Red Hat Enterprise Linux 7

System-Level Authentication Guide

About System-Level Services for Authentication and Identity Management

Last Updated: 2018-06-25

Red Hat Enterprise Linux 7 System-Level Authentication Guide

About System-Level Services for Authentication and Identity Management

Filip Hanzelka
Red Hat Customer Content Services
fhanzelk@redhat.com

Lucie Maňásková
Red Hat Customer Content Services
lmanasko@redhat.com

Aneta Šteflová Petrová
Red Hat Customer Content Services

Marc Muehlfeld
Red Hat Customer Content Services

Tomáš Čapek
Red Hat Customer Content Services

Ella Deon Ballard
Red Hat Customer Content Services

Legal Notice

Copyright © 2018 Red Hat, Inc.

This document is licensed by Red Hat under the Creative Commons Attribution-ShareAlike 3.0 Unported License. If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

This guide covers different applications and services available to configure authentication on local systems. In addition to this guide, you can find documentation on the features and services related to Red Hat Enterprise Linux Identity Management in the following guides: The Linux Domain Identity, Authentication, and Policy Guide documents Red Hat Identity Management, a solution that provides a centralized and unified way to manage identity stores as well as authentication and authorization policies in a Linux-based domain. The Windows Integration Guide documents how to integrate Linux domains with Microsoft Windows Active Directory (AD) using Identity Management. Among other topics, the guide covers various aspects of direct and indirect AD integration, using SSSD to access a Common Internet File System (CIFS), and the realmd system.

Table of Contents

CHAPTER 1. INTRODUCTION TO SYSTEM AUTHENTICATION

One of the cornerstones of establishing a secure network environment is making sure that access is restricted to people who have the right to access the network. If access is allowed, users can *authenticate* to the system, meaning they can verify their identities.

On any Red Hat Enterprise Linux system, there are a number of different services available to create and identify user identities. These can be local system files, services which connect to larger identity domains like Kerberos or Samba, or tools to create those domains.

This guide reviews some common system services and applications which are available to administrators to manage authentication and identities for a local system. Other guides are available which provide more detailed information on creating Linux domains and integrating a Linux system into a Windows domain.

1.1. CONFIRMING USER IDENTITIES

Authentication is the process of confirming an identity. For network interactions, authentication involves the identification of one party by another party. There are many ways to use authentication over networks: simple passwords, certificates, one-time password (OTP) tokens, biometric scans.

Authorization, on the other hand, defines what the authenticated party is allowed to do or access.

Authentication requires that a user presents some kind of *credential* to verify his identity. The kind of credential that is required is defined by the authentication mechanism being used. There are several kinds of authentication for local users on a system:

- *Password-based authentication*. Almost all software permits the user to authenticate by providing a recognized name and password. This is also called *simple authentication*.
- *Certificate-based authentication*. Client authentication based on certificates is part of the SSL protocol. The client digitally signs a randomly generated piece of data and sends both the certificate and the signed data across the network. The server validates the signature and confirms the validity of the certificate.
- *Kerberos authentication*. Kerberos establishes a system of short-lived credentials, called *ticket-granting tickets (TGTs)*. The user presents credentials, that is, user name and password, that identify the user and indicate to the system that the user can be issued a ticket. TGT can then be repeatedly used to request access tickets to other services, like websites and email. Authentication using TGT allows the user to undergo only a single authentication process in this way.
- *Smart card-based authentication*. This is a variant of certificate-based authentication. The smart card (or *token*) stores user certificates; when a user inserts the token into a system, the system can read the certificates and grant access. Single sign-on using smart cards goes through three steps:

    1. A user inserts a smart card into the card reader. Pluggable authentication modules (PAMs) on Red Hat Enterprise Linux detect the inserted smart card.
    2. The system maps the certificate to the user entry and then compares the presented certificates on the smart card, which are encrypted with a private key as explained under the certificate-based authentication, to the certificates stored in the user entry.
    3. If the certificate is successfully validated against the key distribution center (KDC), then the user is allowed to log in.

Smart card-based authentication builds on the simple authentication layer established by Kerberos by adding certificates as additional identification mechanisms as well as by adding physical access requirements.

1.2. AS PART OF PLANNING SINGLE SIGN-ON

The thing about authentication as described in Section 1.1, “Confirming User Identities” is that every secure application requires at least a password to access it. Without a central identity store and every application maintaining its own set of users and credentials, a user has to enter a password for every single service or application he opens. This can require entering a password several times a day, maybe even every few minutes.

Maintaining multiple passwords, and constantly being prompted to enter them, is a hassle for users and administrators. *Single sign-on* is a configuration which allows administrators to create a single password store so that users can log in once, using a single password, and be authenticated to all network resources.

Red Hat Enterprise Linux supports single sign-on for several resources, including logging into workstations, unlocking screen savers, and accessing secured web pages using Mozilla Firefox. With other available system services such as PAM, NSS, and Kerberos, other system applications can be configured to use those identity sources.

Single sign-on is both a convenience to users and another layer of security for the server and the network. Single sign-on hinges on secure and effective authentication. Red Hat Enterprise Linux provides two authentication mechanisms which can be used to enable single sign-on:

- Kerberos-based authentication, through both Kerberos realms and Active Directory domains
- Smart card-based authentication

Both of these methods create a centralized identity store (either through a Kerberos realm or a certificate authority in a public key infrastructure), and the local system services then use those identity domains rather than maintaining multiple local stores.

1.3. AVAILABLE SERVICES

All Red Hat Enterprise Linux systems have some services already available to configure authentication for local users on local systems. These include:

Authentication Setup

- The Authentication Configuration tool (**`authconfig`**) sets up different identity back ends and means of authentication (such as passwords, fingerprints, or smart cards) for the system.

Identity Back End Setup

- The Security System Services Daemon (SSSD) sets up multiple identity providers (primarily LDAP-based directories such as Microsoft Active Directory or Red Hat Enterprise Linux IdM) which can then be used by both the local system and applications for users. Passwords and tickets are cached, allowing both offline authentication and single sign-on by reusing credentials.
- The **`realmd`** service is a command-line utility that allows you to configure an authentication back end, which is SSSD for IdM. The **`realmd`** service detects available IdM domains based on the DNS records, configures SSSD, and then joins the system as an account to a domain.

- Name Service Switch (NSS) is a mechanism for low-level system calls that return information about users, groups, or hosts. NSS determines what source, that is, which modules, should be used to obtain the required information. For example, user information can be located in traditional UNIX files, such as the **`/etc/passwd`** file, or in LDAP-based directories, while host addresses can be read from files, such as the **`/etc/hosts`** file, or the DNS records; NSS locates where the information is stored.

Authentication Mechanisms

- Pluggable Authentication Modules (PAM) provide a system to set up authentication policies. An application using PAM for authentication loads different modules that control different aspects of authentication; which PAM module an application uses is based on how the application is configured. The available PAM modules include Kerberos, Winbind, or local UNIX file-based authentication.

Other services and applications are also available, but these are common ones.

PART I. SYSTEM LOGINS

CHAPTER 2. CONFIGURING SYSTEM AUTHENTICATION

Authentication is the process in which a user is identified and verified to a system. It requires presenting some sort of identity and credentials, such as a user name and password. The system then compares the credentials against the configured authentication service. If the credentials match and the user account is active, then the user is *authenticated*.

Once a user is authenticated, the information is passed to the access control service to determine what the user is permitted to do. Those are the resources the user is *authorized* to access. Note that authentication and authorization are two separate processes.

The system must have a configured list of valid account databases for it to check for user authentication. The information to verify the user can be located on the local system or the local system can reference a user database on a remote system, such as LDAP or Kerberos. A local system can use a variety of different data stores for user information, including Lightweight Directory Access Protocol (LDAP), Network Information Service (NIS), and Winbind. Both LDAP and NIS data stores can use Kerberos to authenticate users.

For convenience and potentially part of single sign-on, Red Hat Enterprise Linux can use the System Security Services Daemon (SSSD) as a central daemon to authenticate the user to different identity back ends or even to ask for a ticket-granting ticket (TGT) for the user. SSSD can interact with LDAP, Kerberos, and external applications to verify user credentials.

This chapter explains what tools are available in Red Hat Enterprise Linux for configuring system authentication:

- the **ipa-client-install** utility and the **realmd** system for Identity Management systems; see Section 2.1, “Identity Management Tools for System Authentication” for more information
- the **authconfig** utility and the authconfig UI for other systems; see Section 2.2, “Using **authconfig**” for more information

2.1. IDENTITY MANAGEMENT TOOLS FOR SYSTEM AUTHENTICATION

You can use the **ipa-client-install** utility and the **realmd** system to automatically configure system authentication on Identity Management machines.

ipa-client-install

The **ipa-client-install** utility configures a system to join the Identity Management domain as a client machine. For more information about **ipa-client-install**, see the *Linux Domain Identity, Authentication, and Policy Guide*.

Note that for Identity Management systems, **ipa-client-install** is preferred over **realmd**.

realmd

The **realmd** system joins a machine to an identity domain, such as an Identity Management or Active Directory domain. For more information about **realmd**, see the *Windows Integration Guide*.

2.2. USING AUTHCONFIG

The **authconfig** tool can help configure what kind of data store to use for user credentials, such as LDAP. On Red Hat Enterprise Linux, **authconfig** has both GUI and command-line options to configure

any user data stores. The **authconfig** tool can configure the system to use specific services — SSSD, LDAP, NIS, or Winbind — for its user database, along with using different forms of authentication mechanisms.

IMPORTANT

To configure Identity Management systems, Red Hat recommends using the **ipa-client-install** utility or the **realmd** system instead of **authconfig**. The **authconfig** utilities are limited and substantially less flexible. For more information, see Section 2.1, “Identity Management Tools for System Authentication”.

The following three **authconfig** utilities are available for configuring authentication settings:

- **authconfig-gtk** provides a full graphical interface.
- **authconfig** provides a command-line interface for manual configuration.
- **authconfig-tui** provides a text-based UI. Note that this utility has been deprecated.

All of these configuration utilities must be run as **root**.

2.2.1. Tips for Using the authconfig CLI

The **authconfig** command-line tool updates all of the configuration files and services required for system authentication, according to the settings passed to the script. Along with providing even more identity and authentication configuration options than can be set through the UI, the **authconfig** tool can also be used to create backup and kickstart files.

For a complete list of **authconfig** options, check the help output and the man page.

There are some things to remember when running **authconfig**:

- With every command, use either the **--update** or **--test** option. One of those options is required for the command to run successfully. Using **--update** writes the configuration changes. The **--test** option displays the changes but does not apply the changes to the configuration.

  If the **--update** option is not used, then the changes are not written to the system configuration files.

- The command line can be used to update existing configuration as well as to set new configuration. Because of this, the command line does not enforce that required attributes are used with a given invocation (because the command may be updating otherwise complete settings).

  When editing the authentication configuration, be very careful that the configuration is complete and accurate. Changing the authentication settings to incomplete or wrong values can lock users out of the system. Use the --test option to confirm that the settings are proper before using the --update option to write them.

- Each enable option has a corresponding disable option.

2.2.2. Installing the authconfig UI

The **authconfig** UI is not installed by default, but it can be useful for administrators to make quick changes to the authentication configuration.

To install the UI, install the **authconfig-gtk** package. This has dependencies on some common system packages, such as the **authconfig** command-line tool, Bash, and Python. Most of those are installed by default.

```
[root@server ~]# yum install authconfig-gtk
Loaded plugins: langpacks, product-id, subscription-manager
Resolving Dependencies
--> Running transaction check
---> Package authconfig-gtk.x86_64 0:6.2.8-8.el7 will be installed
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
======
 Package              Arch         Version            Repository
Size
================================================================================
======
Installing:
 authconfig-gtk       x86_64       6.2.8-8.el7        RHEL-Server
105 k

Transaction Summary
================================================================================
======
Install  1 Package

... 8< ...
```

2.2.3. Launching the authconfig UI

1. Open the terminal and log in as root.
2. Run the **system-config-authentication** command.

IMPORTANT

Any changes take effect immediately when the **authconfig** UI is closed.

There are three configuration tabs in the **Authentication** dialog box:

- **Identity & Authentication**, which configures the resource used as the identity store (the data repository where the user IDs and corresponding credentials are stored).
- **Advanced Options**, which configures authentication methods other than passwords or certificates, like smart cards and fingerprint.
- **Password Options**, which configures password authentication methods.

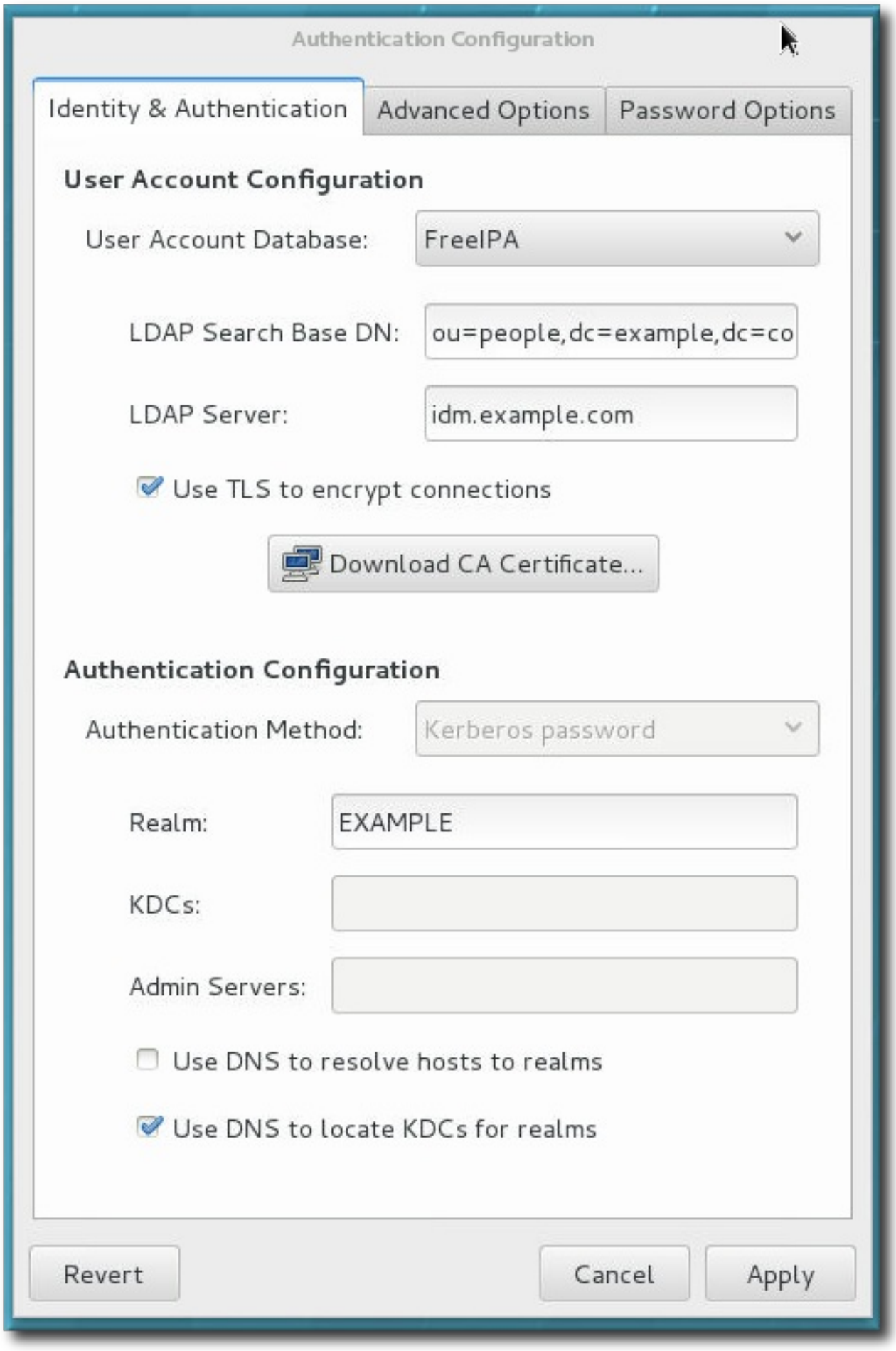


Figure 2.1. authconfig Window

2.2.4. Testing Authentication Settings

It is critical that authentication is fully and properly configured. Otherwise all users (even root) could be locked out of the system, or some users blocked.

The **--test** option prints all of the authentication configuration for the system, for every possible identity and authentication mechanism. This shows both the settings for what is enabled and what areas are disabled.

The **test** option can be run by itself to show the full, current configuration or it can be used with an **authconfig** command to show *how the configuration will be changed* (without actually changing it). This can be very useful in verifying that the proposed authentication settings are complete and correct.

```
[root@server ~]# authconfig --test
caching is disabled
nss_files is always enabled
nss_compat is disabled
nss_db is disabled
nss_hesiod is disabled
 hesiod LHS = ""
 hesiod RHS = ""
nss_ldap is disabled
 LDAP+TLS is disabled
 LDAP server = ""
 LDAP base DN = ""
nss_nis is disabled
 NIS server = ""
 NIS domain = ""
nss_nisplus is disabled
nss_winbind is disabled
 SMB workgroup = "MYGROUP"
 SMB servers = ""
 SMB security = "user"
 SMB realm = ""
 Winbind template shell = "/bin/false"
 SMB idmap range = "16777216-33554431"
nss_sss is enabled by default
nss_wins is disabled
nss_mdns4_minimal is disabled
DNS preference over NSS or WINS is disabled
pam_unix is always enabled
 shadow passwords are enabled
 password hashing algorithm is sha512
pam_krb5 is disabled
 krb5 realm = "#"
 krb5 realm via dns is disabled
 krb5 kdc = ""
 krb5 kdc via dns is disabled
 krb5 admin server = ""
pam_ldap is disabled
 LDAP+TLS is disabled
 LDAP server = ""
 LDAP base DN = ""
 LDAP schema = "rfc2307"
pam_pkcs11 is disabled
 use only smartcard for login is disabled
```

```
 smartcard module = ""
 smartcard removal action = ""
pam_fprintd is disabled
pam_ecryptfs is disabled
pam_winbind is disabled
 SMB workgroup = "MYGROUP"
 SMB servers = ""
 SMB security = "user"
 SMB realm = ""
pam_sss is disabled by default
 credential caching in SSSD is enabled
 SSSD use instead of legacy services if possible is enabled
IPAv2 is disabled
IPAv2 domain was not joined
 IPAv2 server = ""
 IPAv2 realm = ""
 IPAv2 domain = ""
pam_pwquality is enabled (try_first_pass local_users_only retry=3
authtok_type=)
pam_passwdqc is disabled ()
pam_access is disabled ()
pam_mkhomedir or pam_oddjob_mkhomedir is disabled (umask=0077)
Always authorize local users is enabled ()
Authenticate system accounts against network services is disabled
```

2.2.5. Saving and Restoring Configuration Using `authconfig`

Changing authentication settings can be problematic. Improperly changing the configuration can wrongly exclude users who should have access, can cause connections to the identity store to fail, or can even lock all access to a system.

Before editing the authentication configuration, it is strongly recommended that administrators take a backup of all configuration files. This is done with the **`--savebackup`** option.

```
[root@server ~]# authconfig --savebackup=/backups/authconfigbackup20180701
```

The authentication configuration can be restored to any previous saved version using the **`--restorebackup`** option, with the name of the backup to use.

```
[root@server ~]# authconfig --
restorebackup=/backups/authconfigbackup20180701
```

The **`authconfig`** command saves an automatic backup every time the configuration is altered. It is possible to restore the last backup using the **`--restorelastbackup`** option.

```
[root@server ~]# authconfig --restorelastbackup
```

CHAPTER 3. SELECTING THE IDENTITY STORE FOR AUTHENTICATION WITH `AUTHCONFIG`

The **`Identity & Authentication`** tab in the **`authconfig`** UI sets how users should be authenticated. The default is to use local system authentication, meaning the users and their passwords are checked against local system accounts. A Red Hat Enterprise Linux machine can also use external resources which contain the users and credentials, including LDAP, NIS, and Winbind.

3.1. IPAV2

There are two different ways to configure an Identity Management server as an identity back end. For IdM version 2 (Red Hat Enterprise Linux version 6.3 and earlier), version 3 (in Red Hat Enterprise Linux 6.4 and later), and version 4 (in Red Hat Enterprise Linux 7.1 and later), these are configured as IPAv2 providers in **`authconfig`**. For previous IdM versions and for community FreeIPA servers, these are configured as LDAP providers.

3.1.1. Configuring IdM from the UI

1. Open the **`authconfig`** UI.

2. Select **IPAv2** in the **`User Account Database`** drop-down menu.

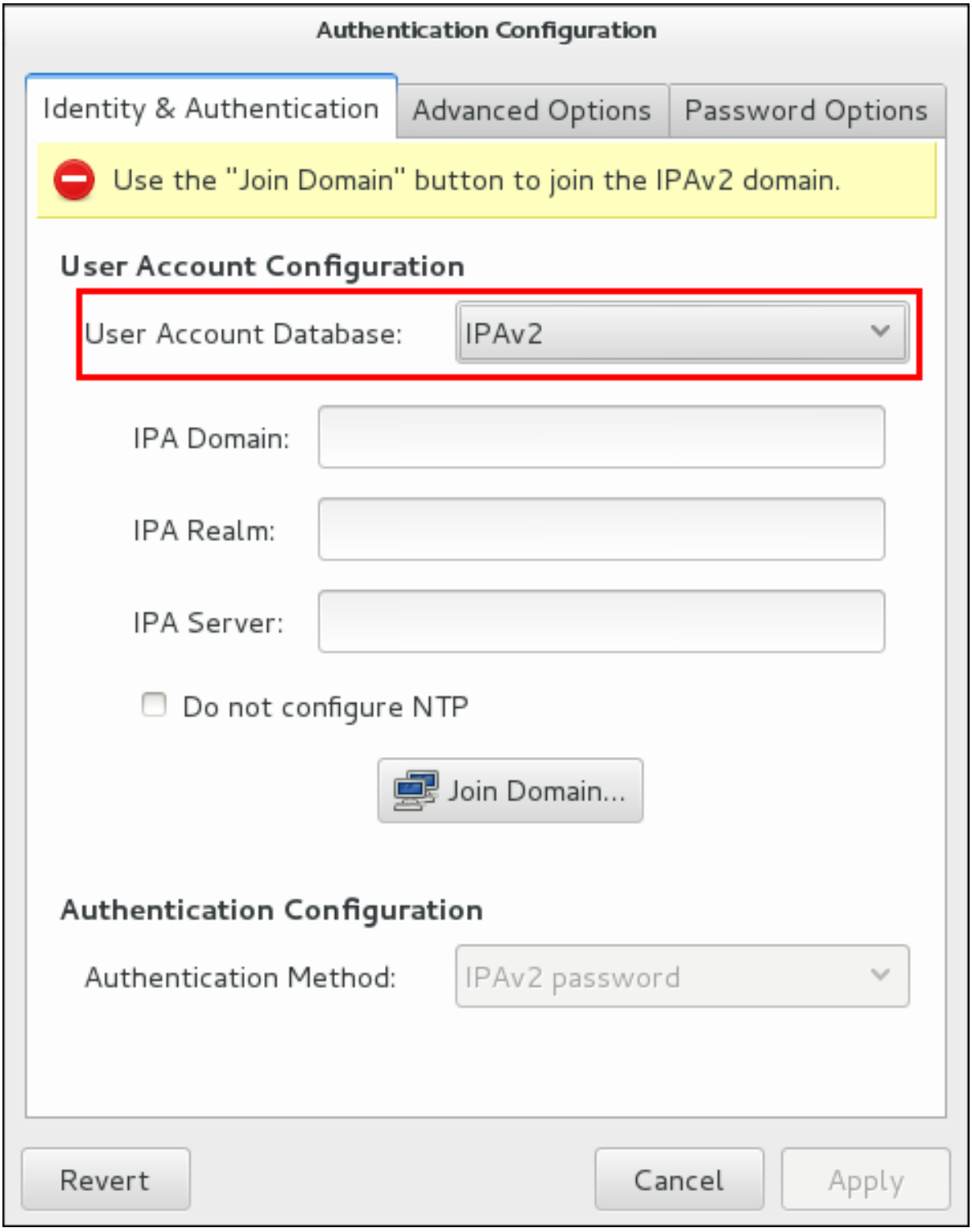


Figure 3.1. Authentication Configuration

3. Set the information that is required to connect to the IdM server.
   - **`IPA Domain`** gives the DNS domain of the IdM domain.
   - **`IPA Realm`** gives the Kerberos domain of the IdM domain.
   - **`IPA Server`** gives the host name of any IdM server within the IdM domain topology.
   - **`Do not configure NTP`** optionally disables NTP services when the client is configured. This is usually not recommended, because the IdM server and all clients need to have synchronized clocks for Kerberos authentication and certificates to work properly. This could

be disabled if the IdM servers are using a different NTP server rather than hosting it within the domain.

4. Click the **Join the domain** button.

   This runs the **ipa-client-install** command and, if necessary, installs the **ipa-client** packages. The installation script automatically configures all system files that are required for the local system and contacts the domain servers to update the domain information.

3.1.2. Configuring IdM from the Command Line

An IdM domain centralizes several common and critical services in a single hierarchy, most notably DNS and Kerberos.

authconfig (much like **realmd** in Chapter 8, *Using **realmd** to Connect to an Identity Domain*) can be used to enroll a system in the IdM domain. That runs the **ipa-client-install** command and, if necessary, installs the **ipa-client** packages. The installation script automatically configures all system files that are required for the local system and contacts the domain servers to update the domain information.

Joining a domain requires three pieces of information to identify the domain: the DNS domain name (**--ipav2domain**), the Kerberos realm name (**--ipav2realm**), and the IdM server to contact (**--ipav2server**). The **--ipav2join** option gives the administrator user name to use to connect to the IdM server; this is typically **admin**.

```
[root@server ~]# authconfig --enableipav2 --ipav2domain=IPAEXAMPLE --
ipav2realm=IPAEXAMPLE --ipav2server=ipaexample.com --ipav2join=admin
```

If the IdM domain is not running its own NTP services, then it is possible to use the **--disableipav2nontp** option to prevent the setup script to use the IdM server as the NTP server. This is generally not recommended, because the IdM server and all clients need to have synchronized clocks for Kerberos authentication and certificates to work properly.

3.2. LDAP AND IDM

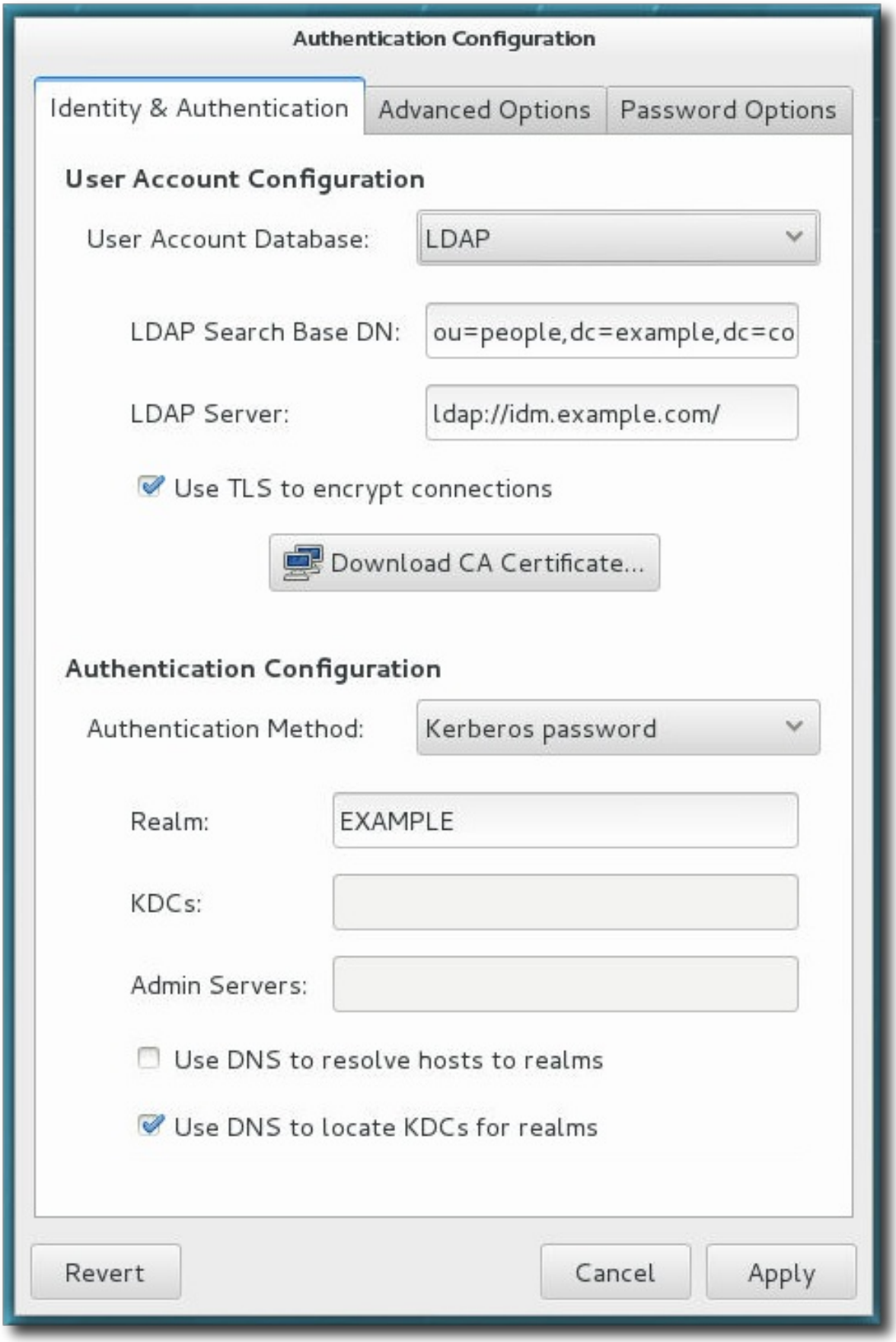
Both standard LDAP directories (such as OpenLDAP and Red Hat Directory Server) can be used as LDAP identity providers. Additionally, older IdM versions and FreeIPA can be configured as identity providers by configuring them as LDAP providers with a related Kerberos server.

Either the openldap-clients package or the sssd package is used to configure an LDAP server for the user database. Both packages are installed by default.

