file system to be shared.

All clients that attempt to access files from the named file system require Kerberos authentication. To access files, the user principal on the NFS client should be authenticated.

4. **(Optional) Mount a file system by using a security mode other than the default.**

   Do not perform this procedure if the default security mode is acceptable.

   - **If the automounter is being used, edit the `auto_master` database to enter a security mode other than the default.**

     file-system `auto_home -nosuid,sec=`*mode*

   - **Manually issue the `mount` command to access the file system by using a non-default mode.**

     `# mount -F nfs -o sec=`*mode* *file-system*

Example 16 Sharing a File System With One Kerberos Security Mode

In this example, authentication with the `krb5` security mode must succeed before any files can be accessed through the NFS service.

```
# share -F nfs -o sec=krb5 /export/home
```

Example 17 Sharing a File System With Multiple Kerberos Security Modes

In this example, all three Kerberos security modes have been selected. The mode that is used is negotiated between the client and the NFS server. If the first mode in the command fails, then the next is tried. For more information, see the `nfssec`(5) man page.

```
# share -F nfs -o sec=krb5:krb5i:krb5p /export/home
```

Configuring Delayed Execution for Access to Kerberos Services

In the default Kerberos environment, credentials expire after a limited amount of time. For processes that can execute at arbitrary times, such as `cron` and `at`, the limited time presents a problem. This procedure describes how to configure the Kerberos environment to support

delayed execution processes that require authenticated services through Kerberos. Oracle Solaris provides PAM modules, uses service keys, and uses `kclient` configuration options to make delayed execution with Kerberos authentication possible and more secure than alternative solutions.

Note - If the `cron` server becomes compromised, an attacker could impersonate users to gain access to target services that are configured for the `cron` server. Therefore, consider that the `cron` host that is configured in this procedure as a more sensitive system, as it provides intermediate services for users.

How to Configure a cron Host for Access to Kerberos Services

This procedure uses the following configuration parameters:

- `cron` host = `host1.example.com`
- NFS server = `host2.example.com`
- LDAP server = `host3.example.com`

1. **Configure the `cron` service to support Kerberos.**

   - **If the `cron` host is not configured for Kerberos, then run the `kclient` command on the system.**

     For more information, see the `kclient(1M)` man page.

     For example, the following command configures the client in the `EXAMPLE.COM` realm. The command includes the `pam_gss_s4u` file in the `/etc/pam.d/cron` service file by using the `include` mechanism.

     ```
     # kclient -s cron:optional -R EXAMPLE.COM
     ```

   - **If the `cron` host is already configured for Kerberos, then you must modify the PAM configuration for the `cron` service on that host manually.**

     Ensure that the PAM configuration for the `cron` service includes the `pam_gss_s4u` file.

     ```
     # cd /etc/pam.d ; cp cron cron.orig
     # pfedit cron
           # PAM include file for optional set credentials
           # through Kerberos keytab and GSS-API S4U support
           auth include          pam_gss_s4u
     ```

2. **Enable the `cron` host to act as a delegate.**

For example:

```
# kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin: modprinc +ok_as_delegate host/host1.example.com@EXAMPLE.COM
Principal host/host1.example.com@EXAMPLE.COM modified.
```

3. **Enable the `cron` host to request tickets for itself on behalf of the user who created the `cron` job.**

```
kadmin: modprinc +ok_to_auth_as_delegate host/host1.example.com@EXAMPLE.COM
Principal host/host1.example.com@EXAMPLE.COM modified.
kadmin: quit
```

4. **In LDAP, configure the `cron` host to specify the services that it uses as a delegate.**

   For example, to enable the `cron` host to access the user's home directory on `host2`, a Kerberized NFS server, add the NFS host to the `krbAllowedToDelegateTo` parameter in the `cron` server's LDAP definition.

   a. **Create the delegate assignment.**

```
# pfedit /tmp/delghost.ldif
dn: krbprincipalname=host/host1.example.com@EXAMPLE.COM,cn=EXAMPLE.COM,
cn=krbcontainer,dc=example,dc=com
changetype: modify
krbAllowedToDelegateTo: nfs/host2.example.com@EXAMPLE.COM
```

   b. **Add the assignment to LDAP.**

```
# ldapmodify -h host3 -D "cn=directory manager" -f delghost.ldif
```

Configuring Cross-Realm Authentication

You have several ways of linking realms together so that users in one realm can be authenticated in another realm. Cross-realm authentication is accomplished by establishing a secret key that is shared between the two realms. The relationship of the realms can be either hierarchal or directional. For more information, see “Kerberos Realm Hierarchy” on page 58.

How to Establish Hierarchical Cross-Realm Authentication

The example in this procedure establishes cross-realm authentication between `CORP.EAST.EXAMPLE.COM` and `EAST.EXAMPLE.COM` in both directions. This procedure must be performed on the master KDC in both realms.

Before You Begin The master KDC for each realm is configured. To fully test the authentication process, you need several clients.

You must assume the `root` role on both KDC servers. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Create ticket-granting ticket service principals for the two realms.**

   You must log in with one of the `admin` principal names that was created when you configured the master KDC.

```
# /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin: addprinc krbtgt/CORP.EAST.EXAMPLE.COM@EAST.EXAMPLE.COM
Enter password for principal krbtgt/CORP.EAST.EXAMPLE.COM@EAST.EXAMPLE.COM:     /** Type
 strong password **/
kadmin: addprinc krbtgt/EAST.EXAMPLE.COM@CORP.EAST.EXAMPLE.COM
Enter password for principal krbtgt/EAST.EXAMPLE.COM@CORP.EAST.EXAMPLE.COM:     /** Type
 strong password **/
kadmin: quit
```

   Note - Save and store these passwords in a safe location.

2. **Add entries to the Kerberos configuration file to define domain names for every realm.**

```
# pfedit /etc/krb5/krb5.conf
[libdefaults]
.
.
[domain_realm]
.corp.east.example.com = CORP.EAST.EXAMPLE.COM
.east.example.com = EAST.EXAMPLE.COM
```

   In this example, domain names for the `CORP.EAST.EXAMPLE.COM` and `EAST.EXAMPLE.COM` realms are defined. The subdomain must precede the domain name in the file, because the file is searched top down.

3. **Copy the Kerberos configuration file to all clients in this realm.**

For cross-realm authentication to work, all systems (including slave KDCs and other servers) must use the master KDC's version of `/etc/krb5/krb5.conf`.

4. **Repeat this procedure in the second realm.**

Note - The password that is specified for each service principal must be identical in both KDCs. Thus, the password for the service principal `krbtgt/CORP.EAST.EXAMPLE.COM@EAST.EXAMPLE.COM` must be the same in both realms.

▼ How to Establish Direct Cross-Realm Authentication

The example in this procedure uses two realms, `CORP.EAST.EXAMPLE.COM` and `SALES.WEST.EXAMPLE.COM`. Cross-realm authentication will be established in both directions. This procedure must be completed on the master KDC in both realms.

Before You Begin The master KDC for each realm is configured. To fully test the authentication process, you need several clients.

You must assume the `root` role on both KDC servers. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Create ticket-granting ticket service principals for the two realms.**

   You must log in with one of the `admin` principal names that was created when you configured the master KDC.

```
# /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin: addprinc krbtgt/CORP.EAST.EXAMPLE.COM@SALES.WEST.EXAMPLE.COM
Enter password for principal
krbtgt/CORP.EAST.EXAMPLE.COM@SALES.WEST.EXAMPLE.COM:      /** Type strong password **/
kadmin: addprinc krbtgt/SALES.WEST.EXAMPLE.COM@CORP.EAST.EXAMPLE.COM
Enter password for principal
krbtgt/SALES.WEST.EXAMPLE.COM@CORP.EAST.EXAMPLE.COM:      /** Type strong password **/
kadmin: quit
```

2. **Add entries in the Kerberos configuration file to define the direct path to the remote realm.**

   This example shows the clients in the `CORP.EAST.EXAMPLE.COM` realm. To add the appropriate definitions in the `SALES.WEST.EXAMPLE.COM` realm, swap the realm names.

```
# pfedit /etc/krb5/krb5.conf
[libdefaults]
```

```
.
.
[capaths]
CORP.EAST.EXAMPLE.COM = {
SALES.WEST.EXAMPLE.COM = .
}

SALES.WEST.EXAMPLE.COM = {
CORP.EAST.EXAMPLE.COM = .
}
```

3. **Copy the Kerberos configuration file to all clients in the current realm.**

   For cross-realm authentication to work, all systems (including slave KDCs and other servers) must use the new version of the Kerberos configuration file, `/etc/krb5/krb5.conf`.

4. **Repeat this procedure for the second realm.**

Synchronizing Clocks Between KDCs and Kerberos Clients

All hosts that participate in the Kerberos authentication system must have their internal clocks synchronized within a specified maximum amount of time (known as *clock skew*). This requirement provides another Kerberos security check. If the clock skew is exceeded between any of the participating hosts, client requests are rejected.

The clock skew also determines how long application servers must keep track of Kerberos protocol messages, in order to recognize and reject replayed requests. So, the longer the clock skew value, the more information that application servers have to collect.

The default value for the maximum clock skew is 300 seconds (five minutes). You can change this default in the `libdefaults` section of the `krb5.conf` file.

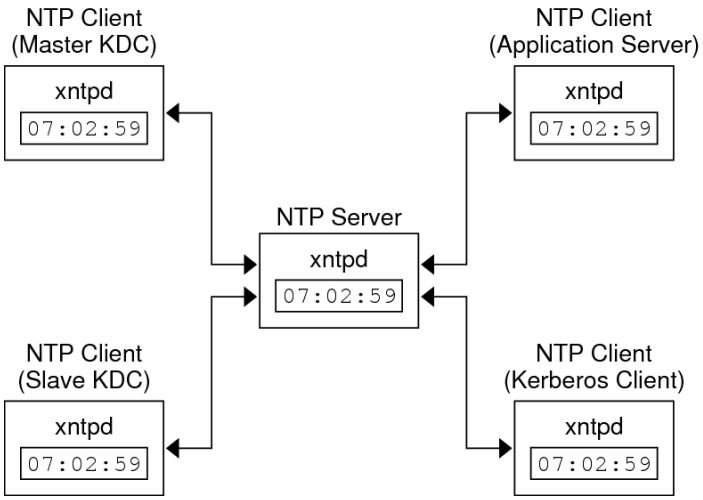
Note - For security reasons, do not increase the clock skew beyond 300 seconds.

Because maintaining synchronized clocks between the KDCs and Kerberos clients is important, use the Precision Time Protocol (PTP) or Network Time Protocol (NTP) software to synchronize the clocks. For a description of these clock synchronization software protocols, see *Enhancing System Performance Using Clock Synchronization and Web Caching in Oracle Solaris 11.3*.

The NTP software is installed by default on most Oracle Solaris systems. You can install the PTP software by using the `pkg install ptp` command.

The following figure shows an example of NTP clock synchronization.

FIGURE 11 Synchronizing Clocks by Using NTP



Ensuring that the KDCs and Kerberos clients maintain synchronized clocks involves implementing the following steps:

1. Setting up a PTP or an NTP server on your Kerberos network. This server can be any system except the master KDC.
2. As you configure the KDCs and Kerberos clients on the network, make them clients of the clock synchronization server. Return to the master KDC to configure the KDC as a client of the clock synchronization server.
3. Enabling the clock synchronization service on all systems.

Swapping a Master KDC and a Slave KDC

The procedures in this section make the swap of a master KDC with a slave KDC easier. You should swap the master KDC with a slave KDC only if the master KDC server fails for some reason, or if the master KDC needs to be re-installed (for example, because new hardware is installed).

How to Configure a Swappable Slave KDC

Perform this procedure in a realm that is using incremental propagation on the slave KDC server that you want to have available to become the master KDC.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Use alias names for the master KDC and the swappable slave KDC during the KDC installation.**

   When you define the host names for the KDCs, make sure that each system has an alias included in DNS. Also, use the alias names when you define the hosts in the `/etc/krb5/krb5.conf` file.

2. **Install a slave KDC.**

   Prior to any swap, this server should function like any other slave KDC in the realm. For more information, see "How to Manually Configure a Slave KDC" on page 80.

3. **After installation, move the master KDC commands.**

   The master KDC commands must not be run from this slave KDC.

```
kdc4# mv /usr/lib/krb5/kprop /usr/lib/krb5/kprop.save
kdc4# mv /usr/lib/krb5/kadmind /usr/lib/krb5/kadmind.save
kdc4# mv /usr/sbin/kadmin.local /usr/sbin/kadmin.local.save
```

How to Swap a Master KDC and a Slave KDC

In this procedure, the master KDC server that is being swapped out is named `kdc1`. The slave KDC that will become the new master KDC is named `kdc4` and the original master KDC becomes a slave KDC. This procedure assumes that you are using incremental propagation.

Before You Begin Complete the procedure "How to Configure a Swappable Slave KDC" on page 141.

You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **On the new master KDC, start `kadmin`.**

```
kdc4# /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin:
```

2. **Create new principals for the `kadmind` service.**

The following example shows the first `addprinc` command on two lines, but it should be typed on one line.

```
kadmin: addprinc -randkey -allow_tgs_req +password_changing_service -clearpolicy \
changepw/kdc4.example.com
Principal "changepw/kdc4.example.com@EXAMPLE.COM" created.
kadmin: addprinc -randkey -allow_tgs_req -clearpolicy kadmin/kdc4.example.com
Principal "kadmin/kdc4.example.com@EXAMPLE.COM" created.
kadmin: quit
```

3. On the new master KDC, force synchronization.

The following steps force a full KDC update on the slave server.

a. Disable the `krb5kdc` service and remove its log file.

```
kdc4# svcadm disable network/security/krb5kdc
kdc4# rm /var/krb5/principal.ulog
```

b. Verify that the update is complete.

```
kdc4# /usr/sbin/kproplog -h
```

c. Restart the KDC service.

```
kdc4# svcadm enable -r network/security/krb5kdc
```

d. Reinitialize the update log for the new master KDC server.

```
kdc4# svcadm disable network/security/krb5kdc
kdc4# rm /var/krb5/principal.ulog
```

4. On the original master KDC, disable the `kadmin` and `krb5kdc` services.

By disabling the `kadmin` service, you prevent any changes from being made to the KDC database.

```
kdc1# svcadm disable network/security/kadmin
kdc1# svcadm disable network/security/krb5kdc
```

5. On the original master KDC, specify the poll time for requesting propagations.

Comment out the `sunw_dbprop_master_ulogsize` entry in `/etc/krb5/kdc.conf` and add an entry that defines the slave's polling interval. This entry sets the poll time to two minutes.

```
kdc1# pfedit /etc/krb5/kdc.conf
[kdcdefaults]
     kdc_ports = 88,750

[realms]
     EXAMPLE.COM= {
```

```
        profile = /etc/krb5/krb5.conf
        database_name = /var/krb5/principal
        acl_file = /etc/krb5/kadm5.acl
        kadmind_port = 749
        max_life = 8h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        sunw_dbprop_enable = true
#            sunw_dbprop_master_ulogsize = 1000
        sunw_dbprop_slave_poll = 2m
    }
```

6. **On the original master KDC, move the master KDC commands and the `kadm5.acl` file.**

   The master KDC commands must not be run from the original master KDC.

```
kdc1# mv /usr/lib/krb5/kprop /usr/lib/krb5/kprop.save
kdc1# mv /usr/lib/krb5/kadmind /usr/lib/krb5/kadmind.save
kdc1# mv /usr/sbin/kadmin.local /usr/sbin/kadmin.local.save
kdc1# mv /etc/krb5/kadm5.acl /etc/krb5/kadm5.acl.save
```

7. **Modify the DNS mappings.**

   a. **On the DNS server, change the alias names for the master KDC.**

      To change the servers, edit the `example.com` zone file and change the entry for `masterkdc`.

```
masterkdc IN CNAME kdc4
```

   b. **Reload the new alias information.**

```
# svcadm refresh network/dns/server
```

8. **On the new master KDC, move the master KDC commands and the slave `kpropd.acl` file.**

   You moved the master KDC commands in Step 3 of "How to Configure a Swappable Slave KDC" on page 141,

```
kdc4# mv /usr/lib/krb5/kprop.save /usr/lib/krb5/kprop
kdc4# mv /usr/lib/krb5/kadmind.save /usr/lib/krb5/kadmind
kdc4# mv /usr/sbin/kadmin.local.save /usr/sbin/kadmin.local
kdc4# mv /etc/krb5/kpropd.acl /etc/krb5/kpropd.acl.save
```

9. **On the new master KDC, create the Kerberos access control list file, `kadm5.acl`.**

   Once populated, the `/etc/krb5/kadm5.acl` file should contain all principal names that are allowed to administer the KDC. The file should also list all of the slaves that can make requests for incremental propagation. For more information, see the `kadm5.acl(4)` man page.

```
kdc4# pfedit /etc/krb5/kadm5.acl
kws/admin@EXAMPLE.COM   *
```

```
kiprop/kdc1.example.com@EXAMPLE.COM p
```

10. On the new master KDC, specify the update log size in the `kdc.conf` file.

Comment out the `sunw_dbprop_slave_poll` entry and add an entry that defines `sunw_dbprop_master_ulogsize`. This entry sets the log size to 1000 entries.

```
kdc4# pfedit /etc/krb5/kdc.conf
[kdcdefaults]
     kdc_ports = 88,750

[realms]
     EXAMPLE.COM= {
        profile = /etc/krb5/krb5.conf
        database_name = /var/krb5/principal
        acl_file = /etc/krb5/kadm5.acl
        kadmind_port = 749
        max_life = 8h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        sunw_dbprop_enable = true
#            sunw_dbprop_slave_poll = 2m
        sunw_dbprop_master_ulogsize = 1000
     }
```

11. On the new master KDC, enable the `kadmin` and `krb5kdc` services.

```
kdc4# svcadm enable -r network/security/krb5kdc
kdc4# svcadm enable -r network/security/kadmin
```

12. Make the original master KDC a slave KDC.

a. Add the `kiprop` service principal.

Adding the `kiprop` principal to the `krb5.keytab` file allows the `kpropd` daemon to authenticate itself for the incremental propagation service.

```
kdc1# /usr/sbin/kadmin -p kws/admin
Authenticating as principal kws/admin@EXAMPLE.COM with password.
Enter password: xxxxxxxx
kadmin: ktadd kiprop/kdc1.example.com
Entry for principal kiprop/kdc1.example.com with kvno 3,
encryption type AES-256 CTS mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal kiprop/kdc1.example.com with kvno 3,
encryption type AES-128 CTS mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal kiprop/kdc1.example.com with kvno 3, encryption type Triple DES
 cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
```

b. Add an entry for each KDC in the `krb5.conf` file to the propagation configuration file.

```
kdc1# pfedit /etc/krb5/kpropd.acl
host/kdc1.example.com@EXAMPLE.COM
host/kdc2.example.com@EXAMPLE.COM
host/kdc3.example.com@EXAMPLE.COM
host/kdc4.example.com@EXAMPLE.COM
```

c. Enable the `kpropd` and `krb5kdc` services.

```
kdc1# svcadm enable -r network/security/krb5_prop
kdc1# svcadm enable -r network/security/krb5kdc
```

Administering the Kerberos Database

The Kerberos database is the backbone of Kerberos and must be maintained properly. This section provides some procedures for administering the Kerberos database, such as backing up and restoring the database, setting up incremental or parallel propagation, and administering the stash file. The steps to initially set up the database are in "How to Manually Configure a Master KDC" on page 75.

Backing Up and Propagating the Kerberos Database

Propagating the Kerberos database from the master KDC to the slave KDCs is one of the most important configuration tasks. If propagation doesn't happen often enough, the master KDC and the slave KDCs will lose synchronization. Then, if the master KDC goes down, the slave KDCs will not have the most recent database information. Also, if a slave KDC has been configured as a master KDC for purposes of load balancing, the clients that use that slave KDC as a master KDC will not have the latest information.

Make sure that propagation occurs often enough or else configure the servers for incremental propagation, based on how often you change the Kerberos database. Incremental propagation is preferred over other propagation methods. Manual propagation of the database requires more administrative overhead and full propagation is inefficient.

Caution - To avoid data loss, you should manually propagate the database if you add significant updates to the Kerberos database before a regularly scheduled propagation.

`kpropd.acl` File

The `kpropd.acl` file on a slave KDC provides a list of host principal names, one name per line, that specifies the systems from which the KDC can receive an updated database through propagation. If the master KDC is used to propagate all the slave KDCs, the `kpropd.acl` file on each slave needs to contain only the host principal name of the master KDC.

However, the Kerberos installation and subsequent configuration steps described in this guide instruct you to use the same `kpropd.acl` file on the master KDC and the slave KDCs. This file contains all the KDC host principal names. This configuration enables you to propagate from any KDC, in case the propagating KDCs become temporarily unavailable. The identical copy on all KDCs eases maintenance.

`kprop_script` Command

The `kprop_script` command uses the `kprop` command to propagate the Kerberos database to other KDCs. If the `kprop_script` command is run on a slave KDC, it propagates the slave KDC's copy of the Kerberos database to other KDCs. The `kprop_script` accepts a list of host names for arguments, separated by spaces, which indicate the KDCs to propagate.

When `kprop_script` is run, it creates a backup of the Kerberos database to the `/var/krb5/slave_datatrans` file and copies the file to the specified KDCs. The Kerberos database is locked until the propagation is finished.

Backing Up the Kerberos Database

When you configure the master KDC, you set up the `kprop_script` command in a `cron` job to automatically back up the Kerberos database to the `/var/krb5/slave_datatrans` dump file and propagate it to the slave KDCs. But, as with any file, the Kerberos database can become corrupted. Data corruption is not an issue on a slave KDC, because the next automatic propagation of the database installs a fresh copy. However, if corruption occurs on the master KDC, the corrupted database is propagated to all of the slave KDCs during the next propagation. The corrupted backup also overwrites the previous uncorrupted backup file on the master KDC.

To protect against this scenario, set up a `cron` job to periodically copy the `slave_datatrans` dump file to another location or to create another separate backup copy by using the `dump` command of `kdb5_util`. Then, if your database becomes corrupted, you can restore the most recent backup on the master KDC by using the `load` command of `kdb5_util`.

Another important note: Because the database dump file contains principal keys, you need to protect the file from being accessed by unauthorized users. By default, the database dump file has read and write permissions only as `root`. To protect against unauthorized access, propagate

the database dump file with the `kprop` command, because this command encrypts the data that is being transferred. Also, `kprop` propagates the data only to the slave KDCs, which minimizes the chance of accidentally sending the database dump file to unauthorized hosts.

EXAMPLE 18 Manually Backing Up the Kerberos Database

You use the `dump` command of the `kdb5_util` command to back up the database. Run this command in a directory that is owned by `root`.

```
# /usr/sbin/kdb5_util dump
```

In the following example, the Kerberos database is backed up to a file called `dumpfile`. Because the `-verbose` option is specified, each principal is printed as it is backed up. Because no principals are specified, the entire database is backed up.

```
# kdb5_util dump -verbose /var/user/kadmin/dumpfile
kadmin/kdc1.corp.example.com@CORP.EXAMPLE.COM
krbtgt/CORP.EXAMPLE.COM@CORP.EXAMPLE.COM
kadmin/history@CORP.EXAMPLE.COM
pak/admin@CORP.EXAMPLE.COM
pak@CORP.EXAMPLE.COM
changepw/kdc1.corp.example.com@CORP.EXAMPLE.COM
```

In the following example, the dump contains only the `pak` and `pak/admin` principals.

```
# kdb5_util dump -verbose pakfile pak/admin@CORP.EXAMPLE.COM pak@CORP.EXAMPLE.COM
pak/admin@CORP.EXAMPLE.COM
pak@CORP.EXAMPLE.COM
```

For more information, see the `kdb5_util`(1M) man page.

▼ How to Restore a Backup of the Kerberos Database

Before You Begin On the KDC master, you must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **On the master, stop the KDC daemons.**

   ```
   kdc1# svcadm disable network/security/krb5kdc
   kdc1# svcadm disable network/security/kadmin
   ```

2. **Restore the Kerberos database by using the `load` command of the `kdb_util` command.**

   For example, load the `dumpfile` backup from Example 18, "Manually Backing Up the Kerberos Database," on page 147.

```
# /usr/sbin/kdb5_util load /var/user/kadmin/dumpfile
```

3. **Start the KDC daemons.**

```
kdc1# svcadm enable -r network/security/krb5kdc
kdc1# svcadm enable -r network/security/kadmin
```

Example 19 Restoring the Kerberos Database

In the following example, the database called `database1` is restored into the current directory from the `dumpfile` file. Because the `-update` option is not specified, a new database is created.

```
# kdb5_util load -d database1 dumpfile
```

How to Convert a Kerberos Database After a Server Upgrade

If your KDC database was created on a server that was running an old release, converting the database enables you to take advantage of the improved database format.

Before You Begin Use this procedure only if the database is using an older format.

On the KDC master, you must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **On the master, stop the KDC daemons.**

```
kdc1# svcadm disable network/security/krb5kdc
kdc1# svcadm disable network/security/kadmin
```

2. **Create a directory to store a temporary copy of the database.**

```
kdc1# mkdir /var/krb5/tmp
kdc1# chmod 700 /var/krb5/tmp
```

3. **Dump the KDC database.**

```
kdc1# kdb5_util dump /var/krb5/tmp/prdb.txt
```

4. **Save copies of the current database files.**

```
kdc1# cd /var/krb5
kdc1# mv princ* tmp/
```

5. **Load the database.**

```
kdc1# kdb5_util load /var/krb5/tmp/prdb.txt
```

6. **Start the KDC daemons.**

```
kdc1# svcadm enable -r network/security/krb5kdc
kdc1# svcadm enable -r network/security/kadmin
```

How to Reconfigure a Master KDC to Use Incremental Propagation

The steps in this procedure can be used to reconfigure an existing master KDC to use incremental propagation. This procedure uses the following configuration parameters:

- Realm name = `EXAMPLE.COM`
- DNS domain name = `example.com`
- Master KDC = `kdc1.example.com`
- Slave KDC = `kdc2.example.com`
- `admin` principal = `kws/admin`

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Configure incremental propagation in the `kdc.conf` file.**

   Enable incremental propagation and select the number of updates the KDC master keeps in the log. For more information, see the `kdc.conf`(4) man page.

```
kdc1# pfedit /etc/krb5/kdc.conf
[kdcdefaults]
     kdc_ports = 88,750

[realms]
     EXAMPLE.COM= {
        profile = /etc/krb5/krb5.conf
        database_name = /var/krb5/principal
        acl_file = /etc/krb5/kadm5.acl
        kadmind_port = 749
        max_life = 8h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        sunw_dbprop_enable = true
        sunw_dbprop_master_ulogsize = 1000
      }
```

2. **Create the `kiprop` principal.**

   The `kiprop` principal is used to authenticate the master KDC server and to authorize updates from the master KDC.

```
kdc1# /usr/sbin/kadmin -p kws/admin
```

```
Enter password: xxxxxxxx
kadmin: addprinc -randkey kiprop/kdc1.example.com
Principal "kiprop/kdc1.example.com@EXAMPLE.COM" created.
kadmin: addprinc -randkey kiprop/kdc2.example.com
Principal "kiprop/kdc2.example.com@EXAMPLE.COM" created.
kadmin:
```

3. **On the master KDC, add a `kiprop` entry to `kadm5.acl`.**

   This entry allows the master KDC to receive requests for incremental propagation from the `kdc2` server.

```
kdc1# pfedit /etc/krb5/kadm5.acl
*/admin@EXAMPLE.COM *
kiprop/kdc2.example.com@EXAMPLE.COM p
```

4. **Comment out the `kprop` line in the `root` crontab file.**

   This step prevents the master KDC from propagating its copy of the KDC database.

```
kdc1# crontab -e
#ident  "@(#)root       1.20    01/11/06 SMI"
#
# The root crontab should be used to perform accounting data collection.
#
# The rtc command is run to adjust the real time clock if and when
# daylight savings time changes.
#
10 3 * * * /usr/sbin/logadm
15 3 * * 0 /usr/lib/fs/nfs/nfsfind
1 2 * * * [ -x /usr/sbin/rtc ] && /usr/sbin/rtc -c > /dev/null 2>&1
30 3 * * * [ -x /usr/lib/gss/gsscred_clean ] && /usr/lib/gss/gsscred_clean
#10 3 * * * /usr/lib/krb5/kprop_script kdc2.example.com
```

5. **Restart `kadmind`.**

```
kdc1# svcadm restart network/security/kadmin
```

6. **Reconfigure all slave KDC servers that use incremental propagation.**

   For complete instructions, see "How to Reconfigure a Slave KDC to Use Incremental Propagation" on page 150.

How to Reconfigure a Slave KDC to Use Incremental Propagation

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Add entries to `kdc.conf`.**

   The first new entry enables incremental propagation. The second new entry sets the poll time to two minutes.

```
kdc2# pfedit /etc/krb5/kdc.conf
[kdcdefaults]
kdc_ports = 88,750

[realms]
EXAMPLE.COM= {
profile = /etc/krb5/krb5.conf
database_name = /var/krb5/principal
acl_file = /etc/krb5/kadm5.acl
kadmind_port = 749
max_life = 8h 0m 0s
max_renewable_life = 7d 0h 0m 0s
sunw_dbprop_enable = true
sunw_dbprop_slave_poll = 2m
}
```

2. **Add the `kiprop` principal to the `krb5.keytab` file.**

```
kdc2# /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin: ktadd kiprop/kdc2.example.com
Entry for principal kiprop/kdc2.example.com with kvno 3,
encryption type AES-256 CTS mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal kiprop/kdc2.example.com with kvno 3,
encryption type AES-128 CTS mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal kiprop/kdc2.example.com with kvno 3, encryption type Triple DES cbc
mode with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
```

3. **Restart `kpropd`.**

```
kdc2# svcadm restart network/security/krb5_prop
```

How to Verify That the KDC Servers Are Synchronized

If incremental propagation has been configured, this procedure ensures that the information about the slave KDC has been updated.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **On the KDC master server, run the `kproplog` command.**

   ```
   kdc1# /usr/sbin/kproplog -h
   ```

2. **On a KDC slave server, run the `kproplog` command.**

   ```
   kdc2# /usr/sbin/kproplog -h
   ```

3. **Check that the last serial number and the last timestamp values match.**

Example 20 Verifying That KDC Servers Are Synchronized

The following is a sample of results from running the `kproplog` command on the master KDC server.

```
kdc1# /usr/sbin/kproplog -h

Kerberos update log (/var/krb5/principal.ulog)
Update log dump:
Log version #: 1
Log state: Stable
Entry block size: 2048
Number of entries: 2500
First serial #: 137966
Last serial #: 140465
First time stamp: Wed Dec 4 00:59:27 2013
Last time stamp: Wed Dec 4 01:06:13 2013
```

The following is a sample of results from running the `kproplog` command on a slave KDC server.

```
kdc2# /usr/sbin/kproplog -h

Kerberos update log (/var/krb5/principal.ulog)
Update log dump:
Log version #: 1
Log state: Stable
Entry block size: 2048
Number of entries: 0
First serial #: None
Last serial #: 140465
First time stamp: None
Last time stamp: Wed Dec 4 01:06:13 2013
```

Notice that the values for the last serial number and the last timestamp are identical, which indicates that the slave is synchronized with the master KDC server.

In the slave KDC server output, notice that no update entries exist in the slave KDC server's update log. No entries exist because the slave KDC server does not keep a set of updates, unlike the master KDC server. Also, the KDC slave server does not include information about the first serial number or the first timestamp because this is not relevant information.

Manually Propagating the Kerberos Database to the Slave KDCs

Typically, a `cron` job propagates the Kerberos database to slave KDCs. If you need to synchronize a slave KDC with the master KDC outside the periodic `cron` job, you have two options, the `/usr/lib/krb5/kprop_script` and the `kprop` command. For more information, review the script and the kprop(1M) man page.

Caution - Do not use these commands if incremental propagation is enabled on the slave KDC.

How to Manually Propagate the Kerberos Database to a Slave KDC

1. **Verify that incremental propagation is not enabled on the slave KDC.**

```
slave# grep sunw_dbprop_enable /etc/krb5/kdc.conf
sunw_dbprop_enable = true
```

2. **If the value is `true`, disable incremental propagation and restart the `krb5_prop` service.**

```
slave# cp /etc/krb5/kdc.conf /etc/krb5/kdc.conf.sav
slave# pfedit /etc/krb5/kdc.conf
...
sunw_dbprop_enable = false
...

slave# svcadm restart krb5_prop
```

3. **On the master KDC, use one of the following commands to propagate the master KDC database to the slave KDC.**

   - **The `kprop_script` command backs up the database before synchronizing the slave KDC.**

     ```
     master# /usr/lib/krb5/kprop_script slave-KDC
     ```

   - **The `kprop` command propagates the current database backup without first making a new backup of the Kerberos database.**

     ```
     master# /usr/lib/krb5/kprop -f /var/krb5/slave_datatrans slave-KDC
     ```

4. **(Optional) After manual propagation is complete, restore the original `krb5.conf` file.**

```
slave# mv /etc/krb5/kdc.conf.sav /etc/krb5/kdc.conf
```

Setting Up Parallel Propagation for Kerberos

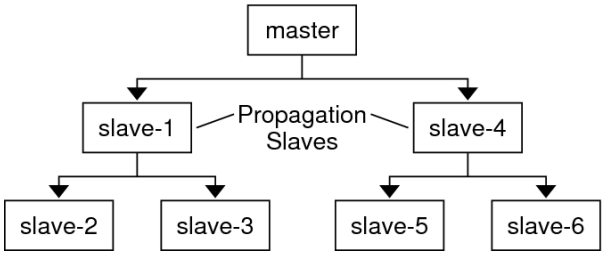
In most cases, the master KDC is used exclusively to propagate its Kerberos database to the slave KDCs. However, if your site has many slave KDCs, you might consider load-sharing the propagation process, known as *parallel propagation*.

Caution - Do not configure parallel propagation if you are using incremental propagation.

Parallel propagation enables specific slave KDCs to share the propagation duties with the master KDC. This sharing of duties enables the propagation to be done faster and to lighten the work for the master KDC.

For example, say your site has one master KDC and six slave KDCs (shown in Figure 12, “Example of Parallel Propagation Configuration in Kerberos,” on page 154), where `slave-1` through `slave-3` consist of one logical grouping and `slave-4` through `slave-6` consist of another logical grouping. To set up parallel propagation, you could have the master KDC propagate the database to `slave-1` and `slave-4`. In turn, those KDC slaves could propagate the database to the KDC slaves in their group.

FIGURE 12 Example of Parallel Propagation Configuration in Kerberos



Configuration Steps for Setting Up Parallel Propagation

The high-level configuration steps to enable parallel propagation are as follows:

1. On the master KDC, change the `kprop_script` entry in its `cron` job to include arguments for only the KDC slaves that will perform the succeeding propagation (the *propagation slaves*).
2. On each propagation slave, add a `kprop_script` entry to its `cron` job, which must include arguments for the slaves to propagate. To successfully propagate in parallel, the `cron` job should be set up to run after the propagation slave is itself propagated with the new Kerberos database.

Note - The amount of time necessary for a propagation slave to be propagated depends on factors such as network bandwidth and the size of the Kerberos database.

---

3. On each slave KDC, set up the appropriate permissions to be propagated by adding the host principal name of its propagating KDC to its `kpropd.acl` file.

EXAMPLE 21 Setting Up Parallel Propagation in Kerberos

Using the example in Figure 12, “Example of Parallel Propagation Configuration in Kerberos,” on page 154, the master KDC's `kprop_script` entry would look similar to the following:

```
0 3 * * * /usr/lib/krb5/kprop_script slave-1.example.com slave-4.example.com
```

The `slave-1`'s `kprop_script` entry would look similar to the following:

```
0 4 * * * /usr/lib/krb5/kprop_script slave-2.example.com slave-3.example.com
```

Note that the propagation on the slave starts an hour after it is propagated by the master.

The `kpropd.acl` file on the propagation slaves would contain the following entry:

```
host/master.example.com@EXAMPLE.COM
```

The `kpropd.acl` file on the KDC slaves being propagated by `slave-1` would contain the following entry:

```
host/slave-1.example.com@EXAMPLE.COM
```

Administering the Stash File for the Kerberos Database

The *stash file* contains the master key for the Kerberos database, which is automatically created when you create a Kerberos database. If the stash file gets corrupted, you can use the `stash` command of the `kdb5_util` utility to replace the corrupted file. The only time you should need to remove a stash file is after removing the Kerberos database with the `destroy` command of `kdb5_util`. Because the stash file is not automatically removed with the database, you have to remove the stash file manually.

To remove the stash file, use the `rm` command:

```
# rm stash-file
```

Where *stash-file* is the path to the stash file. By default, the stash file is located at `/var/krb5/.k5.`*realm*.

Note - If you need to re-create the stash file, you can use the `-f` option of the `kdb5_util` command.

How to Create, Use, and Store a New Master Key for the Kerberos Database

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Create a new master key.**

   This command adds a new, randomly generated master key. The `-s` option requests that the new master key be stored in the default keytab.

   ```
   # kdb5_util add_mkey -s

   Creating new master key for master key principal 'K/M@EXAMPLE.COM'
   You will be prompted for a new database Master Password.
   It is important that you NOT FORGET this password.
   Enter KDC database master key:     /** Type strong password **/
   Re-enter KDC database master key to verify: xxxxxxxx
   ```

2. **Verify that the new master key exists.**

   ```
   # kdb5_util list_mkeys
   Master keys for Principal: K/M@EXAMPLE.COM
   KVNO: 2, Enctype: AES-256 CTS mode with 96-bit SHA-1 HMAC, No activate time set
   KVNO: 1, Enctype: AES-128 CTS mode with 96-bit SHA-1 HMAC, Active on: Fri Dec 31 18:00:
   00 CST 2011 *
   ```

   The asterisk in this output identifies the currently active master key.

3. **Set a time for the newly created master key to become active.**

   ```
   # date
   Fri Oct 9 17:57:00 CDT 2015
   # kdb5_util use_mkey 2 'now+2days'
   # kdb5_util list_mkeys
   Master keys for Principal: K/M@EXAMPLE.COM
   KVNO: 2, Enctype: AES-256 CTS mode with 96-bit SHA-1 HMAC,
   ```

```
Active on: Fri Oct 9 17:57:15 CDT 2015
KVNO: 1, Enctype: AES-128 CTS mode with 96-bit SHA-1 HMAC,
Active on: Fri Dec 31 18:00:00 CST 2011 *
```

In this example, the date is set to two days in the future to allow time for the new master key to propagate to all of the KDCs. Adjust the date as appropriate for your environment.

4. **(Optional) After creating a new principal, verify that the new master key is being used.**

```
# kadmin.local -q 'getprinc tamiko' | egrep 'Principal|MKey'
Authenticating as principal root/admin@EXAMPLE.COM with password.
Principal: tamiko@EXAMPLE.COM
MKey: vno 2
```

In this example, `MKey: vno 2` indicates that the principal's secret key is protected by newly created master key, 2.

5. **Re-encrypt the user principal secret keys with the new master key.**

If you add a pattern argument to the end of the command, the principals that match the pattern will be updated. Add the `-n` option to this command syntax to identify which principals will be updated.

```
# kdb5_util update_princ_encryption -f -v
Principals whose keys WOULD BE re-encrypted to master key vno 2:
updating: host/kdc1.example.com@EXAMPLE.COM
skipping:   tamiko@EXAMPLE.COM
updating: kadmin/changepw@EXAMPLE.COM
updating: kadmin/history@EXAMPLE.COM
updating: kdc/admin@EXAMPLE.COM
updating: host/kdc2.example.com@EXAMPLE.COM
6 principals processed: 5 updated, 1 already current
```

6. **Purge the old master key.**

After a master key is no longer used to protect any principal secret keys, it can be purged from the master key principal. This command will not purge the key if the key is still being used by any principals. Add the `-n` option to this command to verify that the correct master key will be purged.

```
# kdb5_util purge_mkeys -f -v
Purging the following master key(s) from K/M@EXAMPLE.COM:
KVNO: 1
1 key(s) purged.
```

7. **Verify that the old master key has been purged.**

```
# kdb5_util list_mkeys
Master keys for Principal: K/M@EXAMPLE.COM
KVNO: 2, Enctype: AES-256 CTS mode with 96-bit SHA-1 HMAC,
Active on: Sun Oct 4 17:57:15 CDT 2015 *
```

8. **Update the stash file.**

```
# kdb5_util stash
Using existing stashed keys to update stash file.
```

9. **Verify that the stash file has been updated.**

```
# klist -kt /var/krb5/.k5.EXAMPLE.COM
Keytab name: FILE:.k5.EXAMPLE.COM
KVNO Timestamp              Principal
---- --------------- ------------------------------------------------------
2 10/11/2015 18:03 K/M@EXAMPLE.COM
```

Increasing Security on Kerberos Servers

This section provides advice about increasing security on Kerberos application servers and on KDC servers.