3.2.1. Configuring LDAP Authentication from the UI

1. Open the **authconfig** UI, as in Section 2.2.3, “Launching the authconfig UI”.

2. Select **LDAP** in the **User Account Database** drop-down menu.



3. Set the information that is required to connect to the LDAP server.

   - **`LDAP Search Base DN`** gives the root suffix or *distinguished name* (DN) for the user directory. All of the user entries used for identity or authentication exist below this parent entry. For example, **`ou=people,dc=example,dc=com`**.

This field is optional. If it is not specified, the System Security Services Daemon (SSSD) attempts to detect the search base using the ***namingContexts*** and ***defaultNamingContext*** attributes in the LDAP server's configuration entry.

- **LDAP Server** gives the URL of the LDAP server. This usually requires both the host name and port number of the LDAP server, such as **ldap://ldap.example.com:389**.

  Entering the secure protocol by using a URL starting with **ldaps://** enables the **Download CA Certificate** button, which retrieves the issuing CA certificate for the LDAP server from whatever certificate authority issued it. The CA certificate must be in the privacy enhanced mail (PEM) format.

- If you use a insecure standard port connection (URL starting with **ldap://**), you can use the **Use TLS to encrypt connections** check box to encrypt communication with the LDAP server using **STARTTLS**. Selecting this check box also enables the **Download CA Certificate** button.

NOTE

You do not need to select the **Use TLS to encrypt connections** check box if the server URL uses the LDAPS (LDAP over SSL) secure protocol as the communication is already encrypted.

4. Select the authentication method. LDAP allows simple password authentication or Kerberos authentication.

   Using Kerberos is described in Section 4.3.1, “Configuring Kerberos Authentication from the UI”.

   The **LDAP password** option uses PAM applications to use LDAP authentication. This option requires a secure connection to be set either by using LDAPS or TLS to connect to the LDAP server.

3.2.2. Configuring LDAP User Stores from the Command Line

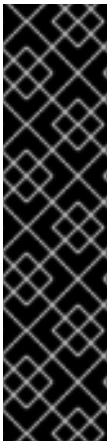
To use an LDAP identity store, use the **--enableldap**. To use LDAP as the authentication source, use **--enableldapauth** and then the requisite connection information, like the LDAP server name, base DN for the user suffix, and (optionally) whether to use TLS. The **authconfig** command also has options to enable or disable RFC 2307bis schema for user entries, which is not possible through the **authconfig** UI.

Be sure to use the full LDAP URL, including the protocol (**ldap** or **ldaps**) and the port number. Do *not* use a secure LDAP URL (**ldaps**) with the **--enableldaptls** option.

```
authconfig --enableldap --enableldapauth --
ldapserver=ldap://ldap.example.com:389,ldap://ldap2.example.com:389 --
ldapbasedn="ou=people,dc=example,dc=com" --enableldaptls --
ldaploadcacert=https://ca.server.example.com/caCert.crt --update
```

Instead of using **--ldapauth** for LDAP password authentication, it is possible to use Kerberos with the LDAP user store. These options are described in Section 4.3.2, “Configuring Kerberos Authentication from the Command Line”.

3.3. NIS



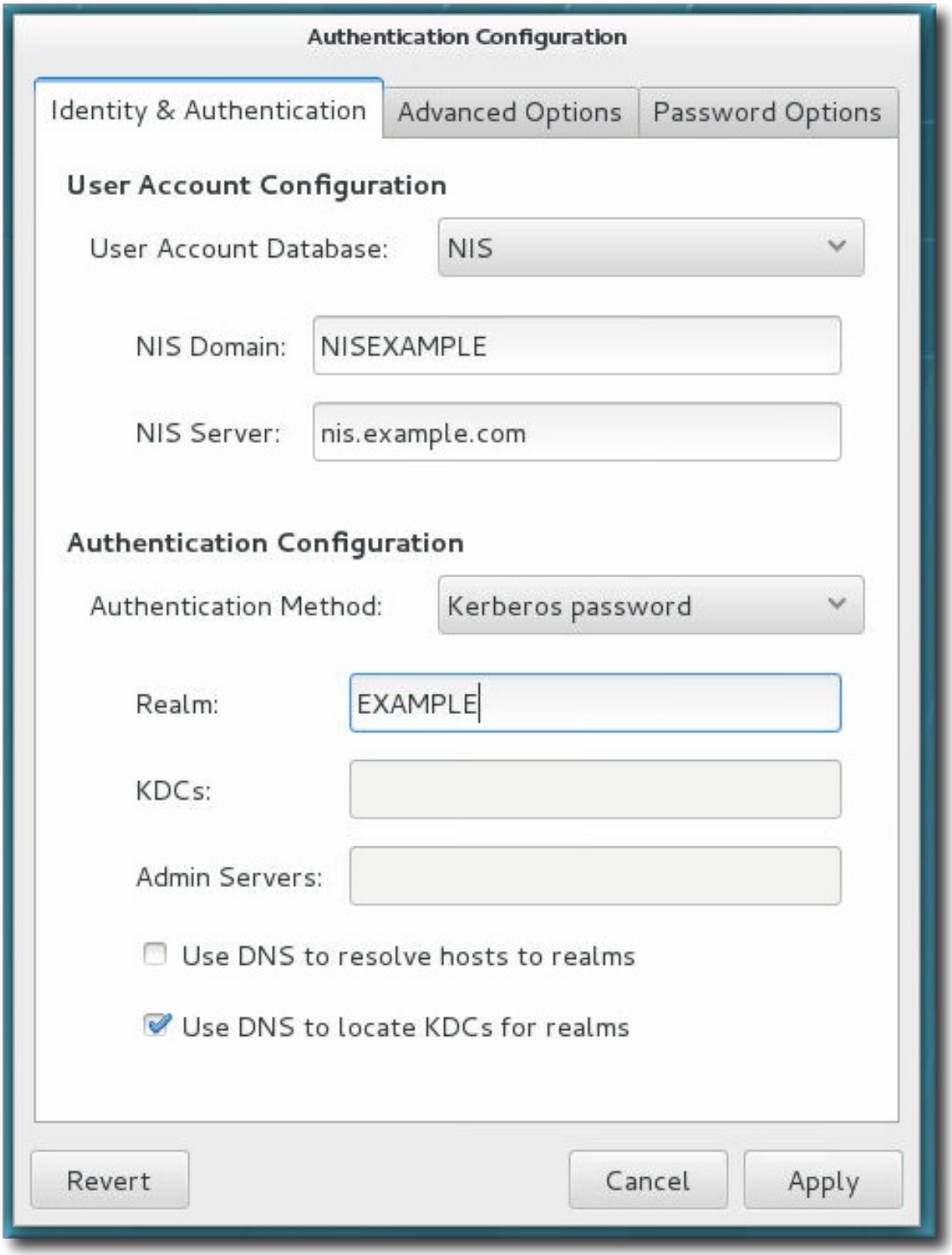
IMPORTANT

Before NIS can be configured as an identity store, NIS itself must be configured for the environment:

- A NIS server must be fully configured with user accounts set up.
- The **ypbind** package must be installed on the local system. This is required for NIS services, but is not installed by default.
- The **portmap** and **ypbind** services are started and enabled to start at boot time. This should be configured as part of the **ypbind** package installation.

3.3.1. Configuring NIS Authentication from the UI

1. Open the **authconfig** UI, as in Section 2.2.3, “Launching the authconfig UI”.
2. Select **NIS** in the **User Account Database** drop-down menu.



3. Set the information to connect to the NIS server, meaning the NIS domain name and the server host name. If the NIS server is not specified, the **authconfig** daemon scans for the NIS server.

4. Select the authentication method. NIS allows simple password authentication or Kerberos authentication.

   Using Kerberos is described in Section 4.3.1, “Configuring Kerberos Authentication from the UI”.

3.3.2. Configuring NIS from the Command Line

To use a NIS identity store, use the **--enablenis**. This automatically uses NIS authentication, unless

the Kerberos parameters are explicitly set (Section 4.3.2, “Configuring Kerberos Authentication from the Command Line”). The only parameters are to identify the NIS server and NIS domain; if these are not used, then the **authconfig** service scans the network for NIS servers.

```
[root@server ~]# authconfig --enablenis --nisdomain=EXAMPLE --
nisserver=nis.example.com --update
```

3.4. WINBIND

Samba must be configured before Winbind can be configured as an identity store for a system. A Samba server must be set up and used for user accounts, or Samba must be configured to use Active Directory as a back end identity store.

Configuring Samba is covered in the Samba project documentation. Specifically configuring Samba as an integration point with Active Directory is also covered in the Red Hat Enterprise Linux Windows Integration Guide.

3.4.1. Enabling Winbind in the authconfig GUI

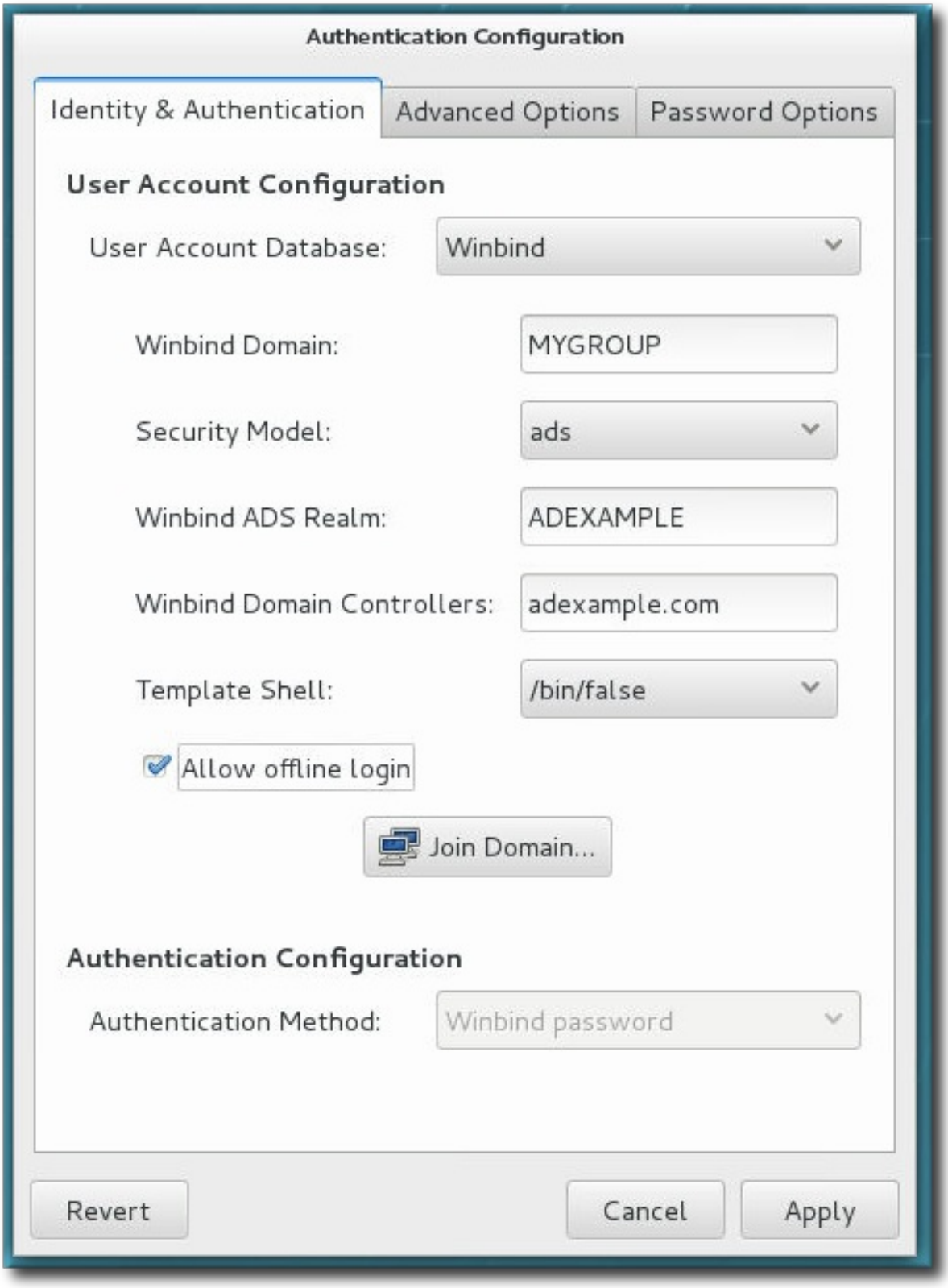
1. Install the **samba-winbind** package. This is required for Windows integration features in Samba services, but is not installed by default.

   ```
   [root@server ~]# yum install samba-winbind
   ```

2. Open the **authconfig** UI.

   ```
   [root2server ~]# authconfig-gtk
   ```

3. In the **Identity & Authentication** tab, select **Winbind** in the **User Account Database** drop-down menu.



4. Set the information that is required to connect to the Microsoft Active Directory domain controller.
   - **`Winbind Domain`** gives the Windows domain to connect to.

     This should be in the Windows 2000 format, such as **`DOMAIN`**.
   - **`Security Model`** sets the security model to use for Samba clients. **`authconfig`** supports four types of security models:

- **ads** configures Samba to act as a domain member in an Active Directory Server realm. To operate in this mode, the **krb5-server** package must be installed and Kerberos must be configured properly.
- **domain** has Samba validate the user name and password by authenticating it through a Windows primary or backup domain controller, much like a Windows server.
- **server** has a local Samba server validate the user name and password by authenticating it through another server, such as a Windows server. If the server authentication attempt fails, the system then attempts to authenticate using **user** mode.
- **user** requires a client to log in with a valid user name and password. This mode does support encrypted passwords.

  The user name format must be *domain**user*, such as **EXAMPLE\jsmith**.

NOTE

When verifying that a given user exists in the Windows domain, always use the **domain\user_name** format and escape the backslash (\) character. For example:

```
[root@server ~]# getent passwd domain\\user
DOMAIN\user:*:16777216:16777216:Name
Surname:/home/DOMAIN/user:/bin/bash
```

  This is the default option.

- **Winbind ADS Realm** gives the Active Directory realm that the Samba server will join. This is only used with the **ads** security model.
- **Winbind Domain Controllers** gives the host name or IP address of the domain controller to use to enroll the system.
- **Template Shell** sets which login shell to use for Windows user account settings.
- **Allow offline login** allows authentication information to be stored in a local cache. The cache is referenced when a user attempts to authenticate to system resources while the system is offline.

3.4.2. Enabling Winbind in the Command Line

Windows domains have several different security models, and the security model used in the domain determines the authentication configuration for the local system. For user and server security models, the Winbind configuration requires only the domain (or workgroup) name and the domain controller host names.

The **--winbindjoin** parameter sets the user to use to connect to the Active Directory domain, and **--enablelocalauthorize** sets local authorization operations to check the **/etc/passwd** file.

After running the **authconfig** command, join the Active Directory domain.

```
[root@server ~]# authconfig --enablewinbind --enablewinbindauth --
smbsecurity=user|server  --enablewinbindoffline --
```

```
smbservers=ad.example.com --smbworkgroup=EXAMPLE --update --
enablelocauthorize --winbindjoin=admin
[root@server ~]# net join ads
```

NOTE

The user name format must be *domain**user*, such as **EXAMPLE\jsmith**.

When verifying that a given user exists in the Windows domain, always use the *domain**user* formats and escape the backslash (\) character. For example:

```
[root@server ~]# getent passwd domain\\user
DOMAIN\user:*:16777216:16777216:Name
Surname:/home/DOMAIN/user:/bin/bash
```

For ads and domain security models, the Winbind configuration allows additional configuration for the template shell and realm (ads only). For example:

```
[root@server ~]# authconfig --enablewinbind --enablewinbindauth --
smbsecurity ads  --enablewinbindoffline --smbservers=ad.example.com --
smbworkgroup=EXAMPLE --smbrealm EXAMPLE.COM --winbindtemplateshell=/bin/sh
--update
```

There are a lot of other options for configuring Windows-based authentication and the information for Windows user accounts, such as name formats, whether to require the domain name with the user name, and UID ranges. These options are listed in the **authconfig** help.

CHAPTER 4. CONFIGURING AUTHENTICATION MECHANISMS

Red Hat Enterprise Linux supports several different authentication methods. They can be configured using the **authconfig** tool or, in some cases, also using Identity Management tools.

4.1. CONFIGURING LOCAL AUTHENTICATION USING AUTHCONFIG

The **Local Authentication Options** area defines settings for local system accounts, not the users stored on the back end. These settings define user-based authorization to system services (as defined in **/etc/security/access.conf**). Otherwise, authorization policies can be defined within the identity provider or the services themselves.

4.1.1. Enabling Local Access Control in the UI

Enable local access control sets the system to check the **/etc/security/access.conf** file for local user authorization rules. This is PAM authorization.

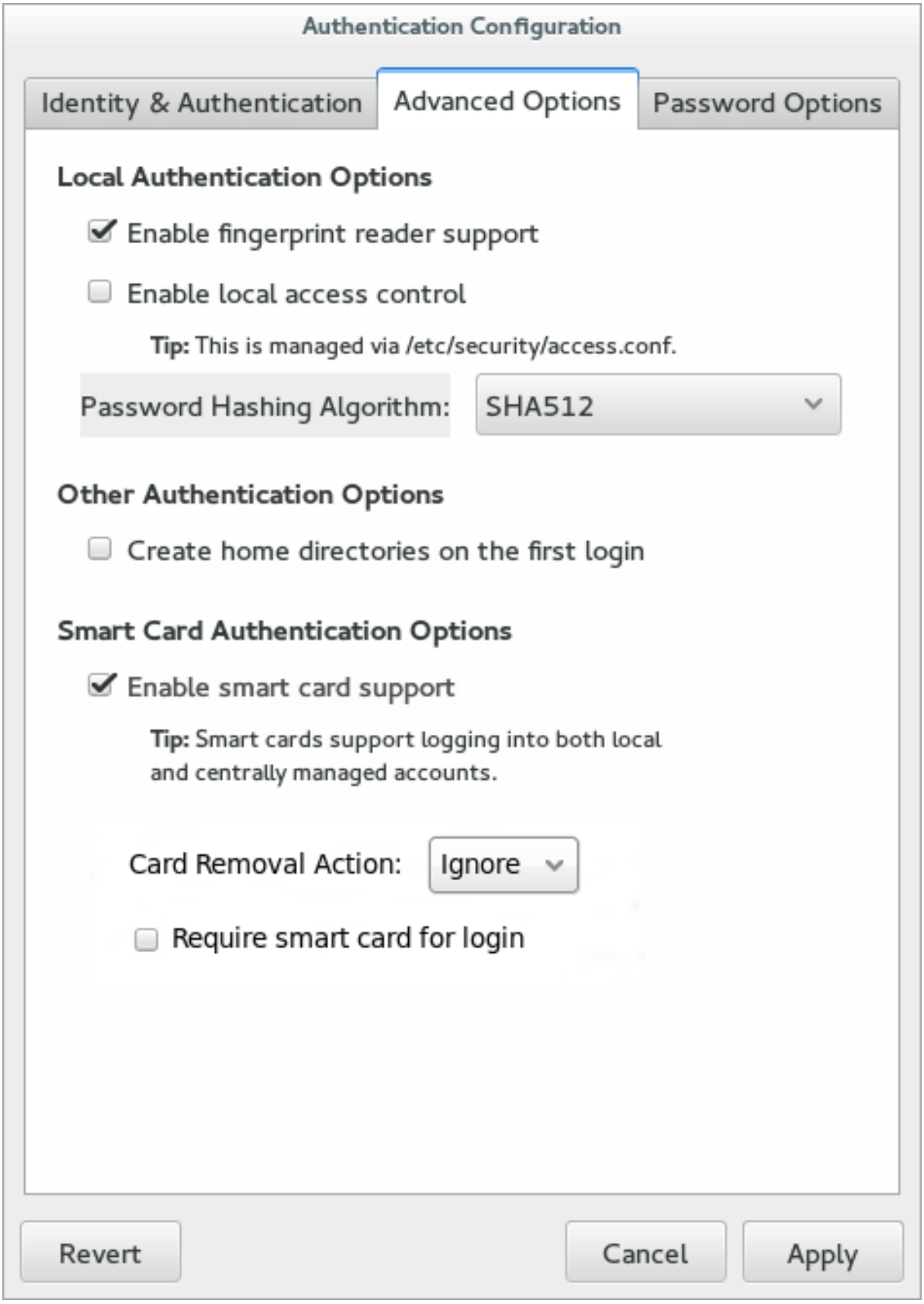


Figure 4.1. Local Accounts Fields

4.1.2. Configuring Local Access Control in the Command Line

There are two options for **authconfig** to enable local authorization controls. **--enablelocauthorize** skips network authentication and only checks local files for system users. **--enablepamaccess** configures the system to look for system authorization policies in

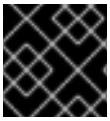
/etc/security/access.conf.

```
[root@server ~]# authconfig --enablelocauthorize --enablepamaccess --
update
```

4.2. CONFIGURING SYSTEM PASSWORDS USING AUTHCONFIG

4.2.1. Password Security

If passwords are stored in plain text format, they are vulnerable to cracking, unauthorized access, or tampering. To prevent this, cryptographic hashing algorithms can be used to securely store password hash digests. The recommended (and also default) hashing algorithm supported in IdM is SHA-512, which uses 64-bit words and also salt and stretching for extra security. To ensure backward compatibility, the SHA-256, DES, BigCrypt, and MD5 hashing algorithms are also supported.



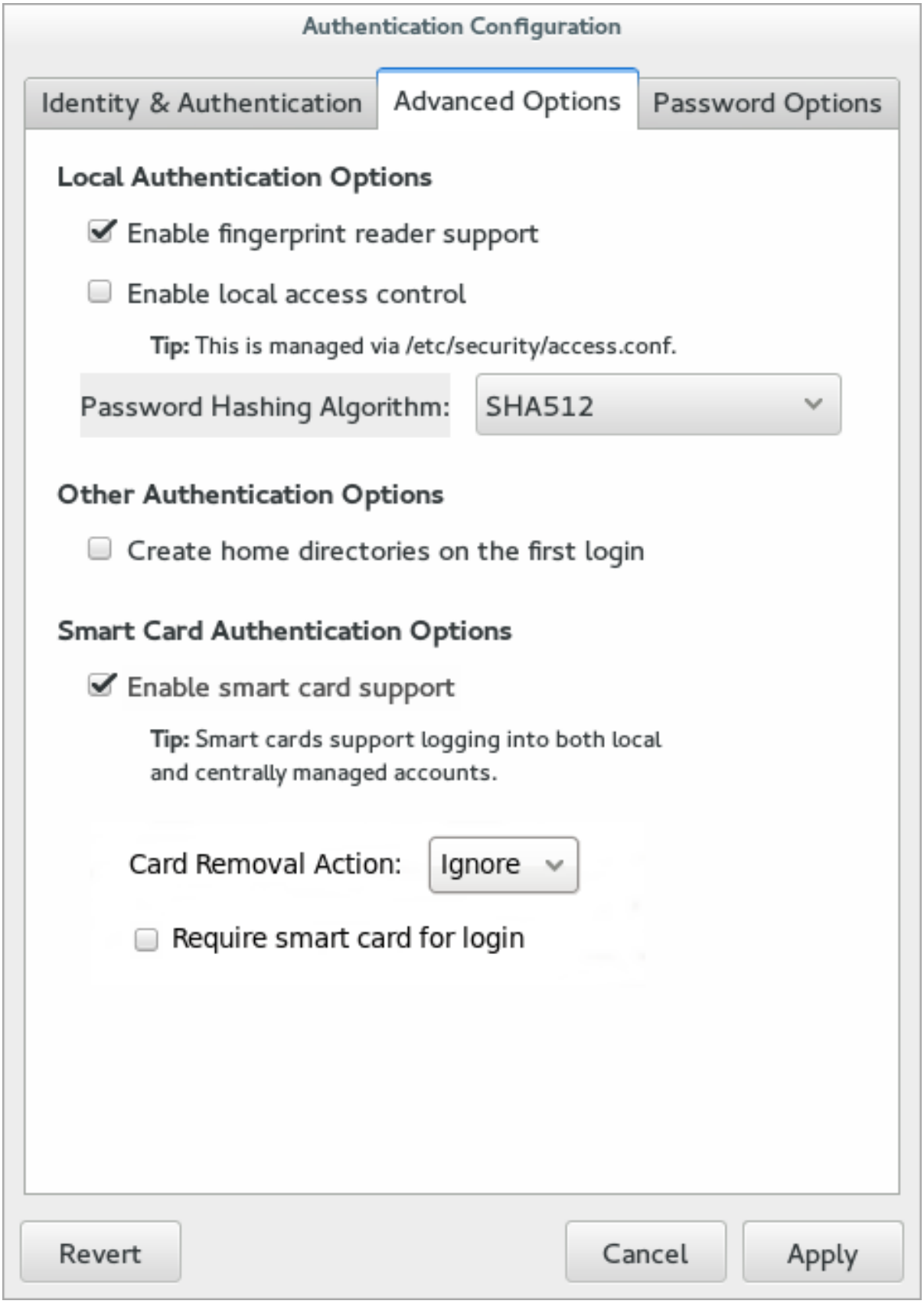
IMPORTANT

If you do not need backward compatibility, only use SHA-512 as it is more secure.

4.2.1.1. Configuring Password Hashing in the UI

The **Local Authentication Options** tab sets how local passwords are stored on the system. The **Password Hashing Algorithm** drop-down menu sets the algorithm to securely store passwords hashes.

1. Open the **authconfig** UI, as in Section 2.2.3, “Launching the authconfig UI”.
2. Open the **Advanced Options** tab.
3. Select the algorithm to use in the **Password Hashing Algorithm** drop-down menu.



4. Click the **Apply** button.

4.2.1.2. Configuring Password Hashing on the Command Line

To set or change the hashing algorithm used to securely store user passwords digests, use the **--passalgo** option and the short name for the algorithm. The following example uses the SHA-512 algorithm:

```
[root@server ~]# authconfig --passalgo=sha512 --update
```

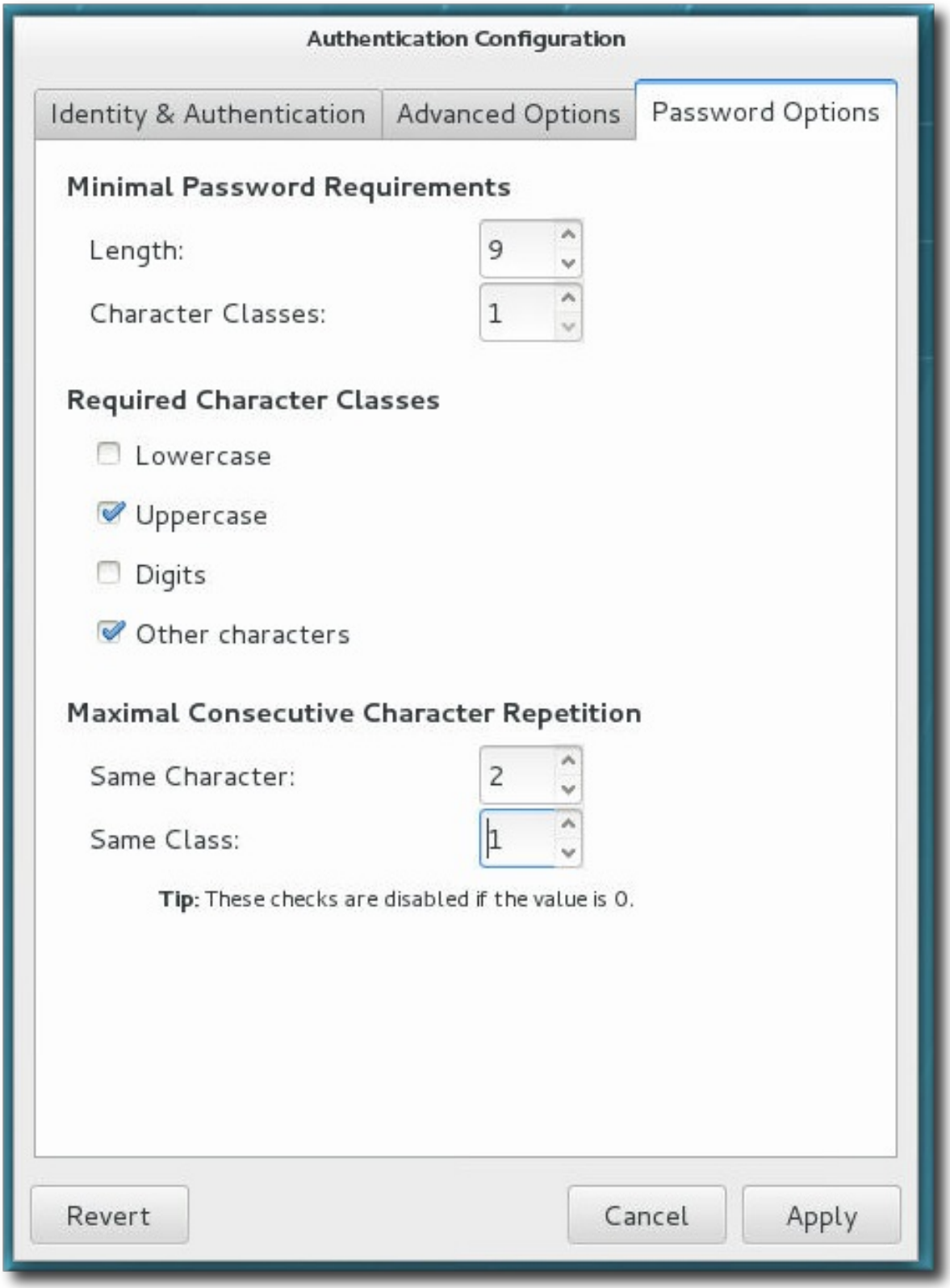
4.2.2. Password Complexity

Password complexity sets how strong a password must be for it to be allowed to be set for a local user account. Complexity is a combination of length and a variation of character classes. One way to look at it is that there are two parts to setting policy for complex passwords: identifying what types of characters can be used in a password (such as upper and lower case letters and special characters) and how those characters can be used within the password (how long must it be and how often can those characters be repeated).

4.2.2.1. Configuring Password Complexity in the UI

1. Open the **authconfig** UI, as in Section 2.2.3, “Launching the authconfig UI”.

2. Open the **Password Options** tab.



3. Set the *minimum* requirements for the password:
   - The minimum length of the password
   - The minimum number of character classes which must be used in the password.
4. Enable characters classes which *must* be used for passwords. For example, an uppercase letter can be used with any password, but if the **`Uppercase`** check box is selected, then an uppercase letter must be used in every password.

5. Set the number of times that a character or character class can be repeated consecutively. (If this is set to zero, then there is no repeat limit.)

   For the **Same Character** field, this sets how often a single letter or character can be repeated. If this is set to 2, for example, then *ssecret* is allowed but *sssecret* is rejected.

   Likewise, **Same Class** sets a limit on how many times any character from a character class (uppercase, number, special character) can be repeated. If this is set to 3, for example, *secret!!* is allowed but *secret!!@* or *secret1234* would be rejected.

6. Click the **Apply** button.

4.2.2.2. Configuring Password Complexity in the Command Line

When defining password complexity in the comment line, there are two halves to setting the requirements. The first is setting the requirements on how a password is constructed — its length, can characters be repeated, and how many different types of characters must be used:

- The minimum length (**--passminlen**).
- The minimum number of different types of characters which must be used (**--passminclass**).
- The number of times a character can be repeated consecutively (**--passmaxrepeat**). Setting this to zero means there is no repeat limit.
- The number of time the same type of character (such as a number) can be used in a row (**--passmaxclassrepeat**). Setting this to zero means there is no repeat limit.

The second half is defining what types or classes of characters are allowed to be used for passwords. All character types are implicitly allowed; using the **--enablereq***Type* option means that a given class is absolutely required or the password is rejected. (Conversely, types can be explicitly denied, as well.)

- Uppercase letters (**--enablerequpper**)
- Lowercase letters (**--enablereqlower**)
- Numbers (**--enablereqdigit**)
- Special characters (**--enablereqother**)

For example, this sets a minimum length of nine characters, does not allow characters or classes to be repeated more than twice, and requires both uppercase and special characters.

```
[root@server ~]# authconfig --passminlen=9 --passminclass=3 --
passmaxrepeat=2 -passmaxclassrepeat=2 --enablerequpper --enablereqother --
update
```

4.3. CONFIGURING KERBEROS (WITH LDAP OR NIS) USING AUTHCONFIG

Both LDAP and NIS authentication stores support Kerberos authentication methods. Using Kerberos has a couple of benefits:

- It uses a security layer for communication while still allowing connections over standard ports.
- It automatically uses credentials caching with SSSD, which allows offline logins.

NOTE

Using Kerberos authentication requires the **krb5-libs** and **krb5-workstation** packages.

4.3.1. Configuring Kerberos Authentication from the UI

The **Kerberos password** option from the **Authentication Method** drop-down menu automatically opens the fields required to connect to the Kerberos realm.

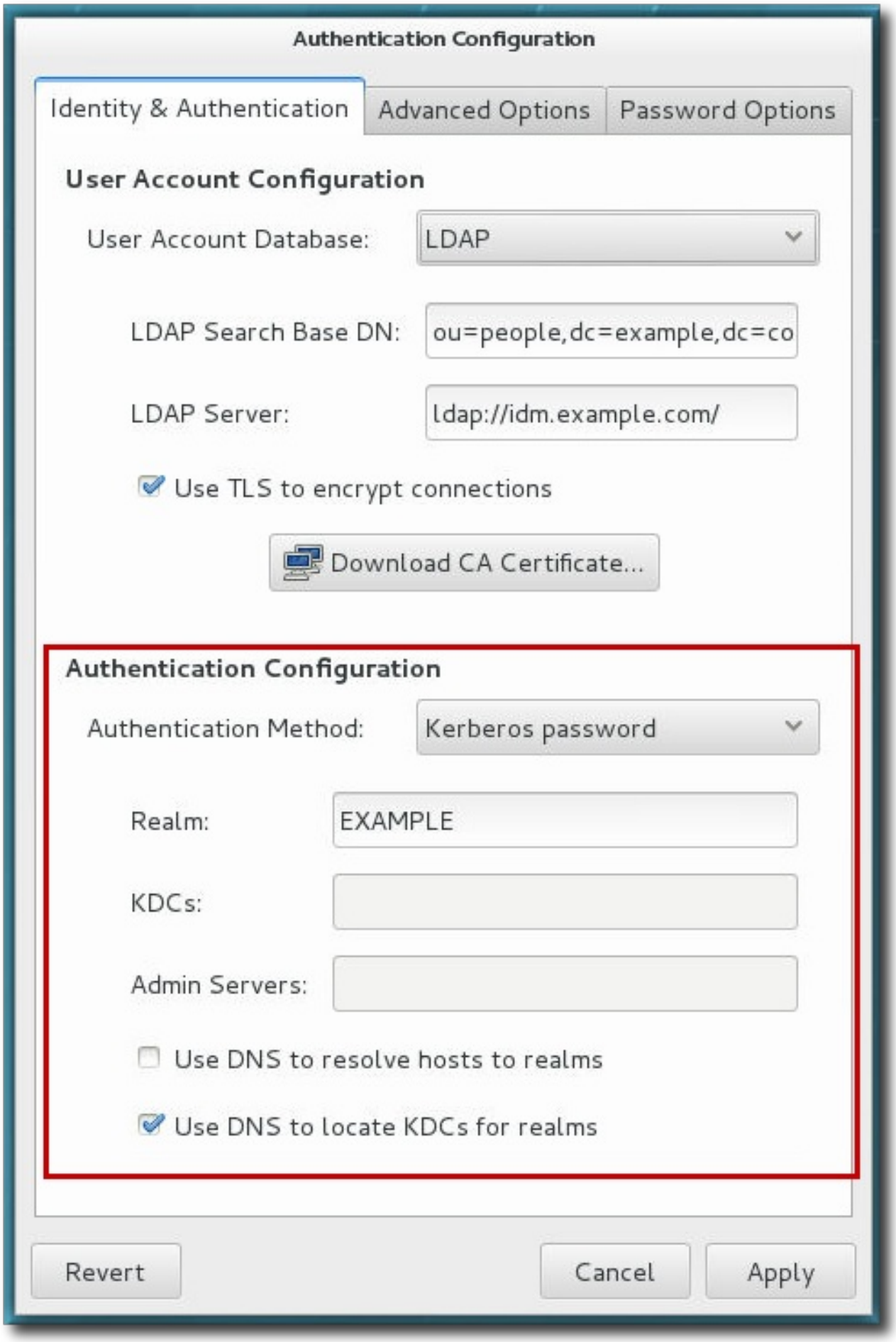


Figure 4.2. Kerberos Fields

- **Realm** gives the name for the realm for the Kerberos server. The realm is the network that uses Kerberos, composed of one or more *key distribution centers* (KDC) and a potentially large number of clients.
- **KDCs** gives a comma-separated list of servers that issue Kerberos tickets.
- **Admin Servers** gives a list of administration servers running the **kadmind** process in the realm.
- Optionally, use DNS to resolve server host name and to find additional KDCs within the realm.

4.3.2. Configuring Kerberos Authentication from the Command Line

Both LDAP and NIS allow Kerberos authentication to be used in place of their native authentication mechanisms. At a minimum, using Kerberos authentication requires specifying the realm, the KDC, and the administrative server. There are also options to use DNS to resolve client names and to find additional admin servers.

```
[root@server ~]# authconfig NIS or LDAP options --enablekrb5 --krb5realm
EXAMPLE --krb5kdc kdc.example.com:88,server.example.com:88 --
krb5adminserver server.example.com:749 --enablekrb5kdcdns --
enablekrb5realmdns --update
```

4.4. SMART CARDS

Authentication based on smart cards is an alternative to password-based authentication. User credentials are stored on the smart card, and special software and hardware is then used to access them. In order to authenticate using a smart card, the user must place the smart card into a smart card reader and then supply the PIN code for the smart card.

IMPORTANT

The following sections describe how to configure a single system for smart card authentication with local users by using the pam_pkcs11 and pam_krb5 packages. Note that these packages are now deprecated, as described in Deprecated Functionality in the *7.4 Release Notes*.

To configure smart card authentication centrally, use the enhanced smart card functionality provided by the System Security Services Daemon (SSSD). For details, see Smart-card Authentication in Identity Management in the *Linux Domain Identity, Authentication, and Policy Guide* for Red Hat Identity Management.

4.4.1. Configuring Smart Cards Using `authconfig`

Once the **Enable smart card support** option is selected, additional controls for configuring behavior of smart cards appear.

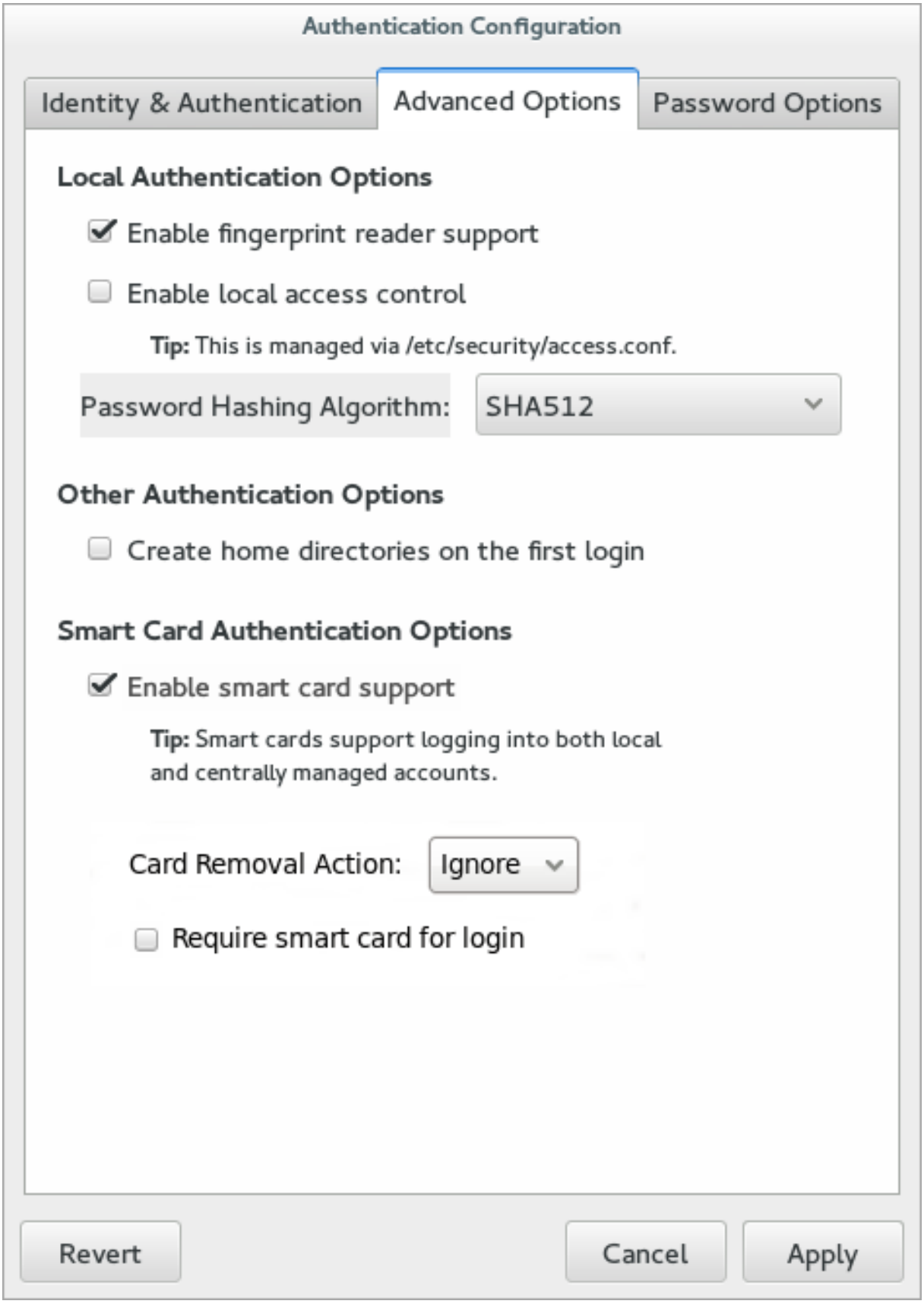


Figure 4.3. Smart Card Options

Note that smart card login for Red Hat Enterprise Linux servers and workstations is not enabled by default and must be enabled in the system settings.

NOTE

Using single sign-on when logging into Red Hat Enterprise Linux requires these packages:

- nss-tools
- nss-pam-ldapd
- esc
- pam_pkcs11
- pam_krb5
- opensc
- pcsc-lite-ccid
- gdm
- authconfig
- authconfig-gtk
- krb5-libs
- krb5-workstation
- krb5-pkinit
- pcsc-lite
- pcsc-lite-libs

4.4.1.1. Enabling Smart Card Authentication from the UI

1. Log into the system as root.
2. Download the root CA certificates for the network in base 64 format, and install them on the server. The certificates are installed in the appropriate system database using the **certutil** command. For example:

```
[root@server ~]# certutil -A -d /etc/pki/nssdb -n "root CA cert" -t
"CT,C,C" -i /tmp/ca_cert.crt
```

NOTE

Do not be concerned that the imported certificate is not displayed in the **authconfig** UI later during the process. You cannot see the certificate in the UI; it is obtained from the **/etc/pki/nssdb/** directory during authentication.

3. In the top menu, select the **Application** menu, select **Sundry**, and then click **Authentication**.
4. Open the **Advanced Options** tab.

5. Click the **Enable Smart Card Support** check box.

6. There are two behaviors that can be configured for smart cards:

   - The **Card removal action** menu sets the response that the system takes if the smart card is removed during an active session. The **Ignore** option means that the system continues functioning as normal if the smart card is removed, while **Lock** immediately locks the screen.

   - The **Require smart card for login** check box sets whether a smart card is required for logins. When this option is selected, all other methods of authentication are blocked.

     WARNING

     Do not select this until *after* you have successfully logged in using a smart card.

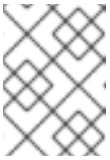
7. By default, the mechanisms to check whether a certificate has been revoked (Online Certificate Status Protocol, or OCSP, responses) are disabled. To validate whether a certificate has been revoked before its expiration period, enable OCSP checking by adding the **ocsp_on** option to the ***cert_policy*** directive.

   1. Open the **pam_pkcs11.conf** file.

      ```
      vim /etc/pam_pkcs11/pam_pkcs11.conf
      ```

   2. Change every ***cert_policy*** line so that it contains the **ocsp_on** option.

      ```
      cert_policy = ca, ocsp_on, signature;
      ```

      NOTE

      Because of the way the file is parsed, there *must* be a space between **cert_policy** and the equals sign. Otherwise, parsing the parameter fails.

8. If the smart card has not yet been enrolled (set up with personal certificates and keys), enroll the smart card.

9. If the smart card is a CAC card, create the **.k5login** file in the CAC user's home directory. The **.k5login** file is required to have the Microsoft Principal Name on the CAC card.

10. Add the following line to the **/etc/pam.d/smartcard-auth** and **/etc/pam.d/system-auth** files:

    ```
    auth    optional    pam_krb5.so use_first_pass no_subsequent_prompt
    preauth_options=X509_user_identity=PKCS11:/usr/lib64/pkcs11/opensc-
    pkcs11.so
    ```

If the OpenSC module does not work as expected, use the module from the coolkey package: **/usr/lib64/pkcs11/libcoolkeypk11.so**. In this case, consider contacting Red Hat Technical Support or filing a Bugzilla report about the problem.

11. Configure the **/etc/krb5.conf** file. The settings vary depending on whether you are using a CAC card or a Gemalto 64K card.

    - With CAC cards, specify all the root certificates related to the CAC card usage in **pkinit_anchors**. In the following example **/etc/krb5.conf** file for configuring a CAC card, *EXAMPLE.COM* is the realm name for the CAC cards, and *kdc.server.hostname.com* is the KDC server host name.

```
[logging]
  default = FILE:/var/log/krb5libs.log
  kdc = FILE:/var/log/krb5kdc.log
  admin_server = FILE:/var/log/kadmind.log

[libdefaults]
  dns_lookup_realm = false
  dns_lookup_kdc = false
  ticket_lifetime = 1h
  renew_lifetime = 6h
  forwardable = true

  default_realm = EXAMPLE.COM
[realms]
  EXAMPLE.COM = {
    kdc = kdc.server.hostname.com
    admin_server = kdc.server.hostname.com
    pkinit_anchors = FILE:/etc/pki/nssdb/ca_cert.pem
    pkinit_anchors = FILE:/etc/pki/nssdb/CAC_CA_cert.pem
    pkinit_anchors = FILE:/etc/pki/nssdb/CAC_CA_email_cert.pem
    pkinit_anchors = FILE:/etc/pki/nssdb/CAC_root_ca_cert.pem
    pkinit_cert_match = CAC card specific information
  }

[domain_realm]
  EXAMPLE.COM = EXAMPLE.COM
  .EXAMPLE.COM = EXAMPLE.COM

  .kdc.server.hostname.com = EXAMPLE.COM
  kdc.server.hostname.com = EXAMPLE.COM

[appdefaults]
    pam = {
      debug = true
      ticket_lifetime = 1h
      renew_lifetime = 3h
      forwardable = true
      krb4_convert = false
      mappings = username on the CAC card     Principal name on
the card
    }
```

- In the following example **/etc/krb5.conf** file for configuring a Gemalto 64K card, *EXAMPLE.COM* is the realm created on the KDC server, *kdc-ca.pem* is the CA certificate, and *kdc.server.hostname.com* is the KDC server host name.

```
[logging]
  default = FILE:/var/log/krb5libs.log
  kdc = FILE:/var/log/krb5kdc.log
  admin_server = FILE:/var/log/kadmind.log

[libdefaults]
  dns_lookup_realm = false
  dns_lookup_kdc = false
  ticket_lifetime = 15m
  renew_lifetime = 6h
  forwardable = true

  default_realm = EXAMPLE.COM
[realms]
  EXAMPLE.COM = {
    kdc = kdc.server.hostname.com
    admin_server = kdc.server.hostname.com
    pkinit_anchors = FILE:/etc/pki/nssdb/kdc-ca.pem
    pkinit_cert_match = <KU>digitalSignature
    pkinit_kdc_hostname = kdc.server.hostname.com
  }

[domain_realm]
  EXAMPLE.COM = EXAMPLE.COM
  .EXAMPLE.COM = EXAMPLE.COM

  .kdc.server.hostname.com = EXAMPLE.COM
  kdc.server.hostname.com = EXAMPLE.COM

[appdefaults]
    pam = {
      debug = true
      ticket_lifetime = 1h
      renew_lifetime = 3h
      forwardable = true
      krb4_convert = false
    }
```

NOTE

When a smart card is inserted, the **pklogin_finder** utility, when run in debug mode, first maps the login ID to the certificates on the card and then attempts to output information about the validity of certificates:

```
pklogin_finder debug
```

The command is useful for diagnosing problems with using a smart card to log into the system.

4.4.1.2. Configuring Smart Card Authentication from the Command Line

All that is required to use smart cards with a system is to set the **--enablesmartcard** option:

```
[root@server ~]# authconfig --enablesmartcard --update
```

There are other configuration options for smart cards, such as changing the default smart card module, setting the behavior of the system when the smart card is removed, and requiring smart cards for login.

A value of **0** instructs the system to lock out a user immediately if the smart card is removed; a setting of **1** ignores it if the smart card is removed:

```
[root@server ~]# authconfig --enablesmartcard --smartcardaction=0 --update
```

Once smart card authentication has been successfully configured and tested, then the system can be configured to require smart card authentication for users rather than simple password-based authentication.

```
[root@server ~]# authconfig --enablerequiresmartcard --update
```

WARNING

Do not use the **--enablerequiresmartcard** option until you have successfully authenticated to the system using a smart card. Otherwise, users may be unable to log into the system.

4.4.2. Smart Card Authentication in Identity Management

Red Hat Identity Management supports smart card authentication for IdM users. For more information, see the Linux Domain Identity, Authentication, and Policy Guide.

4.4.3. Supported Smart Cards

The following smart cards and readers are supported on Red Hat Enterprise Linux.

Smart Cards

- Athena ASECard Crypto smart, pkcs15-unit
- ATOS (Siemens) CardOS 5.0
- Gemalto ID Classic 230 / TOP IM CY2 64kv2
- Gemalto Cyberflex Access 64k V2c
- Gemalto GemPCKey USB form factor key
- Giesecke & Devrient (G&D) SmartCafe Expert 6.0 (SCP03)
- Giesecke & Devrient (G&D) SmartCafe Expert 7.0 (SCP03)

- Safenet 330J
- Safenet SC650 (SCP01)
- Siemens Card CardOS M4.4
- Yubikey 4

Readers

- HP Keyboard KUS1206 with built in Smart Card reader
- Omnikey 3121 reader
- Omnikey 3121 with PID 0x3022 reader
- Reiner SCT cyberJack RFID komfort reader
- SCR331 CCID reader

4.5. ONE-TIME PASSWORDS

One-time password (OTP) is a password that is valid for only one authentication session; it becomes invalid after use. Unlike traditional static passwords that stay the same for a longer period of time, OTPs keep changing. OTPs are used as part of two-factor authentication: the first step requires the user to authenticate with a traditional static password, and the second step prompts for an OTP issued by a recognized authentication token.

Authentication using an OTP combined with a static password is considered safer than authentication using a static password alone. Because an OTP can only be used for successful authentication once, even if a potential intruder intercepts the OTP during login, the intercepted OTP will already be invalid by that point.

One-Time Passwords in Red Hat Enterprise Linux
Red Hat Identity Management supports OTP authentication for IdM users. For more information, see the Linux Domain Identity, Authentication, and Policy Guide.

4.6. CONFIGURING FINGERPRINTS USING `AUTHCONFIG`

4.6.1. Using Fingerprint Authentication in the UI

When there is appropriate hardware available, the **`Enable fingerprint reader support`** option allows fingerprint scans to be used to authenticate local users in addition to other credentials.

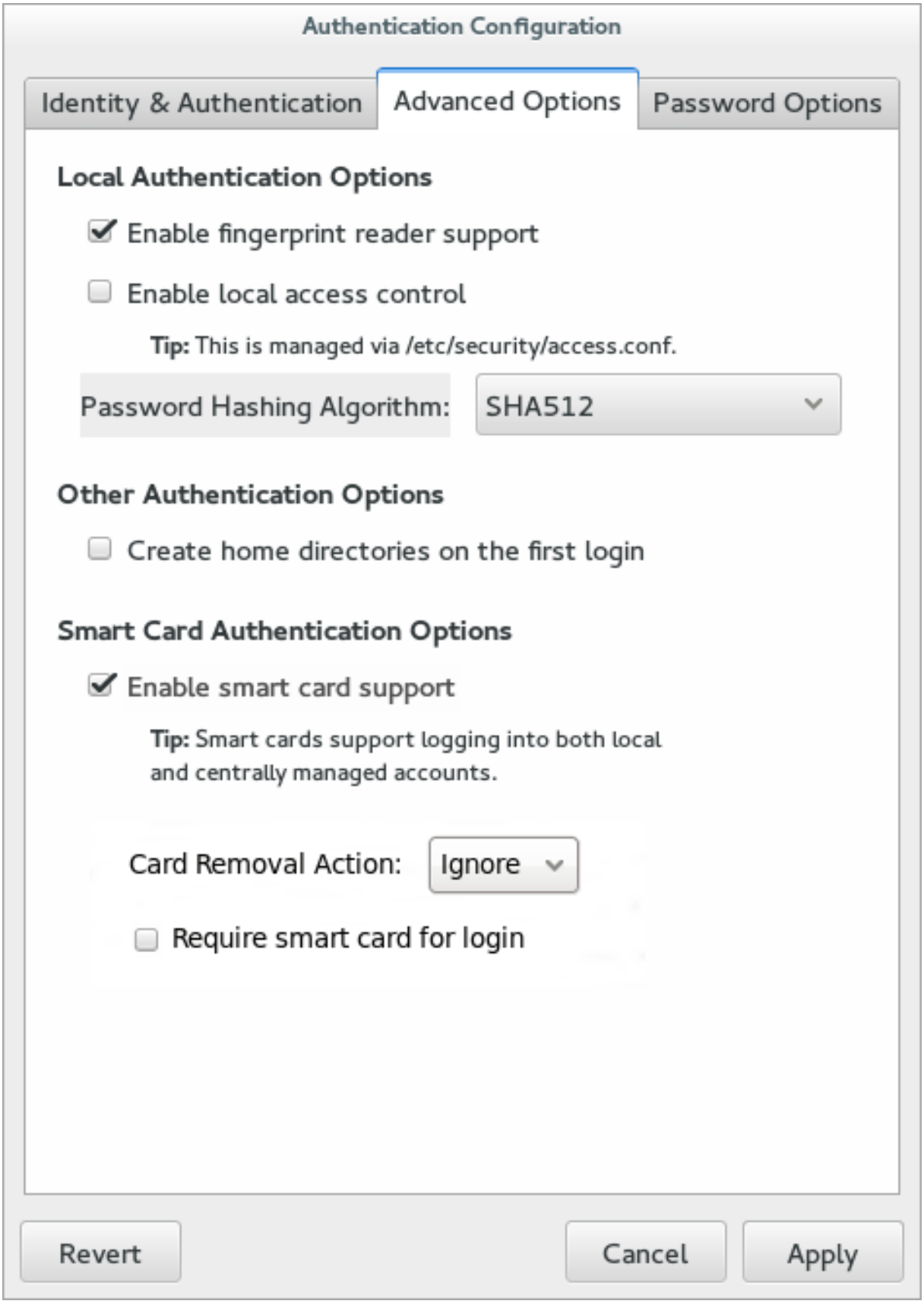


Figure 4.4. Fingerprint Options

4.6.2. Configuring Fingerprint Authentication in the Command Line

There is one option to enable support for fingerprint readers. This option can be used alone or in conjunction with other **`authconfig`** settings, like LDAP user stores.

```
[root@server ~]# authconfig --enablefingerprint --update
```

CHAPTER 5. MANAGING KICKSTART AND CONFIGURATION FILES USING AUTHCONFIG

The **--update** option updates all of the configuration files with the configuration changes. There are a couple of alternative options with slightly different behavior:

- **--kickstart** writes the updated configuration to a kickstart file.
- **--test** displays the full configuration with changes, but does not edit any configuration files.

Additionally, **authconfig** can be used to back up and restore previous configurations. All archives are saved to a unique subdirectory in the **/var/lib/authconfig/** directory. For example, the **--savebackup** option gives the backup directory as **2011-07-01**:

```
[root@server ~]# authconfig --savebackup=2011-07-01
```

This backs up all of the authentication configuration files beneath the **/var/lib/authconfig/backup-2011-07-01** directory.

Any of the saved backups can be used to restore the configuration using the **--restorebackup** option, giving the name of the manually saved configuration:

```
[root@server ~]# authconfig --restorebackup=2011-07-01
```

Additionally, **authconfig** automatically makes a backup of the configuration before it applies any changes (with the **--update** option). The configuration can be restored from the most recent automatic backup, without having to specify the exact backup, using the **--restorelastbackup** option.

CHAPTER 6. ENABLING CUSTOM HOME DIRECTORIES USING AUTHCONFIG

If LDAP users have home directories that are not in **/home** and the system is configured to create home directories the first time users log in, then these directories are created with the wrong permissions.

1. Apply the correct SELinux context and permissions from the **/home** directory to the home directory that is created on the local system. For example:

   ```
   [root@server ~]# semanage fcontext -a -e /home /home/locale
   ```

2. Install the **oddjob-mkhomedir** package on the system.

   This package provides the **pam_oddjob_mkhomedir.so** library, which the **authconfig** command uses to create home directories. The **pam_oddjob_mkhomedir.so** library, unlike the default **pam_mkhomedir.so** library, can create SELinux labels.

   The **authconfig** command automatically uses the **pam_oddjob_mkhomedir.so** library if it is available. Otherwise, it will default to using **pam_mkhomedir.so**.

3. Make sure the **oddjobd** service is running.

4. Run the **authconfig** command and enable home directories. In the command line, this is done through the **--enablemkhomedir** option.

   ```
   [root@server ~]# authconfig --enablemkhomedir --update
   ```

   In the UI, there is an option in the **Advanced Options** tab (**Create home directories on the first login**) to create a home directory automatically the first time that a user logs in.

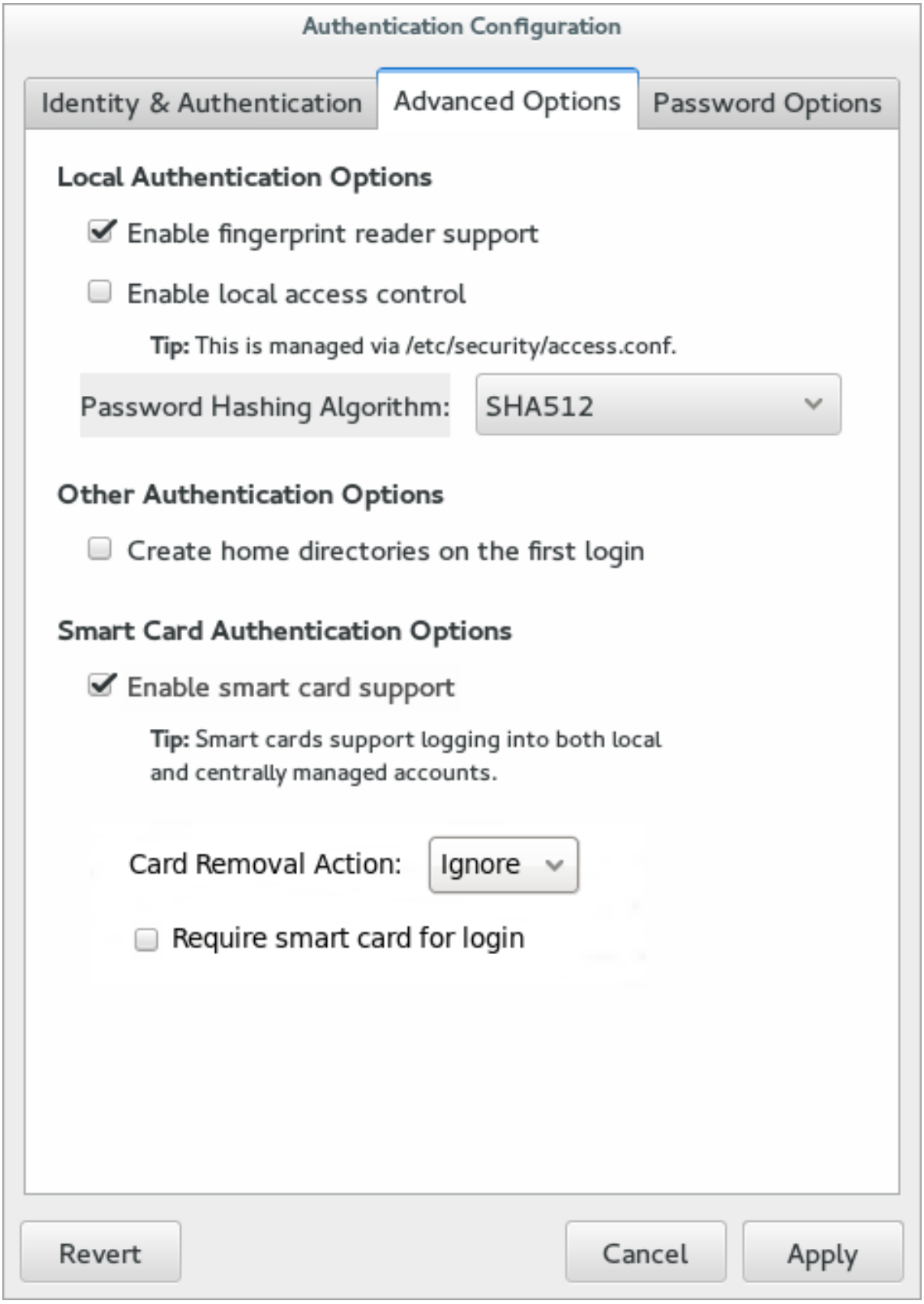


Figure 6.1. Home Directory Option

This option is beneficial with accounts that are managed centrally, such as with LDAP. However, this option should not be selected if a system like automount is used to manage user home directories.