Restricting Access to KDC Servers

Both master KDC servers and slave KDC servers have copies of the KDC database stored locally. Restricting access to these servers so that the databases are secure is important to the overall security of the Kerberos installation.

- Restrict physical access to the hardware that supports the KDC.

  Make sure that the KDC server and its monitor are located in a secure facility. Regular users should not be able to access this server in any way.
- Store KDC database backups on local disks or on the KDC slaves.

  Make tape backups of your KDC only if the tapes are stored securely. Follow the same practice for copies of keytab files.

  The recommended practice is to store these files on a local file system that is not shared with other systems. The storage file system can be on either the master KDC server or any of the slave KDCs.

Using a Dictionary File to Increase Password Security

A dictionary file can be used by the Kerberos service to prevent words in the dictionary from being used as passwords for new credentials. Preventing the use of dictionary terms as

passwords makes it harder for someone else to guess any password. By default, the `/var/krb5/kadm5.dict` file is used, but it is empty.

You need to add a line to the KDC configuration file, `kdc.conf` to instruct the service to use a dictionary file. In this example, the administrator uses the dictionary that is included with the `spell` utility, then restarts the Kerberos services. For a full description of the configuration file, see the `kdc.conf(4)` man page.

```
kdc1# pfedit /etc/krb5/kdc.conf
[kdcdefaults]
     kdc_ports = 88,750

[realms]
     EXAMPLE.COM = {
        profile = /etc/krb5/krb5.conf
        database_name = /var/krb5/principal
        acl_file = /etc/krb5/kadm5.acl
        kadmind_port = 749
        max_life = 8h 0m 0s
        max_renewable_life = 7d 0h 0m 0s
        sunw_dbprop_enable = true
        sunw_dbprop_master_ulogsize = 1000
        dict_file = /usr/share/lib/dict/words
     }
kdc1#
kdc1# svcadm restart -r network/security/krb5kdc
kdc1# svcadm restart -r network/security/kadmin
```

♦ ♦ ♦ CHAPTER 5

Administering Kerberos Principals and Policies

This chapter provides procedures for administering principals and the policies that are associated with them. This chapter also shows how to administer a host's keytab file.

This chapter covers the following topics:

For background information about principals and policies, see Chapter 2, "About the Kerberos Service" and Chapter 3, "Planning for the Kerberos Service".

Ways to Administer Kerberos Principals and Policies

The Kerberos database on the master KDC contains all of your realm's Kerberos principals, their passwords, policies, and other administrative information. To create and delete principals, and to modify their attributes, you can use either the `kadmin` or `gkadmin` command.

The `kadmin` command provides an interactive command-line interface that enables you to maintain Kerberos principals, policies, and keytab files. You can also run scripts that automate principal creation. The `kadmin` command has a local version and a remote version:

- `kadmin` – Uses Kerberos authentication to operate securely from anywhere on the network
- `kadmin.local` – Must be run directly on the master KDC

The capabilities of the two versions are identical. You must configure the database enough with the local version before you can use the remote version.

The Oracle Solaris release also provides an interactive graphical user interface (GUI), `gkadmin`.

This section describes the scripting capabilities of the `kadmin.local` command, and compares the command-line and GUI interfaces.

Automating the Creation of New Kerberos Principals

You can use the `kadmin.local` command in a script to automate the creation of new Kerberos principals. Automation is useful when you want to add many new principals to the database.

The following shell script line shows how to automate the creation of new principals:

```
awk '{ print "ank +needchange -pw", $2, $1 }' < /tmp/princnames |
time /usr/sbin/kadmin.local> /dev/null
```

The preceding example is split over two lines for readability.

- The script reads in a file called `princnames`. This file contains principal names and their passwords, and adds them to the Kerberos database.

  You would have to create the `princnames` file, which contains a principal name and its password on each line, separated by one or more spaces.
- The `ank` command adds a new key. `ank` is an alias for the `add_principal` command.
- The `+needchange` option configures the principal so that the user who is the principal is prompted for a new password at first login.

  Requiring a password change helps to ensure that the passwords in the `princnames` file are not a security risk.

You can build more elaborate scripts. For example, your script could use the information in the name service to obtain the list of user names for the principal names. What you do and how you do it is determined by your site's needs and your scripting expertise.

`gkadmin` GUI

The Kerberos `gkadmin` GUI provides most of the capabilities of the CLI, and includes online help. The following table describes the differences between the CLI and the GUI.

TABLE 8 Command-Line Equivalents of the `gkadmin` GUI

gkadmin GUI Procedure	**Equivalent `kadmin` Command**
View the list of principals.	`list_principals` or `get_principals`
View a principal's attributes.	`get_principal`
Create a new principal.	`add_principal`
Duplicate a principal.	No command-line equivalent
Modify a principal.	`modify_principal` or `change_password`
Delete a principal.	`delete_principal`
Set up defaults for creating new principals.	No command-line equivalent

gkadmin GUI Procedure	Equivalent `kadmin` Command
View the list of policies.	`list_policies` or `get_policies`
View a policy's attributes.	`get_policy`
Create a new policy.	`add_policy`
Duplicate a policy.	No command-line equivalent
Modify a policy.	`modify_policy`
Delete a policy.	`delete_policy`

The Kerberos `gkadmin` GUI provides context-sensitive help. For browser access, use the Oracle URL (`http://docs.oracle.com/cd/E23824_01/html/821-1456/aadmin-23.html`). You can copy this file to a site URL. Specify the online help URL in the `krb5.conf` file when configuring a host to use the `gkadmin` GUI.

Administering Kerberos Principals

This section provides examples of using the `kadmin` command and the `gkadmin` GUI to administer principals. For more information, see the `kadmin`(1M) and `gkadmin`(1M) man pages.

Viewing Kerberos Principals and Their Attributes

The following examples show how to list principals and their attributes. You can use wildcards to construct the lists. For information about possible wildcards, review the definition of `expression` in the `kadmin`(1M) man page.

EXAMPLE 22 Viewing Kerberos Principals

In this example, the `list_principals` subcommand is used to list all the principals that match `kadmin*`. Without an argument, `list_principals` lists all the principals that are defined in the Kerberos database.

```
# /usr/sbin/kadmin
kadmin: list_principals kadmin*
kadmin/changepw@EXAMPLE.COM
kadmin/kdc1.example.com@EXAMPLE.COM
kadmin/history@EXAMPLE.COM
```

EXAMPLE 23 Viewing the Attributes of Kerberos Principals

The following example displays the attributes of the `jdb/admin` principal.

```
kadmin: get_principal jdb/admin
Principal: jdb/admin@EXAMPLE.COM

Expiration date: [never]
Last password change: [never]

Password expiration date: Fri Sep 13 11:50:10 PDT 2013
Maximum ticket life: 1 day 16:00:00
Maximum renewable life: 1 day 16:00:00
Last modified: Thu Aug 15 13:30:30 PST 2013 (host/admin@EXAMPLE.COM)
Last successful authentication: [never]
Last failed authentication: [never]
Failed password attempts: 0
Number of keys: 1
Key: vno 1, AES-256 CTS mode with 96-bit SHA-1 HMAC, no salt
Key: vno 1, AES-128 CTS mode with 96-bit SHA-1 HMAC, no salt
Key: vno 1, Triple DES with HMAC/sha1, no salt
Key: vno 1, ArcFour with HMAC/md5, no salt
Attributes: REQUIRES_HW_AUTH
Policy: [none]
kadmin: quit
```

EXAMPLE 24 Using the `gkadmin` GUI to List and Set Defaults for Kerberos Principals

In this example, the administrator wants to show a new administrator the list of principals and their attributes, so uses the `gkadmin` GUI. The administrator also sets new defaults for future principals.

```
# /usr/sbin/gkadmin
```

The window displays the Principal Name, Password, Realm, and Master KDC fields.

The administrator navigates to the list of all principal names, then shows the new administrator how to use the case-sensitive filter.

Then, the administrator clicks the Edit menu and chooses Properties. After clicking Require Password Change, the administrator applies the change.

To see the attributes for a current principal, the administrator navigates to the list of principals and chooses a principal from the list. The first dialog box displays basic attributes. The administrator clicks the Next button to display all the attributes.

Creating a New Kerberos Principal

If a new principal requires a new policy, you must create the new policy before you create the principal. For policy creation, see Example 31, “Creating a New Kerberos Password Policy,” on page 168. Most Kerberos policies specify password requirements.

EXAMPLE 25 Creating a New Kerberos Principal

The following example creates a new principal called `pak` and sets the principal's policy to `testuser`. The other required values, such as encryption type, use default values.

```
# /usr/sbin/kadmin
kadmin: add_principal -policy testuser pak
Enter password for principal "pak@EXAMPLE.COM": xxxxxxxx
Re-enter password for principal "pak@EXAMPLE.COM": xxxxxxxx
Principal "pak@EXAMPLE.COM" created.
kadmin: quit
```

Typically, few users are privileged to administer the Kerberos database. If this new principal needs administrative privileges, continue with "Modifying Principals' Kerberos Administration Privileges" on page 166.

Modifying a Kerberos Principal

The following examples show how to modify the password attributes of a Kerberos principal.

EXAMPLE 26 Modifying the Maximum Number of Password Retries for a Kerberos Principal

```
# /usr/sbin/kadmin
kadmin: modify_principal jdb
kadmin: maxtries=5
kadmin: quit
```

EXAMPLE 27 Modifying the Password of a Kerberos Principal

```
# /usr/sbin/kadmin
kadmin: change_password jdb
Enter password for principal "jdb": xxxxxxxx
Re-enter password for principal "jdb": xxxxxxxx
Password for "jdb@EXAMPLE.COM" changed.
kadmin: quit
```

Deleting a Kerberos Principal

The following example shows how to delete a principal. Comply with the online message before continuing.

```
# /usr/sbin/kadmin
kadmin: delete_principal pak
Are you sure you want to delete the principal "pak@EXAMPLE.COM"? (yes/no): yes
```

```
Principal "pak@EXAMPLE.COM" deleted.
Make sure that you have removed this principal from all ACLs before reusing.
kadmin: quit
```

To remove this principal from all ACLs, see “Modifying Principals' Kerberos Administration Privileges” on page 166.

Duplicating a Kerberos Principal by Using the `gkadmin` GUI

You can duplicate a principal by using the `gkadmin` GUI. No command-line equivalent exists for this procedure.

1. After starting the GUI with the `/usr/sbin/gkadmin` command, click the Principals tab and select a principal to duplicate.
2. Click the Duplicate button. This action duplicates all of the attributes of the selected principal except for the principal name and password.
3. Specify a new name and password, then click Save to create an exact duplicate.
4. To modify the duplicate, specify values for the principal's attributes.

   Three windows contain attribute information. Choose Context-Sensitive Help from the Help menu to get information about the various attributes in each window.

Typically, few users are privileged to administer the Kerberos database. If this new principal needs administrative privileges, continue with “Modifying Principals' Kerberos Administration Privileges” on page 166.

Modifying Principals' Kerberos Administration Privileges

The few users who are privileged to administer the Kerberos database are specified in the Kerberos access control list (ACL). This list is maintained as entries in a file, `/etc/krb5/kadm5.acl`. For more information, see the `kadm5.acl(4)` man page.

To add entries to the `kadm5.acl` file, use the `pfedit` command.

```
# pfedit /usr/krb5/kadm5.acl
```

An entry in the `kadm5.acl` file has the following format:

principal privileges [*principal-target*]

- *principal* – Specifies the principal to which the privileges are granted. Any part of the principal name can include the '*' wildcard, which is useful for providing the same

privileges for a group of principals. For example, if you want to specify all principals with the `admin` instance, you would use `*/admin@`*realm*.

Note that a common use of an `admin` instance is to grant separate privileges (such as administrative access to the Kerberos database) to a separate Kerberos principal. For example, the user `jdb` might have a principal for administrative use, called `jdb/admin`. By having two principals, the user `jdb` obtains `jdb/admin` tickets only when administrative privileges are required.

- *privileges* – Specifies which operations can be performed by the principal. This field consists of a string of one or more of the following list of characters. If the character is uppercase or unspecified, then the operation is disallowed. If the character is lowercase, then the operation is allowed.
  - [`A`]`a` – [Dis]allows the addition of principals or policies.
  - [`C`]`c` – [Dis]allows the changing of passwords for principals.
  - [`D`]`d` – [Dis]allows the deletion of principals or policies.
  - [`I`]`i` – [Dis]allows inquiries to the Kerberos database.
  - [`L`]`l` – [Dis]allows the listing of principals or policies.
  - [`M`]`m` – [Dis]allows the modification of principals or policies.
  - `x` or `*` – Allows all privileges (`admcil`).
- *principal-target* – When a principal is specified in this field, the principal's privileges apply to this principal only. To assign privileges to a group of principals, use the '*' wildcard in *principal-target*.

EXAMPLE 28 Modifying the Privileges of a Kerberos Principal

The following entry in the `kadm5.acl` file gives any principal in the `EXAMPLE.COM` realm with the `admin` instance all the privileges on the Kerberos database:

```
*/admin@EXAMPLE.COM *
```

The following entry in the `kadm5.acl` file gives the `jdb@EXAMPLE.COM` principal the privileges to list and inquire about any principal that has the `root` instance.

```
jdb@EXAMPLE.COM li */root@EXAMPLE.COM
```

Administering Kerberos Policies

This section provides examples of using the `kadmin` command and the `gkadmin` GUI to administer Kerberos policies. Most Kerberos policies specify password requirements.

The steps for administering policies are similar to the steps for administering principals. For more information, see the `kadmin`(1M) and `gkadmin`(1M) man pages.

EXAMPLE 29 Viewing the List of Kerberos Policies

In this example, the `list_policies` subcommand is used to list all the policies that match `*user*`. Without an argument, `list_policies` lists all the policies that are defined in the Kerberos database.

```
# kadmin
kadmin: list_policies *user*
testuser
financeuser
kadmin: quit
```

EXAMPLE 30 Viewing the Attributes of a Kerberos Policy

In this example, the `get_policy` subcommand is used to view the attributes of the `financeuser` policy.

```
# /usr/sbin/kadmin.local
kadmin.local: get_policy financeuser
Policy: financeuser
Maximum password life: 13050000
Minimum password life: 10886400
Minimum password length: 8
Minimum number of password character classes: 2
Number of old keys kept: 3
Reference count: 8
Maximum password failures before lockout: 5
Password failure count reset interval: 200
Password lockout duration: 300
kadmin: quit
```

The Reference count is the number of principals that are assigned this policy.

EXAMPLE 31 Creating a New Kerberos Password Policy

In this example, the `add_policy` subcommand is used to create the `build11` policy. This policy requires at least three character classes in a password.

```
# kadmin
kadmin: add_policy -minclasses 3 build11
kadmin: quit
```

EXAMPLE 32 Handling a Kerberos Account Lockout Policy

In this example, three authentication failures during a span of `300` seconds triggers an account lockout of `900` seconds.

```
kadmin: add_policy -maxfailure 3 -failurecountinterval "300 seconds"\
```

```
-lockoutduration "900 seconds" default
```

To release the lock within the 15 minutes requires administrative action.

```
# /usr/sbin/kadmin -p kws/admin
Enter password: xxxxxxxx
kadmin: modify_principal -unlock principal
```

EXAMPLE 33 Modifying a Kerberos Policy

In this example, the `modify_policy` subcommand is used to change the minimum length of a password to eight characters for the `build11` policy.

```
# kadmin
kadmin: modify_policy -minlength 8 build11
kadmin: quit
```

EXAMPLE 34 Deleting a Kerberos Policy

In this example, the `delete_policy` subcommand is used to delete the `build11` policy.

1. The administrator removes the policy from all principals that use it.

   ```
   # kadmin
   kadmin: modify_principal -policy build11 *admin*
   ```

2. Then, the administrator deletes the policy.

   ```
   kadmin: delete_policy build11
   Are you sure you want to delete the policy "build11"? (yes/no): yes
   kadmin: quit
   ```

The `delete_policy` command fails if the policy is assigned to a principal.

EXAMPLE 35 Duplicating a Kerberos Policy by Using the `gkadmin` GUI

In the `gkadmin` GUI, you can duplicate a selected policy by clicking the Duplicate button. In the Policy Name field, name the new policy. You can also modify the policy attributes that you duplicated. The steps are similar to the steps in “Duplicating a Kerberos Principal by Using the `gkadmin` GUI” on page 166.

Administering Keytab Files

Every host that provides a service must have a local file, called a keytab file, which is short for “key table”. The keytab contains the principal for the appropriate service, called a service

key. A service key is used by a service to authenticate itself to the KDC and is known only by Kerberos and the service itself. For example, if you have a Kerberized NFS server, that server must have a keytab file that contains the service key for the `nfs` service principal.

To add a service key to a keytab file, you add the appropriate service principal to a host's keytab file by using the `ktadd` command in a `kadmin` process. Because you are adding a service principal to a keytab file, the principal must already exist in the Kerberos database. On application servers that provide Kerberized services, the keytab file is `/etc/krb5/krb5.keytab` by default.

A keytab is analogous to a user's password. Just as users must protect their passwords, application servers must protect their keytab files. You should always store keytab files on a local disk, and make them readable only by the `root` user. Also, you should never send a keytab file over an unsecured network.

Special circumstances can require you to add a `root` principal to a host's keytab file. If you want a user on a Kerberos client to mount Kerberized NFS file systems that require `root`-equivalent access, you must add the client's `root` principal to the client's keytab file. Otherwise, users must use the `kinit` command as `root` to obtain credentials for the client's `root` principal whenever they want to mount a Kerberized NFS file system with `root` access, even when they are using the automounter.

Caution - Mounting NFS servers as `root` is a security risk.

You can also use the `ktutil` command to administer keytab files. This interactive command enables you to manage a local host's keytab file without having Kerberos administration privileges, because this command does not interact with the Kerberos database as `kadmin` does. After a principal is added to a keytab file, you can use `ktutil` to view the keylist in a keytab file or to temporarily disable authentication for a service.

Note - When you change a principal in a keytab file by using the `ktadd` command in `kadmin`, a new key is generated and added to the keytab file.

Adding a Kerberos Service Principal to a Keytab File

You can add a principal to a keytab file after ensuring that the principal exists in the Kerberos database. For more information, see "Viewing Kerberos Principals and Their Attributes" on page 163.

On the host that needs a principal added to its keytab file, you run the `ktadd` command in a `kadmin` process. For more information, see the `kadmin`(1M) man page.

```
# /usr/sbin/kadmin
kadmin: ktadd [-e enctype] [-k keytab] [-q] [principal | -glob principal-exp]
```

-e *enctype*	Overrides the list of encryption types defined in the `krb5.conf` file.
-k *keytab*	Specifies the keytab file. By default, `/etc/krb5/krb5.keytab` is used.
-q	Displays less verbose information.
principal	Specifies the principal to be added to the keytab file. You can add the following service principals: `host`, `root`, `nfs`, and `ftp`.
-glob *principal-exp*	Specifies the principal expressions. All principals that match the *principal-exp* are added to the keytab file. The rules for principal expression are the same as for the `list_principals` command of `kadmin`. For the possible expressions, review the `expression` definition in the `kadmin`(1M) man page

EXAMPLE 36 Adding a Service Principal to a Keytab File

In this example, `denver`'s `host` principal is added to `denver`'s keytab file so that the KDC can authenticate `denver`'s network services.

```
denver # /usr/sbin/kadmin
kadmin: ktadd host/denver.example.com
Entry for principal host/denver.example.com with kvno 3,
encryption type AES-256 CTS
mode with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/denver.example.com with kvno 3,
encryption type AES-128 CTS mode
with 96-bit SHA-1 HMAC added to keytab WRFILE:/etc/krb5/krb5.keytab.
Entry for principal host/denver.example.com with kvno 3,
encryption type Triple DES cbc mode
with HMAC/sha1 added to keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
```

Removing a Service Principal From a Keytab File

You can remove a principal from a keytab file. On the host that needs a principal removed from its keytab file, you first view the list of principals. See “Displaying the Principals in a Keytab File” on page 172.