If home directories were created before the home directory configuration was changed, then correct the permissions and SELinux contexts. For example:

```
[root@server ~]# semanage fcontext -a -e /home /home/locale
# restorecon -R -v /home/locale
```

PART II. IDENTITY AND AUTHENTICATION STORES

CHAPTER 7. CONFIGURING SSSD

7.1. INTRODUCTION TO SSSD

7.1.1. How SSSD Works

The System Security Services Daemon (SSSD) is a system service to access remote directories and authentication mechanisms. It connects a local system (an SSSD *client*) to an external back-end system (a *provider*). This provides the SSSD client with access to identity and authentication remote services using an SSSD provider. For example, these remote services include: an LDAP directory, an Identity Management (IdM) or Active Directory (AD) domain, or a Kerberos realm.

For this purpose, SSSD:

1. Connects the client to an identity store to retrieve authentication information.
2. Uses the obtained authentication information to create a local cache of users and credentials on the client.

Users on the local system are then able to authenticate using the user accounts stored in the external back-end system.

SSSD does not create user accounts on the local system. Instead, it uses the identities from the external data store and lets the users access the local system.

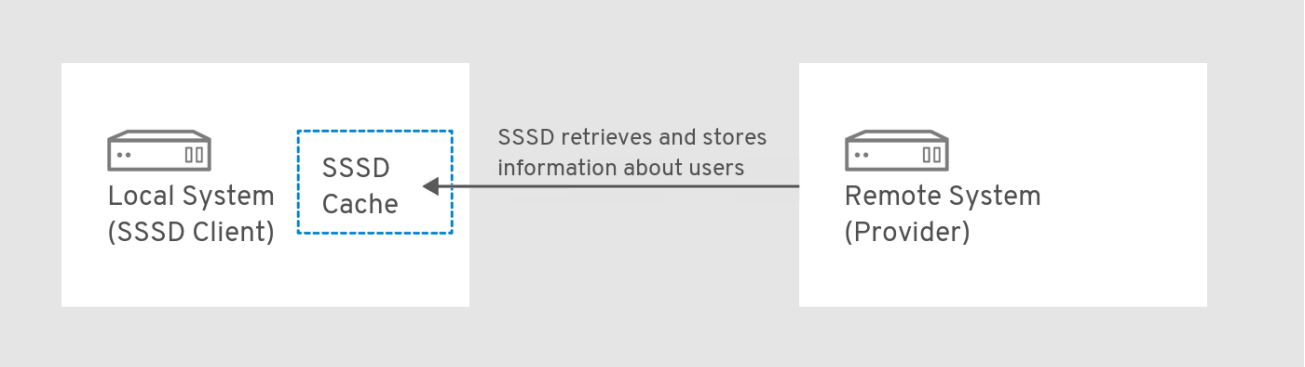


Figure 7.1. How SSSD works

SSSD can also provide caches for several system services, such as Name Service Switch (NSS) or Pluggable Authentication Modules (PAM).

7.1.2. Benefits of Using SSSD

Reduced load on identity and authentication servers

When requesting information, SSSD clients contact SSSD, which checks its cache. SSSD contacts the servers only if the information is not available in the cache.

Offline authentication

SSSD optionally keeps a cache of user identities and credentials retrieved from remote services. In this setup, users can successfully authenticate to resources even if the remote server or the SSSD client are offline.

A single user account: improved consistency of the authentication process

With SSSD, it is not necessary to maintain both a central account and a local user account for offline authentication.

Remote users often have multiple user accounts. For example, to connect to a virtual private network (VPN), remote users have one account for the local system and another account for the VPN system.

Thanks to caching and offline authentication, remote users can connect to network resources simply by authenticating to their local machine. SSSD then maintains their network credentials.

7.2. USING MULTIPLE SSSD CONFIGURATION FILES ON A PER-CLIENT BASIS

The default configuration file for SSSD is **`/etc/sssd/sssd.conf`**. Apart from this file, SSSD can read its configuration from all **`*.conf`** files in the **`/etc/sssd/conf.d/`** directory.

For example, this allows you to use the default **`/etc/sssd/sssd.conf`** file on all clients and add additional settings in further configuration files to extend the functionality individually on a per-client basis.

How SSSD Processes the Configuration Files

SSSD reads the configuration files in this order:

1. The primary **`/etc/sssd/sssd.conf`** file
2. Other **`*.conf`** files in **`/etc/sssd/conf.d/`**, in alphabetical order

If the same parameter appears in multiple configuration files, SSSD uses the last read parameter.

NOTE

SSSD does not read hidden files (files starting with **`.`**) in the **`conf.d`** directory.

7.3. CONFIGURING IDENTITY AND AUTHENTICATION PROVIDERS FOR SSSD

7.3.1. Introduction to Identity and Authentication Providers for SSSD

SSSD Domains. Identity and Authentication Providers

Identity and authentication providers are configured as *domains* in the SSSD configuration file. A single domain can be used as:

- An *identity provider* (for user information)
- An *authentication provider* (for authentication requests)
- An *access control provider* (for authorization requests)
- A combination of these providers (if all the corresponding operations are performed within a single server)

You can configure multiple domains for SSSD. At least one domain must be configured, otherwise SSSD will not start.

The **access_provider** option in the **/etc/sssd/sssd.conf** file sets the access control provider used for the domain. By default, the option is set to **permit**, which always allows all access. See the sssd.conf(5) man page for details.

Proxy Providers

A proxy provider works as an intermediary relay between SSSD and resources that SSSD would otherwise not be able to use. When using a proxy provider, SSSD connects to the proxy service, and the proxy loads the specified libraries.

Using a proxy provider, you can configure SSSD to use:

- Alternative authentication methods, such as a fingerprint scanner
- Legacy systems, such as NIS
- A local system account defined in **/etc/passwd** and remote authentication

Available Combinations of Identity and Authentication Providers

Table 7.1. Available Combinations of Identity and Authentication Providers

Identity Provider	Authentication Provider
Identity Management [a]	Identity Management [a]
Active Directory [a]	Active Directory [a]
LDAP	LDAP
LDAP	Kerberos
proxy	proxy
proxy	LDAP
proxy	Kerberos

[a] An extension of the LDAP provider type.

Note that this guide does not describe all provider types. See the following additional resources for more information:

- To configure an SSSD client for Identity Management, Red Hat recommends using the **ipa-client-install** utility. See Installing and Uninstalling Identity Management Clients in the *Linux Domain Identity, Authentication, and Policy Guide*.
- To configure an SSSD client for Identity Management manually without **ipa-client-install**, see Installing and Uninstalling an Identity Management Client Manually in Red Hat Knowledgebase.

- To configure Active Directory to be used with SSSD, see Using Active Directory as an Identity Provider for SSSD in the *Windows Integration Guide*.

7.3.2. Configuring an LDAP Domain for SSSD

Prerequisites

- Install SSSD.

```
# yum install sssd
```

Configure SSSD to Discover the LDAP Domain

1. Open the **/etc/sssd/sssd.conf** file.
2. Create a **[domain]** section for the LDAP domain:

```
[domain/LDAP_domain_name]
```

3. Specify if you want to use the LDAP server as an identity provider, an authentication provider, or both.
   a. To use the LDAP server as an identity provider, set the ***id_provider*** option to **ldap**.
   b. To use the LDAP server as an authentication provider, set the ***auth_provider*** option to **ldap**.

   For example, to use the LDAP server as both:

```
[domain/LDAP_domain_name]
id_provider = ldap
auth_provider = ldap
```

4. Specify the LDAP server. Choose one of the following:
   a. To explicitly define the server, specify the server's URI with the ***ldap_uri*** option:

```
[domain/LDAP_domain_name]
id_provider = ldap
auth_provider = ldap

ldap_uri = ldap://ldap.example.com
```

   The ***ldap_uri*** option also accepts the IP address of the server. However, using an IP address instead of the server name might cause TLS/SSL connections to fail. See Configuring an SSSD Provider to Use an IP Address in the Certificate Subject Name in Red Hat Knowledgebase.

   b. To configure SSSD to discover the server dynamically using DNS service discovery, see Section 7.4.3, “Configuring DNS Service Discovery”.

   Optionally, specify backup servers in the ***ldap_backup_uri*** option as well.

5. Specify the LDAP server's search base in the ***ldap_search_base*** option:

```
[domain/LDAP_domain_name]
id_provider = ldap
auth_provider = ldap

ldap_uri = ldap://ldap.example.com
ldap_search_base = dc=example,dc=com
```

6. Specify a way to establish a secure connection to the LDAP server. The recommended method is to use a TLS connection. To do this, enable the ***ldap_id_use_start_tls*** option, and use these CA certificate-related options:

   - ***ldap_tls_reqcert*** specifies if the client requests a server certificate and what checks are performed on the certificate
   - ***ldap_tls_cacert*** specifies the file containing the certificate

```
[domain/LDAP_domain_name]
id_provider = ldap
auth_provider = ldap

ldap_uri = ldaps://ldap.example.com
ldap_search_base = dc=example,dc=com

ldap_id_use_start_tls = true
ldap_tls_reqcert = demand
ldap_tls_cacert = /etc/pki/tls/certs/ca-bundle.crt
```

NOTE

SSSD always uses an encrypted channel for authentication, which ensures that passwords are never sent over the network unencrypted. With **ldap_id_use_start_tls = true**, identity lookups (such as commands based on the **id** or **getent** utilities) are also encrypted.

7. Add the new domain to the ***domains*** option in the **[sssd]** section. The option lists the domains that SSSD queries. For example:

```
domains = LDAP_domain_name, domain2
```

Additional Resources

The above procedure shows the basic options for an LDAP provider. For more details, see:

- the sssd.conf(5) man page, which describes global options available for all types of domains
- the sssd-ldap(5) man page, which describes options specific to LDAP

7.3.3. Configuring a Proxy Provider for SSSD

Prerequisites

- Install SSSD.

```
# yum install sssd
```

Configure SSSD to Discover the Proxy Domain

1. Open the **/etc/sssd/sssd.conf** file.
2. Create a **[domain]** section for the proxy provider:

```
[domain/proxy_name]
```

3. To specify an authentication provider:
   a. Set the ***auth_provider*** option to **proxy**.
   b. Use the ***proxy_pam_target*** option to specify a PAM service as the authentication proxy.

   For example:

```
[domain/proxy_name]
auth_provider = proxy
proxy_pam_target = sssdpamproxy
```

IMPORTANT

Ensure that the proxy PAM stack does *not* recursively include **pam_sss.so**.

4. To specify an identity provider:
   a. Set the ***id_provider*** option to **proxy**.
   b. Use the ***proxy_lib_name*** option to specify an NSS library as the identity proxy.

   For example:

```
[domain/proxy_name]
id_provider = proxy
proxy_lib_name = nis
```

5. Add the new domain to the ***domains*** option in the **[sssd]** section. The option lists the domains that SSSD queries. For example:

```
domains = proxy_name, domain2
```

Additional Resources

The above procedure shows the basic options for a proxy provider. For more details, see the sssd.conf(5) man page, which describes global options available for all types of domains and other proxy-related options.

7.3.4. Configuring a Kerberos Authentication Provider

Prerequisites

- Install SSSD.

```
# yum install sssd
```

Configure SSSD to Discover the Kerberos Domain

1. Open the **/etc/sssd/sssd.conf** file.
2. Create a **[domain]** section for the SSSD domain.

   ```
   [domain/Kerberos_domain_name]
   ```

3. Specify an identity provider. For example, for details on configuring an LDAP identity provider, see Section 7.3.2, “Configuring an LDAP Domain for SSSD”.

   If the Kerberos principal names are not available in the specified identity provider, SSSD constructs the principals using the format *username@REALM*.

4. Specify the Kerberos authentication provider details:

   a. Set the ***auth_provider*** option to **krb5**.

   ```
   [domain/Kerberos_domain_name]
   id_provider = ldap
   auth_provider = krb5
   ```

   b. Specify the Kerberos server:

      i. To explicitly define the server, use the ***krb5_server*** option. The options accepts the host name or IP address of the server:

      ```
      [domain/Kerberos_domain_name]
      id_provider = ldap
      auth_provider = krb5

      krb5_server = kdc.example.com
      ```

      ii. To configure SSSD to discover the server dynamically using DNS service discovery, see Section 7.4.3, “Configuring DNS Service Discovery”.

   Optionally, specify backup servers in the ***krb5_backup_server*** option as well.

   c. If the Change Password service is not running on the KDC specified in ***krb5_server*** or ***krb5_backup_server***, use the ***krb5_passwd*** option to specify the server where the service is running.

   ```
   [domain/Kerberos_domain_name]
   id_provider = ldap
   auth_provider = krb5

   krb5_server = kdc.example.com
   krb5_backup_server = kerberos.example.com
   krb5_passwd = kerberos.admin.example.com
   ```

   If ***krb5_passwd*** is not used, SSSD uses the KDC specified in ***krb5_server*** or ***krb5_backup_server***.

d. Use the ***krb5_realm*** option to specify the name of the Kerberos realm.

```
[domain/Kerberos_domain_name]
id_provider = ldap
auth_provider = krb5

krb5_server = kerberos.example.com
krb5_backup_server = kerberos2.example.com
krb5_passwd = kerberos.admin.example.com
krb5_realm = EXAMPLE.COM
```

5. Add the new domain to the ***domains*** option in the **[sssd]** section. The option lists the domains that SSSD queries. For example:

```
domains = Kerberos_domain_name, domain2
```

Additional Resources
The above procedure shows the basic options for a Kerberos provider. For more details, see:

- the sssd.conf(5) man page, which describes global options available for all types of domains
- the sssd-krb5(5) man page, which describes options specific to Kerberos

7.4. ADDITIONAL CONFIGURATION FOR IDENTITY AND AUTHENTICATION PROVIDERS

7.4.1. Adjusting User Name Formats

7.4.1.1. Defining the Regular Expression for Parsing Full User Names

SSSD parses full user name strings into the user name and domain components. By default, SSSD interprets full user names in the format **user_name@domain_name** based on the following regular expression in Python syntax:

```
(?P<name>[^@]+)@?(?P<domain>[^@]*$)
```

NOTE

For Identity Management and Active Directory providers, the default user name format is ***user_name*@*domain_name*** or ***NetBIOS_name**user_name***.

To adjust how SSSD interprets full user names:

1. Open the **/etc/sssd/sssd.conf** file.
2. Use the ***re_expression*** option to define a custom regular expression.
   a. To define the regular expressions globally for all domains, add ***re_expression*** to the **[sssd]** section of **sssd.conf**.

b. To define the regular expressions individually for a particular domain, add ***re_expression*** to the corresponding domain section of **sssd.conf**.

For example, to configure a regular expression for the *LDAP* domain:

```
[domain/LDAP]
[... file truncated ...]
re_expression = (?P<domain>[^\\]*?)\\?(?P<name>[^\\]+$)
```

For details, see the descriptions for ***re_expression*** in the **SPECIAL SECTIONS** and **DOMAIN SECTIONS** parts of the sssd.conf(5) man page.

7.4.1.2. Defining How SSSD Prints Full User Names

If the **use_fully_qualified_names** option is enabled in the **/etc/sssd/sssd.conf** file, SSSD prints full user names in the format **name@domain** based on the following expansion by default:

```
%1$s@%2$s
```

NOTE

If **use_fully_qualified_names** is not set or is explicitly set to **false** for trusted domains, only the user name is printed, without the domain component.

To adjust the format in which SSSD prints full user names:

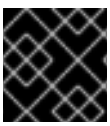
1. Open the **/etc/sssd/sssd.conf** file.
2. Use the ***full_name_format*** option to define the expansion for the full user name format:
   a. To define the expansion globally for all domains, add ***full_name_format*** to the **[sssd]** section of **sssd.conf**.
   b. To define the expansion individually for a particular domain, add ***full_name_format*** to the corresponding domain section of **sssd.conf**.

For details, see the descriptions for ***full_name_format*** in the **SPECIAL SECTIONS** and **DOMAIN SECTIONS** parts of the sssd.conf(5) man page.

In some name configurations, SSSD could strip the domain component of the name, which can cause authentication errors. Because of this, if you set ***full_name_format*** to a non-standard value, a warning will prompt you to change it to a more standard format.

7.4.2. Enabling Offline Authentication

SSSD does not cache user credentials by default. When processing authentication requests, SSSD always contacts the identity provider. If the provider is unavailable, user authentication fails.



IMPORTANT

SSSD never caches passwords in plain text. It stores only a hash of the password.

To ensure that users can authenticate even when the identity provider is unavailable, enable credential caching:

1. Open the **/etc/sssd/sssd.conf** file.
2. In a domain section, add the ***cache_credentials = true*** setting:

```
[domain/domain_name]
cache_credentials = true
```

3. *Optional, but recommended.* Configure a time limit for how long SSSD allows offline authentication if the identity provider is unavailable.
   a. Configure the PAM service to work with SSSD. See Section 7.5.2, “Configuring Services: PAM”.
   b. Use the ***offline_credentials_expiration*** option to specify the time limit. For example, to specify that users are able to authenticate offline for 3 days since the last successful login:

```
[pam]
offline_credentials_expiration = 3
```

For details on ***offline_credentials_expiration***, see the sssd.conf(5) man page.

7.4.3. Configuring DNS Service Discovery

If the identity or authentication server is not explicitly defined in the **/etc/sssd/sssd.conf** file, SSSD can discover the server dynamically using *DNS service discovery* [1].

For example, if **sssd.conf** includes the ***id_provider = ldap*** setting, but the ***ldap_uri*** option does not specify any host name or IP address, SSSD uses DNS service discovery to discover the server dynamically.

NOTE

SSSD cannot dynamically discover backup servers, only the primary server.

Configuring SSSD for DNS Service Discovery

1. Open the **/etc/sssd/sssd.conf** file.
2. Set the primary server value to **_srv_**. For an LDAP provider, the primary server is set using the ***ldap_uri*** option:

```
[domain/domain_name]
id_provider = ldap
ldap_uri = _srv_
```

3. Enable service discovery in the password change provider by setting a service type:

```
[domain/domain_name]
id_provider = ldap
```

```
ldap_uri = _srv_

chpass_provider = ldap
ldap_chpass_dns_service_name = ldap
```

4. *Optional.* By default, the service discovery uses the domain portion of the system host name as the domain name. To use a different DNS domain, specify the domain name in the ***dns_discovery_domain*** option.

5. *Optional.* By default, the service discovery scans for the LDAP service type. To use a different service type, specify the type in the ***ldap_dns_service_name*** option.

6. *Optional.* By default, SSSD attempts to look up an IPv4 address. If the attempt fails, SSSD attempts to look up an IPv6 address. To customize this behavior, use the ***lookup_family_order*** option. See the sssd.conf(5) man page for details.

7. For every service with which you want to use service discovery, add a DNS record to the DNS server:

```
_service._protocol._domain TTL priority weight port host_name
```

7.4.4. Defining Access Control Using the `simple` Access Provider

The **simple** access provider allows or denies access based on a list of user names or groups. It enables you to restrict access to specific machines.

For example, on company laptops, you can use the **simple** access provider to restrict access to only a specific user or a specific group. Other users or groups will not be allowed to log in even if they authenticate successfully against the configured authentication provider.

Configuring `simple` Access Provider Rules

1. Open the **/etc/sssd/sssd.conf** file.

2. Set the ***access_provider*** option to **simple**:

```
[domain/domain_name]
access_provider = simple
```

3. Define the access control rules for users. Choose one of the following:

   a. To allow access to users, use the ***simple_allow_users*** option.

   b. To deny access to users, use the ***simple_deny_users*** option.

IMPORTANT

Allowing access to specific users is considered safer than denying. If you deny access to specific users, you automatically allow access to everyone else.

4. Define the access control rules for groups. Choose one of the following:

a. To allow access to groups, use the ***simple_allow_groups*** option.

b. To deny access to groups, use the ***simple_deny_groups*** option.

IMPORTANT

Allowing access to specific groups is considered safer than denying. If you deny access to specific groups, you automatically allow access to everyone else.

The following example allows access to **user1**, **user2**, and members of **group1**, while denying access to all other users.

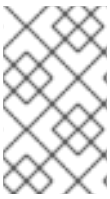
```
[domain/domain_name]
access_provider = simple
simple_allow_users = user1, user2
simple_allow_groups = group1
```

For details, see the sssd-simple(5) man page.

7.4.5. Defining Access Control Using the LDAP Access Filter

When the ***access_provider*** option is set in **/etc/sssd/sssd.conf**, SSSD uses the specified access provider to evaluate which users are granted access to the system. If the access provider you are using is an extension of the LDAP provider type, you can also specify an LDAP access control filter that a user must match in order to be allowed access to the system.

For example, when using an Active Directory (AD) server as the access provider, you can restrict access to the Linux system only to specified AD users. All other users that do not match the specified filter will be denied access.



NOTE

The access filter is applied on the LDAP user entry only. Therefore, using this type of access control on nested groups might not work. To apply access control on nested groups, see Section 7.4.4, “Defining Access Control Using the **simple** Access Provider”.

IMPORTANT

When using offline caching, SSSD checks if the user's most recent online login attempt was successful. Users who logged in successfully during the most recent online login will still be able to log in offline, even if they do not match the access filter.

Configuring SSSD to Apply an LDAP Access Filter

1. Open the **/etc/sssd/sssd.conf** file.
2. In the **[domain]** section, specify the LDAP access control filter.
   - For an LDAP access provider, use the ***ldap_access_filter*** option. See the sssd-ldap(5) man page for details.

- For an AD access provider, use the ***ad_access_filter*** option. See the sssd-ad(5) man page for details.

For example, to allow access only to AD users who belong to the **admins** user group and have a ***unixHomeDirectory*** attribute set:

```
[domain/AD_domain_name]
access provider = ad
[... file truncated ...]
ad_access_filter = (&
(memberOf=cn=admins,ou=groups,dc=example,dc=com)
(unixHomeDirectory=*))
```

SSSD can also check results by the ***authorizedService*** or ***host*** attribute in an entry. In fact, all options — LDAP filter, ***authorizedService***, and ***host*** — can be evaluated, depending on the user entry and the configuration. The **ldap_access_order** parameter lists all access control methods to use, in order of how they should be evaluated.

```
[domain/example.com]
access_provider = ldap
ldap_access_filter = memberOf=cn=allowedusers,ou=Groups,dc=example,dc=com
ldap_access_order = filter, host, authorized_service
```

The attributes in the user entry to use to evaluate authorized services or allowed hosts can be customized. Additional access control parameters are listed in the **sssd-ldap(5)** man page.

7.5. CONFIGURING SYSTEM SERVICES FOR SSSD

SSSD provides interfaces towards several system services. Most notably:

Name Service Switch (NSS)

See Section 7.5.1, “Configuring Services: NSS”.

Pluggable Authentication Modules (PAM)

See Section 7.5.2, “Configuring Services: PAM”.

OpenSSH

See Configuring SSSD to Provide a Cache for the OpenSSH Services in the *Linux Domain Identity, Authentication, and Policy Guide*.

autofs

See Section 7.5.3, “Configuring Services: **autofs**”.

sudo

See Section 7.5.4, “Configuring Services: **sudo**”.

7.5.1. Configuring Services: NSS

How SSSD Works with NSS

The Name Service Switch (NSS) service maps system identities and services with configuration sources: it provides a central configuration store where services can look up sources for various configuration and name resolution mechanisms.

SSSD can use NSS as a provider for several types of NSS maps. Most notably:

- User information (the **passwd** map)
- Groups (the **groups** map)
- Netgroups (the **netgroups** map)
- Services (the **services** map)

Prerequisites

- Install SSSD.

```
# yum install sssd
```

Configure NSS Services to Use SSSD

1. Use the **authconfig** utility to enable SSSD:

```
[root@server ~]# authconfig --enablesssd --update
```

This updates the **/etc/nsswitch.conf** file to enable the following NSS maps to use SSSD:

```
passwd:     files sss
shadow:     files sss
group:      files sss

netgroup:   files sss
```

2. Open **/etc/nsswitch.conf** and add **sss** to the **services** map line:

```
services: file sss
```

Configure SSSD to Work with NSS

1. Open the **/etc/sssd/sssd.conf** file.
2. In the **[sssd]** section, make sure that NSS is listed as one of the services that works with SSSD.

```
[sssd]
[... file truncated ...]
services = nss, pam
```

3. In the **[nss]** section, configure how SSSD interacts with NSS. For example:

```
[nss]
filter_groups = root
filter_users = root
```

```
entry_cache_timeout = 300
entry_cache_nowait_percentage = 75
```

For a complete list of available options, see **NSS configuration options** in the sssd.conf(5) man page.

4. Restart SSSD.

```
# systemctl restart sssd.service
```

Test That the Integration Works Correctly

Display information about a user with these commands:

- **id *user***
- **getent passwd *user***

7.5.2. Configuring Services: PAM

WARNING

A mistake in the PAM configuration file can lock users out of the system completely. Always back up the configuration files before performing any changes, and keep a session open so that you can revert any changes.

Configure PAM to Use SSSD

- Use the **authconfig** utility to enable SSSD:

```
# authconfig --enablesssdauth --update
```

This updates the PAM configuration to reference the SSSD modules, usually in the **/etc/pam.d/system-auth** and **/etc/pam.d/password-auth** files. For example:

```
[... file truncated ...]
auth  required pam_env.so
auth  sufficient pam_unix.so nullok try_first_pass
auth  requisite pam_succeed_if.so uid >= 500 quiet
auth        sufficient pam_sss.so use_first_pass
auth  required pam_deny.so
[... file truncated ...]
```

For details, see the pam.conf(5) or pam(8) man pages.

Configure SSSD to Work with PAM

1. Open the **/etc/sssd/sssd.conf** file.

2. In the **[sssd]** section, make sure that NSS is listed as one of the services that works with SSSD.

```
[sssd]
[... file truncated ...]
services = nss, pam
```

3. In the **[pam]** section, configure how SSSD interacts with PAM. For example:

```
[pam]
offline_credentials_expiration = 2
offline_failed_login_attempts = 3
offline_failed_login_delay = 5
```

   For a complete list of available options, see **PAM configuration options** in the sssd.conf(5) man page.

4. Restart SSSD.

```
# systemctl restart sssd.service
```

Test That the Integration Works Correctly

- Try logging in as a user.
- Use the **sssctl user-checks *user_name* auth** command to check your SSSD configuration. For details, use the **sssctl user-checks --help** command.

7.5.3. Configuring Services: autofs

How SSSD Works with automount

The **automount** utility can mount and unmount NFS file systems automatically (on-demand mounting), which saves system resources. For details on **automount**, see **autofs** in the Storage Administration Guide.

You can configure **automount** to point to SSSD. In this setup:

1. When a user attempts to mount a directory, SSSD contacts LDAP to obtain the required information about the current **automount** configuration.
2. SSSD stores the information required by **automount** in a cache, so that users can mount directories even when the LDAP server is offline.

Configure autofs to Use SSSD

1. Install the autofs package.

```
# yum install autofs
```

2. Open the **/etc/nsswitch.conf** file.
3. On the **automount** line, change the location where to look for the **automount** map information from **ldap** to **sss**:

```
automount: files sss
```

Configure SSSD to Work with `autofs`

1. Open the **`/etc/sssd/sssd.conf`** file.
2. In the **`[sssd]`** section, add **`autofs`** to the list of services that SSSD manages.

```
[sssd]
services = nss,pam,autofs
```

3. Create a new **`[autofs]`** section. You can leave it empty.

```
[autofs]
```

   For a list of available options, see **`AUTOFS configuration options`** in the sssd.conf(5) man page.

4. Make sure an LDAP domain is available in **`sssd.conf`**, so that SSSD can read the **`automount`** information from LDAP. See Section 7.3.2, “Configuring an LDAP Domain for SSSD”.

   The **`[domain]`** section of **`sssd.conf`** accepts several **`autofs`**-related options. For example:

```
[domain/LDAP]
[... file truncated ...]
autofs_provider=ldap
ldap_autofs_search_base=cn=automount,dc=example,dc=com
ldap_autofs_map_object_class=automountMap
ldap_autofs_entry_object_class=automount
ldap_autofs_map_name=automountMapName
ldap_autofs_entry_key=automountKey
ldap_autofs_entry_value=automountInformation
```

   For a complete list of available options, see **`DOMAIN SECTIONS`** in the sssd.conf(5) man page.

   If you do not provide additional **`autofs`** options, the configuration depends on the identity provider settings.

5. Restart SSSD.

```
# systemctl restart sssd.service
```

Test the Configuration

- Use the **`automount -m`** command to print the maps from SSSD.

7.5.4. Configuring Services: `sudo`

How SSSD Works with `sudo`

The **sudo** utility gives administrative access to specified users. For more information about **sudo**, see The **sudo** Command in the *System Administrator's Guide*.

You can configure **sudo** to point to SSSD. In this setup:

1. When a user attempts a **sudo** operation, SSSD contacts LDAP to obtain the required information about the current **sudo** configuration.