Then, you run the `ktadd` command in a `kadmin` process. For more information, see the `kadmin`(1M) man page.

```
# /usr/sbin/kadmin
kadmin: ktremove [-k keytab] [-q] principal [kvno | all | old ]
```

-k *keytab*	Specifies the keytab file. By default, `/etc/krb5/krb5.keytab` is used.
`-q`	Displays less verbose information.
principal	Specifies the principal to be removed from the keytab file.
kvno	Removes all entries for the specified principal whose key version number matches *kvno*.
`all`	Removes all entries for the specified principal.
`old`	Removes all entries for the specified principal except those principals with the highest key version number.

EXAMPLE 37 Removing a Service Principal From a Keytab File

In this example, `denver`'s `host` principal is removed from `denver`'s keytab file.

```
denver # /usr/sbin/kadmin
kadmin: ktremove host/denver.example.com@EXAMPLE.COM
kadmin: Entry for principal host/denver.example.com@EXAMPLE.COM with kvno 3
removed from keytab WRFILE:/etc/krb5/krb5.keytab.
kadmin: quit
```

Displaying the Principals in a Keytab File

On the host with the keytab file, run the `ktutil` command, read the keytab, then list the principals. Principals are also known as the *keylist*.

Note - Although you can create keytab files that are owned by other users, using the default location for the keytab file requires `root` ownership.

EXAMPLE 38 Displaying the Keylist (Principals) in a Keytab File

The following example displays the keylist in the `/etc/krb5/krb5.keytab` file on the `denver` host.

```
denver # /usr/bin/ktutil
ktutil: read_kt /etc/krb5/krb5.keytab
ktutil: list
slot KVNO Principal
```

```
---- ---- ------------------------------------
1    5 host/denver@EXAMPLE.COM
ktutil: quit
```

Temporarily Disabling a Kerberos Service on a Host

At times, you might need to temporarily disable the authentication mechanism for a service, such as `ssh` or `ftp`, on a network application server. For example, you might want to stop users from logging in to a system while you are performing maintenance procedures.

Note - By default, most services require authentication. If a service does not require authentication, then disabling authentication has no effect. The service is still available.

▼ How to Temporarily Disable Authentication for a Kerberos Service on a Host

The `ktutil` command enables a user without `kadmin` privileges to disable a service. This user can also restore the service. For more information, see the `ktutil(1)` man page.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Save the current keytab file to a temporary file.**

   You use this temporary file to re-enable authentication in Step 9.

2. **On the host with the keytab file, start the `ktutil` command.**

   Note - Although you can create keytab files that are owned by other users, using the default location for the keytab file requires `root` ownership.

   ```
   # /usr/bin/ktutil
   ```

3. **Read the keytab file into the keylist buffer.**

   ```
   ktutil: read_kt keytab
   ```

4. **Display the keylist buffer.**

   ```
   ktutil: list
   ```

The current keylist buffer is displayed. Note the slot number for the service that you want to disable.

5. **Temporarily disable a host's service by removing the specific service principal from the keylist buffer.**

   ```
   ktutil: delete_entry slot-number
   ```

   Where *slot-number* specifies the slot number of the service principal to be deleted in the `list` output.

6. **Write the keylist buffer to a new keytab file.**

   ```
   ktutil: write_kt new-keytab
   ```

7. **Quit the `ktutil` command.**

   ```
   ktutil: quit
   ```

8. **Use the new keytab file to disable the principal's authentication.**

   ```
   # mv new-keytab keytab
   ```

9. **(Optional) To re-enable the service, copy the temporary keytab file back to its original location.**

   ```
   # cp original-keytab keytab
   ```

Example 39 Temporarily Disabling a Kerberos Host

In this example, the `host` service on the `denver` host is temporarily disabled. To re-enable the host service on `denver`, the administrator copies the saved keytab file to its original location.

```
denver # cp /etc/krb5/krb5.keytab /etc/krb5/krb5.keytab.save
denver # /usr/bin/ktutil
ktutil:read_kt /etc/krb5/krb5.keytab
ktutil:list
slot KVNO Principal
---- ---- ---------------------------------------
1    8 root/denver@EXAMPLE.COM
2    5 host/denver@EXAMPLE.COM
ktutil:delete_entry 2
ktutil:list
slot KVNO Principal
---- ---- ---------------------------------------
1    8 root/denver@EXAMPLE.COM
ktutil:write_kt /etc/krb5/nodenverhost.krb5.keytab
ktutil: quit
denver # cp /etc/krb5/nodenverhost.krb5.keytab /etc/krb5/krb5.keytab
```

The host is unavailable until the user copies the saved file back to its original location.

```
denver # cp /etc/krb5/krb5.keytab.save /etc/krb5/krb5.keytab
```

♦ ♦ ♦ CHAPTER 6

Using Kerberos Applications

This chapter explains how to use the Kerberos-based commands and services that are provided in Oracle Solaris. You should already be familiar with these commands in their original versions before you read about them here.

Because this chapter is intended for application users, it includes information about tickets: obtaining, viewing, and destroying them. This chapter also includes information about choosing or changing a Kerberos password.

This chapter covers the following topics:

For an overview of Kerberos, see Chapter 2, "About the Kerberos Service".

Kerberos Ticket Management

This section explains how to obtain, view, and destroy tickets. For an introduction to tickets, see "How the Kerberos Service Works" on page 40.

In Oracle Solaris, Kerberos is built into the `login` command. However, to obtain tickets automatically, you must configure the PAM service for the applicable login services. For more information, see the `pam_krb5`(5) man page.

The `ssh` command can be set up to forward copies of your tickets to the other systems, so you do not have to explicitly ask for tickets to get access to those systems. For security reasons, your administrator might prevent this. For more information, see the discussion about agent forwarding in the `ssh`(1) man page.

For information about ticket lifetimes, see "Ticket Lifetimes" on page 190.

Creating a Kerberos Ticket

If PAM is configured properly, a ticket is created automatically when you log in, and you need not do anything special to obtain a ticket. However, you might need to create a ticket if your ticket expires. Also, you might need to use a different principal besides your default principal, for example, if you use `ssh -l` to log in to a system as someone else.

To create a ticket, use the `kinit` command.

```
% /usr/bin/kinit
```

The `kinit` command prompts you for your password. For the full syntax of the `kinit` command, see the `kinit(1)` man page.

EXAMPLE 40 Creating a Kerberos Ticket

This example shows a user, `kdoe`, creating a ticket on her own system.

```
% kinit
Password for kdoe@CORP.EXAMPLE.COM: xxxxxxxx
```

In this example, the user `kdoe` creates a ticket that is valid for three hours with the `-l` option.

```
% kinit -l 3h kdoe@EXAMPLE.ORG
Password for kdoe@EXAMPLE.ORG: xxxxxxxx
```

Viewing Kerberos Tickets

Not all tickets are alike. For example, one ticket might be `forwardable`, another ticket might be `postdated`, and a third ticket might be both forwardable and postdated. You can see which tickets you have, and what their attributes are, by using the `klist` command with the `-f` option:

```
% /usr/bin/klist -f
```

The following symbols indicate the attributes that are associated with each ticket, as displayed by `klist`:

A	Preauthenticated
D	Postdatable
d	Postdated

F	Forwardable
f	Forwarded
I	Initial
i	Invalid
P	Proxiable
p	Proxy
R	Renewable

"Types of Tickets" on page 188 describes the various attributes that a ticket can have.

EXAMPLE 41 Viewing Kerberos Tickets

This example shows that the user `kdoe` has an `initial` ticket, which is `forwardable` (F) and `postdated` (d), but not yet validated (i).

```
% /usr/bin/klist -f
Ticket cache: /tmp/krb5cc_74287
Default principal: kdoe@EXAMPLE.COM

Valid starting                Expires                Service principal
09 Feb 14 15:09:51  09 Feb 14 21:09:51  nfs/EXAMPLE.COM@EXAMPLE.COM
renew until 10 Feb 14 15:12:51, Flags: Fdi
```

The following example shows that the user `kdoe` has two tickets that were `forwarded` (f) to the user's host from another host. The tickets are also `forwardable` (F).

```
% klist -f
Ticket cache: /tmp/krb5cc_74287
Default principal: kdoe@EXAMPLE.COM

Valid starting                Expires                Service principal
07 Feb 14 06:09:51  09 Feb 14 23:33:51  host/EXAMPLE.COM@EXAMPLE.COM
renew until 10 Feb 14 17:09:51, Flags: fF

Valid starting                Expires                Service principal
08 Feb 14 08:09:51  09 Feb 14 12:54:51  nfs/EXAMPLE.COM@EXAMPLE.COM
renew until 10 Feb 14 15:22:51, Flags: fF
```

The following example shows how to display the encryption types of the session key and the ticket by using the -e option. The -a option is used to map the system address to a system name if the name service can do the conversion.

```
% klist -fea
```

```
Ticket cache: /tmp/krb5cc_74287
Default principal: kdoe@EXAMPLE.COM

Valid starting                Expires                Service principal
07 Feb 14 06:09:51  09 Feb 14 23:33:51  krbtgt/EXAMPLE.COM@EXAMPLE.COM
renew until 10 Feb 14 17:09:51, Flags: FRIA
Etype(skey, tkt): AES-256 CTS mode with 96-bit SHA-1 HMAC
Addresses: client.example.com
```

Destroying Kerberos Tickets

If you want to destroy all Kerberos tickets acquired during your current session, use the `kdestroy` command. The command destroys your credential cache, which destroys all your credentials and tickets. While this destruction is not usually necessary, running `kdestroy` reduces the chance of the credential cache being compromised during times that you are not logged in.

To destroy your tickets, use the `kdestroy` command.

```
% /usr/bin/kdestroy
```

The `kdestroy` command destroys *all* your tickets. You cannot use this command to selectively destroy a particular ticket.

If you are going to be away from your system, you should either use the `kdestroy` command or lock the screen with a screen saver.

Kerberos Password Management

With the Kerberos service configured, you now have two passwords: your regular Oracle Solaris password and a Kerberos password. You can make both passwords the same, or they can be different.

Changing Your Password

If PAM is properly configured, you can change your Kerberos password in two ways.

- Use the `passwd` command. With the Kerberos service configured, the `passwd` command also automatically prompts for a new Kerberos password.

By using `passwd`, you can set both your UNIX and Kerberos passwords at the same time. However, you can change only one password with `passwd` and leave the other password untouched.

Note - The behavior of `passwd` depends on how the PAM module is configured. You might be required to change both passwords in some configurations. For some sites, the UNIX password must be changed, while other sites require the Kerberos password to change.

- Use the `kpasswd` command. `kpasswd` changes only Kerberos passwords. You must use `passwd` if you want to change your UNIX password.

  A primary use for `kpasswd` is to change a password for a Kerberos principal that is not a valid UNIX user. For example, `jdoe/admin` is a Kerberos principal but not an actual UNIX user, so you must use `kpasswd` to change the password.

After you change your password, the password must propagate through the network. Depending on the size of the Kerberos network, the time that is required for the propagation might range from a few minutes to an hour or more. If you need to get new Kerberos tickets shortly after you change your password, try the new password first. If the new password doesn't work, try again using the old password.

Kerberos *policy* defines the criteria for passwords. A policy can be set for each user or a default policy can apply. For information about policies, see "Administering Kerberos Policies" on page 167. For an example that lists the criteria that can be set for Kerberos passwords, see Example 30, "Viewing the Attributes of a Kerberos Policy," on page 168. Password character classes are lowercase, uppercase, numbers, punctuation, and all other characters.

Remote Logins in Kerberos

As in Oracle Solaris, Kerberos provides the `ssh` command for remote access. After installation, the `ssh` command is the only network service that accepts network requests. Therefore, other network services that are modified for Kerberos, such as `rlogin` and `telnet`, are disabled. For more information, see "Kerberos Network Programs" on page 45 and "Kerberos User Commands" on page 181.

Kerberos User Commands

The Kerberos V5 product is a *single sign-on* system, which means that you only have to type your password only once when using network applications. The Kerberos V5 applications perform authentication and optionally encryption for you because Kerberos has been built into

each of a suite of existing, familiar network applications. The Kerberos V5 applications are versions of existing UNIX network applications with Kerberos features added.

Note - In general, the `ssh` application should suffice for your remote login needs. See the `ssh`(1) man page. For information about enabling a deprecated network application, see "Kerberos Network Programs" on page 45.

For the Kerberos features in existing network applications, see the following man pages, and specifically the `OPTIONS` sections:

- `ftp`(1)
- `rcp`(1)
- `rlogin`(1)
- `rsh`(1)
- `telnet`(1)

When you use a Kerberized application to connect to a remote system, the application, the KDC, and the remote system perform a set of rapid negotiations. When these negotiations are completed and your application has proven your identity on your behalf to the remote system, then the remote system grants you access.

For examples of using deprecated remote login commands, see "Using Kerberos Applications (Tasks)" in *Oracle Solaris 11.1 Administration: Security Services*.

♦ ♦ ♦ CHAPTER 7

Kerberos Service Reference

This chapter lists many of the files, commands, and daemons that are part of the Kerberos product.

This chapter covers the following reference information:

Kerberos Files

The following files are used by the Kerberos service.

`~/.gkadmin`	Default values for creating new principals in the Kerberos Administration GUI
`~/.k5login`	List of principals that grant access to a Kerberos account
`/etc/krb5/kadm5.acl`	Kerberos access control list file, which includes principal names of KDC administrators and their Kerberos administration privileges
`/etc/krb5/kdc.conf`	KDC configuration file
`/etc/krb5/kpropd.acl`	Kerberos database propagation configuration file
`/etc/krb5/krb5.conf`	Kerberos realm configuration file

`/etc/krb5/krb5.keytab`	Keytab file for network application servers
`/etc/krb5/warn.conf`	Kerberos ticket expiration warning and automatic renewal configuration file
`/etc/pam.conf`	PAM configuration file
`/tmp/krb5cc_`*UID*	Default credentials cache, where *UID* is the decimal UID of the user
`/tmp/ovsec_adm.`*xxxxxx*	Temporary credentials cache for the lifetime of the password changing operation, where *xxxxxx* is a random string
`/var/krb5/.k5.`*REALM*	KDC stash file, which contains a copy of the KDC master key
`/var/krb5/kadmin.log`	Log file for `kadmind`
`/var/krb5/kdc.log`	Log file for the KDC
`/var/krb5/principal`	Kerberos principal database
`/var/krb5/principal.kadm5`	Kerberos administrative database, which contains policy information
`/var/krb5/principal.kadm5.lock`	Kerberos administrative database lock file
`/var/krb5/principal.ok`	Kerberos principal database initialization file that is created when the Kerberos database is initialized successfully
`/var/krb5/principal.ulog`	Kerberos update log, which contains updates for incremental propagation
`/var/krb5/slave_datatrans`	Backup file of the KDC that the `kprop_script` script uses for propagation
`/var/krb5/slave_datatrans_`*slave*	Temporary dump file that is created when full updates are made to the specified slave
`/var/user/`*$USER*`/krb-warn.conf`	Per-user ticket expiration warning and automatic renewal configuration file

Kerberos Commands

The following commands are included in the Kerberos product. The commands are listed by their man page.

`ftp`(1)	File Transfer Protocol application
`kdestroy`(1)	Destroys Kerberos tickets
`kinit`(1)	Obtains and caches Kerberos ticket-granting tickets
`klist`(1)	Displays current Kerberos tickets
`kpasswd`(1)	Changes a Kerberos password
`ktutil`(1)	Manages Kerberos keytab files
`kvno`(1)	Lists key version numbers for Kerberos principals
`rcp`(1)	Remote file copy application
`rlogin`(1)	Remote login application
`rsh`(1)	Remote shell application
`scp`(1)	Secure remote file copy application
`sftp`(1)	Secure file transfer application
`ssh`(1)	Secure shell for remote login
`telnet`(1)	Kerberized `telnet` application
`kprop`(1M)	Kerberos database propagation program
`gkadmin`(1M)	Kerberos database administration GUI program, which is used to manage principals and policies
`gsscred`(1M)	Manage `gsscred` table entries
`kadmin`(1M)	Remote Kerberos database administration program that requires Kerberos authentication, which is used to manage principals, policies, and keytab files

kadmin.local(1M)	Local Kerberos database administration program that runs on the master KDC which is used to manage principals, policies, and keytab files
kclient(1M)	Kerberos client installation script which is used with or without an installation profile
kdb5_ldap_util(1M)	Creates LDAP containers for Kerberos databases
kdb5_util(1M)	Creates Kerberos databases and stash files
kdcmgr(1M)	Configures Kerberos master and slave KDCs
kproplog(1M)	Lists a summary of update entries in the update log

Kerberos Daemons

The following daemons are used by the Kerberos product.

/usr/lib/inet/proftpd	Secure File Transfer Protocol daemon
/usr/lib/ssh/sshd	Secure shell daemon
/usr/lib/krb5/kadmind	Kerberos database administration daemon
/usr/lib/krb5/kpropd	Kerberos database propagation daemon
/usr/lib/krb5/krb5kdc	Kerberos ticket processing daemon
/usr/lib/krb5/ktkt_warnd	Kerberos ticket expiration warning and automatic renewal daemon
/usr/sbin/in.rlogind	Remote login daemon
/usr/sbin/in.rshd	Remote shell daemon
/usr/sbin/in.telnetd	telnet daemon

Kerberos Terminology

The following Kerberos terms are used throughout the Kerberos documentation. To grasp Kerberos concepts, an understanding of these terms is essential.

Kerberos-Specific Terminology

You need to understand the terms in this section in order to administer KDCs.

The *Key Distribution Center* or *KDC* is the component of Kerberos that is responsible for issuing credentials. These credentials are created by using information that is stored in the KDC database. Each realm needs at least two KDCs, a master and at least one slave. All KDCs generate credentials, but only the master KDC handles any changes to the KDC database.

A *stash file* contains the master key for the KDC. This key is used when a server is rebooted to automatically authenticate the KDC before starting the `kadmind` and `krb5kdc` commands. Because this file includes the master key, the file and any backups of the file should be kept secure. The file is created with read-only permissions for `root`. To keep the file secure, do not change the permissions. If the file is compromised, then the key could be used to access or modify the KDC database.

Authentication-Specific Terminology

You need to know the terms in this section to understand the authentication process. Programmers and system administrators should be familiar with these terms.

A *client* is the software that runs on a user's workstation. The Kerberos software that runs on the client makes many requests during this process. Therefore, differentiating the actions of this software from the user is important.

The terms *server* and *service* are often used interchangeably. To clarify, the term *server* is used to define the physical machine that Kerberos software is running on. The term *service* corresponds to a particular function that is being supported on a server (for example, `ftp` or `nfs`). Documentation often mentions servers as part of a service, but this definition clouds the meaning of the terms. Therefore, the term *server* refers to the physical machine. The term *service* refers to the software.

The Kerberos product uses two types of keys. One type of key is a password-derived key, which is given to each user principal and is known only to the user and to the KDC. The other type of key is a random key that is not associated with a password and so is not suitable for use by user principals. Random keys are typically used for service principals that have entries in a

keytab and whose session keys are generated by the KDC. Service principals can use random keys since the service can access the key in the keytab which allows it to run non-interactively. Session keys are generated by the KDC (and shared between the client and service) to provide secure transactions between a client and a service.

A *ticket* is an information packet that is used to securely pass the identity of a user to a server or service. A ticket is valid for only a single client and a particular service on a specific server. A ticket contains:

- Principal name of the service
- Principal name of the user
- IP address of the user's host
- Timestamp
- Value which defines the lifetime of the ticket
- Copy of the session key

All of this data is encrypted in the server's service key. The KDC issues the ticket embedded in a credential. After a ticket has been issued, it can be reused until the ticket expires.

A *credential* is a packet of information that includes a ticket and a matching session key. The credential is encrypted with the requesting principal's key. Typically, the KDC generates a credential in response to a ticket request from a client.