2. SSSD stores the **sudo** information in a cache, so that users can perform **sudo** operations even when the LDAP server is offline.

SSSD only caches **sudo** rules which apply to the local system, depending on the value of the ***sudoHost*** attribute. See the sssd-sudo(5) man page for details.

Configure sudo to Use SSSD

1. Open the **/etc/nsswitch.conf** file.

2. Add SSSD to the list on the **sudoers** line.

```
sudoers: files sss
```

Configure SSSD to Work with sudo

1. Open the **/etc/sssd/sssd.conf** file.

2. In the **[sssd]** section, add **sudo** to the list of services that SSSD manages.

```
[sssd]
services = nss,pam,sudo
```

3. Create a new **[sudo]** section. You can leave it empty.

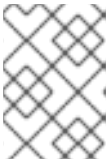
```
[sudo]
```

For a list of available options, see **SUDO configuration options** in the sssd.conf(5) man page.

4. Make sure an LDAP domain is available in **sssd.conf**, so that SSSD can read the **sudo** information from LDAP. See Section 7.3.2, “Configuring an LDAP Domain for SSSD”.

The **[domain]** section for the LDAP domain must include these **sudo**-related parameters:

```
[domain/LDAP]
[... file truncated ...]
sudo_provider = ldap
ldap_sudo_search_base = ou=sudoers,dc=example,dc=com
```



NOTE

Setting Identity Management as the ID provider automatically enables the **sudo** provider. In this situation, it is not necessary to specify ***sudo_provider = ipa***.

For a complete list of available options, see **DOMAIN SECTIONS** in the sssd.conf(5) man page.

For options available for a **sudo** provider, see the sssd-ldap(5) man page.

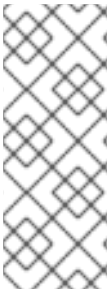
5. Restart SSSD.

```
# systemctl restart sssd.service
```

7.6. SSSD CLIENT-SIDE VIEWS

SSSD enables you to create a client-side view to specify new values for POSIX user or group attributes. The view takes effect only on the local machine where the overrides are configured. You can configure client-side overrides for all **id_provider** values, except **ipa**. If you are using the **ipa** provider, define ID views centrally in IdM. See the corresponding section in the Linux Domain Identity, Authentication, and Policy Guide.

For information about a potential negative impact on the SSSD performance, see the corresponding section in the Linux Domain Identity, Authentication, and Policy Guide.



NOTE

After creating the first override using the **sss_override user-add**, **sss_override group-add**, or **sss_override user-import** command, restart SSSD for the changes to take effect:

```
# systemctl restart sssd
```

7.6.1. Defining a Different Attribute Value for a User Account

As an administrator, you configured an existing host to use accounts from LDAP. However, a user's new ID in LDAP is different from the user's previous ID on the local system. You can configure a client-side view to override the UID instead of changing the permissions on existing files.

To override the UID of the *user* account with UID *6666*:

1. *Optional*. Display the current UID of the *user* account:

```
# id user
uid=1241400014(user_name) gid=1241400014(user_name)
Groups=1241400014(user_name)
```

2. Override the account's UID with *6666*:

```
# sss_override user-add user -u 6666
```

3. Wait until the in-memory cache has been expired. To expire it manually:

```
# sss_cache --users
```

4. Verify that the new UID is applied:

```
# id user
uid=6666(user_name) gid=1241400014(user_name)
Groups=1241400014(user_name)
```

5. *Optional*. Display the overrides for the user:

```
# sss_override user-show user
user@ldap.example.com::6666:::::
```

For a list of attributes you can override, list the command-line options by adding **`--help`** to the command:

```
# sss_override user-add --help
```

7.6.2. Listing All Overrides on a Host

As an administrator, you want to list all user and group overrides on a host to verify that the correct attributes are overridden.

To list all user overrides:

```
# sss_override user-find
user1@ldap.example.com::8000::::/bin/zsh:
user2@ldap.example.com::8001::::/bin/bash:
...
```

To list all group overrides:

```
# sss_override group-find
group1@ldap.example.com::7000
group2@ldap.example.com::7001
...
```

7.6.3. Removing a Local Override

You previously created an override for the shell of the *user* account, that is defined in the global LDAP directory. To remove the override for the account, run:

```
# sss_override user-del user
```

The changes take effect immediately.

To remove an override for a group, run:

```
# sss_override group-del group
```

NOTE

When you remove overrides for a user or group, all overrides for this object are removed.

7.6.4. Exporting and Importing Local Views

Client-side views are stored in the local SSSD cache. You can export user and group views from the cache to a file to create a backup. For example, when you remove the SSSD cache, you can restore the views later again.

To back up user and group views:

```
# sss_override user-export /var/lib/sss/backup/sssd_user_overrides.bak
# sss_override group-export /var/lib/sss/backup/sssd_group_overrides.bak
```

To restore user and group view:

```
# sss_override user-import /var/lib/sss/backup/sssd_user_overrides.bak
# sss_override group-import /var/lib/sss/backup/sssd_group_overrides.bak
```

7.7. DOWNGRADING SSSD

When downgrading — either downgrading the version of SSSD or downgrading the operating system itself — then the existing SSSD cache needs to be removed. If the cache is not removed, then SSSD process is dead but a PID file remains. The SSSD logs show that it cannot connect to any of its associated domains because the cache version is unrecognized.

```
(Wed Nov 28 21:25:50 2012) [sssd] [sysdb_domain_init_internal] (0x0010):
Unknown DB version [0.14], expected [0.10] for domain AD!
```

Users are then no longer recognized and are unable to authenticate to domain services and hosts.

After downgrading the SSSD version:

1. Delete the existing cache database files.

   ```
   [root@server ~]# rm -rf /var/lib/sss/db/*
   ```

2. Restart the SSSD process.

   ```
   [root@server ~]# systemctl restart sssd.service
   ```

7.8. USING NSCD WITH SSSD

SSSD is not designed to be used with the NSCD daemon. Even though SSSD does not directly conflict with NSCD, using both services can result in unexpected behavior, especially with how long entries are cached.

The most common evidence of a problem is conflicts with NFS. When using Network Manager to manage network connections, it may take several minutes for the network interface to come up. During this time, various services attempt to start. If these services start before the network is up and the DNS servers are available, these services fail to identify the forward or reverse DNS entries they need. These services will read an incorrect or possibly empty **`resolv.conf`** file. This file is typically only read once, and so any changes made to this file are not automatically applied. This can cause NFS locking to fail on the machine where the NSCD service is running, unless that service is manually restarted.

To avoid this problem, enable caching for hosts and services in the **`/etc/nscd.conf`** file and rely on the SSSD cache for the **`passwd`**, **`group`**, **`services`**, and **`netgroup`** entries.

Change the **`/etc/nscd.conf`** file:

```
enable-cache hosts yes
```

```
enable-cache passwd no
enable-cache group no
enable-cache netgroup no
enable-cache services no
```

With NSCD answering hosts requests, these entries will be cached by NSCD and returned by NSCD during the boot process. All other entries are handled by SSSD.

7.9. ADDITIONAL RESOURCES

- A complete list of SSSD-related man pages is available in the **SEE ALSO** section in the sssd(8) man page.
- Troubleshooting advice: Section A.1, “Troubleshooting SSSD”.
- A procedure for configuring SSSD to process password expiration warnings sent by the server and display them to users on the local system: Setting Password Expiry in Red Hat Knowledgebase
- An SSSD client can automatically create a GID for every user retrieved from an LDAP server, and at the same time ensure that the GID matches the user's UID unless the GID number is already taken. To see how automatic creation of GIDs can be set up on an SSSD client which is directly integrated into Active Directory, see the corresponding section in the Windows Integration Guide.

[1] DNS service discovery enables applications to check the SRV records in a given domain for certain services of a certain type, and then returns any servers that match the required type. DNS service discovery is defined in RFC 2782.

CHAPTER 8. USING REALMD TO CONNECT TO AN IDENTITY DOMAIN

The **realmd** system provides a clear and simple way to discover and join identity domains. It does not connect to the domain itself but configures underlying Linux system services, such as SSSD or Winbind, to connect to the domain.

The Windows Integration Guide describes using **realmd** to connect to a Microsoft Active Directory (AD) domain. The same procedures apply to using **realmd** to connect to non-AD identity domains. See the corresponding chapter in the Windows Integration Guide.

CHAPTER 9. LDAP SERVERS

LDAP (Lightweight Directory Access Protocol) is a set of open protocols used to access centrally stored information over a network. It is based on the **X.500** standard for directory sharing, but is less complex and resource-intensive. For this reason, LDAP is sometimes referred to as “X.500 Lite”.

Like X.500, LDAP organizes information in a hierarchical manner using directories. These directories can store a variety of information such as names, addresses, or phone numbers, and can even be used in a manner similar to the *Network Information Service* (NIS), enabling anyone to access their account from any machine on the LDAP enabled network.

LDAP is commonly used for centrally managed users and groups, user authentication, or system configuration. It can also serve as a virtual phone directory, allowing users to easily access contact information for other users. Additionally, it can refer a user to other LDAP servers throughout the world, and thus provide an ad-hoc global repository of information. However, it is most frequently used within individual organizations such as universities, government departments, and private companies.

9.1. RED HAT DIRECTORY SERVER

Red Hat Directory Server is an LDAP-compliant server that centralizes user identity and application information. It provides an operating system-independent and network-based registry for storing application settings, user profiles, group data, policies, and access control information.

NOTE

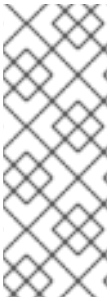
You require a current Red Hat Directory Server subscription to install and update Directory Server.

For further details about setting up and using Directory Server, see:

- *Red Hat Directory Server Installation Guide*
- *Red Hat Directory Server Deployment Guide*
- *Red Hat Directory Server Administration Guide*
- *Red Hat Directory Server Configuration, Command, and File Reference*
- *Red Hat Directory Server Performance Tuning Guide*
- *Red Hat Directory Server Plug-in Guide*

9.2. OPENLDAP

This section covers the installation and configuration of **OpenLDAP 2.4**, an open source implementation of the LDAPv2 and LDAPv3 protocols.

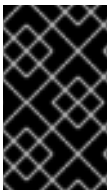


NOTE

Starting with Red Hat Enterprise Linux 7.4, the openldap-server package has been deprecated and will not be included in a future major release of Red Hat Enterprise Linux. For this reason, migrate to Identity Management included in Red Hat Enterprise Linux or to Red Hat Directory Server. For further details about Identity Management, see *Linux Domain Identity, Authentication, and Policy Guide*. For further details about Directory Server, see Section 9.1, "Red Hat Directory Server".

9.2.1. Introduction to LDAP

Using a client-server architecture, LDAP provides a reliable means to create a central information directory accessible from the network. When a client attempts to modify information within this directory, the server verifies the user has permission to make the change, and then adds or updates the entry as requested. To ensure the communication is secure, the *Transport Layer Security* (TLS) cryptographic protocol can be used to prevent an attacker from intercepting the transmission.



IMPORTANT

The OpenLDAP suite in Red Hat Enterprise Linux 7.5 and later no longer uses Mozilla implementation of *Network Security Services* (NSS). Instead, it uses the *OpenSSL*. OpenLDAP continues to work with existing NSS database configuration.

IMPORTANT

Due to the vulnerability described in Resolution for POODLE SSLv3.0 vulnerability (CVE-2014-3566) for components that do not allow SSLv3 to be disabled via configuration settings, Red Hat recommends that you do not rely on the **SSLv3** protocol for security. OpenLDAP is one of the system components that do not provide configuration parameters that allow **SSLv3** to be effectively disabled. To mitigate the risk, it is recommended that you use the `stunnel` command to provide a secure tunnel, and disable **stunnel** from using **SSLv3**. For more information on using **stunnel**, see the Red Hat Enterprise Linux 7 Security Guide.

The LDAP server supports several database systems, which gives administrators the flexibility to choose the best suited solution for the type of information they are planning to serve. Because of a well-defined client *Application Programming Interface* (API), the number of applications able to communicate with an LDAP server is numerous, and increasing in both quantity and quality.

9.2.1.1. LDAP Terminology

The following is a list of LDAP-specific terms that are used within this chapter:

entry

A single unit within an LDAP directory. Each entry is identified by its unique *Distinguished Name* (DN).

attribute

Information directly associated with an entry. For example, if an organization is represented as an LDAP entry, attributes associated with this organization might include an address, a fax number, and so on. Similarly, people can be represented as entries with common attributes such as personal telephone number or email address.

An attribute can either have a single value, or an unordered space-separated list of values. While certain attributes are optional, others are required. Required attributes are specified using the **`objectClass`** definition, and can be found in schema files located in the **`/etc/openldap/slapd.d/cn=config/cn=schema/`** directory.

The assertion of an attribute and its corresponding value is also referred to as a *Relative Distinguished Name* (RDN). Unlike distinguished names that are unique globally, a relative distinguished name is only unique per entry.

LDIF

The *LDAP Data Interchange Format* (LDIF) is a plain text representation of an LDAP entry. It takes the following form:

```
[id] dn: distinguished_name
attribute_type: attribute_value…
attribute_type: attribute_value…
…
```

The optional *id* is a number determined by the application that is used to edit the entry. Each entry can contain as many *attribute_type* and *attribute_value* pairs as needed, as long as they are all defined in a corresponding schema file. A blank line indicates the end of an entry.

9.2.1.2. OpenLDAP Features

OpenLDAP suite provides a number of important features:

- *LDAPv3 Support* — Many of the changes in the protocol since LDAP version 2 are designed to make LDAP more secure. Among other improvements, this includes the support for Simple Authentication and Security Layer (SASL), Transport Layer Security (TLS), and Secure Sockets Layer (SSL) protocols.
- *LDAP Over IPC* — The use of inter-process communication (IPC) enhances security by eliminating the need to communicate over a network.
- *IPv6 Support* — OpenLDAP is compliant with Internet Protocol version 6 (IPv6), the next generation of the Internet Protocol.
- *LDIFv1 Support* — OpenLDAP is fully compliant with LDIF version 1.
- *Updated C API* — The current C API improves the way programmers can connect to and use LDAP directory servers.
- *Enhanced Standalone LDAP Server* — This includes an updated access control system, thread pooling, better tools, and much more.

9.2.1.3. OpenLDAP Server Setup

The typical steps to set up an LDAP server on Red Hat Enterprise Linux are as follows:

1. Install the OpenLDAP suite. See Section 9.2.2, “Installing the OpenLDAP Suite” for more information on required packages.
2. Customize the configuration as described in Section 9.2.3, “Configuring an OpenLDAP Server”.

3. Start the **slapd** service as described in Section 9.2.5, “Running an OpenLDAP Server”.
4. Use the **ldapadd** utility to add entries to the LDAP directory.
5. Use the **ldapsearch** utility to verify that the **slapd** service is accessing the information correctly.

9.2.2. Installing the OpenLDAP Suite

The suite of OpenLDAP libraries and tools is provided by the following packages:

Table 9.1. List of OpenLDAP packages

Package	Description
openldap	A package containing the libraries necessary to run the OpenLDAP server and client applications.
openldap-clients	A package containing the command line utilities for viewing and modifying directories on an LDAP server.
openldap-servers	A package containing both the services and utilities to configure and run an LDAP server. This includes the *Standalone LDAP Daemon*, **slapd**.
compat-openldap	A package containing the OpenLDAP compatibility libraries.

Additionally, the following packages are commonly used along with the LDAP server:

Table 9.2. List of commonly installed additional LDAP packages

Package	Description
nss-pam-ldapd	A package containing **nslcd**, a local LDAP name service that allows a user to perform local LDAP queries.
mod_ldap	A package containing the **mod_authnz_ldap** and **mod_ldap** modules. The **mod_authnz_ldap** module is the LDAP authorization module for the Apache HTTP Server. This module can authenticate users' credentials against an LDAP directory, and can enforce access control based on the user name, full DN, group membership, an arbitrary attribute, or a complete filter string. The **mod_ldap** module contained in the same package provides a configurable shared memory cache, to avoid repeated directory access across many HTTP requests, and also support for SSL/TLS. Note that this package is provided by the Optional channel. See Adding the Optional and Supplementary Repositories in the *System Administrator's Guide* for more information on Red Hat additional channels.

To install these packages, use the **yum** command in the following form:

```
yum install package…
```

For example, to perform the basic LDAP server installation, type the following at a shell prompt:

```
~]# yum install openldap openldap-clients openldap-servers
```

Note that you must have superuser privileges (that is, you must be logged in as **root**) to run this command. For more information on how to install new packages in Red Hat Enterprise Linux, see Installing Packages in the *System Administrator's Guide*.

9.2.2.1. Overview of OpenLDAP Server Utilities

To perform administrative tasks, the openldap-servers package installs the following utilities along with the **slapd** service:

Table 9.3. List of OpenLDAP server utilities

Command	Description
slapacl	Allows you to check the access to a list of attributes.
slapadd	Allows you to add entries from an LDIF file to an LDAP directory.
slapauth	Allows you to check a list of IDs for authentication and authorization permissions.
slapcat	Allows you to pull entries from an LDAP directory in the default format and save them in an LDIF file.
slapdn	Allows you to check a list of Distinguished Names (DNs) based on available schema syntax.
slapindex	Allows you to re-index the **slapd** directory based on the current content. Run this utility whenever you change indexing options in the configuration file.
slappasswd	Allows you to create an encrypted user password to be used with the **ldapmodify** utility, or in the **slapd** configuration file.
slapschema	Allows you to check the compliance of a database with the corresponding schema.
slaptest	Allows you to check the LDAP server configuration.

For a detailed description of these utilities and their usage, see the corresponding manual pages as referred to in the section called “Installed Documentation”.

IMPORTANT

Although only **root** can run **slapadd**, the **slapd** service runs as the **ldap** user. Because of this, the directory server is unable to modify any files created by **slapadd**. To correct this issue, after running the **slapdadd** utility, type the following at a shell prompt:

```
~]# chown -R ldap:ldap /var/lib/ldap
```

WARNING

To preserve the data integrity, stop the **slapd** service before using **slapadd**, **slapcat**, or **slapindex**. You can do so by typing the following at a shell prompt:

```
~]# systemctl stop slapd.service
```

For more information on how to start, stop, restart, and check the current status of the **slapd** service, see Section 9.2.5, “Running an OpenLDAP Server”.

9.2.2.2. Overview of OpenLDAP Client Utilities

The openldap-clients package installs the following utilities which can be used to add, modify, and delete entries in an LDAP directory:

Table 9.4. List of OpenLDAP client utilities

Command	Description
ldapadd	Allows you to add entries to an LDAP directory, either from a file, or from standard input. It is a symbolic link to **ldapmodify -a**.
ldapcompare	Allows you to compare given attribute with an LDAP directory entry.
ldapdelete	Allows you to delete entries from an LDAP directory.
ldapexop	Allows you to perform extended LDAP operations.
ldapmodify	Allows you to modify entries in an LDAP directory, either from a file, or from standard input.
ldapmodrdn	Allows you to modify the RDN value of an LDAP directory entry.
ldappasswd	Allows you to set or change the password for an LDAP user.
ldapsearch	Allows you to search LDAP directory entries.

Command	Description
ldapurl	Allows you to compose or decompose LDAP URLs.
ldapwhoami	Allows you to perform a **whoami** operation on an LDAP server.

With the exception of **ldapsearch**, each of these utilities is more easily used by referencing a file containing the changes to be made rather than typing a command for each entry to be changed within an LDAP directory. The format of such a file is outlined in the man page for each utility.

9.2.2.3. Overview of Common LDAP Client Applications

Although there are various graphical LDAP clients capable of creating and modifying directories on the server, none of them is included in Red Hat Enterprise Linux. Popular applications that can access directories in a read-only mode include **Mozilla Thunderbird**, **Evolution**, or **Ekiga**.

9.2.3. Configuring an OpenLDAP Server

By default, the OpenLDAP configuration is stored in the **/etc/openldap/** directory. The following table highlights the most important directories and files within this directory:

Table 9.5. List of OpenLDAP configuration files and directories

Path	Description
/etc/openldap/ldap.conf	The configuration file for client applications that use the OpenLDAP libraries. This includes **ldapadd**, **ldapsearch**, **Evolution**, and so on.
/etc/openldap/slapd.d/	The directory containing the **slapd** configuration.

Note that OpenLDAP no longer reads its configuration from the **/etc/openldap/slapd.conf** file. Instead, it uses a configuration database located in the **/etc/openldap/slapd.d/** directory. If you have an existing **slapd.conf** file from a previous installation, you can convert it to the new format by running the following command:

```
~]# slaptest -f /etc/openldap/slapd.conf -F /etc/openldap/slapd.d/
```

The **slapd** configuration consists of LDIF entries organized in a hierarchical directory structure, and the recommended way to edit these entries is to use the server utilities described in Section 9.2.2.1, “Overview of OpenLDAP Server Utilities”.

IMPORTANT

An error in an LDIF file can render the **slapd** service unable to start. Because of this, it is strongly advised that you avoid editing the LDIF files within the **/etc/openldap/slapd.d/** directly.

9.2.3.1. Changing the Global Configuration

Global configuration options for the LDAP server are stored in the **/etc/openldap/slapd.d/cn=config.ldif** file. The following directives are commonly used:

olcAllows

The **olcAllows** directive allows you to specify which features to enable. It takes the following form:

```
olcAllows: feature…
```

It accepts a space-separated list of features as described in Table 9.6, "Available **olcAllows** options". The default option is **bind_v2**.

Table 9.6. Available olcAllows options

Option	Description
bind_v2	Enables the acceptance of LDAP version 2 bind requests.
bind_anon_cred	Enables an anonymous bind when the Distinguished Name (DN) is empty.
bind_anon_dn	Enables an anonymous bind when the Distinguished Name (DN) is *not* empty.
update_anon	Enables processing of anonymous update operations.
proxy_authz_anon	Enables processing of anonymous proxy authorization control.

Example 9.1. Using the olcAllows directive

```
olcAllows: bind_v2 update_anon
```

olcConnMaxPending

The **olcConnMaxPending** directive allows you to specify the maximum number of pending requests for an anonymous session. It takes the following form:

```
olcConnMaxPending: number
```

The default option is **100**.

Example 9.2. Using the olcConnMaxPending directive

```
olcConnMaxPending: 100
```

olcConnMaxPendingAuth

The **olcConnMaxPendingAuth** directive allows you to specify the maximum number of pending requests for an authenticated session. It takes the following form:

```
olcConnMaxPendingAuth: number
```

The default option is **1000**.

Example 9.3. Using the olcConnMaxPendingAuth directive

```
olcConnMaxPendingAuth: 1000
```

olcDisallows

The **olcDisallows** directive allows you to specify which features to disable. It takes the following form:

```
olcDisallows: feature…
```

It accepts a space-separated list of features as described in Table 9.7, "Available **olcDisallows** options". No features are disabled by default.

Table 9.7. Available olcDisallows options

Option	Description
bind_anon	Disables the acceptance of anonymous bind requests.
bind_simple	Disables the simple bind authentication mechanism.
tls_2_anon	Disables the enforcing of an anonymous session when the STARTTLS command is received.
tls_authc	Disallows the STARTTLS command when authenticated.

Example 9.4. Using the olcDisallows directive

```
olcDisallows: bind_anon
```

olcIdleTimeout

The **olcIdleTimeout** directive allows you to specify how many seconds to wait before closing an idle connection. It takes the following form:

```
olcIdleTimeout: number
```

This option is disabled by default (that is, set to **0**).

Example 9.5. Using the olcIdleTimeout directive

```
olcIdleTimeout: 180
```

olcLogFile

The **olcLogFile** directive allows you to specify a file in which to write log messages. It takes the following form:

```
olcLogFile: file_name
```

The log messages are written to standard error by default.

Example 9.6. Using the olcLogFile directive

```
olcLogFile: /var/log/slapd.log
```

olcReferral

The **olcReferral** option allows you to specify a URL of a server to process the request in case the server is not able to handle it. It takes the following form:

```
olcReferral: URL
```

This option is disabled by default.

Example 9.7. Using the olcReferral directive

```
olcReferral: ldap://root.openldap.org
```

olcWriteTimeout

The **olcWriteTimeout** option allows you to specify how many seconds to wait before closing a connection with an outstanding write request. It takes the following form:

```
olcWriteTimeout
```

This option is disabled by default (that is, set to **0**).

Example 9.8. Using the olcWriteTimeout directive

```
olcWriteTimeout: 180
```

9.2.3.2. The Front End Configuration

The OpenLDAP front end configuration is stored in the **etc/openldap/slapd.d/cn=config/olcDatabase={-1}frontend.ldif** file and defines global database options, such as access control lists (ACL). For details, see the Global Database Options section in the slapd-config(5) man page.

9.2.3.3. The Monitor Back End

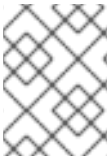
The **/etc/openldap/slapd.d/cn=config/olcDatabase={1}monitor.ldif** file controls the OpenLDAP monitor back end. If enabled, it is automatically generated and dynamically updated by OpenLDAP with information about the running status of the daemon. The suffix is **cn=Monitor** and cannot be changed. For further details, see the slapd-monitor(5) man page.

9.2.3.4. Database-Specific Configuration

By default, the OpenLDAP server uses the **hdb** database back end. Besides that it uses a hierarchical database layout which supports subtree renames, it is identical to the **bdb** back end and uses the same configuration options. The configuration for this database back end is stored in the **/etc/openldap/slapd.d/cn=config/olcDatabase={2}hdb.ldif** file.

For a list of other back end databases, see the slapd.backends(5) man page. Database-specific settings you find in the man page for the individual back ends. For example:

```
# man slapd-hdb
```



NOTE

The **bdb** and **hdb** back ends are deprecated. Consider using the **mdb** back end for new installations instead.

The following directives are commonly used in a database-specific configuration:

olcReadOnly

The **olcReadOnly** directive allows you to use the database in a read-only mode. It takes the following form:

```
olcReadOnly: boolean
```

It accepts either **TRUE** (enable the read-only mode), or **FALSE** (enable modifications of the database). The default option is **FALSE**.

Example 9.9. Using the olcReadOnly directive

```
olcReadOnly: TRUE
```

olcRootDN

The **olcRootDN** directive allows you to specify the user that is unrestricted by access controls or administrative limit parameters set for operations on the LDAP directory. It takes the following form:

```
olcRootDN: distinguished_name
```

It accepts a *Distinguished Name* (DN). The default option is **cn=Manager,dn=my-domain,dc=com**.

Example 9.10. Using the olcRootDN directive

```
olcRootDN: cn=root,dn=example,dn=com
```

-

olcRootPW

The **olcRootPW** directive allows you to set a password for the user that is specified using the **olcRootDN** directive. It takes the following form:

```
olcRootPW: password
```

It accepts either a plain text string, or a hash. To generate a hash, type the following at a shell prompt:

```
~]$ slappaswd
New password:
Re-enter new password:
{SSHA}WczWsyPEnMchFf1GRTweq2q7XJcvmSxD
```

Example 9.11. Using the olcRootPW directive

```
olcRootPW: {SSHA}WczWsyPEnMchFf1GRTweq2q7XJcvmSxD
```

olcSuffix

The **olcSuffix** directive allows you to specify the domain for which to provide information. It takes the following form:

```
olcSuffix: domain_name
```

It accepts a *fully qualified domain name* (FQDN). The default option is **dc=my-domain,dc=com**.

Example 9.12. Using the olcSuffix directive

```
olcSuffix: dc=example,dc=com
```

9.2.3.5. Extending Schema

Since OpenLDAP 2.3, the **/etc/openldap/slapd.d/** directory also contains LDAP definitions that were previously located in **/etc/openldap/schema/**. It is possible to extend the schema used by OpenLDAP to support additional attribute types and object classes using the default schema files as a guide. However, this task is beyond the scope of this chapter. For more information on this topic, see http://www.openldap.org/doc/admin/schema.html.

9.2.3.6. Establishing a Secure Connection

The OpenLDAP suite and servers can be secured using the Transport Layer Security (TLS) framework. TLS is a cryptographic protocol designed to provide communication security over the network. OpenLDAP suite in Red Hat Enterprise Linux 7 uses OpenSSL as the TLS implementation.

To establish a secure connection using TLS, obtain the required certificates. Then, a number of options must be configured on both the client and the server. At minimum, a server must be configured with the

Certificate Authority (CA) certificates and also its own server certificate and private key. The clients must be configured with the name of the file containing all the trusted CA certificates.

Typically, a server only needs to sign a single CA certificate. A client may want to connect to a variety of secure servers, therefore it is common to specify a list of several trusted CAs in its configuration.

Server Configuration
This section lists global configuration directives for **slapd** that need to be specified in the **/etc/openldap/slapd.d/cn=config.ldif** file on an OpenLDAP server in order to establish TLS.

While the old style configuration uses a single file, normally installed as **/usr/local/etc/openldap/slapd.conf**, the new style uses a **slapd** back end database to store the configuration. The configuration database normally resides in the **/usr/local/etc/openldap/slapd.d/** directory.

The following directives are also valid for establishing SSL. In addition to TLS directives, you need to enable a port dedicated to SSL on the server side – typically it is port 636. To do so, edit the **/etc/sysconfig/slapd** file and append the **ldaps:///** string to the list of URLs specified with the **SLAPD_URLS** directive.

olcTLSCACertificateFile

The **olcTLSCACertificateFile** directive specifies the file encoded with privacy-enhanced mail (PEM) schema that contains trusted CA certificates. The directive takes the following form:

```
olcTLSCACertificateFile: path
```

Replace *path* with the path to the CA certificate file.

olcTLSCACertificatePath

The **olcTLSCACertificatePath** directive specifies the path to a directory containing individual CA certificates in separate files. This directory must be specially managed with the OpenSSL **c_rehash** utility that generates symbolic links with the hashed names that point to the actual certificate files. In general, it is simpler to use the **olcTLSCACertificateFile** directive instead.

The directive takes the following form:

```
olcTLSCACertificatePath: path
```

Replace *path* with a path to the directory containing the CA certificate files. The specified directory must be managed with the OpenSSL **c_rehash** utility.

olcTLSCertificateFile

The **olcTLSCertificateFile** directive specifies the file that contains the **slapd** server certificate. The directive takes the following form:

```
olcTLSCertificateFile: path
```

Replace *path* with a path to the server certificate file of the **slapd** service.

olcTLSCertificateKeyFile

The **olcTLSCertificateKeyFile** directive specifies the file that contains the private key that matches the certificate stored in the file specified with **olcTLSCertificateFile**. Note that the

current implementation does not support encrypted private keys, and therefore the containing file must be sufficiently protected. The directive takes the following form:

```
olcTLSCertificateKeyFile: path
```

Replace *path* with a path to the private key file.

Client Configuration

Specify the following directives in the **/etc/openldap/ldap.conf** configuration file on the client system. Most of these directives are parallel to the server configuration options. Directives in **/etc/openldap/ldap.conf** are configured on a system-wide basis, however, individual users may override them in their **~/.ldaprc** files.

The same directives can be used to establish an SSL connection. The **ldaps://** string must be used instead of **ldap://** in OpenLDAP commands such as **ldapsearch**. This forces commands to use the default port for SSL, port 636, configured on the server.

TLS_CACERT

The **TLS_CACERT** directive specifies a file containing certificates for all of the Certificate Authorities the client will recognize. This is equivalent to the **olcTLSCACertificateFile** directive on a server. **TLS_CACERT** should always be specified before **TLS_CACERTDIR** in **/etc/openldap/ldap.conf**. The directive takes the following form:

```
TLS_CACERT path
```

Replace *path* with a path to the CA certificate file.

TLS_CACERTDIR

The **TLS_CACERTDIR** directive specifies the path to a directory that contains Certificate Authority certificates in separate files. As with **olcTLSCACertificatePath** on a server, the specified directory must be managed with the OpenSSL **c_rehash** utility.

```
TLS_CACERTDIR directory
```

Replace *directory* with a path to the directory containing CA certificate files.

TLS_CERT

The **TLS_CERT** specifies the file that contains a client certificate. This directive can only be specified in a user's **~/.ldaprc** file. The directive takes the following form:

```
TLS_CERT path
```

Replace *path* with a path to the client certificate file.

TLS_KEY

The **TLS_KEY** specifies the file that contains the private key that matches the certificate stored in the file specified with the **TLS_CERT** directive. As with **olcTLSCertificateFile** on a server, encrypted key files are not supported, so the file itself must be carefully protected. This option is only configurable in a user's **~/.ldaprc** file.

The **TLS_KEY** directive takes the following form:

```
TLS_KEY path
```

Replace *path* with a path to the client certificate file.

9.2.3.7. Setting Up Replication

Replication is the process of copying updates from one LDAP server (*provider*) to one or more other servers or clients (*consumers*). A provider replicates directory updates to consumers, the received updates can be further propagated by the consumer to other servers, so a consumer can also act simultaneously as a provider. Also, a consumer does not have to be an LDAP server, it may be just an LDAP client. In OpenLDAP, you can use several replication modes, most notable are *mirror* and *sync*. For more information on OpenLDAP replication modes, see the *OpenLDAP Software Administrator's Guide* installed with openldap-servers package (see the section called “Installed Documentation”).

To enable a chosen replication mode, use one of the following directives in **/etc/openldap/slapd.d/** on both provider and consumers.

olcMirrorMode

The **olcMirrorMode** directive enables the mirror replication mode. It takes the following form:

```
olcMirrorMode on
```

This option needs to be specified both on provider and consumers. Also a **serverID** must be specified along with **syncrepl** options. Find a detailed example in the *18.3.4. MirrorMode* section of the *OpenLDAP Software Administrator's Guide* (see the section called “Installed Documentation”).

olcSyncrepl

The **olcSyncrepl** directive enables the sync replication mode. It takes the following form:

```
olcSyncrepl on
```

The sync replication mode requires a specific configuration on both the provider and the consumers. This configuration is thoroughly described in the *18.3.1. Syncrepl* section of the *OpenLDAP Software Administrator's Guide* (see the section called “Installed Documentation”).

9.2.3.8. Loading Modules and Back ends

You can enhance the **slapd** service with dynamically loaded modules. Support for these modules must be enabled with the **--enable-modules** option when configuring **slapd**. Modules are stored in files with the *.la* extension:

```
module_name.la
```

Back ends store or retrieve data in response to LDAP requests. Back ends may be compiled statically into **slapd**, or when module support is enabled, they may be dynamically loaded. In the latter case, the following naming convention is applied:

```
back_backend_name.la
```

To load a module or a back end, use the following directive in **/etc/openldap/slapd.d/**:

olcModuleLoad

The **olcModuleLoad** directive specifies a dynamically loadable module to load. It takes the following form:

```
olcModuleLoad: module
```

Here, *module* stands either for a file containing the module, or a back end, that will be loaded.

9.2.4. SELinux Policy for Applications Using LDAP

SELinux is an implementation of a mandatory access control mechanism in the Linux kernel. By default, SELinux prevents applications from accessing an OpenLDAP server. To enable authentication through LDAP, which is required by several applications, the **allow_ypbind** SELinux Boolean needs to be enabled. Certain applications also demand an enabled **authlogin_nsswitch_use_ldap** Boolean in this scenario. Execute the following commands to enable the aforementioned Booleans:

```
~]# setsebool -P allow_ypbind=1
```

```
~]# setsebool -P authlogin_nsswitch_use_ldap=1
```

The **-P** option makes this setting persistent across system reboots. See the Red Hat Enterprise Linux 7 SELinux User's and Administrator's Guide for more detailed information about SELinux.

9.2.5. Running an OpenLDAP Server

This section describes how to start, stop, restart, and check the current status of the **Standalone LDAP Daemon**. For more information on how to manage system services in general, see Managing Services with systemd in the *System Administrator's Guide*.

9.2.5.1. Starting the Service

To start the **slapd** service in the current session, type the following at a shell prompt as **root**:

```
~]# systemctl start slapd.service
```

To configure the service to start automatically at the boot time, use the following command as **root**:

```
~]# systemctl enable slapd.service
ln -s '/usr/lib/systemd/system/slapd.service' '/etc/systemd/system/multi-
user.target.wants/slapd.service'
```

9.2.5.2. Stopping the Service

To stop the running **slapd** service in the current session, type the following at a shell prompt as **root**:

```
~]# systemctl stop slapd.service
```

To prevent the service from starting automatically at the boot time, type as **root**:

```
~]# systemctl disable slapd.service
rm '/etc/systemd/system/multi-user.target.wants/slapd.service'
```

9.2.5.3. Restarting the Service

To restart the running **slapd** service, type the following at a shell prompt:

```
~]# systemctl restart slapd.service
```

This stops the service and immediately starts it again. Use this command to reload the configuration.

9.2.5.4. Verifying the Service Status

To verify that the **slapd** service is running, type the following at a shell prompt:

```
~]$ systemctl is-active slapd.service
active
```

9.2.6. Configuring a System to Authenticate Using OpenLDAP

In order to configure a system to authenticate using OpenLDAP, make sure that the appropriate packages are installed on both LDAP server and client machines. For information on how to set up the server, follow the instructions in Section 9.2.2, “Installing the OpenLDAP Suite” and Section 9.2.3, “Configuring an OpenLDAP Server”. On a client, type the following at a shell prompt:

```
~]# yum install openldap openldap-clients nss-pam-ldapd
```

9.2.6.1. Migrating Old Authentication Information to LDAP Format

The migrationtools package provides a set of shell and Perl scripts to help you migrate authentication information into an LDAP format. To install this package, type the following at a shell prompt:

```
~]# yum install migrationtools
```

This will install the scripts to the **/usr/share/migrationtools/** directory. Once installed, edit the **/usr/share/migrationtools/migrate_common.ph** file and change the following lines to reflect the correct domain, for example:

```
# Default DNS domain
$DEFAULT_MAIL_DOMAIN = "example.com";

# Default base
$DEFAULT_BASE = "dc=example,dc=com";
```

Alternatively, you can specify the environment variables directly on the command line. For example, to run the **migrate_all_online.sh** script with the default base set to **dc=example,dc=com**, type:

```
~]# export DEFAULT_BASE="dc=example,dc=com" \
/usr/share/migrationtools/migrate_all_online.sh
```

To decide which script to run in order to migrate the user database, see Table 9.8, “Commonly used LDAP migration scripts”.

Table 9.8. Commonly used LDAP migration scripts

Existing Name Service	Is LDAP Running?	Script to Use
/etc flat files	yes	**migrate_all_online.sh**
/etc flat files	no	**migrate_all_offline.sh**
NetInfo	yes	**migrate_all_netinfo_online.sh**
NetInfo	no	**migrate_all_netinfo_offline.sh**
NIS (YP)	yes	**migrate_all_nis_online.sh**
NIS (YP)	no	**migrate_all_nis_offline.sh**

For more information on how to use these scripts, see the **README** and the **migration-tools.txt** files in the **/usr/share/doc/migrationtools-*version*/** directory.

9.2.7. Additional Resources

The following resources offer additional information on the Lightweight Directory Access Protocol. Before configuring LDAP on your system, it is highly recommended that you review these resources, especially the *OpenLDAP Software Administrator's Guide*.

Installed Documentation

The following documentation is installed with the openldap-servers package:

- **/usr/share/doc/openldap-servers-*version*/guide.html** — A copy of the *OpenLDAP Software Administrator's Guide*.
- **/usr/share/doc/openldap-servers-*version*/README.schema** — A README file containing the description of installed schema files.

Additionally, there is also a number of manual pages that are installed with the openldap, openldap-servers, and openldap-clients packages:

Client Applications

- ldapadd(1) — The manual page for the **ldapadd** command describes how to add entries to an LDAP directory.
- ldapdelete(1) — The manual page for the **ldapdelete** command describes how to delete entries within an LDAP directory.
- ldapmodify(1) — The manual page for the **ldapmodify** command describes how to modify entries within an LDAP directory.

- ldapsearch(1) — The manual page for the **ldapsearch** command describes how to search for entries within an LDAP directory.
- ldappasswd(1) — The manual page for the **ldappasswd** command describes how to set or change the password of an LDAP user.
- ldapcompare(1) — Describes how to use the **ldapcompare** tool.
- ldapwhoami(1) — Describes how to use the **ldapwhoami** tool.
- ldapmodrdn(1) — Describes how to modify the RDNs of entries.

Server Applications

- slapd(8C) — Describes command line options for the LDAP server.

Administrative Applications

- slapadd(8C) — Describes command line options used to add entries to a **slapd** database.
- slapcat(8C) — Describes command line options used to generate an LDIF file from a **slapd** database.
- slapindex(8C) — Describes command line options used to regenerate an index based upon the contents of a **slapd** database.
- slappasswd(8C) — Describes command line options used to generate user passwords for LDAP directories.

Configuration Files

- ldap.conf(5) — The manual page for the **ldap.conf** file describes the format and options available within the configuration file for LDAP clients.
- slapd-config(5) — Describes the format and options available within the **/etc/openldap/slapd.d** configuration directory.

Other Resources

- *OpenLDAP and Mozilla NSS Compatibility Layer* Implementation details of NSS database backwards compatibility.
- *How do I use TLS/SSL?* Information on how to configure OpenLDAP to use OpenSSL.

PART III. SECURE APPLICATIONS

CHAPTER 10. USING PLUGGABLE AUTHENTICATION MODULES (PAM)

Pluggable authentication modules (PAMs) are a common framework for authentication and authorization. Most system applications in Red Hat Enterprise Linux depend on underlying PAM configuration for authentication and authorization.

10.1. ABOUT PAM

Pluggable Authentication Modules (PAMs) provide a centralized authentication mechanism which system application can use to relay authentication to a centrally configured framework.

PAM is pluggable because there is a PAM module for different types of authentication sources (such as Kerberos, SSSD, NIS, or the local file system). Different authentication sources can be prioritized.

This modular architecture offers administrators a great deal of flexibility in setting authentication policies for the system. PAM is a useful system for developers and administrators for several reasons:

- PAM provides a common authentication scheme that can be used with a wide variety of applications.
- PAM provides significant flexibility and control over authentication for system administrators.
- PAM provides a single, fully-documented library which allows developers to write programs without having to create their own authentication schemes.

10.1.1. Other PAM Resources

PAM has an extensive documentation set with much more detail about both using PAM and writing modules to extend or integrate PAM with other applications. Almost all of the major modules and configuration files with PAM have their own man pages. Additionally, the **/usr/share/doc/pam-version#/** directory contains a *System Administrators' Guide*, a *Module Writers' Manual*, and the *Application Developers' Manual*, as well as a copy of the PAM standard, DCE-RFC 86.0.

The libraries for PAM are available at http://www.linux-pam.org. This is the primary distribution website for the Linux-PAM project, containing information on various PAM modules, frequently asked questions, and additional PAM documentation.

10.1.2. Custom PAM Modules

New PAM modules can be created or added at any time for use by PAM-aware applications. PAM-aware programs can immediately use the new module and any methods it defines without being recompiled or otherwise modified. This allows developers and system administrators to use a selection of authentication modules, as well as tests, for different programs without recompiling them.

Documentation on writing modules is included in the **/usr/share/doc/pam-devel-version#/** directory.

10.2. ABOUT PAM CONFIGURATION FILES

Each PAM-aware application or *service* has a file in the **/etc/pam.d/** directory. Each file in this directory has the same name as the service to which it controls access. For example, the **login** program defines its service name as **login** and installs the **/etc/pam.d/login** PAM configuration

file.

WARNING

It is highly recommended to configure PAMs using the **authconfig** tool instead of manually editing the PAM configuration files.

10.2.1. PAM Configuration File Format

Each PAM configuration file contains a group of directives that define the module (the authentication configuration area) and any controls or arguments with it.

The directives all have a simple syntax that identifies the module purpose (interface) and the configuration settings for the module.

```
module_interface control_flag module_name module_arguments
```

In a PAM configuration file, the module interface is the first field defined. For example:

```
auth required pam_unix.so
```

A PAM *interface* is essentially the type of authentication action which that specific module can perform. Four types of PAM module interface are available, each corresponding to a different aspect of the authentication and authorization process:

- **auth** — This module interface authenticates users. For example, it requests and verifies the validity of a password. Modules with this interface can also set credentials, such as group memberships.
- **account** — This module interface verifies that access is allowed. For example, it checks if a user account has expired or if a user is allowed to log in at a particular time of day.
- **password** — This module interface is used for changing user passwords.
- **session** — This module interface configures and manages user sessions. Modules with this interface can also perform additional tasks that are needed to allow access, like mounting a user's home directory and making the user's mailbox available.

An individual module can provide any or all module interfaces. For instance, **pam_unix.so** provides all four module interfaces.

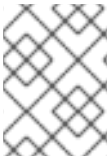
The module name, such as **pam_unix.so**, provides PAM with the name of the library containing the specified module interface. The directory name is omitted because the application is linked to the appropriate version of **libpam**, which can locate the correct version of the module.

All PAM modules generate a success or failure result when called. *Control flags* tell PAM what to do with the result. Modules can be listed (*stacked*) in a particular order, and the control flags determine how important the success or failure of a particular module is to the overall goal of authenticating the user to the service.

There are several simple flags[2], which use only a keyword to set the configuration:

- **required** — The module result must be successful for authentication to continue. If the test fails at this point, the user is not notified until the results of all module tests that reference that interface are complete.
- **requisite** — The module result must be successful for authentication to continue. However, if a test fails at this point, the user is notified immediately with a message reflecting the first failed **required** *or* **requisite** module test.
- **sufficient** — The module result is ignored if it fails. However, if the result of a module flagged **sufficient** is successful *and* no previous modules flagged **required** have failed, then no other results are required and the user is authenticated to the service.
- **optional** — The module result is ignored. A module flagged as **optional** only becomes necessary for successful authentication when no other modules reference the interface.
- **include** — Unlike the other controls, this does not relate to how the module result is handled. This flag pulls in all lines in the configuration file which match the given parameter and appends them as an argument to the module.

Module interface directives can be *stacked*, or placed upon one another, so that multiple modules are used together for one purpose.



NOTE

If a module's control flag uses the **sufficient** or **requisite** value, then the order in which the modules are listed is important to the authentication process.

Using stacking, the administrator can require specific conditions to exist before the user is allowed to authenticate. For example, the **setup** utility normally uses several stacked modules, as seen in its PAM configuration file:

```
[root@MyServer ~]# cat /etc/pam.d/setup

auth       sufficient pam_rootok.so
auth       include system-auth
account    required pam_permit.so
session    required pam_permit.so
```

- **auth sufficient pam_rootok.so** — This line uses the **pam_rootok.so** module to check whether the current user is root, by verifying that their UID is 0. If this test succeeds, no other modules are consulted and the command is executed. If this test fails, the next module is consulted.
- **auth include system-auth** — This line includes the content of the **/etc/pam.d/system-auth** module and processes this content for authentication.
- **account required pam_permit.so** — This line uses the **pam_permit.so** module to allow the root user or anyone logged in at the console to reboot the system.
- **session required pam_permit.so** — This line is related to the session setup. Using **pam_permit.so**, it ensures that the **setup** utility does not fail.

PAM uses *arguments* to pass information to a pluggable module during authentication for some modules.

For example, the **pam_pwquality.so** module checks how strong a password is and can take several arguments. In the following example, **enforce_for_root** specifies that even password of the root user must successfully pass the strength check and **retry** defines that a user will receive three opportunities to enter a strong password.

```
password requisite pam_pwquality.so enforce_for_root retry=3
```

Invalid arguments are generally ignored and do not otherwise affect the success or failure of the PAM module. Some modules, however, may fail on invalid arguments. Most modules report errors to the **journald** service. For information on how to use **journald** and the related **journalctl** tool, see the System Administrator's Guide.

NOTE

The **journald** service was introduced in Red Hat Enterprise Linux 7.1. In previous versions of Red Hat Enterprise Linux, most modules report errors to the **/var/log/secure** file.

10.2.2. Annotated PAM Configuration Example

Example 10.1, “Simple PAM Configuration” is a sample PAM application configuration file:

Example 10.1. Simple PAM Configuration

```
#%PAM-1.0
auth  required pam_securetty.so
auth  required pam_unix.so nullok
auth  required pam_nologin.so
account  required pam_unix.so
password required pam_pwquality.so retry=3
password required pam_unix.so shadow nullok use_authtok
session  required pam_unix.so
```

- The first line is a comment, indicated by the hash mark (#) at the beginning of the line.
- Lines two through four stack three modules for login authentication.

  auth required pam_securetty.so — This module ensures that *if* the user is trying to log in as root, the TTY on which the user is logging in is listed in the **/etc/securetty** file, *if* that file exists.

  If the TTY is not listed in the file, any attempt to log in as root fails with a **Login incorrect** message.

  auth required pam_unix.so nullok — This module prompts the user for a password and then checks the password using the information stored in **/etc/passwd** and, if it exists, **/etc/shadow**.

  The argument **nullok** instructs the **pam_unix.so** module to allow a blank password.

- **auth required pam_nologin.so** — This is the final authentication step. It checks whether the **/etc/nologin** file exists. If it exists and the user is not root, authentication fails.

NOTE

In this example, all three **auth** modules are checked, even if the first **auth** module fails. This prevents the user from knowing at what stage their authentication failed. Such knowledge in the hands of an attacker could allow them to more easily deduce how to crack the system.

- **account required pam_unix.so** — This module performs any necessary account verification. For example, if shadow passwords have been enabled, the account interface of the **pam_unix.so** module checks to see if the account has expired or if the user has not changed the password within the allowed grace period.

- **password required pam_pwquality.so retry=3** — If a password has expired, the password component of the **pam_pwquality.so** module prompts for a new password. It then tests the newly created password to see whether it can easily be determined by a dictionary-based password cracking program.

  The argument **retry=3** specifies that if the test fails the first time, the user has two more chances to create a strong password.

- **password required pam_unix.so shadow nullok use_authtok** — This line specifies that if the program changes the user's password, using the **password** interface of the **pam_unix.so** module.

  - The argument **shadow** instructs the module to create shadow passwords when updating a user's password.

  - The argument **nullok** instructs the module to allow the user to change their password *from* a blank password, otherwise a null password is treated as an account lock.

  - The final argument on this line, **use_authtok**, provides a good example of the importance of order when stacking PAM modules. This argument instructs the module not to prompt the user for a new password. Instead, it accepts any password that was recorded by a previous password module. In this way, all new passwords must pass the **pam_pwquality.so** test for secure passwords before being accepted.

- **session required pam_unix.so** — The final line instructs the session interface of the **pam_unix.so** module to manage the session. This module logs the user name and the service type to **/var/log/secure** at the beginning and end of each session. This module can be supplemented by stacking it with other session modules for additional functionality.

10.3. PAM AND ADMINISTRATIVE CREDENTIAL CACHING

A number of graphical administrative tools in Red Hat Enterprise Linux, such as the GNOME's **control-center**, provide users with elevated privileges for up to five minutes using the **pam_timestamp.so** module. It is important to understand how this mechanism works, because a user who walks away from a terminal while **pam_timestamp.so** is in effect leaves the machine open to manipulation by anyone with physical access to the console.

In the PAM timestamp scheme, the graphical administrative application prompts the user for the root password when it is launched. When the user has been authenticated, the **pam_timestamp.so** module

creates a timestamp file. By default, this is created in the **/var/run/sudo/** directory. If the timestamp file already exists, graphical administrative programs do not prompt for a password. Instead, the **pam_timestamp.so** module freshens the timestamp file, reserving an extra five minutes of unchallenged administrative access for the user.

You can verify the actual state of the timestamp file by inspecting the file in the **/var/run/sudo/***user* directory. For the desktop, the relevant file is **unknown:root**. If it is present and its timestamp is less than five minutes old, the credentials are valid.

The existence of the timestamp file is indicated by an authentication icon, which appears in the notification area of the panel.

Figure 10.1. The Authentication Icon

10.3.1. Removing the Timestamp File

Before abandoning a console where a PAM timestamp is active, it is recommended that the timestamp file be destroyed. To do this from a graphical environment, click the authentication icon on the panel. This causes a dialog box to appear. Click the **Forget Authorization** button to destroy the active timestamp file.

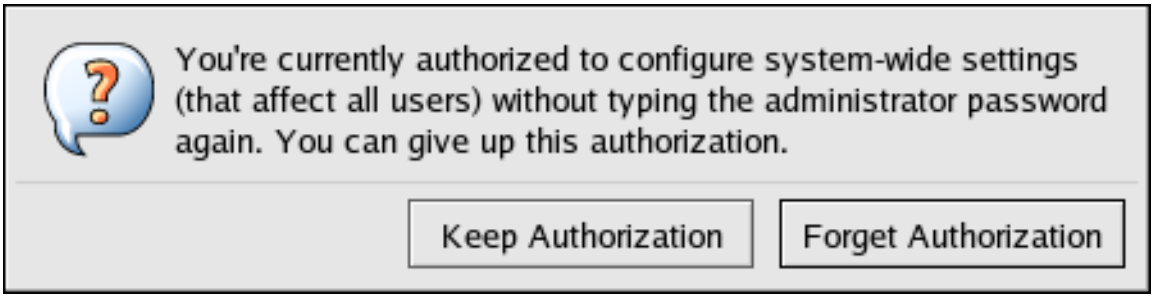


Figure 10.2. Dismiss Authentication Dialog

The PAM timestamp file has some important characteristics:

- If logged in to the system remotely using **ssh**, use the **/sbin/pam_timestamp_check -k root** command to destroy the timestamp file.
- Run the **/sbin/pam_timestamp_check -k root** command from the same terminal window where the privileged application was launched.
- The logged in user who originally invoked the **pam_timestamp.so** module must be the user who runs the **/sbin/pam_timestamp_check -k** command. Do not run this command as root.
- Killing the credentials on the desktop without using the **Forget Authorization** action on the icon can be done with the **/sbin/pam_timestamp_chec** command.

  ```
  /sbin/pam_timestamp_check -k root </dev/null >/dev/null 2>/dev/null
  ```

  Any other method only removes the credentials from the PTY where the command was run.

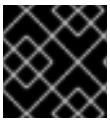
Refer to the **pam_timestamp_check** man page for more information about destroying the timestamp file using **pam_timestamp_check**.

10.3.2. Common pam_timestamp Directives

The **pam_timestamp.so** module accepts several directives, with two used most commonly:

- **timestamp_timeout** — Specifies the period (in seconds) for which the timestamp file is valid. The default value is 300 (five minutes).
- **timestampdir** — Specifies the directory in which the timestamp file is stored. The default value is **/var/run/sudo/**.

10.4. RESTRICTING DOMAINS FOR PAM SERVICES

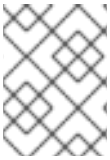


IMPORTANT

This feature requires SSSD to be running on the system.

SSSD enables you to restrict which domains can be accessed by PAM services. SSSD evaluates authentication requests from PAM services based on the user the particular PAM service is running as. Whether the PAM service can access an SSSD domain depends on whether the PAM service user is able to access the domain.

An example use case is an environment where external users are allowed to authenticate to an FTP server. The FTP server is running as a separate non-privileged user that should only be able to authenticate to a selected SSSD domain, separate from internal company accounts. With this feature, the administrator can allow the FTP user to only authenticate to selected domains specified in the FTP PAM configuration file.



NOTE

This functionality is similar to legacy PAM modules, such as **pam_ldap**, which were able to use a separate configuration file as a parameter for a PAM module.

Options to Restrict Access to Domains

The following options are available to restrict access to selected domains:

pam_trusted_users in /etc/sssd/sssd.conf

This option accepts a list of numerical UIDs or user names representing the PAM services that are to be trusted by SSSD. The default setting is **all**, which means all service users are trusted and can access any domain.

pam_public_domains in /etc/sssd/sssd.conf

This option accepts a list of public SSSD domains. Public domains are domains accessible even for untrusted PAM service users. The option also accepts the **all** and **none** values. The default value is **none**, which means no domains are public and untrusted service users therefore cannot access any domain.

domains for PAM configuration files

This option specifies a list of domains against which a PAM service can authenticate. If you use **domains** without specifying any domain, the PAM service will not be able to authenticate against any domain, for example:

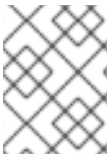
```
auth     required   pam_sss.so domains=
```

If **domains** is not used in the PAM configuration file, the PAM service is able to authenticate against all domains, on the condition that the service is running under a trusted user.

The **domains** option in the **/etc/sssd/sssd.conf** SSSD configuration file also specifies a list of domains to which SSSD attempts to authenticate. Note that the **domains** option in a PAM configuration file cannot extend the list of domains in **sssd.conf**, it can only restrict the **sssd.conf** list of domains by specifying a shorter list. Therefore, if a domain is specified in the PAM file but not in **sssd.conf**, the PAM service will not be able to authenticate against the domain.

The default settings **pam_trusted_users = all** and **pam_public_domains = none** specify that all PAM service users are trusted and can access any domain. The **domains** option for PAM configuration files can be used in this situation to restrict the domains that can be accessed.

If you specify a domain using **domains** in the PAM configuration file while **sssd.conf** contains **pam_public_domains**, it might be required to specify the domain in **pam_public_domains** as well. If **pam_public_domains** is used but does not include the required domain, the PAM service will not be able to successfully authenticate against the domain if it is running under an untrusted user.



NOTE

Domain restrictions defined in a PAM configuration file only apply to authentication actions, not to user lookups.

For more information about the **pam_trusted_users** and **pam_public_domains** options, see the sssd.conf(5) man page. For more information about the **domains** option used in PAM configuration files, see the pam_sss(8) man page.

Example 10.2. Restricting Domains for a PAM Service

To restrict the domains against which a PAM service can authenticate:

1. Make sure SSSD is configured to access the required domain or domains. The domains against which SSSD can authenticate are defined in the **domains** option in the **/etc/sssd/sssd.conf** file.

   ```
   [sssd]
   domains = domain1, domain2, domain3
   ```

2. Specify the domain or domains to which a PAM service will be able to authenticate. To do this, set the **domains** option in the PAM configuration file. For example:

   ```
   auth        sufficient    pam_sss.so forward_pass domains=domain1
   account     [default=bad success=ok user_unknown=ignore]
   pam_sss.so
   password    sufficient    pam_sss.so use_authtok
   ```

The PAM service is now only allowed to authenticate against **domain1**.

[2] There are many complex control flags that can be set. These are set in *attribute=value* pairs; a complete list of attributes is available in the **pam.d** manpage.

CHAPTER 11. USING KERBEROS

Maintaining system security and integrity within a network is critical, and it encompasses every user, application, service, and server within the network infrastructure. It requires an understanding of everything that is running on the network and the manner in which these services are used. At the core of maintaining this security is maintaining *access* to these applications and services and enforcing that access.

Kerberos is an authentication protocol significantly safer than normal password-based authentication. With Kerberos, passwords are never sent over the network, even when services are accessed on other machines.

Kerberos provides a mechanism that allows both users and machines to identify themselves to network and receive defined, limited access to the areas and services that the administrator configured. Kerberos *authenticates* entities by verifying their identity, and Kerberos also secures this authenticating data so that it cannot be accessed and used or tampered with by an outsider.

11.1. ABOUT KERBEROS

Kerberos uses symmetric-key cryptography[3] to authenticate users to network services, which means passwords are never actually sent over the network.

Consequently, when users authenticate to network services using Kerberos, unauthorized users attempting to gather passwords by monitoring network traffic are effectively thwarted.

11.1.1. The Basics of How Kerberos Works

Most conventional network services use password-based authentication schemes, where a user supplies a password to access a given network server. However, the transmission of authentication information for many services is unencrypted. For such a scheme to be secure, the network has to be inaccessible to outsiders, and all computers and users on the network must be trusted and trustworthy.

With simple, password-based authentication, a network that is connected to the Internet cannot be assumed to be secure. Any attacker who gains access to the network can use a simple packet analyzer, or *packet sniffer*, to intercept user names and passwords, compromising user accounts and, therefore, the integrity of the entire security infrastructure.

Kerberos eliminates the transmission of unencrypted passwords across the network and removes the potential threat of an attacker sniffing the network.

Rather than authenticating each user to each network service separately as with simple password authentication, Kerberos uses symmetric encryption and a trusted third party (a *key distribution center* or KDC) to authenticate users to a suite of network services. The computers managed by that KDC and any secondary KDCs constitute a *realm*.

When a user authenticates to the KDC, the KDC sends a set of credentials (a *ticket*) specific to that session back to the user's machine, and any Kerberos-aware services look for the ticket on the user's machine rather than requiring the user to authenticate using a password.

As shown in Figure 11.1, “Kerberos Authentication”, each user is identified to the KDC with a unique identity, called a *principal*. When a user on a Kerberos-aware network logs into his workstation, his principal is sent to the KDC as part of a request for a *ticket-granting ticket* (or TGT) from the authentication server. This request can be sent by the login program so that it is transparent to the user or can be sent manually by a user through the **kinit** program after the user logs in.