An *authenticator* is information that the server uses to authenticate the client user principal. An authenticator includes the principal name of the user, a timestamp, and other data. Unlike a ticket, an authenticator is used only once, usually when access to a service is requested. An authenticator is encrypted by using the session key shared by the client and server. Typically, the client creates the authenticator and sends it with the server's or service's ticket in order to authenticate to the server or service.

Types of Tickets

Tickets have properties that govern how they can be used. These properties are assigned to the ticket when it is created, although you can modify a ticket's properties later. For example, a ticket can change from being `forwardable` to being `forwarded`. You can view ticket properties with the `klist` command. See "Viewing Kerberos Tickets" on page 178.

Tickets can be described by one or more of the following terms:

Forwardable/ forwarded
: A forwardable ticket can be sent from one host to another host, obviating the need for a client to reauthenticate itself. For example, if the user `david` obtains a forwardable ticket while on user `jennifer`'s host, `david` can log in to his own host without having to get a new ticket (and thus authenticate himself again). See Example 40, "Creating a Kerberos Ticket," on page 178 for an example of a forwardable ticket.

Initial	An initial ticket is a ticket that is issued directly, not based on a ticket-granting ticket. Some services, such as applications that change passwords, can require tickets to be marked “initial” in order to assure themselves that the client can demonstrate a knowledge of its secret key. An initial ticket indicates that the client has recently authenticated itself instead of relying on a ticket-granting ticket, which might be older.
Invalid	An invalid ticket is a postdated ticket that has not yet become usable. An invalid ticket will be rejected by an application server until it becomes validated. To be validated, a ticket must be presented to the KDC by the client in a ticket-granting service request, with the `VALIDATE` flag set, after its start time has passed.
Postdatable/ postdated	A postdated ticket is a ticket that does not become valid until some specified time after its creation. Such a ticket is useful, for example, for batch jobs that are intended to be run late at night, because the ticket, if stolen, cannot be used until the batch job is to be run. When a postdated ticket is issued, it is issued as invalid and remains that way until its start time has passed, and the client requests validation by the KDC. A postdated ticket is normally valid until the expiration time of the ticket-granting ticket. However, if the ticket is marked renewable, its lifetime is normally set to be equal to the duration of the full life of the ticket-granting ticket.
Proxiable/proxy	At times, principal might need to allow a service to perform an operation on its behalf. The principal name of the proxy must be specified when the ticket is created. Oracle Solaris does not support proxiable or proxy tickets. A proxiable ticket is similar to a forwardable ticket except that it is valid only for a single service. A forwardable ticket grants the service the complete use of the client's identity. A forwardable ticket can therefore be thought of as a sort of super-proxy.
Renewable	Because tickets with very long lives are a security risk, tickets can be designated as renewable. A renewable ticket has two expiration times: the time at which the current instance of the ticket expires, and the maximum lifetime for any ticket, which is one week. If a client wants to continue to use a ticket, the client renews it before the first expiration occurs. For example, suppose a ticket can be valid for one hour, with all tickets having a maximum lifetime of 10 hours. If the client that is holding the ticket wants to keep it for more than an hour, the client must renew it within that hour. When a ticket reaches the maximum ticket lifetime (10 hours), it automatically expires and cannot be renewed.

For information about how to view the attributes of tickets, see “Viewing Kerberos Tickets” on page 178.

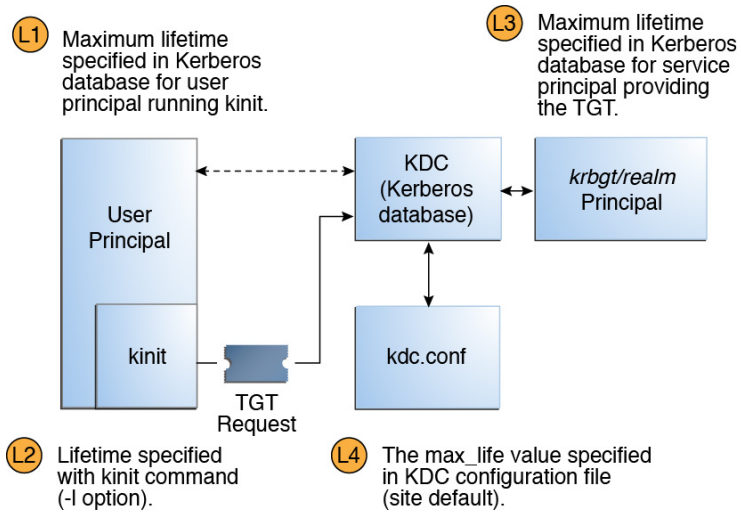
Ticket Lifetimes

When a principal obtains a ticket, including a ticket-granting ticket (TGT), the ticket's lifetime is set as the smallest of the following lifetime values:

- The lifetime value that is specified by the `-l` option of `kinit`, if `kinit` is used to get the ticket. By default, `kinit` uses the maximum lifetime value.
- The maximum lifetime value (`max_life`) that is specified in the `kdc.conf` file.
- The maximum lifetime value that is specified in the Kerberos database for the service principal that provides the ticket. In the case of `kinit`, the service principal is `krbtgt/`*realm*.
- The maximum lifetime value that is specified in the Kerberos database for the user principal that requests the ticket.

The following figure shows how a TGT's lifetime is determined and where the four lifetime values originate. When any principal obtains a ticket, the ticket's lifetime is determined similarly. The two differences are that `kinit` does not provide a lifetime value, and the service principal that provides the ticket provides a maximum lifetime value instead of the `krbtgt/`*realm* principal.

FIGURE 13 How a TGT's Lifetime Is Determined



Ticket Lifetime = Minimum value of L1, L2, L3, and L4

The renewable ticket lifetime is determined from the minimum of four renewable lifetime values, as follows:

- The renewable lifetime value that is specified by the `-r` option of `kinit`, if `kinit` is used to obtain or renew the ticket.
- The maximum renewable lifetime value (`max_renewable_life`) that is specified in the `kdc.conf` file.
- The maximum lifetime renewable value that is specified in the Kerberos database for the service principal that provides the ticket. In the case of `kinit`, the service principal is `krbtgt/`*realm*.
- The maximum lifetime renewable value that is specified in the Kerberos database for the user principal that requests the ticket.

Kerberos Principal Names

Each ticket is identified by a principal name. The principal name can identify a user or a service. The following examples show typical principal names:

`changepw/kdc1.example.com@EXAMPLE.COM`	A principal for the master KDC server that allows access to the KDC when you are changing passwords.
`clntconfig/admin@EXAMPLE.COM`	A principal that is used by the `kclient` installation utility.
`ftp/boston.example.com@EXAMPLE.COM`	A principal that is used by the `ftp` service. This principal can be used instead of a `host` principal.
`host/boston.example.com@EXAMPLE.COM`	A principal that is used by the Kerberized applications (`klist` and `kprop`, for example) and services (such as `ftp` and `telnet`). This principal is called a `host` or service principal. The principal is used to authenticate NFS mounts. This principal is also used by a client to verify that the TGT that is issued to the client is from the correct KDC.
`K/M@EXAMPLE.COM`	The master key name principal. One master key name principal is associated with each master KDC.
`kadmin/history@EXAMPLE.COM`	A principal that includes a key used to keep password histories for other principals. Each master KDC has one of these principals.
`kadmin/kdc1.example.com@EXAMPLE.COM`	A principal for the master KDC server that allows access to the KDC by using `kadmind`.

`kadmin/changepw.example.com@EXAMPLE.COM`	A principal that is used to accept password change requests from clients that are not running an Oracle Solaris release.
`krbtgt/EXAMPLE.COM@EXAMPLE.COM`	This principal is used when you generate a ticket-granting ticket.
`krbtgt/EAST.EXAMPLE.COM@WEST.EXAMPLE.COM`	This principal is an example of a cross-realm ticket-granting ticket.
`nfs/boston.example.com@EXAMPLE.COM`	A principal that is used by the NFS service. This principal can be used instead of a `host` principal.
`root/boston.example.com@EXAMPLE.COM`	A principal that is associated with the `root` account on a client. This principal is called a `root` principal and provides `root` access to NFS mounted file systems..
username`@EXAMPLE.COM`	A principal for a user.
username`/admin@EXAMPLE.COM`	An `admin` principal that can be used to administer the KDC database.

♦ ♦ ♦ CHAPTER 8

Kerberos Error Messages and Troubleshooting

This chapter provides resolutions for error messages that you might receive when you use the Kerberos service. This chapter also provides some troubleshooting tips for various problems.

This chapter covers the following topics:

Kerberos Error Messages

This section provides information about Kerberos error messages, including why each error occurs and a way to fix it.

gkadmin GUI Error Messages

`Unable to view the list of principals or policies; use the Name field.`

Cause: The `admin` principal that you logged in with does not have the list privilege (`l`) in the Kerberos ACL file (`kadm5.acl`). So, you cannot view the principal list or policy list.

Solution: You must type the principal and policy names in the Name field to work on them, or you need to log in with a principal that has the appropriate privileges.

`JNI: Java * failed`

Cause: A serious problem exists with the Java Native Interface that is used by the `gkadmin` GUI.

Solution: Exit `gkadmin` and restart it. If the problem persists, report a bug.

Common Kerberos Error Messages (A-M)

This section provides an alphabetical list (A-M) of common error messages for the Kerberos commands, Kerberos daemons, PAM framework, GSS interface, the NFS service, and the Kerberos library.

`Bad lifetime value`

Cause: The lifetime value provided is not valid or incorrectly formatted.

Solution: Make sure that the value provided is consistent with the Time Formats section in the `kinit(1)` man page.

`Bad start time value`

Cause: The start time value provided is not valid or incorrectly formatted.

Solution: Make sure that the value provided is consistent with the Time Formats section in the `kinit(1)` man page.

`Cannot contact any KDC for requested realm`

Cause: No KDC responded in the requested realm.

Solution: Make sure that at least one KDC (either the master or a slave) is reachable or that the `krb5kdc` daemon is running on the KDCs. Check the `/etc/krb5/krb5.conf` file for the list of configured KDCs (`kdc =` *kdc-name*).

`Cannot determine realm for host: host is '`*hostname*`'`

Cause: Kerberos cannot determine the realm name for the host.

Solution: Make sure that there is a default realm name, or that the domain name mappings are set up in the Kerberos configuration file (`krb5.conf`).

`Cannot find a kadmin KDC entry in krb5.conf(4) or DNS Service Location records for realm '`*realmname*`'`

`Cannot find a kpassword KDC entry in krb5.conf(4) or DNS Service Location records for realm '`*realmname*`'`

`Cannot find a master KDC entry in krb5.conf(4) or DNS Service Location records for realm '`*realmname*`'`

`Cannot find any KDC entries in krb5.conf(4) or DNS Service Location records for realm '`*realmname*`'`

Cause: Either the `krb5.conf` file or the DNS server record are incorrectly configured.

Solution: Make sure that the Kerberos configuration file (`/etc/krb5/krb5.conf`) or that the DNS server records for the KDC are configured properly.

`Cannot find address for '`*`hostname`*`': '`*`error-string`*`'`

Cause: No address was found in the DNS records for the given hostname.

Solution: Fix the host record in DNS or correct the error in the DNS lookup process.

`cannot initialize realm` *realm-name*

Cause: The KDC might not have a stash file.

Solution: Make sure that the KDC has a stash file. If not, create a stash file by using the `kdb5_util` command, and try restarting the `krb5kdc` command.

`Cannot resolve KDC for requested realm`

Cause: Kerberos cannot determine any KDC for the realm.

Solution: Make sure that the Kerberos configuration file (`krb5.conf`) specifies a KDC in the `realm` section.

`Cannot resolve network address for KDCs '`*`hostname`*`' discovered via DNS Service Location records for realm '`*`realm-name`*`'`

`Cannot resolve network address for KDCs '`*`hostname`*`' specified in krb5.conf(4) for realm '`*`realm-name`*`'`

Cause: Either the `krb5.conf` file or the DNS server record is incorrectly configured.

Solution: Make sure that the Kerberos configuration file (`/etc/krb5/krb5.conf`) and the DNS server records for the KDC are configured properly.

`Can't open/find Kerberos configuration file`

Cause: The Kerberos configuration file (`krb5.conf`) was unavailable.

Solution: Make sure that the `krb5.conf` file is available in the correct location and has the correct permissions. This file should be writable by `root` and readable by everyone else.

`Client '`*`principal`*`' pre-authentication failed`

Cause: Authentication failed for the principal.

Solution: Make sure that the user is using the correct password.

`Client or server has a null key`

Cause: The principal has a null key.

Solution: Modify the principal to have a non-null key by using the `cpw` command of `kadmin`.

`Communication failure with server while initializing kadmin interface`

Cause: The server that was specified for the master KDC did not have the `kadmind` daemon running.

Solution: Make sure that you specified the correct server name for the master KDC. If you specified the correct server name, make sure that `kadmind` is running on the master KDC that you specified.

`Credentials cache file permissions incorrect`

Cause: You do not have the appropriate read or write permissions on the credentials cache (`/tmp/krb5cc_`*uid*).

Solution: Make sure that you have read and write permissions on the credentials cache.

`Credentials cache I/O operation failed` *XXX*

Cause: Kerberos had a problem writing to the system's credentials cache (`/tmp/krb5cc_`*uid*).

Solution: Make sure that the credentials cache has not been removed, and that there is space left on the device by using the `df` command.

`Decrypt integrity check failed`

Cause: You might have an invalid ticket.

Solution: Verify both of the following conditions:

- Make sure that your credentials are valid. Destroy your tickets with `kdestroy`, and create new tickets with `kinit`.
- Make sure that the target host has a keytab file with the correct version of the service key. Use `kadmin` to view the key version number of the service principal (for example,

host/*FQDN-hostname*) in the Kerberos database. Also, use the `klist -k` command on the target host to make sure that it has the same key version number.

`Decrypt integrity check failed for client 'principal' and server 'hostname'`

Cause: You might have an invalid ticket.

Solution: Make sure that your credentials are valid. Destroy your tickets with the `kdestroy` command, and create new tickets with the `kinit` command.

`failed to obtain credentials cache`

Cause: During `kadmin` initialization, a failure occurred when `kadmin` tried to obtain credentials for the `admin` principal.

Solution: Make sure that you used the correct principal and password when you executed the `kadmin` command.

`Field is too long for this implementation`

Cause: The message size that was being sent by a Kerberized application was too long. This error could be generated if the transport protocol is UDP. which has a default maximum message size 65535 bytes. In addition, there are limits on individual fields within a protocol message that is sent by the Kerberos service.

Solution: Verify that you have not restricted the transport to UDP in the KDC server's `/etc/krb5/kdc.conf` file.

`GSS-API (or Kerberos) error`

Cause: This message is a generic GSS-API or Kerberos error message and can be caused by several different problems.

Solution: Check the `/var/krb5/kdc.log` file to find the more specific error message that was logged when this error occurred.

`Improper format of Kerberos configuration file`

Cause: The Kerberos configuration file has invalid entries.

Solution: Make sure that all the relations in the `krb5.conf` file are followed by the “=” sign and a value. Also, verify that the brackets are present in pairs for each subsection.

`Invalid credential was supplied`

`Service key not available`

Cause: The service ticket in the credentials cache may be incorrect.

Solution: Destroy current credential cache and rerun the `kinit` command before trying to use this service.

`Invalid flag for file lock mode`

Cause: An internal Kerberos error occurred.

Solution: Please report a bug.

`Invalid message type specified for encoding`

Cause: Kerberos did not recognize the message type that was sent by the Kerberized application.

Solution: If you are using a Kerberized application that was developed by your site or a vendor, make sure that it is using Kerberos correctly.

`kadmin: Bad encryption type while changing host/`*FQDN*`'s key`

Cause: More default encryption types are included in the base release in newer releases. Clients can request encryption types that might not be supported by a KDC running an older version of the software.

Solution: Set `permitted_enctypes` in `krb5.conf` on the client to not include the `aes256` encryption type. This step will need to be done on each new client.

`KDC can't fulfill requested option`

Cause: The KDC did not allow the requested option. A possible problem might be that postdating or forwardable options were being requested, and the KDC did not allow them. Another problem might be that you requested the renewal of a TGT, but you didn't have a renewable TGT.

Solution: Determine if you are either requesting an option that the KDC does not allow or a type of ticket that is not available.

`KDC reply did not match expectation: KDC not found. Probably got an unexpected realm referral`

Cause: The KDC reply did not contain the expected principal name, or other values in the response were incorrect.

Solution: Make sure that the KDC you are communicating with complies with RFC4120, that the request you are sending is a Kerberos V5 request, and that the KDC is available.

`kdestroy: Could not obtain principal name from cache`

Cause: The credentials cache is missing or corrupted.

Solution: Check that the cache location provided is correct. Remove and obtain a new TGT by using `kinit`, if necessary.

`kdestroy: No credentials cache file found while destroying cache`

Cause: The credentials cache (`/tmp/krb5c_`*uid*) is missing or corrupted.

Solution: Check that the cache location provided is correct. Remove and obtain a new TGT using `kinit`, if necessary.

`kdestroy: TGT expire warning NOT deleted`

Cause: The credentials cache is missing or corrupted.

Solution: Check that the cache location provided is correct. Remove and obtain a new TGT using `kinit`, if necessary.

`Kerberos authentication failed`

Cause: The Kerberos password is either incorrect or the password might not be synchronized with the UNIX password.

Solution: If the passwords are not synchronized, then you must specify the Kerberos password to complete authentication. It is possible that the user has forgotten their original password.

`Key version` *number* `is not available for principal` *principal*

Cause: The key version of the keys does not match the version for the keys on the application server.

Solution: Check the version of the keys on the application server using the `klist -k` command.

`Key version number for principal in key table is incorrect`

Cause: A principal's key version in the server's keytab file is different from the version in the Kerberos database. Either a service's key has been changed, or you might be using an old service ticket.

Solution: If a service's key has been changed (for example, by using `kadmin`), you need to extract the new key and store it in the host's keytab file where the service is running.

Or, you might be using an old service ticket that has an older key. You might want to run the `kdestroy` command and then the `kinit` command again.

`kinit: gethostname failed`

Cause: An error in the local network configuration is causing `kinit` to fail.

Solution: Make sure that the host is configured correctly.

`login: load_modules: can not open module /usr/lib/security/pam_krb5.so.1`

Cause: Either the Kerberos PAM module is missing or it is not a valid executable binary.

Solution: Make sure that the Kerberos PAM module is in the `/usr/lib/security` directory and that it is a valid executable binary. Also, make sure that your PAM configuration file for `login` contains the correct path to `pam_krb5.so.1`.

`Looping detected getting initial creds: '`*client-principal*`' requesting ticket '`*service-principal*`'. Max loops is `*value*`. Make sure a KDC is available.`

Cause: Kerberos made several attempts to get the initial tickets but failed.

Solution: Make sure that at least one KDC is responding to authentication requests.

`Master key does not match database`

Cause: The loaded database dump was not created from a database that contains the master key. The master key is located in `/var/krb5/.k5.`*REALM*.

Solution: Make sure that the master key in the loaded database dump matches the master key that is located in `/var/krb5/.k5.`*REALM*.

`Matching credential not found`

Cause: The matching credential for your request was not found. Your request requires credentials that are unavailable in the credentials cache.

Solution: Destroy your tickets with `kdestroy`, and create new tickets with `kinit`.

`Message out of order`

Cause: Messages that were sent when using sequential-order privacy arrived out of order. Some messages might have been lost in transit.

Solution: You must reinitialize the Kerberos session.

`Message stream modified`

Cause: The computed checksum and the message checksum do not match. The message might have been modified in transit, which can indicate a security leak.

Solution: Make sure that the messages are being sent across the network correctly. Because this message can also indicate the possible tampering of messages while they are being sent, destroy your tickets and reinitialize the Kerberos services that you are using.

Common Kerberos Error Messages (N-Z)

This section provides an alphabetical list (N-Z) of common error messages for the Kerberos commands, Kerberos daemons, PAM framework, GSS interface, the NFS service, and the Kerberos library.

`No credentials cache file found`

Cause: Kerberos could not find the credentials cache (`/tmp/krb5cc_`*uid*).

Solution: Make sure that the credential file exists and is readable. If it is not, try running the `kinit` command again.

`No credentials were supplied, or the credentials were unavailable or inaccessible`
`No credential cache found`

Cause: The user's credential cache is incorrect or does not exist.