The KDC then checks for the principal in its database. If the principal is found, the KDC creates a TGT, encrypts it using the user's key, and sends the TGT to that user.

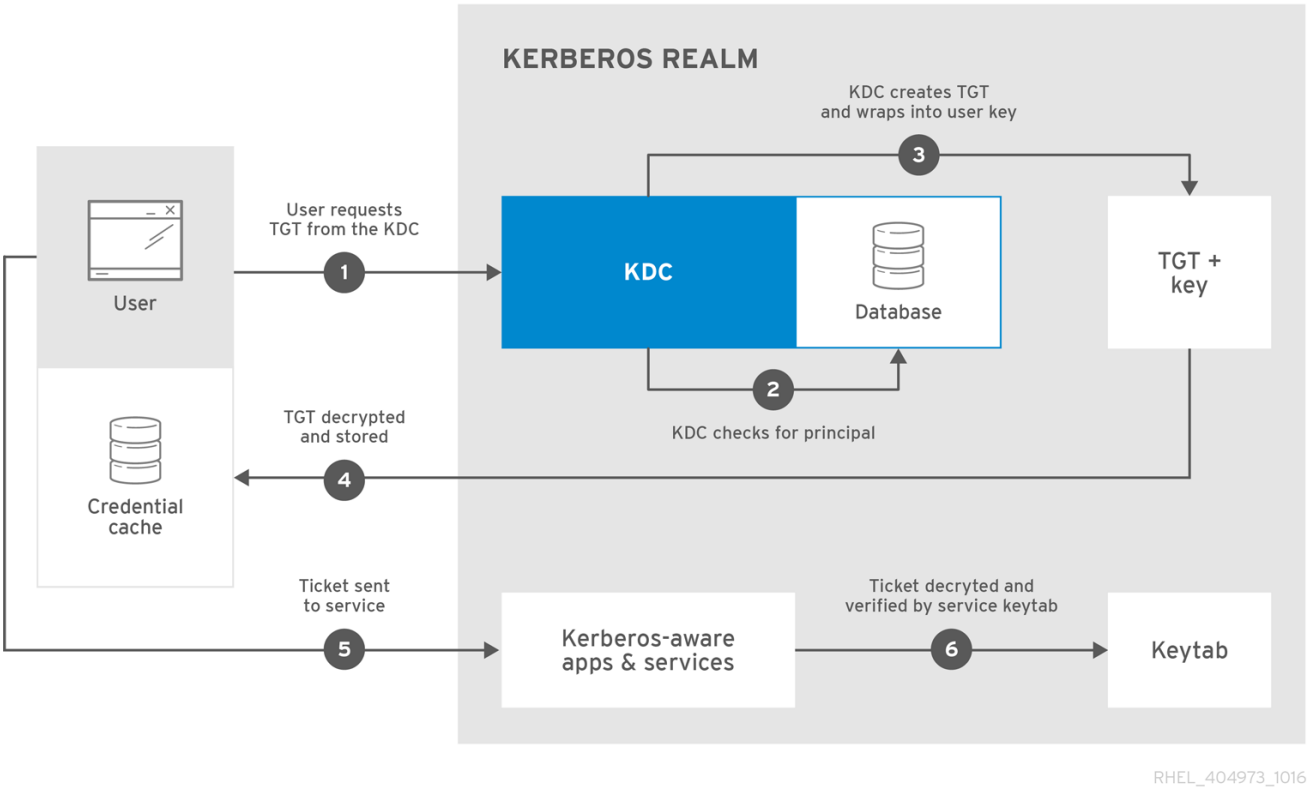


Figure 11.1. Kerberos Authentication

The login or **`kinit`** program on the client then decrypts the TGT using the user's key, which it computes from the user's password. The user's key is used only on the client machine and is *not* transmitted over the network. The ticket (or credentials) sent by the KDC are stored in a local store, the *credential cache (ccache)*, which can be checked by Kerberos-aware services. Red Hat Enterprise Linux 7 supports the following types of credential caches:

- KEYRING; the persistent KEYRING ccache type is the default for Red Hat Enterprise Linux 7
- FILE
- DIR
- MEMORY

After authentication, servers can check an unencrypted list of recognized principals and their keys rather than checking **`kinit`**; this is kept in a *keytab*.

The TGT is set to expire after a certain period of time (usually 10 to 24 hours) and is stored in the client machine's credential cache. An expiration time is set so that a compromised TGT is of use to an attacker for only a short period of time. After the TGT has been issued, the user does not have to enter their password again until the TGT expires or until they log out and log in again.

Whenever the user needs access to a network service, the client software uses the TGT to request a new ticket for that specific service from the ticket-granting server (TGS). The service ticket is then used to authenticate the user to that service transparently.

11.1.2. About Kerberos Principal Names

The principal identifies not only the user or service, but also the realm that the entity belongs to. A principal name has two parts, the identifier and the realm:

```
identifier@REALM
```

For a user, the *identifier* is only the Kerberos user name. For a service, the *identifier* is a combination of the service name and the host name of the machine it runs on:

```
service/FQDN@REALM
```

The *service* name is a case-sensitive string that is specific to the service type, like **host**, **ldap**, **http**, and **DNS**. Not all services have obvious principal identifiers; the **sshd** daemon, for example, uses the host service principal.

The host principal is usually stored in **/etc/krb5.keytab**.

When Kerberos requests a ticket, it always resolves the domain name aliases (DNS CNAME records) to the corresponding DNS address (A or AAAA records). The host name from the address record is then used when service or host principals are created.

For example:

```
www.example.com  CNAME  web-01.example.com
web-01.example.com  A  192.0.2.145
```

A service attempts to connect to the host using its CNAME alias:

```
$ ssh www.example.com
```

The Kerberos server requests a ticket for the resolved host name, **web-01.example.com@EXAMPLE.COM**, so the host principal must be **host/web-01.example.com@EXAMPLE.COM**.

11.1.3. About the Domain-to-Realm Mapping

When a client attempts to access a service running on a particular server, it knows the name of the service (*host*) and the name of the server (*foo.example.com*), but because more than one realm can be deployed on the network, it must guess at the name of the Kerberos realm in which the service resides.

By default, the name of the realm is taken to be the DNS domain name of the server in all capital letters.

```
foo.example.org → EXAMPLE.ORG
foo.example.com → EXAMPLE.COM
foo.hq.example.com → HQ.EXAMPLE.COM
```

In some configurations, this will be sufficient, but in others, the realm name which is derived will be the name of a non-existent realm. In these cases, the mapping from the server's DNS domain name to the name of its realm must be specified in the **domain_realm** section of the client system's **/etc/krb5.conf** file. For example:

```
[domain_realm]
.example.com = EXAMPLE.COM
example.com = EXAMPLE.COM
```

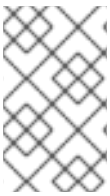
The configuration specifies two mappings. The first mapping specifies that any system in the example.com DNS domain belongs to the *EXAMPLE.COM* realm. The second specifies that a system with the exact name example.com is also in the realm. The distinction between a domain and a specific host is marked by the presence or lack of an initial period character. The mapping can also be stored directly in DNS using the "_kerberos TXT" records, for example:

```
$ORIGIN example.com
_kerberos  TXT "EXAMPLE.COM"
```

11.1.4. Environmental Requirements

Kerberos relies on being able to resolve machine names. Thus, it requires a working domain name service (DNS). Both DNS entries and hosts on the network must be properly configured, which is covered in the Kerberos documentation in **/usr/share/doc/krb5-server-***version-number*.

Applications that accept Kerberos authentication require time synchronization. You can set up approximate clock synchronization between the machines on the network using a service such as **ntpd**. For information on the **ntpd** service, see the documentation in **/usr/share/doc/ntp-***version-number***/html/index.html** or the ntpd(8) man page.



NOTE

Kerberos clients running Red Hat Enterprise Linux 7 support automatic time adjustment with the KDC and have no strict timing requirements. This enables better tolerance to clocking differences when deploying IdM clients with Red Hat Enterprise Linux 7.

11.1.5. Considerations for Deploying Kerberos

Although Kerberos removes a common and severe security threat, it is difficult to implement for a variety of reasons:

- Kerberos assumes that each user is trusted but is using an untrusted host on an untrusted network. Its primary goal is to prevent unencrypted passwords from being transmitted across that network. However, if anyone other than the proper user has access to the one host that issues tickets used for authentication — the KDC — the entire Kerberos authentication system are at risk.

- For an application to use Kerberos, its source must be modified to make the appropriate calls into the Kerberos libraries. Applications modified in this way are considered to be *Kerberos-aware*. For some applications, this can be quite problematic due to the size of the application or its design. For other incompatible applications, changes must be made to the way in which the server and client communicate. Again, this can require extensive programming. Closed source applications that do not have Kerberos support by default are often the most problematic.

- To secure a network with Kerberos, one must either use Kerberos-aware versions of *all* client and server applications that transmit passwords unencrypted, or not use that client and server application at all.

- Migrating user passwords from a standard UNIX password database, such as **/etc/passwd** or **/etc/shadow**, to a Kerberos password database can be tedious. There is no automated mechanism to perform this task. Migration methods can vary substantially depending on the particular way Kerberos is deployed. That is why it is recommended that you use the Identity Management feature; it has specialized tools and methods for migration.

WARNING

The Kerberos system can be compromised if a user on the network authenticates against a non-Kerberos aware service by transmitting a password in plain text. The use of non-Kerberos aware services (including telnet and FTP) is highly discouraged. Other encrypted protocols, such as SSH or SSL-secured services, are preferred to unencrypted services, but this is still not ideal.

11.1.6. Additional Resources for Kerberos

Kerberos can be a complex service to implement, with a lot of flexibility in how it is deployed. Table 11.1, "External Kerberos Documentation" and Table 11.2, "Important Kerberos Man Pages" list of a few of the most important or most useful sources for more information on using Kerberos.

Table 11.1. External Kerberos Documentation

Documentation	Location
Kerberos V5 Installation Guide (in both PostScript and HTML)	**/usr/share/doc/krb5-server-*version-number***
Kerberos V5 System Administrator's Guide (in both PostScript and HTML)	**/usr/share/doc/krb5-server-*version-number***
Kerberos V5 UNIX User's Guide (in both PostScript and HTML)	**/usr/share/doc/krb5-workstation-*version-number***
"Kerberos: The Network Authentication Protocol" web page from MIT	http://web.mit.edu/kerberos/www/
Designing an Authentication System: a Dialogue in Four Scenes, originally by Bill Bryant in 1988, modified by Theodore Ts'o in 1997. This document is a conversation between two developers who are thinking through the creation of a Kerberos-style authentication system. The conversational style of the discussion makes this a good starting place for people who are completely unfamiliar with Kerberos.	http://web.mit.edu/kerberos/www/dialogue.html
An article for making a network Kerberos-aware.	http://www.ornl.gov/~jar/HowToKerb.html

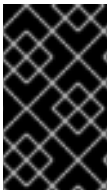
Any of the manpage files can be opened by running **man** *command_name*.

Table 11.2. Important Kerberos Man Pages

Manpage	Description
Client Applications	
kerberos	An introduction to the Kerberos system which describes how credentials work and provides recommendations for obtaining and destroying Kerberos tickets. The bottom of the man page references a number of related man pages.
kinit	Describes how to use this command to obtain and cache a ticket-granting ticket.
kdestroy	Describes how to use this command to destroy Kerberos credentials.
klist	Describes how to use this command to list cached Kerberos credentials.
Administrative Applications	
kadmin	Describes how to use this command to administer the Kerberos V5 database.
kdb5_util	Describes how to use this command to create and perform low-level administrative functions on the Kerberos V5 database.
Server Applications	
krb5kdc	Describes available command line options for the Kerberos V5 KDC.
kadmind	Describes available command line options for the Kerberos V5 administration server.
Configuration Files	
krb5.conf	Describes the format and options available within the configuration file for the Kerberos V5 library.
kdc.conf	Describes the format and options available within the configuration file for the Kerberos V5 AS and KDC.

11.2. CONFIGURING THE KERBEROS KDC

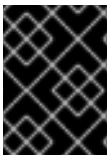
Install the master KDC first and then install any necessary secondary servers after the master is set up.



IMPORTANT

Setting up Kerberos KDC manually is not recommended. The recommended way to introduce Kerberos into Red Hat Enterprise Linux environments is to use the Identity Management feature.

11.2.1. Configuring the Master KDC Server



IMPORTANT

The KDC system should be a dedicated machine. This machine needs to be very secure — if possible, it should not run any services other than the KDC.

1. Install the required packages for the KDC:

```
[root@server ~]# yum install krb5-server krb5-libs krb5-workstation
```

2. Edit the **/etc/krb5.conf** and **/var/kerberos/krb5kdc/kdc.conf** configuration files to reflect the realm name and domain-to-realm mappings. For example:

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 default_realm = EXAMPLE.COM
 dns_lookup_realm = false
 dns_lookup_kdc = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 allow_weak_crypto = true

[realms]
  EXAMPLE.COM = {
  kdc = kdc.example.com.:88
  admin_server = kdc.example.com
  default_domain = example.com
 }

[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

A simple realm can be constructed by replacing instances of *EXAMPLE.COM* and *example.com* with the correct domain name — being certain to keep uppercase and lowercase names in the correct format — and by changing the KDC from *kerberos.example.com* to the name of the Kerberos server. By convention, all realm names are uppercase and all DNS host names and domain names are lowercase. The man pages of these configuration files have full details about the file formats.

3. Create the database using the **kdb5_util** utility.

```
[root@server ~]# kdb5_util create -s
```

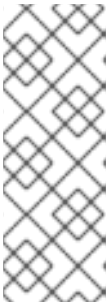
The **create** command creates the database that stores keys for the Kerberos realm. The **-s** argument creates a *stash* file in which the master server key is stored. If no stash file is present from which to read the key, the Kerberos server (**krb5kdc**) prompts the user for the master server password (which can be used to regenerate the key) every time it starts.

4. Edit the **/var/kerberos/krb5kdc/kadm5.acl** file. This file is used by **kadmind** to determine which principals have administrative access to the Kerberos database and their level of access. For example:

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

Most users are represented in the database by a single principal (with a *NULL*, or empty, instance, such as *joe@EXAMPLE.COM*). In this configuration, users with a second principal with an instance of *admin* (for example, *joe/admin@EXAMPLE.COM*) are able to exert full administrative control over the realm's Kerberos database.

After **kadmind** has been started on the server, any user can access its services by running **kadmin** on any of the clients or servers in the realm. However, only users listed in the **kadm5.acl** file can modify the database in any way, except for changing their own passwords.



NOTE

The **kadmin** utility communicates with the **kadmind** server over the network, and uses Kerberos to handle authentication. Consequently, the first principal must already exist before connecting to the server over the network to administer it. Create the first principal with the **kadmin.local** command, which is specifically designed to be used on the same host as the KDC and does not use Kerberos for authentication.

5. Create the first principal using **kadmin.local** at the KDC terminal:

```
[root@server ~]# kadmin.local -q "addprinc username/admin"
```

6. Start Kerberos using the following commands:

```
[root@server ~]# systemctl start krb5kdc.service
[root@server ~]# systemctl start kadmin.service
```

7. Add principals for the users using the **addprinc** command within **kadmin**. **kadmin** and **kadmin.local** are command line interfaces to the KDC. As such, many commands — such as **addprinc** — are available after launching the **kadmin** program. Refer to the **kadmin** man page for more information.

8. Verify that the KDC is issuing tickets. First, run **kinit** to obtain a ticket and store it in a credential cache file. Next, use **klist** to view the list of credentials in the cache and use **kdestroy** to destroy the cache and the credentials it contains.

NOTE

By default, **kinit** attempts to authenticate using the same system login user name (not the Kerberos server). If that user name does not correspond to a principal in the Kerberos database, **kinit** issues an error message. If that happens, supply **kinit** with the name of the correct principal as an argument on the command line:

```
kinit principal
```

11.2.2. Setting up Secondary KDCs

When there are multiple KDCs for a given realm, one KDC (the *master KDC*) keeps a writable copy of the realm database and runs **kadmind**. The master KDC is also the realm's *admin server*. Additional secondary KDCs keep read-only copies of the database and run **kpropd**.

The master and slave propagation procedure entails the master KDC dumping its database to a temporary dump file and then transmitting that file to each of its slaves, which then overwrite their previously received read-only copies of the database with the contents of the dump file.

To set up a secondary KDC:

1. Install the required packages for the KDC:

```
[root@slavekdc ~]# yum install krb5-server krb5-libs krb5-
workstation
```

2. Copy the master KDC's **krb5.conf** and **kdc.conf** files to the secondary KDC.

3. Start **kadmin.local** from a root shell on the master KDC.

    1. Use the **kadmin.local add_principal** command to create a new entry for the master KDC's *host* service.

```
[root@slavekdc ~]# kadmin.local -r EXAMPLE.COM
 Authenticating as principal root/admin@EXAMPLE.COM with
password.
kadmin: add_principal -randkey host/masterkdc.example.com
Principal "host/masterkdc.example.com@EXAMPLE.COM" created.
kadmin: ktadd host/masterkdc.example.com
Entry for principal host/masterkdc.example.com with kvno 3,
encryption type Triple DES cbc mode with HMAC/sha1 added to
keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/masterkdc.example.com with kvno 3,
encryption type ArcFour with HMAC/md5 added to keytab
WRFILE:/etc/krb5.keytab.
Entry for principal host/masterkdc.example.com with kvno 3,
encryption type DES with HMAC/sha1 added to keytab
WRFILE:/etc/krb5.keytab.
Entry for principal host/masterkdc.example.com with kvno 3,
encryption type DES cbc mode with RSA-MD5 added to keytab
WRFILE:/etc/krb5.keytab.
kadmin: quit
```

2. Use the **kadmin.local ktadd** command to set a random key for the service and store the random key in the master's default keytab file.

NOTE

This key is used by the **kprop** command to authenticate to the secondary servers. You will only need to do this once, regardless of how many secondary KDC servers you install.

4. Start **kadmin** from a root shell on the secondary KDC.

   1. Use the **kadmin.local add_principal** command to create a new entry for the secondary KDC's *host* service.

```
[root@slavekdc ~]# kadmin -p jsmith/admin@EXAMPLE.COM -r
EXAMPLE.COM
Authenticating as principal jsmith/admin@EXAMPLE.COM with
password.
Password for jsmith/admin@EXAMPLE.COM:
kadmin: add_principal -randkey host/slavekdc.example.com
Principal "host/slavekdc.example.com@EXAMPLE.COM" created.
kadmin: ktadd host/slavekdc.example.com@EXAMPLE.COM
Entry for principal host/slavekdc.example.com with kvno 3,
encryption type Triple DES cbc mode with HMAC/sha1 added to
keytab WRFILE:/etc/krb5.keytab.
Entry for principal host/slavekdc.example.com with kvno 3,
encryption type ArcFour with HMAC/md5 added to keytab
WRFILE:/etc/krb5.keytab.
Entry for principal host/slavekdc.example.com with kvno 3,
encryption type DES with HMAC/sha1 added to keytab
WRFILE:/etc/krb5.keytab.
Entry for principal host/slavekdc.example.com with kvno 3,
encryption type DES cbc mode with RSA-MD5 added to keytab
WRFILE:/etc/krb5.keytab.
kadmin: quit
```

   2. Use the **kadmin.local ktadd** command to set a random key for the service and store the random key in the secondary KDC server's default keytab file. This key is used by the **kpropd** service when authenticating clients.

5. With its service key, the secondary KDC could authenticate any client which would connect to it. Obviously, not all potential clients should be allowed to provide the **kprop** service with a new realm database. To restrict access, the **kprop** service on the secondary KDC will only accept updates from clients whose principal names are listed in **/var/kerberos/krb5kdc/kpropd.acl**.

   Add the master KDC's host service's name to that file.

```
[root@slavekdc ~]# echo host/masterkdc.example.com@EXAMPLE.COM >
/var/kerberos/krb5kdc/kpropd.acl
```

6. Once the secondary KDC has obtained a copy of the database, it will also need the master key which was used to encrypt it. If the KDC database's master key is stored in a stash file on the master KDC (typically named **/var/kerberos/krb5kdc/.k5.REALM**), either copy it to the

secondary KDC using any available secure method, or create a dummy database and identical stash file on the secondary KDC by running **kdb5_util create -s** and supplying the same password. The dummy database will be overwritten by the first successful database propagation.

7. Ensure that the secondary KDC's firewall allows the master KDC to contact it using TCP on port 754 (*krb5_prop*), and start the **kprop** service.

8. Verify that the **kadmin** service is *disabled*.

9. Perform a manual database propagation test by dumping the realm database on the master KDC to the default data file which the **kprop** command will read (**/var/kerberos/krb5kdc/slave_datatrans**).

```
[root@masterkdc ~]# kdb5_util dump
/var/kerberos/krb5kdc/slave_datatrans
```

10. Use the **kprop** command to transmit its contents to the secondary KDC.

```
[root@slavekdc ~]# kprop slavekdc.example.com
```

11. Using **kinit**, verify that the client system is able to correctly obtain the initial credentials from the KDC.

    The **/etc/krb5.conf** for the client should list only the secondary KDC in its list of KDCs.

```
[realms]
  EXAMPLE.COM = {
  kdc = slavekdc.example.com.:88
  admin_server = kdc.example.com
  default_domain = example.com
 }
```

12. Create a script which dumps the realm database and runs the **kprop** command to transmit the database to each secondary KDC in turn, and configure the **cron** service to run the script periodically.

11.2.3. Kerberos Key Distribution Center Proxy

In some deployments, only the HTTPS port (443 using TCP) is accessible and not the default Kerberos ports. Clients can obtain Kerberos credentials using the IdM HTTPS service as a proxy. This reverse proxy enables accessing Kerberos-authenticated services through HTTPS.

Kerberos Key Distribution Center Proxy (KKDCP) provides this functionality in IdM.

Configuring KKDCP in Your Deployment

On an IdM server, KKDCP is enabled by default.

On an IdM client, you must enable KKDCP:

1. Reconfigure the **/etc/krb5.conf** file as described in Section 11.3, “Configuring a Kerberos Client”.

2. Restart the SSSD service:

```
# systemctl restart sssd.service
```

Verifying That KKDCP Is Enabled on an IdM Server

The KKDCP is automatically enabled each time the Apache web server starts, if the attribute and value pair **ipaConfigString=kdcProxyEnabled** exists in the directory. In this situation, the symbolic link **/etc/httpd/conf.d/ipa-kdc-proxy.conf** is created.

To verify if the KKDCP feature is enabled, check that the symbolic link exists:

```
$ ls -l /etc/httpd/conf.d/ipa-kdc-proxy.conf
lrwxrwxrwx. 1 root root 36 Aug 15 09:37 /etc/httpd/conf.d/ipa-kdc-
proxy.conf -> /etc/ipa/kdcproxy/ipa-kdc-proxy.conf
```

Disabling KKDCP on an IdM Server

1. Remove the **ipaConfigString=kdcProxyEnabled** attribute and value pair from the directory:

```
# ipa-ldap-updater /usr/share/ipa/kdcproxy-disable.uldif
```

2. Restart the **httpd** service on the IdM server:

```
# systemctl restart httpd.service
```

Additional Resources

- For details on configuring KKDCP for an Active Directory realm, see Configure IPA server as a KDC Proxy for AD Kerberos communication in Red Hat Knowledgebase.

11.3. CONFIGURING A KERBEROS CLIENT

All that is required to set up a Kerberos 5 client is to install the client packages and provide each client with a valid **krb5.conf** configuration file. While **ssh** and **slogin** are the preferred methods of remotely logging in to client systems, Kerberos-aware versions of **rsh** and **rlogin** are still available, with additional configuration changes.

1. Install the **krb5-libs** and **krb5-workstation** packages on all of the client machines.

```
[root@server ~]# yum install krb5-workstation krb5-libs
```

2. Supply a valid **/etc/krb5.conf** file for each client. Usually this can be the same **krb5.conf** file used by the Kerberos Distribution Center (KDC). For example:

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 default_realm = EXAMPLE.COM
 dns_lookup_realm = false
 dns_lookup_kdc = false
```

```
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 allow_weak_crypto = true

[realms]
  EXAMPLE.COM = {
  kdc = kdc.example.com.:88
  admin_server = kdc.example.com
  default_domain = example.com
 }

[domain_realm]
 .example.com = EXAMPLE.COM
 example.com = EXAMPLE.COM
```

In some environments, the KDC is only accessible using an HTTPS Kerberos Key Distribution Center Proxy (KKDCP). In this case, make the following changes:

1. Assign the URL of the KKDCP instead of the host name to the **kdc** and **admin_server** options in the **[realms]** section:

   ```
   [realms]
   EXAMPLE.COM = {
     kdc = https://kdc.example.com/KdcProxy
     admin_server = https://kdc.example.com/KdcProxy
     kpasswd_server = https://kdc.example.com/KdcProxy
     default_domain = example.com
   }
   ```

   For redundancy, the parameters **kdc**, **admin_server**, and **kpasswd_server** can be added multiple times using different KKDCP servers.

2. On IdM clients, restart the **sssd** service to make the changes take effect:

   ```
   [root@server ~]# systemctl restart sssd
   ```

3. To use Kerberos-aware **rsh** and **rlogin** services, install the **rsh** package.

4. Before a workstation can use Kerberos to authenticate users who connect using **ssh**, **rsh**, or **rlogin**, it must have its own host principal in the Kerberos database. The **sshd**, **kshd**, and **klogind** server programs all need access to the keys for the host service's principal.

   1. Using **kadmin**, add a host principal for the workstation on the KDC. The instance in this case is the host name of the workstation. Use the **-randkey** option for the **kadmin**'s **addprinc** command to create the principal and assign it a random key:

      ```
      addprinc -randkey host/server.example.com
      ```

   2. The keys can be extracted for the workstation by running **kadmin** *on the workstation itself* and using the **ktadd** command.

      ```
      ktadd -k /etc/krb5.keytab host/server.example.com
      ```

5. To use other Kerberos-aware network services, install the krb5-server package and start the services. The Kerberos-aware services are listed in Table 11.3, “Common Kerberos-aware Services”.

Table 11.3. Common Kerberos-aware Services

Service Name	Usage Information
ssh	OpenSSH uses GSS-API to authenticate users to servers if the client's and server's configuration both have **GSSAPIAuthentication** enabled. If the client also has **GSSAPIDelegateCredentials** enabled, the user's credentials are made available on the remote system. OpenSSH also contains the **sftp** tool, which provides an FTP-like interface to SFTP servers and can use GSS-API.
IMAP	The **cyrus-imap** package uses Kerberos 5 if it also has the **cyrus-sasl-gssapi** package installed. The **cyrus-sasl-gssapi** package contains the Cyrus SASL plugins which support GSS-API authentication. Cyrus IMAP functions properly with Kerberos as long as the **cyrus** user is able to find the proper key in **/etc/krb5.keytab**, and the root for the principal is set to **imap** (created with **kadmin**). An alternative to **cyrus-imap** can be found in the **dovecot** package, which is also included in Red Hat Enterprise Linux. This package contains an IMAP server but does not, to date, support GSS-API and Kerberos.

11.4. SETTING UP A KERBEROS CLIENT FOR SMART CARDS

Smart cards can be used with Kerberos, but it requires additional configuration to recognize the X.509 (SSL) user certificates on the smart cards:

1. Install the required PKI/OpenSSL package, along with the other client packages:

```
[root@server ~]# yum install krb5-pkinit
[root@server ~]# yum install krb5-workstation krb5-libs
```

2. Edit the **/etc/krb5.conf** configuration file to add a parameter for the public key infrastructure (PKI) to the **[realms]** section of the configuration. The ***pkinit_anchors*** parameter sets the location of the CA certificate bundle file.