Solution: The user must run `kinit` before trying to start the service.

`No credentials were supplied, or the credentials were unavailable or inaccessible`
`No principal in keytab ('`*filename*`') matches desired name` *principal*

Cause: An error occurred during an attempt to authenticate the server.

Solution: Make sure that the host or service principal is in the server's keytab file.

`Operation requires “`*privilege*`” privilege`

Cause: The `admin` principal that was being used is not assigned the appropriate privilege in the `kadm5.acl` file.

Solution: Use a principal that has the appropriate privileges. Or, configure the principal that was being used to have the appropriate privileges. Usually, a principal with `/admin` as part of its name has the appropriate privileges.

`PAM-KRB5 (auth): krb5_verify_init_creds failed: Key table entry not found`

Cause: The remote application tried to read the host's service principal in the local `/etc/krb5/krb5.keytab` file, but one does not exist.

Solution: Add the host's service principal to the host's keytab file.

`Permission denied in replay cache code`

Cause: The system's replay cache could not be opened. Your server might have been first run under a user ID different than your current user ID.

Solution: Make sure that the replay cache has the appropriate permissions. The replay cache is stored on the host where the Kerberized server application is running. The replay cache file is called `/var/krb5/rcache/rc_`*service_name_uid* for non-`root` users. For root users the replay cache file is called `/var/krb5/rcache/root/rc_`*service_name*.

`Protocol version mismatch`

Cause: Most likely, a Kerberos V4 request was sent to the KDC. The Kerberos service supports only the Kerberos V5 protocol.

Solution: Make sure that your applications are using the Kerberos V5 protocol.

`Request is a replay`

Cause: The request has already been sent to this server and processed. The tickets might have been stolen, and someone else is trying to reuse the tickets.

Solution: Wait for a few minutes, and reissue the request.

`Requested principal and ticket don't match: Requested principal is '`*service-principal*`' and TGT principal is '`*TGT-principal*`'`

Cause: The service principal that you are connecting to and the service ticket that you have do not match.

Solution: Make sure that DNS is functioning properly. If you are using another vendor's software, make sure that the software is using principal names correctly.

`Server refused to negotiate authentication, which is required for encryption. Good bye.`

Cause: The remote application is not capable or has been configured not to accept Kerberos authentication from the client.

Solution: Provide a remote application that can negotiate authentication or configure the application to use the appropriate flags to turn on authentication.

`Server rejected authentication (during sendauth exchange)`

Cause: The server that you are trying to communicate with rejected the authentication. Most often, this error occurs during Kerberos database propagation. Some common causes might be problems with the `kpropd.acl` file, DNS, or the keytab file.

Solution: If you get this error when you are running applications other than `kprop`, investigate whether the server's keytab file is correct.

`Target name principal '`*principal*`' does not match` *service-principal*

Cause: The service principal that is being used does not match the service principal that the application server is using.

Solution: On the application server, make sure that the service principal is included in the keytab file. For the client, make sure that the correct service principal is being used.

`The ticket isn't for us`

`Ticket/authenticator do not match.`

Cause: The principal name in the request might not have matched the service principal's name. Either because the ticket was being sent with an FQDN name of the principal while the service expected a non-FQDN name, or a non-FQDN name was sent when the service expected an FQDN name.

Solution: If you get this error when you are running applications other than `kprop`, investigate whether the server's keytab file is correct.

`Truncated input file detected`

Cause: The database dump file that was being used in the operation is not a complete dump file.

Solution: Create the dump file again, or use a different database dump file.

Kerberos Troubleshooting

This section provides troubleshooting information for the Kerberos software.

Problems With Key Version Numbers

Sometimes, the key version number (KVNO) used by the KDC and the service principal keys stored in `/etc/krb5/krb5.keytab` for services hosted on the system do not match. The KVNO can get out of synchronization when a new set of keys are created on the KDC without updating the keytab file with the new keys. After diagnosing the problem, refresh the `krb5.keytab` file.

1. List the keytab entries.

   The KVNO for each principal is the first item in each entry.

   ```
   # klist -k
   Keytab name: FILE:/etc/krb5/krb5.keytab
   KVNO Principal
   ---- --------------------------------------------------------------------------
   2 host/denver.example.com@EXAMPLE.COM
   2 host/denver.example.com@EXAMPLE.COM
   2 host/denver.example.com@EXAMPLE.COM
   2 nfs/denver.example.com@EXAMPLE.COM
   2 nfs/denver.example.com@EXAMPLE.COM
   2 nfs/denver.example.com@EXAMPLE.COM
   2 nfs/denver.example.com@EXAMPLE.COM
   ```

2. Acquire an initial credential by using the `host` key.

   ```
   # kinit -k
   ```

3. Determine the KVNO that is used by the KDC.

   ```
   # kvno nfs/denver.example.com
   nfs/denver.example.com@EXAMPLE.COM: kvno = 3
   ```

   Note that the KVNO listed here is 3 instead of 2.

Problems With the Format of the `krb5.conf` File

If the `krb5.conf` file is not formatted properly, then the following error message might be displayed in a terminal window or recorded in the log file:

```
Improper format of Kerberos configuration file while initializing krb5 library
```

If the format is incorrect, then the associated services could be vulnerable to attack. You must fix the problem before you allow Kerberos features to be used.

Problems Propagating the Kerberos Database

If propagating the Kerberos database fails, try `/usr/bin/rlogin -x` between the slave KDC and master KDC, and from the master KDC to the slave KDC server.

If the KDCs are secure by default, then the `rlogin` command is disabled and cannot be used to troubleshoot this problem. To enable `rlogin` on a KDC, you must enable the `eklogin` service.

```
# svcadm enable svc:/network/login:eklogin
```

After you finish troubleshooting the problem, disable the `eklogin` service.

```
# svcadm disable svc:/network/login:eklogin
```

If remote access does not work, problems are likely due to the keytab files on the KDCs. If remote access does work, the problem is not in the keytab file or the name service, because `rlogin` and the propagation software use the same `host/`*host-name* principal. In this case, make sure that the `kpropd.acl` file is correct.

Problems Mounting a Kerberized NFS File System

- If mounting a Kerberized NFS file system fails, make sure that the `/var/rcache/root` file exists on the NFS server. If the file system is not owned by `root`, remove it and try the mount again.
- If you have a problem accessing a Kerberized NFS file system, make sure that the `gssd` service is enabled on your system and the NFS server.
- If you see either the `invalid argument` or `bad directory` error message when you are trying to access a Kerberized NFS file system, the problem might be that you are not using a fully qualified DNS name when you are trying to mount the NFS file system. The host that is being mounted is not the same as the host name part of the service principal in the server's keytab file.

  This problem might also occur if your server has multiple Ethernet interfaces, and you have set up DNS to use a “name per interface” scheme instead of a “multiple address records per host” scheme. For the Kerberos service, you should set up multiple address records per host as follows[1]:

  ```
  my.host.name.   A       1.2.3.4
  A       1.2.4.4
  A       1.2.5.4

  my-en0.host.name.       A       1.2.3.4
  my-en1.host.name.       A       1.2.4.4
  ```

[1]Ken Hornstein, “Kerberos FAQ,” [http://www.cmf.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html#kerbdns], accessed 10 March 2010.

```
my-en2.host.name.       A       1.2.5.4

4.3.2.1         PTR     my.host.name.
4.4.2.1         PTR     my.host.name.
4.5.2.1         PTR     my.host.name.
```

In this example, the setup allows one reference to the different interfaces and a single service principal instead of three service principals in the server's keytab file.

Problems Authenticating as the `root` User

If authentication fails when you try to become `root` on your system and you have already added the `root` principal to your host's keytab file, investigate two areas. First, make sure that the `root` principal in the keytab file has a fully qualified system name as its instance. If it does, check the `/etc/resolv.conf` file to make sure that the system is correctly set up as a DNS client.

Observing Mapping From GSS Credentials to UNIX Credentials

To be able to monitor the credential mappings, first uncomment this line from the `/etc/gss/gsscred.conf` file.

```
SYSLOG_UID_MAPPING=yes
```

Next, make the `gssd` service read the `/etc/gss/gsscred.conf` file.

```
# pkill -HUP gssd
```

Now you can monitor the credential mappings as `gssd` requests them. The mappings are recorded by the `syslog` daemon, if the `syslog.conf` file is configured for the `auth` system facility with the `debug` severity level.

Note - If the `rsyslog` service instance is enabled, the mappings are recorded by the `rsyslog` daemon.

Using DTrace With the Kerberos Service

The Kerberos mechanism supports several DTrace probes for decoding various protocol messages. For a list, see Appendix A, “DTrace Probes for Kerberos”. DTrace probes have a distinct advantage over other protocol inspectors, because DTrace enables a privileged user to easily look at unencrypted Kerberos and application data.

The following examples indicate what can be viewed with Kerberos DTrace probes.

EXAMPLE 42 Using DTrace to Track Kerberos Messages

The following script uses DTrace probes to display details about Kerberos messages that are sent and received by the system. Note that the fields that are displayed are based the structures that are assigned to the `krb_message-recv` and `krb_message-send` probes. For more information, see "Definitions of Kerberos DTrace Probes" on page 224.

```
kerberos$target:::krb_message-recv
{
    printf("<- krb message recved: %s\n", args[0]->krb_message_type);
    printf("<- krb message remote addr: %s\n", args[1]->kconn_remote);
    printf("<- krb message ports: local %d remote %d\n",
            args[1]->kconn_localport, args[1]->kconn_remoteport);
    printf("<- krb message protocol: %s transport: %s\n",
            args[1]->kconn_protocol, args[1]->kconn_type);
}

kerberos$target:::krb_message-send
{
    printf("-> krb message sent: %s\n", args[0]->krb_message_type);
    printf("-> krb message remote addr: %s\n", args[1]->kconn_remote);
    printf("-> krb message ports: local %d remote %d\n",
            args[1]->kconn_localport, args[1]->kconn_remoteport);
    printf("-> krb message protocol: %s transport: %s\n",
            args[1]->kconn_protocol, args[1]->kconn_type);
    printf("\n");
}

kerberos$target:::krb_error-read
{
    printf("<- krb error code: %s\n", args[1]->kerror_error_code);
    printf("<- krb error client: %s server: %s\n", args[1]->kerror_client,
            args[1]->kerror_server);
    printf("<- krb error e-text: %s\n", args[1]->kerror_e_text);
    printf("\n");
}
```

The preceding script can be called from the command line or with the `krb5kdc` daemon. The following is an example of calling the script, which is named `krb-dtrace.d`, from the command line. The command causes Kerberos to send and receive messages. Note that `LD_NOLAZYLOAD=1` is needed to force the loading of the Kerberos `mech_krb5.so` library that contains the Kerberos DTrace probes.

```
# LD_NOLAZYLOAD=1 dtrace -s ./krb\-dtrace.d -c kinit
dtrace: script './krb-dtrace' matched 4 probes
kinit: Client 'root@DEV.ORACLE.COM' not found in Kerberos database while g
etting initial credentials
dtrace: pid 3750 has exited
```

```
 CPU     ID                    FUNCTION:NAME
   2  74782 k5_trace_message_send:krb_message-send -> krb message sent: KR
B_AS_REQ(10)
-> krb message remote addr: 10.229.168.163
-> krb message ports: local 62029 remote 88
-> krb message protocol: ipv4 transport: udp


   2  74781 k5_trace_message_recv:krb_message-recv <- krb message recved:
KRB_ERROR(30)
<- krb message remote addr: 10.229.168.163
<- krb message ports: local 62029 remote 88
<- krb message protocol: ipv4 transport: udp

   2  74776     krb5_rd_error:krb_error-read <- krb error code: KDC_ERR_C_
PRINCIPAL_UNKNOWN(6)
<- krb error client: root@DEV.ORACLE.COM server: krbtgt/DEV.ORACLE.COM@DEV
.ORACLE.COM
<- krb error e-text: CLIENT_NOT_FOUND
```

To use the script with the `krb5kdc` daemon, the `svc:/network/security/krb5kdc:default` service must be enabled and online. Note that the following command does not use `LD_NOLAZYLOAD=1` because the `mech_krb5.so` library loads the `krb5kdc` daemon.

```
# dtrace -s ./krb\-dtrace.d -p $(pgrep -x krb5kdc)
```

EXAMPLE 43 Using DTrace to View Kerberos Pre-Authentication Types

The following example shows what pre-authentication is chosen by the client. The first step is to create a DTrace script, like the following:

```
cat krbtrace.d
kerberos$target:::krb_message-recv
{
printf("<- krb message recved: %s\n", args[0]->krb_message_type);
printf("<- krb message remote addr: %s\n", args[1]->kconn_remote);
printf("<- krb message ports: local %d remote %d\n",
args[1]->kconn_localport, args[1]->kconn_remoteport);
printf("<- krb message protocol: %s transport: %s\n",
args[1]->kconn_protocol, args[1]->kconn_type);
}

kerberos$target:::krb_message-send
{
printf("-> krb message sent: %s\n", args[0]->krb_message_type);
printf("-> krb message remote addr: %s\n", args[1]->kconn_remote);
printf("-> krb message ports: local %d remote %d\n",
args[1]->kconn_localport, args[1]->kconn_remoteport);
printf("-> krb message protocol: %s transport: %s\n",
args[1]->kconn_protocol, args[1]->kconn_type);
printf("\n");
```

```
}

kerberos$target:::krb_kdc_req-make
{
printf("-> krb kdc_req make msg type: %s\n", args[0]->krb_message_type);
printf("-> krb kdc_req make pre-auths: %s\n", args[1]->kdcreq_padata_types);
printf("-> krb kdc_req make auth data: %s\n", args[1]->kdcreq_authorization_data);
printf("-> krb kdc_req make client: %s server: %s\n", args[1]->kdcreq_client,
args[1]->kdcreq_server );
}

kerberos$target:::krb_kdc_req-read
{
/* printf("<- krb kdc_req msg type: %s\n", args[0]->krb_message_type); */
printf("<- krb kdc_req client: %s server: %s\n", args[1]->kdcreq_client,
args[1]->kdcreq_server );
printf("\n");
}

kerberos$target:::krb_kdc_rep-read
{
/* printf("<- krb kdc_rep msg type: %s\n", args[0]->krb_message_type); */
printf("<- krb kdc_rep client: %s server: %s\n", args[1]->kdcrep_client,
args[1]->kdcrep_enc_server );
printf("\n");
}

kerberos$target:::krb_ap_req-make
{
printf("-> krb ap_req make server: %s client: %s\n", args[2]->kticket_server,
args[2]->kticket_enc_client );
}

kerberos$target:::krb_error-read
{
printf("<- krb error code: %s\n", args[1]->kerror_error_code);
printf("<- krb error client: %s server: %s\n", args[1]->kerror_client,
args[1]->kerror_server);
printf("<- krb error e-text: %s\n", args[1]->kerror_e_text);
printf("\n");
}
```

Next, execute the `krbtrace.d` script as a privileged user on the Kerberos system by typing the following command:

```
# LD_BIND_NOW=1 dtrace -qs krbtrace.d -c "kinit -k"
.
.
-> krb kdc_req make pre-auths: FX_COOKIE(133) ENC_TIMESTAMP(2) REQ_ENC_PA_REP(149)
```

The pre-authentication types are displayed in the output. For more information about the various pre-authentication types, see RFC 4120.

EXAMPLE 44 Using DTrace to Dump a Kerberos Error Message

```
# dtrace -n 'krb_error-make {
printf("\n{");
printf("\n\tctime = %Y", (uint64_t)(args[1]->kerror_ctime * 1000000000));
printf("\n\tcusec = %d", args[1]->kerror_cusec);
printf("\n\tstime = %Y", (uint64_t)(args[1]->kerror_stime * 1000000000));
printf("\n\tsusec = %d", args[1]->kerror_susec);
printf("\n\terror_code = %s", args[1]->kerror_error_code);
printf("\n\tclient = %s", args[1]->kerror_client);
printf("\n\tserver = %s", args[1]->kerror_server);
printf("\n\te_text = %s", args[1]->kerror_e_text);
printf("\n\te_data = %s", "");
printf("\n}");
}'
dtrace: description 'krb_error-make ' matched 1 probe
CPU     ID                    FUNCTION:NAME
0  78307     krb5_mk_error:krb_error-make
{
ctime = 2012 May 10 12:10:20
cusec = 0
stime = 2012 May 10 12:10:20
susec = 319090
error_code = KDC_ERR_C_PRINCIPAL_UNKNOWN(6)
client = testuser@EXAMPLE.COM
server = krbtgt/EXAMPLE.COM@EXAMPLE.COM
e_text = CLIENT_NOT_FOUND
e_data =
}
```

EXAMPLE 45 Using DTrace to View the Service Ticket for an SSH Server

```
# LD_PRELOAD_32=/usr/lib/gss/mech_krb5.so.1 dtrace -q -n '
kerberos$target:::krb_kdc_req-make {
printf("kdcreq_server: %s",args[1]->kdcreq_server);
}' -c "ssh local@four.example.com" -o dtrace.out
Last login: Wed Sep 10 10:10:20 2014
Oracle Solaris 11 X86    July 2014
$ ^D
# cat dtrace.out
kdcreq_server: host/four.example.com@EXAMPLE.COM
```

EXAMPLE 46 Using DTrace to View the Address and Port of an Unavailable KDC When Requesting an Initial TGT

```
# LD_BIND_NOW=1 dtrace -q -n '
kerberos$target:::krb_message-send {
printf("%s:%d\n",args[1]->kconn_remote, args[1]->kconn_remoteport)
}' -c "kinit local4"
```

```
10.10.10.14:88
10.10.10.14:750
10.10.10.14:88
10.10.10.14:750
10.10.10.14:88
10.10.10.14:750
kinit(v5): Cannot contact any KDC for realm 'EXAMPLE.COM'
while getting initial credentials
```

EXAMPLE 47 Using DTrace to View Requests From Kerberos Principals

```
# LD_BIND_NOW=1 dtrace -qs /opt/kdebug/mykdtrace.d \
-c 'kadmin -p kdc/admin -w test123 -q listprincs'
Authenticating as principal kdc/admin with password.
krb kdc_req msg type: KRB_AS_REQ(10)
krb kdc_req make client: kdc/admin@TEST.NET server:
kadmin/interop1.example.com@TEST.NET
krb message sent: KRB_AS_REQ(10)
krb message recved: KRB_ERROR(30)
Err code: KDC_ERR_PREAUTH_REQUIRED(25)
Err msg client: kdc/admin@TEST.NET server: kadmin/interop1.example.com@TEST.NET
Err e-text: NEEDED_PREAUTH
krb kdc_req msg type: KRB_AS_REQ(10)
krb kdc_req make client: kdc/admin@TEST.NET server:
kadmin/interop1.example.com@TEST.NET
krb message sent: KRB_AS_REQ(10)
krb message recved: KRB_AS_REP(11)
kadmin: Database error! Required KADM5 principal missing while
initializing kadmin interface
krb kdc_req msg type: KRB_AS_REQ(10)
krb kdc_req make client: kdc/admin@TEST.NET server:
kadmin/interop2.example.com@TEST.NET
krb message sent: KRB_AS_REQ(10)
krb message recved: KRB_ERROR(30)
Err code: KDC_ERR_S_PRINCIPAL_UNKNOWN(7)
Err msg client: kdc/admin@TEST.NET server: kadmin/interop2.example.com@TEST.NET
Err e-text: SERVER_NOT_FOUND
krb kdc_req msg type: KRB_AS_REQ(10)
krb kdc_req make client: kdc/admin@TEST.NET server:
kadmin/interop2.example.com@TEST.NET
```

The following script was used to produce the preceding output.

```
kerberos$target:::krb_message-recv
{
printf("krb message recved: %s\n", args[0]->krb_message_type);
}

kerberos$target:::krb_message-send
{
printf("krb message sent: %s\n", args[0]->krb_message_type);
```

```
}

kerberos$target:::krb_kdc_req-make
{
printf("krb kdc_req msg type: %s\n", args[0]->krb_message_type);
printf("krb kdc_req make client: %s server: %s\n", args[1]->kdcreq_client,
args[1]->kdcreq_server );
}

kerberos$target:::krb_ap_req-make
{
printf("krb ap_req make server: %s client: %s\n", args[2]->kticket_server,
args[2]->kticket_enc_client );
}

kerberos$target:::krb_error-read
{
printf("Err code: %s\n", args[1]->kerror_error_code);
printf("Err msg client: %s server: %s\n", args[1]->kerror_client,
args[1]->kerror_server);
printf("Err e-text: %s\n", args[1]->kerror_e_text);
}
```

♦ ♦ ♦ CHAPTER 9

Using Simple Authentication and Security Layer

This chapter includes information about the Simple Authentication and Security Layer (SASL).

About SASL

The Simple Authentication and Security Layer (SASL) is a framework that provides authentication and optional security services to network protocols. An application calls the SASL library, `/usr/lib/libsasl.so`, which provides a glue layer between the application and the various SASL mechanisms. The mechanisms are used in the authentication process and in providing optional security services. The version of SASL is derived from the Cyrus SASL with a few changes.

SASL provides the following services:

- Loading of any plugins
- Determining the necessary security options from the application to aid in the choice of a security mechanism
- Listing of plugins that are available to the application
- Choosing the best mechanism from a list of available mechanisms for a particular authentication attempt
- Routing the authentication data between the application and the chosen mechanism
- Providing information about the SASL negotiation back to the application

SASL Reference

The following section provides information about the implementation of SASL.

SASL Plugins

SASL plugins provide support for security mechanisms, user-canonicalization, and auxiliary property retrieval. By default, the dynamically loaded 32-bit plugins are installed in `/usr/lib/sasl`, and the 64-bit plugins are installed in `/usr/lib/sasl/`*$ISA*. The following security mechanism plugins are provided:

`crammd5.so.1`	CRAM-MD5, which supports authentication only, no authorization
`digestmd5.so.1`	DIGEST-MD5, which supports authentication, integrity, and privacy, as well as authorization
`gssapi.so.1`	GSSAPI, which supports authentication, integrity, and privacy, as well as authorization. The GSSAPI security mechanism requires a functioning Kerberos infrastructure.
`plain.so.1`	PLAIN, which supports authentication and authorization.

In addition, the EXTERNAL security mechanism plugin and the INTERNAL user canonicalization plugins are built into `libsasl.so.1`. The EXTERNAL mechanism supports authentication and authorization. The mechanism supports integrity and privacy if the external security source provides it. The INTERNAL plugin adds the realm name if necessary to the username.

The Oracle Solaris release is not supplying any `auxprop` plugins at this time. For the CRAM-MD5 and DIGEST-MD5 mechanism plugins to be fully operational on the server side, the user must provide an `auxprop` plugin to retrieve clear text passwords. The PLAIN plugin requires additional support to verify the password. The support for password verification can be one of the following: a callback to the server application, an `auxprop` plugin, `saslauthd`, or `pwcheck`. The `saslauthd` and `pwcheck` daemons are not provided in the Oracle Solaris releases. For better interoperability, restrict server applications to those mechanisms that are fully operational by using the `mech_list` SASL option.

SASL Environment Variable

By default, the client authentication name is set to `getenv("LOGNAME")`. This variable can be reset by the client or by the plugin.

SASL Options

The behavior of `libsasl` and the plugins can be modified on the server side by using options that can be set in the `/etc/sasl/`*app*`.conf` file. The variable *app* is the server-defined name for the application. The documentation for the server *app* should specify the application name.

The following options are supported:

`auto_transition` Automatically transitions the user to other mechanisms when the user does a successful plain text authentication.

`auxprop_login` Lists the name of auxiliary property plugins to use.

`canon_user_plugin` Selects the canon_user plugin to use.

`mech_list` Lists the mechanisms that are allowed to be used by the server application.

`pwcheck_method` Lists the mechanisms used to verify passwords. Currently, `auxprop` is the only allowed value.

`reauth_timeout` Sets the length of time, in minutes, that authentication information is cached for a fast reauthentication. This option is used by the DIGEST-MD5 plugin. Setting this option to 0 disables reauthentication.

The following options are not supported:

`plugin_list` Lists available mechanisms. Not used because the option changes the behavior of the dynamic loading of plugins.

`saslauthd_path` Defines the location of the saslauthd door, which is used for communicating with the `saslauthd` daemon. The `saslauthd` daemon is not included in the Oracle Solaris release. So, this option is also not included.

`keytab` Defines the location of the `keytab` file used by the GSSAPI plugin. Use the `KRB5_KTNAME` environment variable instead to set the default `keytab` location.

The following options are options not found in Cyrus SASL. However, they have been added for the Oracle Solaris release:

`use_authid` Acquires the client credentials rather than use the default credentials when creating the GSS client security context. By default, the default client Kerberos identity is used.

`log_level`	Sets the desired level of logging for a server.

♦ ♦ ♦ CHAPTER 10

Configuring Network Services Authentication

This chapter provides information about how to use Secure RPC to authenticate a host and a user across an NFS mount, and covers the following topics:

About Secure RPC

Secure RPC (Remote Procedure Call) protects remote procedures with an authentication mechanism. The Diffie-Hellman authentication mechanism authenticates both the host and the user who is making a request for a service. The authentication mechanism uses Data Encryption Standard (DES) encryption. Applications that use Secure RPC include NFS and the NIS naming service.

NFS Services and Secure RPC

NFS enables several hosts to share files over the network. Under the NFS service, a server holds the data and resources for several clients. The clients have access to the file systems that the server shares with the clients. Users who are logged in to the client systems can access the file systems by mounting the file systems from the server. To the user on the client system, it appears as if the files are local to the client. One of the most common uses of NFS allows systems to be installed in offices, while storing all user files in a central location. Some features of the NFS service, such as the `-nosuid` option to the `mount` command, can be used to prohibit the opening of devices and file systems by unauthorized users.

The NFS service uses Secure RPC to authenticate users who make requests over the network. This process is known as *Secure NFS*. The Diffie-Hellman authentication mechanism, `AUTH_DH`, uses DES encryption to ensure authorized access. The `AUTH_DH` mechanism has also been called `AUTH_DES`. For more information, see the following:

- To set up and administer Secure NFS, see "Administering the Secure NFS System" in *Managing Network File Systems in Oracle Solaris 11.3*.

Kerberos Authentication

Kerberos is an authentication system that was developed at MIT. A client-side and server-side implementation of Kerberos V5, which uses RPCSEC_GSS, is included with this release. For more information, see "How to Configure Kerberos NFS Servers" on page 130.

DES Encryption With Secure NFS

The Data Encryption Standard (DES) encryption functions use a 56-bit key to encrypt data. If two credential users or principals know the same DES key, they can communicate in private by using the key to encipher and decipher text. DES is a relatively fast encryption mechanism.

The risk of using just the DES key is that an intruder can collect enough cipher-text messages that were encrypted with the same key to be able to discover the key and decipher the messages. For this reason, security systems such as Secure NFS need to change the keys frequently.

Diffie-Hellman Authentication and Secure RPC

The Diffie-Hellman (DH) method of authenticating a user is nontrivial for an intruder to crack. The client and the server have their own private key, which they use with the public key to devise a common key. The private key is also known as the *secret key*. The client and the server use the common key to communicate with each other. The common key is encrypted with an agreed-upon encryption function, such as DES.

Authentication is based on the ability of the sending system to use the common key to encrypt the current time. Then, the receiving system can decrypt and check against its current time. The time on the client and the server must be synchronized. The Network Time Protocol (NTP) can be used to synchronize clocks. NTP public domain software from the University of Delaware is included in the Oracle Solaris software. Documentation is available from the NTP Documentation web site.

The public keys and private keys are stored in an NIS database. NIS stores the keys in the `publickey` map. This file contains the public key and the private key for all potential users.

The system administrator is responsible for setting up NIS maps and for generating a public key and a private key for each user. The private key is stored in encrypted form with the user's password. This process makes the private key known only to the user.

Administering Authentication With Secure RPC

By requiring authentication for use of mounted NFS file systems, you increase the security of your network.

The following task map points to procedures that configure Secure RPC for NIS, and NFS.

TABLE 9 Task Map: Administering Authentication With Secure RPC

Task	Description	For Instructions
1. Start the keyserver.	Ensures that keys can be created so that users can be authenticated.	"How to Restart the Secure RPC Keyserver" on page 219
2. Set up credentials on an NIS host.	Ensures that the `root` user on a host can be authenticated in an NIS environment.	"How to Set Up a Diffie-Hellman Key for an NIS Host" on page 220
3. Give an NIS user a key.	Enables a user to be authenticated in an NIS environment.	"How to Set Up a Diffie-Hellman Key for an NIS User" on page 221
4. Share NFS files with authentication.	Enables an NFS server to securely protect shared file systems using authentication.	"How to Share NFS Files With Diffie-Hellman Authentication" on page 222

▼ How to Restart the Secure RPC Keyserver

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Verify that the `keyserv` daemon is running.**

```
# svcs \*keyserv\*
STATE    STIME   FMRI
disabled Dec_14  svc:/network/rpc/keyserv
```

2. **Ensure that the `nobody` user cannot use default keys.**

```
$ pfedit /etc/default/keyserv
# Change the default value of ENABLE_NOBODY_KEYS to NO.
#
#ENABLE_NOBODY_KEYS=YES
ENABLE_NOBODY_KEYS=NO
```

3. **Enable the keyserver service if the service is not online.**

```
# svcadm enable network/rpc/keyserv
```

How to Set Up a Diffie-Hellman Key for an NIS Host

Perform this procedure on every host in the NIS domain.

Before You Begin You must assume the `root` role. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **If the default naming service is not NIS, add the `publickey` map to the naming service.**

   a. **Verify that the value of `config/default` for the naming service is not `nis`.**

   ```
   # svccfg -s name-service/switch listprop config
   config                      application
   config/value_authorization  astring       solaris.smf.value.name-service.switch
   config/default              astring       files
   config/host                 astring       "files nis dns"
   config/printer              astring       "user files nis"
   ```

   If the value of `config/default` is `nis`, you can stop here.

   b. **Set the naming service for `publickey` to `nis`.**

   ```
   # svccfg -s name-service/switch setprop config/publickey = astring: "nis"
   # svccfg -s name-service/switch:default refresh
   ```

   c. **Confirm the `publickey` value.**

   ```
   # svccfg -s name-service/switch listprop
   config                      application
   config/value_authorization  astring       solaris.smf.value.name-service.switch
   config/default              astring       files
   config/host                 astring       "files nis dns"
   config/printer              astring       "user files nis"
   config/publickey            astring       nis
   ```

   On this system, the value of `publickey` is listed because it differs from the default, `files`.

2. **Create a new key pair by using the `newkey` command.**

   ```
   # newkey -h hostname
   ```

   where *hostname* is the name of the client.

Example 48 Setting Up a New Key for `root` on an NIS Client

In the following example, and administrator with the Name Service Security rights profile sets up `earth` as a secure NIS client.

```
# newkey -h earth
Adding new key for unix.earth@example.com
New Password: xxxxxxxx
Retype password: xxxxxxxx
Please wait for the database to get updated...
Your new key has been successfully stored away.
#
```

How to Set Up a Diffie-Hellman Key for an NIS User

Perform this procedure for every user in the NIS domain.

Before You Begin You must be logged in to the NIS master server to generate a new key for a user. You must be assigned the Name Service Security rights profile. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **Create a new key for a user.**

   ```
   # newkey -u username
   ```

   where *username* is the name of the user. The system prompts for a password. You can type a generic password. The private key is stored in an encrypted form by using the generic password.

2. **Tell the user to log in and type the `chkey -p` command.**

   This command allows users to re-encrypt their private keys with a password known only to the user.

Note - The `chkey` command can be used to create a new key pair for a user.

Example 49 Setting Up and Encrypting a New User Key in NIS

In this example, superuser sets up the key.

```
# newkey -u jdoe
Adding new key for unix.12345@example.com
New Password: xxxxxxxx
Retype password: xxxxxxxx
Please wait for the database to get updated...
```

```
Your new key has been successfully stored away.
#
```

Then the user `jdoe` re-encrypts the key with a private password.

```
% chkey -p
Updating nis publickey database.
Reencrypting key for unix.12345@example.com
Please enter the Secure-RPC password for jdoe: xxxxxxxx
Please enter the login password for jdoe: xxxxxxxx
Sending key change request to centralexample...
```

How to Share NFS Files With Diffie-Hellman Authentication

This procedure protects shared file systems on an NFS server by requiring authentication for access.

Before You Begin Diffie-Hellman public key authentication must be enabled on the network. To enable authentication on the network, complete "How to Set Up a Diffie-Hellman Key for an NIS Host" on page 220.

You must become an administrator who is assigned the System Management rights profile to perform this task. For more information, see "Using Your Assigned Administrative Rights" in *Securing Users and Processes in Oracle Solaris 11.3*.

1. **On the NFS server, share a file system with Diffie-Hellman authentication.**

   # share -F nfs -o sec=dh */filesystem*

   where *filesystem* is the file system that is being shared.

   The `-o sec=dh` option means that `AUTH_DH` authentication is now required to access the file system.

2. **On an NFS client, mount a file system with Diffie-Hellman authentication.**

   # mount -F nfs -o sec=dh *server:filesystem mount-point*

server	Is the name of the system that is sharing *filesystem*
filesystem	Is the name of the file system that is being shared, such as `opt`
mount-point	Is the name of the mount point, such as `/opt`

   The `-o sec=dh` option mounts the file system with `AUTH_DH` authentication.

♦ ♦ ♦ APPENDIX A

DTrace Probes for Kerberos

This appendix describes the DTrace probes and argument structures. For examples of their use, see “Using DTrace With the Kerberos Service” on page 206.

DTrace Probes in Kerberos

Probes are program locations or activities to which DTrace can bind a request to perform a set of actions. Probes are defined and implemented by a *provider*. Providers are kernel-loadable modules that enable their probes to trace data.

These probes are for user statically defined tracing (USDT). USDT probes are designed to examine the Kerberos protocol in userland. No kernel probes for statically defined tracing are provided.

You create scripts where appropriate DTrace probes record the information that you want, for example, a stack trace, a timestamp, or the argument of a function. As probes are fired, DTrace gathers the data from the probes and reports it back to you. If you do not specify any actions for a probe, DTrace records each time the probe fires and on what CPU.

Kerberos DTrace probes are modeled after the Kerberos message types that are described in RFC4120: The Kerberos Network Authentication Service (V5) (`http://www.ietf.org/rfc/rfc4120.txt`). The probes are available to consumers of `libkrb5/mech_krb5`, including those applications that use `mech_krb5` through `libgss`. The probes are split between message creation and consumption, and sending and receiving. For more information about `libgss`, see the `libgss`(3LIB) man page.

To use the probes, you specify the `kerberos` provider, the name of the probe (for example `krb_message-recv`), and the arguments. For examples, see “Using DTrace With the Kerberos Service” on page 206.

Definitions of Kerberos DTrace Probes

Probes for KRB_AP_REP:

```
kerberos$pid:::krb_ap_rep-make
kerberos$pid:::krb_ap_rep-read

args[0]        krbinfo_t *
args[1]        kaprepinfo_t *
```

Probes for KRB_AP_REQ:

```
kerberos$pid:::krb_ap_req-make
kerberos$pid:::krb_ap_req-read

args[0]        krbinfo_t *
args[1]        kapreqinfo_t *
args[2]        kticketinfo_t *
args[3]        kauthenticatorinfo_t *
```

Probes for KRB_KDC_REP:

```
kerberos$pid:::krb_kdc_rep-make
kerberos$pid:::krb_kdc_rep-read

args[0]        krbinfo_t *
args[1]        kdcrepinfo_t *
args[2]        kticketinfo_t *
```

Probes for KRB_KDC_REQ:

```
kerberos$pid:::krb_kdc_req-make
kerberos$pid:::krb_kdc_req-read

args[0]        krbinfo_t *
args[1]        kdcreqinfo_t *
```

Probes for KRB_CRED:

```
kerberos$pid:::krb_cred-make
kerberos$pid:::krb_cred-read

args[0]        krbinfo_t *
args[1]        kcredinfo_t *
```

Probes for KRB_ERROR:

```
kerberos$pid:::krb_error-make
kerberos$pid:::krb_error-read

args[0]        krbinfo_t *
args[1]        kerrorinfo_t *
```

Probes for `KRB_PRIV`:

```
kerberos$pid:::krb_priv-make
kerberos$pid:::krb_priv-read

args[0]        krbinfo_t *
args[1]        kprivinfo_t *
```

Probes for `KRB_SAFE`:

```
kerberos$pid:::krb_safe-make
kerberos$pid:::krb_safe-read

args[0]        krbinfo_t *
args[1]        ksafeinfo_t *
```

Probes for sending and receiving messages

```
kerberos$pid:::krb_message-recv
kerberos$pid:::krb_message-send

args[0]        krbinfo_t *
args[1]        kconninfo_t *
```

DTrace Argument Structures in Kerberos

In certain situations, the values of some arguments might be `0`, or empty. The Kerberos argument structures are designed to be generally consistent with RFC4120: The Kerberos Network Authentication Service (V5) (`http://www.ietf.org/rfc/rfc4120.txt`).

Kerberos Message Information in DTrace

```
typedef struct krbinfo {
uint8_t krb_version;                /* protocol version number (5) */
string krb_message_type;            /* Message type (AS_REQ(10), ...) */
uint64_t krb_message_id;            /* message identifier */
uint32_t krb_message_length;        /* message length */
uintptr_t krb_message;              /* raw ASN.1 encoded message */
} krbinfo_t;
```

Note - The Kerberos protocol does not have message identifier. The `krb_message_id` identifier is specific to the Kerberos provider and is designed to link messages between the `make/read` and `send/recv` probes.

Kerberos Connection Information in DTrace

```
typedef struct kconninfo {
string kconn_remote;                /* remote host address */
string kconn_local;                 /* local host address */
uint16_t kconn_localport;           /* local port */
uint16_t kconn_remoteport;          /* remote port */
string kconn_protocol;              /* protocol (ipv4, ipv6) */
string kconn_type;                  /* transport type (udp, tcp) */
} kconninfo_t;
```

Kerberos Authenticator Information in DTrace

```
typedef struct kauthenticatorinfo {
string kauth_client;                /* client principal identifier */
string kauth_cksum_type;            /* type of checksum (des-cbc, ...) */
uint32_t kauth_cksum_length;        /* length of checksum */
uintptr_t kauth_cksum_value;        /* raw checksum data */
uint32_t kauth_cusec;               /* client time, microseconds */
uint32_t kauth_ctime;               /* client time in seconds */
string kauth_subkey_type;           /* sub-key type (des3-cbc-sha1, ...) */
uint32_t kauth_subkey_length;       /* sub-key length */
uintptr_t kauth_subkey_value;       /* sub-key data */
uint32_t kauth_seq_number;          /* sequence number */
string kauth_authorization_data;    /* top-level authorization types
(AD-IF-RELEVANT, ... ) */
} kauthenticatorinfo_t;

typedef struct kticketinfo_t {
string kticket_server;              /* service principal identifier */
uint32_t kticket_enc_part_kvno;     /* key version number */
string kticket_enc_part_etype;      /* enc type of encrypted ticket */
string kticket_enc_flags;           /* ticket flags (forwardable, ...) */
string kticket_enc_key_type;        /* key type (des3-cbc-sha1, ...) */
uint32_t kticket_enc_key_length;    /* key length */
uintptr_t kticket_enc_key_value;    /* key data */
string kticket_enc_client;          /* client principal identifier */
string kticket_enc_transited;       /* list of transited Kerberos realms */
string kticket_enc_transited_type;  /* encoding type */
uint32_t kticket_enc_authtime;      /* time of initial authentication */
uint32_t kticket_enc_starttime;     /* ticket start time in seconds */
uint32_t kticket_enc_endtime;       /* ticket end time in seconds */
uint32_t kticket_enc_renew_till;    /* ticket renewal time in seconds */
string kticket_enc_addresses;       /* addresses associated with ticket */
string kticket_enc_authorization_data;  /* list of top-level auth types */
} kticketinfo_t;
```

```
typedef struct kdcreqinfo {
string kdcreq_padata_types;         /* list of pre-auth types */
string kdcreq_kdc_options;          /* requested ticket flags */
string kdcreq_client;               /* client principal identifier */
string kdcreq_server;               /* server principal identifier */
uint32_t kdcreq_from;               /* requested start time in seconds */
uint32_t kdcreq_till;               /* requested end time in seconds */
uint32_t kdcreq_rtime;              /* requested renewal time in seconds */
uint32_t kdcreq_nonce;              /* nonce for replay detection */
string kdcreq_etype;                /* preferred encryption types */
string kdcreq_addresses;            /* list of requested ticket addresses */
string kdcreq_authorization_data;   /* list of top-level auth types */
uint32_t kdcreq_num_additional_tickets; /* number of additional tickets */
} kdcreqinfo_t;

typedef struct kdcrepinfo {
string kdcrep_padata_types;         /* list of pre-auth types */
string kdcrep_client;               /* client principal identifier */
uint32_t kdcrep_enc_part_kvno;      /* key version number */
string kdcrep_enc_part_etype;       /* enc type of encrypted KDC reply */
string kdcrep_enc_key_type;         /* key type (des3-cbc-sha1, ...) */
uint32_t kdcrep_enc_key_length;     /* key length */
uintptr_t kdcrep_enc_key_value;     /* key data */
string kdcrep_enc_last_req;         /* times of last request of principal */
uint32_t kdcrep_enc_nonce;          /* nonce for replay detection */
uint32_t kdcrep_enc_key_expiration; /* expiration time of client's key */
string kdcrep_enc_flags;            /* ticket flags */
uint32_t kdcrep_enc_authtime;       /* time of authentication of ticket */
uint32_t kdcrep_enc_starttime;      /* ticket start time in seconds */
uint32_t kdcrep_enc_endtime;        /* ticket end time in seconds */
uint32_t kdcrep_enc_renew_till;       /* ticket renewal time in seconds*/
string kdcrep_enc_server;           /* server principal identifier */
string kdcrep_enc_caddr;            /* zero or more client addresses */
} kdcrepinfo_t;

typedef struct kapreqinfo {
string kapreq_ap_options;           /* options (use-session-key,... ) */
uint32_t kapreq_authenticator_kvno; /* key version number */
string kapreq_authenticator_etype;  /* enc type of authenticator */
} kapreqinfo_t;

typedef struct kaprepinfo {
uint32_t kaprep_enc_part_kvno;      /* key version number */
string kaprep_enc_part_etype;       /* enc type of encrypted AP reply */
uint32_t kaprep_enc_ctime;          /* client time in seconds */
uint32_t kaprep_enc_cusec;          /* client time, microseconds portion */
string kaprep_enc_subkey_type;      /* sub-key type */
uint32_t kaprep_enc_subkey_length;  /* sub-key length */
uintptr_t kaprep_enc_subkey_value;  /* sub-key data */
uint32_t kaprep_enc_seq_number;     /* sequence number */
} kaprepinfo_t;
```

```
typedef struct kerrorinfo {
uint32_t kerror_ctime;              /* client time in seconds */
uint32_t kerror_cusec;              /* client time, microseconds */
uint32_t kerror_stime;              /* server time in seconds */
uint32_t kerror_susec;              /* server time, microseconds */
string kerror_error_code;           /* error code (KRB_AP_ERR_SKEW, ...) */
string kerror_client;               /* client principal identifier */
string kerror_server;               /* server principal identifier */
string kerror_e_text;               /* additional error text */
string kerror_e_data;               /* additional error data */
} kerrorinfo_t;

typedef struct ksafeinfo {
uintptr_t ksafe_user_data;          /* raw application specific data */
uint32_t ksafe_timestamp;           /* time of sender in seconds */
uint32_t ksafe_usec;                /* time of sender, microseconds */
uint32_t ksafe_seq_number;          /* sequence number */
string ksafe_s_address;             /* sender's address */
string ksafe_r_address;             /* recipient's address */
string ksafe_cksum_type;            /* checksum type (des-cbc, ...) */
uint32_t ksafe_cksum_length;        /* length of checksum */
uintptr_t ksafe_cksum_value;        /* raw checksum data */
} ksafeinfo_t;

typedef struct kprivinfo {
uint32_t kpriv_enc_part_kvno;       /* key version number */
string kpriv_enc_part_etype;        /* enc type of encrypted message */
uintptr_t kpriv_enc_user_data;      /* raw application specific data */
uint32_t kpriv_enc_timestamp;       /* time of sender in seconds */
uint32_t kpriv_enc_usec;            /* time of sender, microseconds */
uint32_t kpriv_enc_seq_number;      /* sequence number */
string kpriv_enc_s_address;         /* sender's address */
string kpriv_enc_r_address;         /* recipient's address */
} kprivinfo_t;

typedef struct kcredinfo {
uint32_t kcred_enc_part_kvno;       /* key version number */
string kcred_enc_part_etype;        /* enc type of encrypted message */
uint32_t kcred_tickets;             /* number of tickets */
uint32_t kcred_enc_nonce;           /* nonce for replay detection */
uint32_t kcred_enc_timestamp;       /* time of sender in seconds */
uint32_t kcred_enc_usec;            /* time of sender, microseconds */
string kcred_enc_s_address;         /* sender's address */
string kcred_enc_r_address;         /* recipient's address */
} kcredinfo_t;
```

Authentication Services Glossary

access control list (ACL) An access control list (ACL) provides finer-grained file security than traditional UNIX file protection provides. For example, an ACL enables you to allow group read access to a file, while allowing only one member of that group to write to the file.

admin principal In Kerberos, a user principal with a name of the form *username*`/admin` (as in `jdoe/admin`). An admin principal can have more privileges (for example, to change policies) than a regular user principal. See also principal name, user principal.

AES Advanced Encryption Standard. A symmetric block data encryption technique. The U.S. government adopted the Rijndael variant of the algorithm as its encryption standard in October 2000. The default for Kerberos encryption is `aes256-cts-hmac-sha1-96`.

application server See network application server.

authentication In Kerberos, the process of verifying the claimed identity of a principal.

authenticator In Kerberos, authenticators are passed by clients when requesting tickets (from a KDC) and services (from a server). They contain information that is generated by using a session key known only by the client and server, that can be verified as of recent origin, thus indicating that the transaction is secure. When used with a ticket, an authenticator can be used to authenticate a user principal. An authenticator includes the principal name of the user, the IP address of the user's host, and a time stamp. Unlike a ticket, an authenticator can be used only once, usually when access to a service is requested. An authenticator is encrypted by using the session key for that client and that server.

authorization 1. In Kerberos, the process of determining if a principal can use a service, which objects the principal is allowed to access, and the type of access that is allowed for each object.

2. In user rights management, a right that can be assigned to a role or user (or embedded in a rights profile) for performing a class of operations that are otherwise prohibited by security policy. Authorizations are enforced at the user application level, not in the kernel.

client Narrowly, a process that makes use of a network service on behalf of a user; for example, an application that uses `rlogin`. In some cases, a server can itself be a client of some other server or service.

More broadly, a host that a) receives a Kerberos credential, and b) makes use of a service that is provided by a server.

Informally, a Kerberos principal that makes use of a service.

client principal (RPCSEC_GSS API) In Kerberos, a client (a user or an application) that uses RPCSEC_GSS-secured network services. Client principal names are stored in the form of `rpc_gss_principal_t` structures.

clock skew The maximum amount of time that the internal system clocks on all hosts that are participating in the Kerberos authentication system can differ. If the clock skew is exceeded between any of the participating hosts, requests are rejected. Clock skew can be specified in the `krb5.conf` file.

confidentiality See privacy.

consumer In the Cryptographic Framework feature of Oracle Solaris, a consumer is a user of the cryptographic services that come from cryptographic providers. Consumers can be applications, end users, or kernel operations. Kerberos is an example of a consumer and the Cryptographic Framework is its provider.

credential In Kerberos, an information package that includes a ticket and a matching session key. Used to authenticate the identity of a Kerberos principal. See also ticket, session key.

credential cache In Kerberos, a storage space (usually a file) that contains credentials that are received from the KDC.

Diffie-Hellman protocol Also known as public key cryptography. An asymmetric cryptographic key agreement protocol that was developed by Diffie and Hellman in 1976. The protocol enables two users to exchange a secret key over an insecure medium without any prior secrets. Diffie-Hellman is used by Kerberos.

flavor Historically, *security flavor* and *authentication flavor* had the same meaning, as a flavor that indicated a type of authentication (AUTH_UNIX, AUTH_DES, AUTH_KERB). RPCSEC_GSS is also a security flavor, even though it provides integrity and privacy services in addition to authentication.

forwardable ticket In Kerberos, a ticket that a client can use to request a ticket on a remote host without requiring the client to go through the full authentication process on that host. For example, if the user `david` obtains a forwardable ticket while on user `jennifer`'s system, `david` can log in to his own system without being required to get a new ticket (and thus authenticate himself again). See also proxiable ticket.

FQDN Fully qualified domain name. For example, `central.example.com` (as opposed to simply `central`).

GSS-API The Generic Security Service Application Programming Interface. A network layer that provides support for various modular security services, including the Kerberos service.

GSS-API provides for security authentication, integrity, and privacy services. See also authentication, integrity, privacy.

host A system that is accessible over a network.

host principal In Kerberos, a particular instance of a service principal in which the principal (signified by the primary name `host`) is set up to provide a range of network services, such as `ftp` or NFS. An example of a host principal is `host/central.example.com@EXAMPLE.COM`. See also server principal.

initial ticket In Kerberos, a ticket that is issued directly (that is, not based on an existing ticket-granting ticket). Some services, such as applications that change passwords, might require tickets to be marked `initial` so as to assure themselves that the client can demonstrate a knowledge of its secret key. This assurance is important because an initial ticket indicates that the client has recently authenticated itself (instead of relying on a ticket-granting ticket, which might have existed for a long time).

instance In Kerberos, the second part of a principal name, an instance qualifies the principal's primary. In the case of a service principal, the instance is required. The instance is the host's fully qualified domain name, as in `host/central.example.com`. For user principals, an instance is optional. Note, however, that `jdoe` and `jdoe/admin` are unique principals. See also primary, principal name, service principal, user principal.

integrity A security service that, in addition to user authentication, provides for the validity of transmitted data through cryptographic checksumming. See also authentication, privacy.

invalid ticket In Kerberos, a postdated ticket that has not yet become usable. An invalid ticket is rejected by an application server until it becomes validated. To be validated, an invalid ticket must be presented to the KDC by the client in a TGS request, with the `VALIDATE` flag set, after its start time has passed. See also postdated ticket.

KDC Key Distribution Center. A system that has three Kerberos V5 components:

- Principal and key database
- Authentication service
- Ticket-granting service

Each realm has at least one master KDC and should have one or more slave KDCs.

Kerberos An authentication service, the protocol that is used by that service, or the code that is used to implement that service.

The Kerberos implementation in Oracle Solaris that is closely based on Kerberos V5 implementation.

While technically different, “Kerberos” and “Kerberos V5” are often used interchangeably in the Kerberos documentation.

Kerberos (also spelled Cerberus) was a fierce, three-headed mastiff who guarded the gates of Hades in Greek mythology.

Kerberos policy A set of rules that governs password usage in the Kerberos service. Policies can regulate principals' accesses, or ticket parameters, such as lifetime.

key 1. Generally, one of two main types of keys:

- A *symmetric key* – An encryption key that is identical to the decryption key. Symmetric keys are used to encrypt files.
- An *asymmetric key* or *public key* – A key that is used in public key algorithms, such as Diffie-Hellman or RSA. Public keys include a private key that is known only by one user, a public key that is used by the server or general resource, and a private-public key pair that combines the two. A private key is also called a *secret* key. The public key is also called a *shared* key or *common* key.

2. An entry (principal name) in a keytab file. See also keytab file.

3. In Kerberos, an encryption key, of which there are three types:

- A *private key* – An encryption key that is shared by a principal and the KDC, and distributed outside the bounds of the system. See also private key.
- A *service key* – This key serves the same purpose as the private key, but is used by servers and services. See also service key.
- A *session key* – A temporary encryption key that is used between two principals, with a lifetime limited to the duration of a single login session. See also session key.

keystore A keystore holds passwords, passphrases, certificates, and other authentication objects for retrieval by applications. A keystore can be specific to a technology, or a location that several applications use.

keytab file In Kerberos, a key table file that contains one or more keys (principals). A host or service uses a keytab file in the much the same way that a user uses a password.

kvno Key version number in Kerberos. A sequence number that tracks a particular key in order of generation. The highest kvno is the latest and most current key.

LDAP binding LDAP binding enables data exchange with an LDAP directory on an LDAP server, therefore enables processes to search, compare, and modify an LDAP directory using the LDAP protocol.

master KDC The main KDC in each realm, which includes a Kerberos administration server, `kadmind`, and an authentication and ticket-granting daemon, `krb5kdc`. Each realm must have at least one master KDC, and can have many duplicate, or slave, KDCs that provide authentication services to clients. In a multi-master configuration, a realm contains at least two master KDCs.

network application server A server that provides a network application, such as `ftp`. A Kerberos realm can contain several network application servers.

NTP Network Time Protocol. Software from the University of Delaware that enables you to manage precise time or network clock synchronization, or both, in a network environment. You can use NTP to maintain clock skew in a Kerberos environment. See also clock skew.

PAM Pluggable Authentication Module. A framework that allows for multiple authentication mechanisms to be used without having to recompile the services that use them. PAM enables Kerberos session initialization at login.

passphrase A phrase that is used to verify that a private key was created by the passphrase user. A good passphrase is 10-30 characters long, mixes alphabetic and numeric characters, and avoids simple prose and simple names. You are prompted for the passphrase to authenticate use of the private key to encrypt and decrypt communications.

password policy The encryption algorithms that can be used to generate passwords. Can also refer to more general issues around passwords, such as how often the passwords must be changed, how many password attempts are permitted, and other security considerations. Security policy requires passwords. Password policy might dictate password strength, frequency of change, and permitted alphabet, number, and keyboard modifier keys.

policy Generally, a plan or course of action that influences or determines decisions and actions. For computer systems, policy typically means security policy. Your site's security policy is the set of rules that define the sensitivity of the information that is being processed and the measures that are used to protect the information from unauthorized access.

See also Kerberos policy and password policy.

postdated ticket In Kerberos, a postdated ticket does not become valid until some specified time after its creation. Such a ticket is useful, for example, for batch jobs that are intended to run late at night, since the ticket, if stolen, cannot be used until the batch job is run. When a postdated ticket is issued, it is issued as `invalid` and remains that way until a) its start time has passed, and b) the client requests validation by the KDC. A postdated ticket is normally valid until the expiration time of the ticket-granting ticket. However, if the postdated ticket is marked `renewable`, its lifetime is normally set to be equal to the duration of the full life time of the ticket-granting ticket. See also invalid ticket, renewable ticket.

primary In Kerberos, the first part of a principal name. See also instance, principal name, realm.

principal 1. In Kerberos, a uniquely named client/user or server/service instance that participates in a network communication. Kerberos transactions involve interactions between principals (service principals and user principals) or between principals and KDCs. In other words, a principal is a unique entity to which Kerberos can assign tickets. See also principal name, service principal, user principal.

2. (RPCSEC_GSS API) See client principal, server principal.

principal name 1. In Kerberos, the name of a principal, in the format *primary/instance@REALM*. See also instance, primary, realm.

2. (RPCSEC_GSS API) See client principal, server principal.

privacy A security service, in which transmitted data is encrypted before being sent. Privacy also includes data integrity and user authentication. See also authentication, integrity, service.

private key A key that is given to each user principal in Kerberos, and known only to the user of the principal and to the KDC. For user principals, the key is based on the user's password. The Kerberos service is a private-key system. See also key.

privilege In general, a power or capability to perform an operation on a computer system that is beyond the powers of a regular user. A privileged user or privileged application is a user or application that has been granted additional rights.

privileged application An application that can override system controls. The application checks for security attributes, such as specific UIDs, GIDs, authorizations, or privileges.

privileged user A user who is assigned rights beyond the rights of regular user on a computer system.

proxiable ticket In Kerberos, a ticket that can be used by a service on behalf of a client to perform an operation for the client. Thus, the service is said to act as the client's proxy. With the ticket, the service can take on the identity of the client. The service can use a proxiable ticket to obtain a service ticket to another service, but it cannot obtain a ticket-granting ticket. The difference between a proxiable ticket and a forwardable ticket is that a proxiable ticket is only valid for a single operation. See also forwardable ticket.

PTP Precision Time Protocol. Software that enables you to manage precise time or network clock synchronization, or both, in a network environment. You can use PTP to maintain clock skew in a Kerberos environment. See also clock skew.

public-key encryption An encryption scheme in which each user has two keys, one public key and one private key. In public-key encryption, the sender uses the receiver's public key to encrypt the message, and the receiver uses a private key to decrypt it.

realm 1. The logical network that is served by a single Kerberos database and a set of Key Distribution Centers (KDCs).

2. The third part of a principal name. For the principal name `jdoe/admin@CORP.EXAMPLE.COM`, the realm is `CORP.EXAMPLE.COM`. See also principal name.

relation In Kerberos, a configuration variable or relationship that is defined in the `kdc.conf` or `krb5.conf` files.

renewable ticket Because having tickets with very long lives is a security risk, Kerberos tickets can be designated as `renewable`. A renewable ticket has two expiration times: a) the time at which the current instance of the ticket expires, and b) maximum lifetime for any ticket. If a client wants to continue to use a ticket, the client renews the ticket before the first expiration occurs. For example, a ticket can be valid for one hour, with all tickets having a maximum lifetime of ten hours. If the client that holds the ticket wants to keep it for more than an hour, the client must renew the ticket. When a ticket reaches the maximum ticket lifetime, it automatically expires and cannot be renewed.

secret key See private key.

security flavor See flavor.

security mechanism Specific implementation of an encryption algorithm, such as `aes-256` and `aes-512`.

security policy See policy.

security service See service.

server In Kerberos, a principal that provides a resource to network clients. For example, if you `ssh` to the system `central.example.com`, then that system is the server that provides the `ssh` service. See also service principal.

server principal (RPCSEC_GSS API) In Kerberos, a principal that provides a service. The server principal is stored as an ASCII string in the form *service@host*. See also client principal.

service 1. A resource that is provided to network clients, often by more than one server. For example, if you `ssh` to the system `central.example.com`, then that system is the server that provides the `ssh` service.

2. A security service (either integrity or privacy) that provides a level of protection beyond authentication. See also integrity and privacy.

service key An encryption key that is shared by a Kerberos service principal and the KDC, and is distributed outside the bounds of the system. See also key.

service principal In Kerberos, a principal that provides authentication for a service or services. For service principals, the primary name is a name of a service, such as `ftp`, and its instance is the fully qualified host name of the system that provides the service. See also host principal, user principal.

session key In Kerberos, a key that is generated by the authentication service or the ticket-granting service. A session key is generated to provide secure transactions between a client and a service. The lifetime of a session key is limited to a single login session. See also key.

slave KDC A copy of a master KDC, which is capable of performing most functions of the master. Each realm usually has several slave KDCs and one master KDC. A realm that has more than one master KDC is called a multi-master configuration. See also KDC, master KDC.

stash file In Kerberos, a stash file contains an encrypted copy of the master key for the KDC. This master key is used when a server is rebooted to automatically authenticate the KDC before it starts the `kadmind` and `krb5kdc` processes. Because the stash file includes the master key, the stash file and any backups of it should be kept secure. If the encryption is compromised, then the key could be used to access or modify the KDC database.

TGS Ticket-Granting Service. That portion of the KDC that is responsible for issuing tickets.

TGT Ticket-Granting Ticket. A ticket that is issued by the KDC that enables a client to request tickets for other services.

ticket In Kerberos, an information packet that is used to securely pass the identity of a user to a server or service. A ticket is valid for only a single client and a particular service on a specific server. A ticket contains the principal name of the service, the principal name of the user, the IP address of the user's host, a time stamp, and a value that defines the lifetime of the ticket. A ticket is created with a random session key to be used by the client and the service. Once a ticket has been created, it can be reused until the ticket expires. A ticket only serves to authenticate a client when it is presented along with a fresh authenticator. See also authenticator, credential, service, session key.

ticket file See credential cache.

user principal In Kerberos, a principal that is attributed to a particular user. A user principal's primary name is a user name, and its optional instance is a name that is used to described the intended use of the corresponding credentials (for example, `jdoe` or `jdoe/admin`). Also known as a user instance. See also service principal.

Index

D

E

F

G

H

I

K

L

M

N

O

P

R

S

T

U

V

W