```
[realms]
  EXAMPLE.COM = {
    kdc = kdc.example.com.:88
    admin_server = kdc.example.com
    default_domain = example.com
    ...
    pkinit_anchors = FILE:/usr/local/example.com.crt
  }
```

3. Add the PKI module information to the PAM configuration for both smart card authentication (**/etc/pam.d/smartcard-auth**) and system authentication (**/etc/pam.d/system-auth**). The line to be added to both files is as follows:

```
auth    optional    pam_krb5.so use_first_pass no_subsequent_prompt
preauth_options=X509_user_identity=PKCS11:/usr/lib64/pkcs11/opensc-
pkcs11.so
```

If the OpenSC module does not work as expected, use the module from the coolkey package: **/usr/lib64/pkcs11/libcoolkeypk11.so**. In this case, consider contacting Red Hat Technical Support or filing a Bugzilla report about the problem.

11.5. SETTING UP CROSS-REALM KERBEROS TRUSTS

The Kerberos V5 realm is a set of Kerberos principals defined in the Kerberos database on all connected masters and slaves. You must configure cross-realm Kerberos trust if you want principals from different realms to communicate with each other.

A lot of Linux environments, as well as mixed environments, will already have a Kerberos realm deployed for single sign-on, application authentication, and user management. That makes Kerberos a potentially common integration path for different domains and mixed system (such as Windows and Linux) environments, particularly if the Linux environment is not using a more structured domain configuration like Identity Management.

11.5.1. A Trust Relationship

A *trust* means that the users within one realm are trusted to access the resources in another domain *as if they belonged to that realm*. This is done by creating a shared key for a single principal that is held in common by both domains.

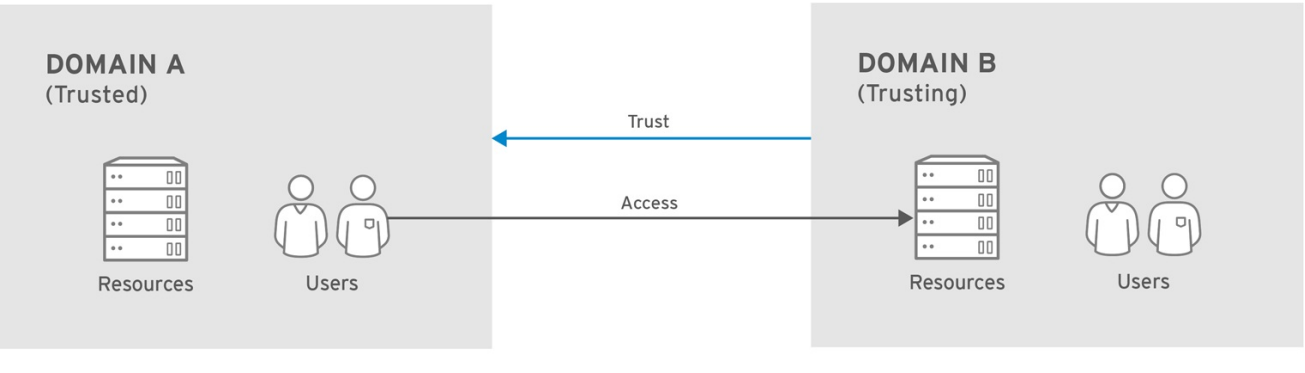


Figure 11.2. Basic Trust

In Figure 11.2, “Basic Trust”, the shared principal would belong to Domain B (**krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM**). When that principal is also added to Domain A, then the clients in Domain A can access the resources in Domain B. The configured principal exists in both realms. That shared principal has three characteristics:

- It exists in both realms.
- When a key is created, the same password is used in both realms.

- The key has the same key version number (**kvno**).

A cross-realm trust is unidirectional by default. This trust is not automatically reciprocated so that the **`B.EXAMPLE.COM`** realm are trusted to authenticate to services in the **`A.EXAMPLE.COM`** realm. To establish trust in the other direction, both realms would need to share keys for the **`krbtgt/A.EXAMPLE.COM@B.EXAMPLE.COM`** service — an entry with a reverse mapping from the previous example.

A realm can have multiple trusts, both realms that it trusts and realms it is trusted by. With Kerberos trusts, the trust can flow in a chain. If Realm A trusts Realm B and Realm B trusts Realm C, Realm A implicitly trusts Realm C, as well. The trust flows along realms; this is a *transitive* trust.

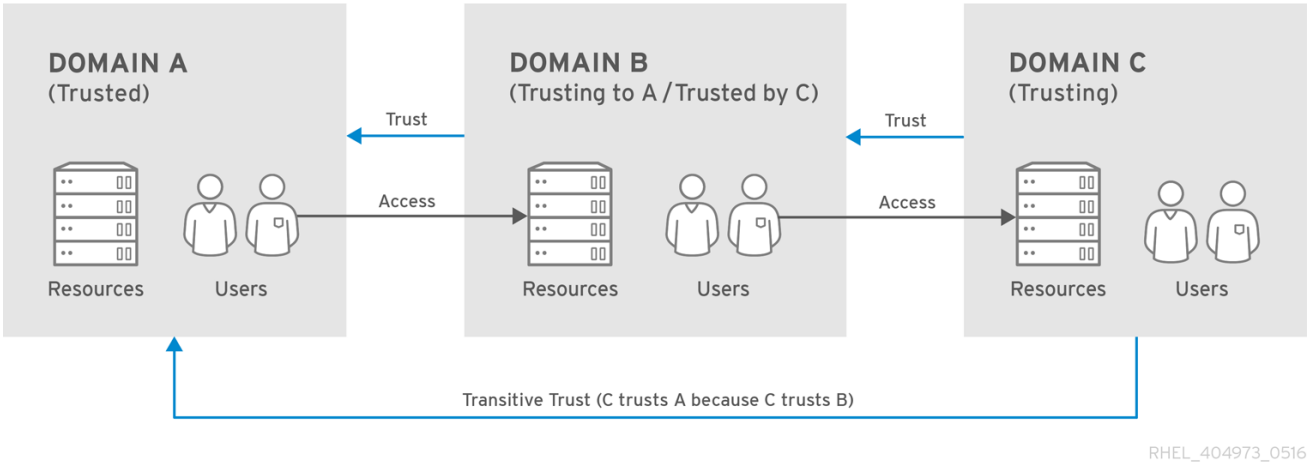


Figure 11.3. Transitive Trust

The direction of a transitive trust is the *trust flow*. The trust flow has to be defined, first by recognizing to what realm a service belongs and then by identifying what realms a client must contact to access that service.

A Kerberos principal name is structured in the format *service/hostname@REALM*. The *service* is generally a protocol, such as LDAP, IMAP, HTTP, or host. The *hostname* is the fully-qualified domain name of the host system, and the *REALM* is the Kerberos realm to which it belongs. Kerberos clients typically use the host name or DNS domain name for Kerberos realm mapping. This mapping can be explicit or implicit. Explicit mapping uses the **`[domain_realm]`** section of the **`/etc/krb5.conf`** file. With implicit mapping, the domain name is converted to upper case; the converted name is then assumed to be the Kerberos realm to search.

When traversing a trust, Kerberos assumes that each realm is structured like a hierarchical DNS domain, with a root domain and subdomains. This means that the trust flows up to a shared root. Each step, or *hop*, has a shared key. In Figure 11.4, “Trusts in the Same Domain”, SALES.EXAMPLE.COM shares a key with EXAMPLE.COM, and EXAMPLE.COM shares a key with EVERYWHERE.EXAMPLE.COM.

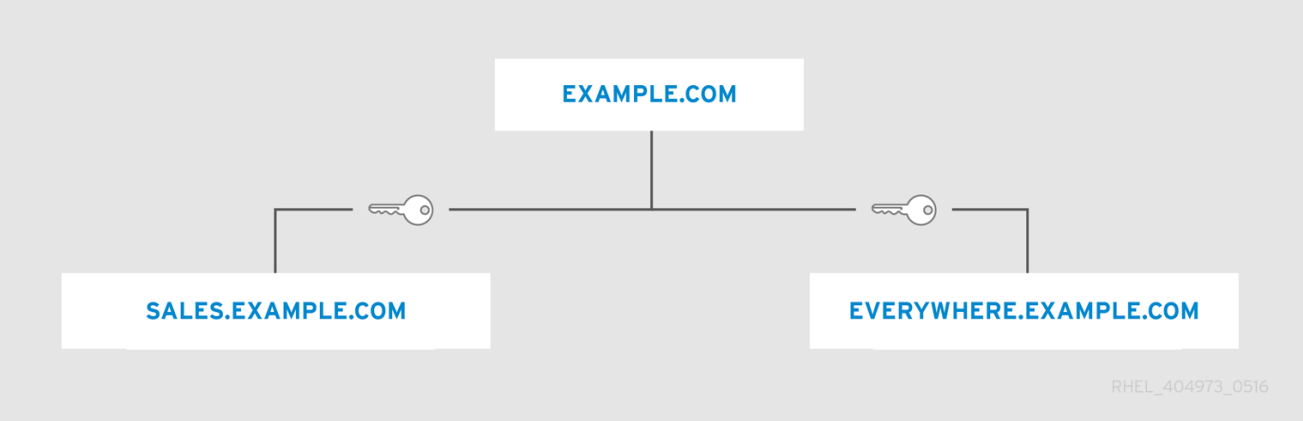


Figure 11.4. Trusts in the Same Domain

The client treats the realm name as a DNS name, and it determines its trust path by stripping off elements of its own realm name until it reaches the root name. It then begins prepending names until it reaches the service's realm.

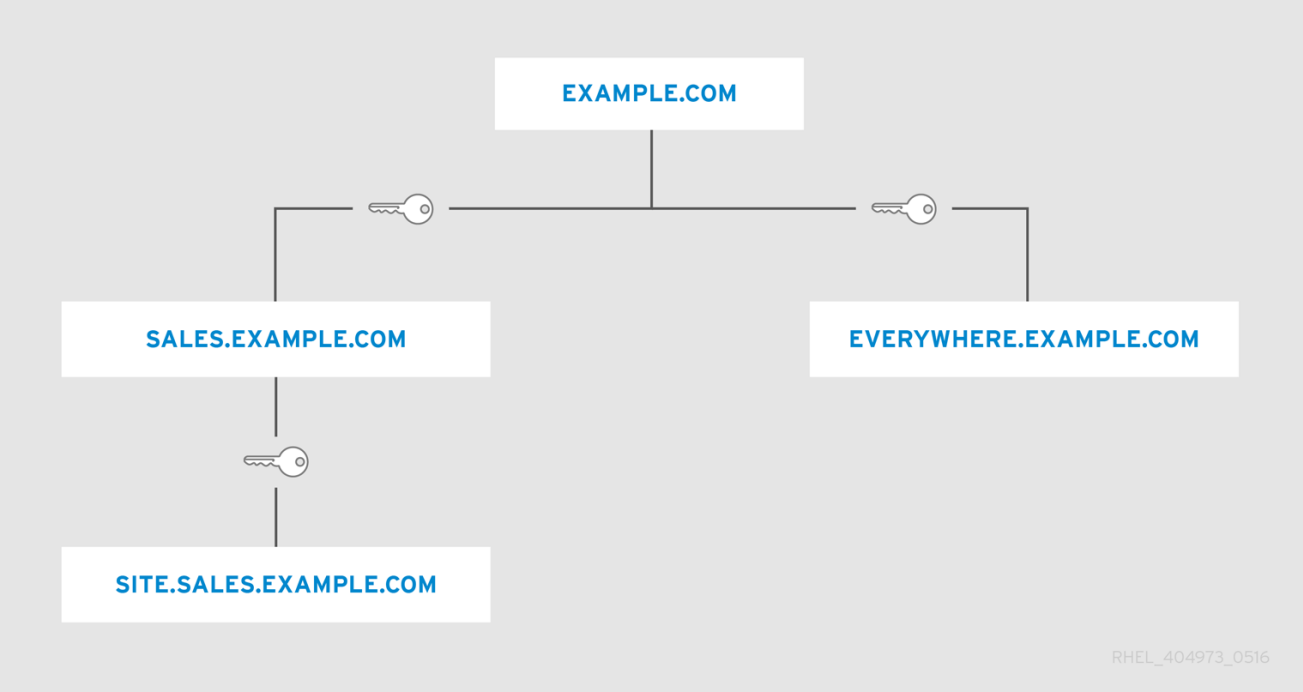


Figure 11.5. Child/Parent Trusts in the Same Domain

This is a nature of trusts being transitive. SITE.SALES.EXAMPLE.COM only has a single shared key, with SALES.EXAMPLE.COM. But because of a series of small trusts, there is a large trust flow that allows trust to go from SITE.SALES.EXAMPLE.COM to EVERYWHERE.EXAMPLE.COM.

That trust flow can even go between completely different domains by creating a shared key at the domain level, where the sites share no common suffix.

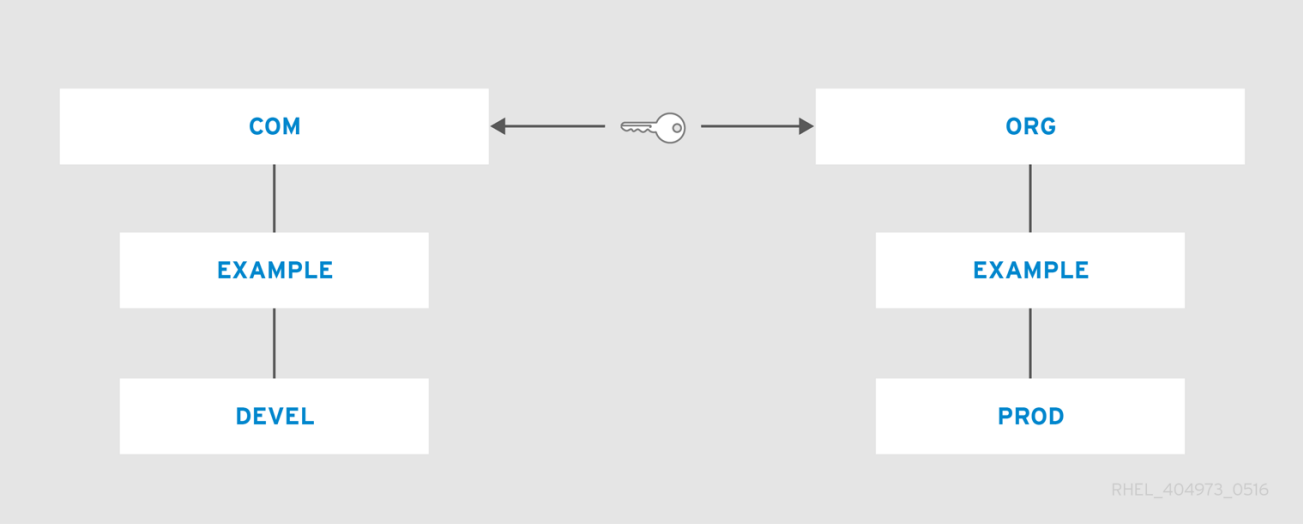


Figure 11.6. Trusts in Different Domains

The `[capaths]` section

It is also possible to reduce the number of hops and represent very complex trust flows by explicitly defining the flow. The **`[capaths]`** section of the **`/etc/krb5.conf`** file defines the trust flow between different realms.

The format of the **`[capaths]`** section is relatively straightforward: there is a main entry for each realm where a client has a principal, and then inside each realm section is a list of intermediate realms from which the client must obtain credentials.

For example, **`[capaths]`** can be used to specify the following process for obtaining credentials:

1. With credentials from Realm A, the client obtains a **`krbtgt/A@A`** ticket from the KDC of Realm A. Using this ticket, the client then asks for the **`krbtgt/B@A`** ticket.

   The **`krbtgt/B@A`** ticket issued by the KDC of Realm A is a *cross-realm ticket granting ticket*. It allows the client to ask the KDC of Realm B for a ticket to a service principal of Realm B.

2. With the **`krbtgt/B@A`** ticket, the client asks for the **`krbtgt/C@B`** cross-realm ticket.

3. With the **`krbtgt/C@B`** ticket issued by the KDC of Realm B, the client asks for the **`krbtgt/D@C`** cross-realm ticket.

4. With the **`krbtgt/D@C`** ticket issued by the KDC of Realm C, the client asks the KDC of Realm D for a ticket to a service principal in Realm D.

After this, the credentials cache contains tickets for **`krbtgt/A@A`**, **`krbtgt/B@A`**, **`krbtgt/C@B`**, **`krbtgt/D@C`**, and **`service/hostname@D`**. To obtain the **`service/hostname@D`** ticket, it was required to obtain the three intermediate cross-realm tickets.

For more information on the **`[capaths]`** section, including examples of the **`[capaths]`** configuration, see the krb5.conf(5) man page.

11.5.2. Setting up a Realm Trust

In this example, the Kerberos realms are **`A.EXAMPLE.COM`** and **`B.EXAMPLE.COM`**.

Create the entry for the shared principal for the *B* realm in the *A* realm, using **`kadmin`**.

-

```
[root@server ~]# kadmin -r A.EXAMPLE.COM
kadmin: add_principal krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM
Enter password for principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM":
Re-enter password for principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM":
Principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM" created.
quit
```

That means that the A realm will trust the B principal.

IMPORTANT

It is recommended that you choose very strong passwords for cross-realm principals. Unlike many other passwords, for which the user can be prompted as often as several times a day, the system will not request the password for cross-realm principal frequently from you, which is why it does not need to be easy to memorize.

To create a bidirectional trust, then create principals going the reverse way. Create a principal for the *A* realm in the *B* realm, using **kadmin**.

```
[root@server ~]# kadmin -r B.EXAMPLE.COM
kadmin: add_principal krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM
Enter password for principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM":
Re-enter password for principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM":
Principal "krbtgt/B.EXAMPLE.COM@A.EXAMPLE.COM" created.
quit
```

Use the **get_principal** command to verify that both entries have matching key version numbers (**kvno** values) and encryption types.

IMPORTANT

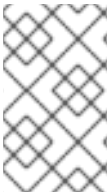
A common, but incorrect, situation is for administrators to try to use the **add_principal** command's **-randkey** option to assign a random key instead of a password, dump the new entry from the database of the first realm, and import it into the second. This will not work unless the master keys for the realm databases are identical, as the keys contained in a database dump are themselves encrypted using the master key.

[3] A system where both the client and the server share a common key that is used to encrypt and decrypt network communication.

CHAPTER 12. WORKING WITH CERTMONGER

Part of managing machine authentication is managing machine certificates. The **certmonger** service manages certificate life cycle for applications and, if properly configured, can work together with a certificate authority (CA) to renew certificates.

The **certmonger** daemon and its command-line clients simplify the process of generating public/private key pairs, creating certificate requests, and submitting requests to the CA for signing. The **certmonger** daemon monitors certificates for expiration and can renew certificates that are about to expire. The certificates that **certmonger** monitors are tracked in files stored in a configurable directory. The default location is **/var/lib/certmonger/requests**.



NOTE

The **certmonger** daemon cannot revoke certificates. A certificate can only be revoked by a relevant Certificate Authority, which needs to invalidate the certificate and update its Certificate Revocation List.

12.1. CERTMONGER AND CERTIFICATE AUTHORITIES

By default, **certmonger** can automatically obtain three kinds of certificates that differ in what authority source the certificate employs:

- Self-signed certificate

  Generating a self-signed certificate does not involve any CA, because each certificate is signed using the certificate's own key. The software that is verifying a self-signed certificate needs to be instructed to trust that certificates directly in order to verify it.

  To obtain a self-signed certificate, run the **selfsign-getcert** command.

- Certificate from the Dogtag Certificate System CA as part of Red Hat Enterprise Linux IdM

  To obtain a certificate using an IdM server, run the **ipa-getcert** command

- Certificate signed by a local CA present on the system

  The software that is verifying a certificate signed by a local signer needs to be instructed to trust certificates from this local signer in order to verify them.

  To obtain a locally-signed certificate, run the **local-getcert** command.

Other CAs can also use **certmonger** to manage certificates, but support must be added to **certmonger** by creating special *CA helpers*. For more information on how to create CA helpers, see the **certmonger** project documentation at https://pagure.io/certmonger/blob/master/f/doc/submit.txt.

12.2. REQUESTING A SELF-SIGNED CERTIFICATE WITH CERTMONGER

To request a certificate with **certmonger**, use the **getcert request** utility.

Certificates and keys are stored locally in plain text files with the **.pem** extension or in an NSS database, identified by the certificate nickname. When requesting a certificate, then, the request should identify the location where the certificate will be stored and the nickname of the certificate. For example:

```
[root@server ~]# selfsign-getcert request -d /etc/pki/nssdb -n Server-Cert
```

The **/etc/pki/nssdb** file is the global NSS database, and **Server-Cert** is the nickname of this certificate. The certificate nickname must be unique within this database.

The options you can provide with the command to generate a certificate vary depending on what kind of certificate you are requesting and the required configuration for the final certificate, as well as other settings:

- **-r** automatically renews the certificate when its expiration date is close if the key pair already exists. This option is used by default.
- **-f** stores the certificate in the given file.
- **-k** either stores the key in the given file or, if the key file already exists, uses the key in the file.
- **-K** gives the Kerberos principal name of the service that will be using the certificate; **-K** is required when requesting a certificate from an IdM server and optional when requesting a self-signed or locally-signed certificate
- **-N** gives the subject name.
- **-D** requests a DNS domain name to be included in the certificate as a **subjectAltName** value.
- **-U** sets the extended key usage flag.
- **-A** requests an IP address to be included in the certificate as a **subjectAltName** value.
- **-I** sets a name for the task. **certmonger** uses this name to refer to the combination of storage locations and request options, and it is also displayed in the output of the **getcert list** command. If you do not specify this option, **certmonger** assigns an automatically-generated name for the task.

A real CA, such as the one in IdM, can ignore anything that you specify in the signing request using the **-K**, **-N**, **-D**, **-U**, and **-A** options according to the CA's own policies. For example, IdM requires that **-K** and **-N** agree with the local host name. Certificates generated using the **selfsign-getcert** and **local-getcert** commands, on the other hand, agree with the options that you specify because these commands do not enforce any policy.

Example 12.1. Using certmonger for a Service

```
[root@server ~]# selfsign-getcert request -f
/etc/httpd/conf/ssl.crt/server.crt -k /etc/httpd/conf/ssl.key/server.key
-N CN=`hostname --fqdn` -D `hostname` -U id-kp-serverAuth
```

12.3. REQUESTING A CA-SIGNED CERTIFICATE THROUGH SCEP

The Simple Certificate Enrollment Protocol (SCEP) automates and simplifies the process of certificate management with the CA. It lets a client request and retrieve a certificate over HTTP directly from the

CA's SCEP service. This process is secured by a one-time PIN that is usually valid only for a limited time.

The following example adds a SCEP CA configuration to **certmonger**, requests a new certificate, and adds it to the local NSS database.

1. Add the CA configuration to **certmonger**:

```
[root@server ~]# getcert add-scep-ca -c CA_Name -u SCEP_URL
```

   - **-c**: Mandatory nickname for the CA configuration. The same value can later be passed to other **getcert** commands.
   - **-u**: URL to the server's SCEP interface.
   - Mandatory parameter when using an HTTPS URL:

     -R *CA_Filename*: Location of the PEM-formatted copy of the SCEP server's CA certificate, used for the HTTPS encryption.

2. Verify that the CA configuration has been successfully added:

```
[root@server ~]# getcert list-cas -c CA_Name
CA 'CA_Name':
       is-default: no
       ca-type: EXTERNAL
       helper-location: /usr/libexec/certmonger/scep-submit -u
http://SCEP_server_enrollment_interface_URL
       SCEP CA certificate thumbprint (MD5): A67C2D4B 771AC186
FCCA654A 5E55AAF7
       SCEP CA certificate thumbprint (SHA1): FBFF096C 6455E8E9
BD55F4A5 5787C43F 1F512279
```

   The CA configuration was successfully added, when the CA certificate thumbprints were retrieved over SCEP and shown in the command's output. When accessing the server over unencrypted HTTP, manually compare the thumbprints with the ones displayed at the SCEP server to prevent a Man-in-the-middle attack.

3. Request a certificate from the CA:

```
[root@server ~]# getcert request -I Task_Name -c CA_Name -d
/etc/pki/nssdb -n Certificate_Name -N cn="Subject Name" -L one-
time_PIN
```

   - **-I**: Name of the task. The same value can later be passed to the**getcert list** command.
   - **-c**: CA configuration to submit the request to.
   - **-d**: Directory with the NSS database to store the certificate and key.
   - **-n**: Nickname of the certificate, used in the NSS database.
   - **-N**: Subject name in the CSR.

- **-L**: Time-limited one-time PIN issued by the CA.

4. Right after submitting the request, you can verify that a certificate was issued and correctly stored in the local database:

```
[root@server ~]# getcert list -I TaskName
 Request ID 'Task_Name':
        status: MONITORING
        stuck: no
        key pair storage:
type=NSSDB,location='/etc/pki/nssdb',nickname='TestCert',token='NSS
Certificate DB'
        certificate:
type=NSSDB,location='/etc/pki/nssdb',nickname='TestCert',token='NSS
Certificate DB'
        signing request thumbprint (MD5): 503A8EDD DE2BE17E 5BAA3A57
D68C9C1B
        signing request thumbprint (SHA1): B411ECE4 D45B883A
75A6F14D 7E3037F1 D53625F4
        CA: AD-Name
        issuer: CN=windows-CA,DC=ad,DC=example,DC=com
        subject: CN=Test Certificate
        expires: 2018-05-06 10:28:06 UTC
        key usage: digitalSignature,keyEncipherment
        eku: iso.org.dod.internet.security.mechanisms.8.2.2
        certificate template/profile: IPSECIntermediateOffline
        pre-save command:
        post-save command:
        track: yes
 auto-renew: yes
```

The status **MONITORING** signifies a successful retrieval of the issued certificate. The **getcert-list(1)** man page lists other possible states and their meanings.

12.4. STORING CERTIFICATES IN NSS DATABASES

By default, **certmonger** uses **.pem** files to store the key and the certificate. To store the key and the certificate in an NSS database, specify the **-d** and **-n** with the command you use for requesting the certificate.

- **-d** sets the security database location
- **-n** gives the certificate nickname which is used for the certificate in the NSS database

NOTE

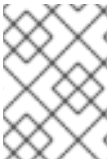
The **-d** and **-n** options are used instead of the **-f** and **-k** options that give the **.pem** file.

For example:

```
[root@server ~]# selfsign-getcert request -d /export/alias -n ServerCert
...
```

Requesting a certificate using **ipa-getcert** and **local-getcert** allows you to specify another two options:

- **-F** gives the file where the certificate of the CA is to be stored.
- **-a** gives the location of the NSS database where the certificate of the CA is to be stored.



NOTE

If you request a certificate using **selfsign-getcert**, there is no need to specify the **-F** and **-a** options because generating a self-signed certificate does not involve any CA.

Supplying the **-F** option, the **-a** option, or both with **local-getcert** allows you to obtain a copy of the CA certificate that is required in order to verify a certificate issued by the local signer. For example:

```
[root@server ~]# local-getcert request -F /etc/httpd/conf/ssl.crt/ca.crt -
n ServerCert -f /etc/httpd/conf/ssl.crt/server.crt -k
/etc/httpd/conf/ssl.key/server.key
```

12.5. TRACKING CERTIFICATES WITH CERTMONGER

certmonger can monitor expiration date of a certificate and automatically renew the certificate at the end of its validity period. To track a certificate in this way, run the **getcert start-tracking** command.

NOTE

It is not required that you run **getcert start-tracking** after running **getcert request**, because the **getcert request** command by default automatically tracks and renews the requested certificate. The **getcert start-tracking** command is intended for situations when you have already obtained the key and certificate through some other process, and therefore you have to manually instruct **certmonger** to start the tracking.

The **getcert start-tracking** command takes several options:

- **-r** automatically renews the certificate when its expiration date is close if the key pair already exists. This option is used by default.
- **-I** sets a name for the tracking request. **certmonger** uses this name to refer to the combination of storage locations and request options, and it is also displayed in the output of the **getcert list** command. If you do not specify this option, **certmonger** assigns an automatically generated a name for the task.

```
[root@server ~]# getcert start-tracking -I cert1-tracker -d /export/alias
-n ServerCert
```

To cancel tracking for a certificate, run the **stop-tracking** command.

CHAPTER 13. CONFIGURING APPLICATIONS FOR SINGLE SIGN-ON

Some common applications, such as browsers and email clients, can be configured to use Kerberos tickets, SSL certifications, or tokens as a means of authenticating users.

The precise procedures to configure any application depend on that application itself. The examples in this chapter (Mozilla Thunderbird and Mozilla Firefox) are intended to give you an idea of how to configure a user application to use Kerberos or other credentials.

13.1. CONFIGURING FIREFOX TO USE KERBEROS FOR SINGLE SIGN-ON

Firefox can use Kerberos for single sign-on (SSO) to intranet sites and other protected websites. For Firefox to use Kerberos, it first has to be configured to send Kerberos credentials to the appropriate KDC.

Even after Firefox is configured to pass Kerberos credentials, it still requires a valid Kerberos ticket to use. To generate a Kerberos ticket, use the **kinit** command and supply the user password for the user on the KDC.

```
[jsmith@host ~] $ kinit
Password for jsmith@EXAMPLE.COM:
```

To configure Firefox to use Kerberos for SSO:

1. In the address bar of Firefox, type **about:config** to display the list of current configuration options.
2. In the **Filter** field, type **negotiate** to restrict the list of options.
3. Double-click the **network.negotiate-auth.trusted-uris** entry.
4. Enter the name of the domain against which to authenticate, including the preceding period (.). If you want to add multiple domains, enter them in a comma-separated list.

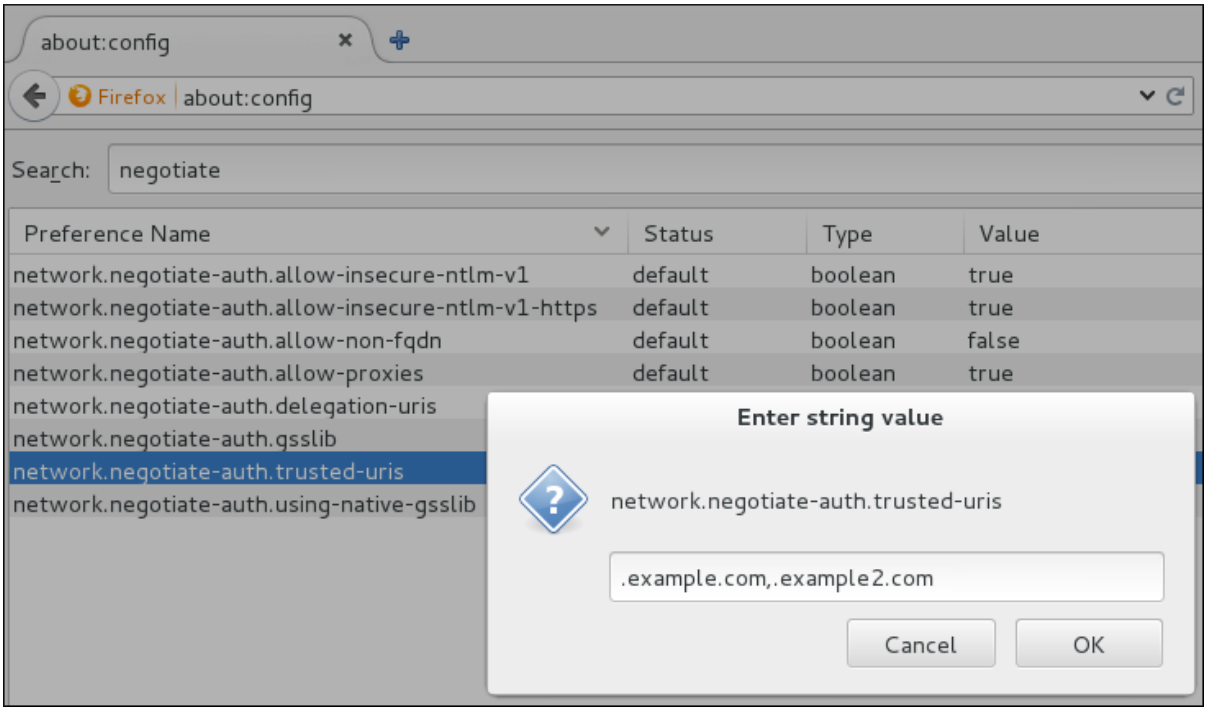


Figure 13.1. Manual Firefox Configuration

IMPORTANT

It is not recommended to configure delegation using the **`network.negotiate-auth.delegation-uris`** entry in the Firefox configuration options because this enables every Kerberos-aware server to act as the user.

NOTE

For information about configuring Firefox to use Kerberos in Identity Management, see the corresponding section in the Linux Domain Identity, Authentication, and Policy Guide.

13.2. CERTIFICATE MANAGEMENT IN FIREFOX

To manage certificates in Firefox, open the **`Certificate Manager`**.

1. In Mozilla Firefox, open the Firefox menu and click **`Preferences`**.

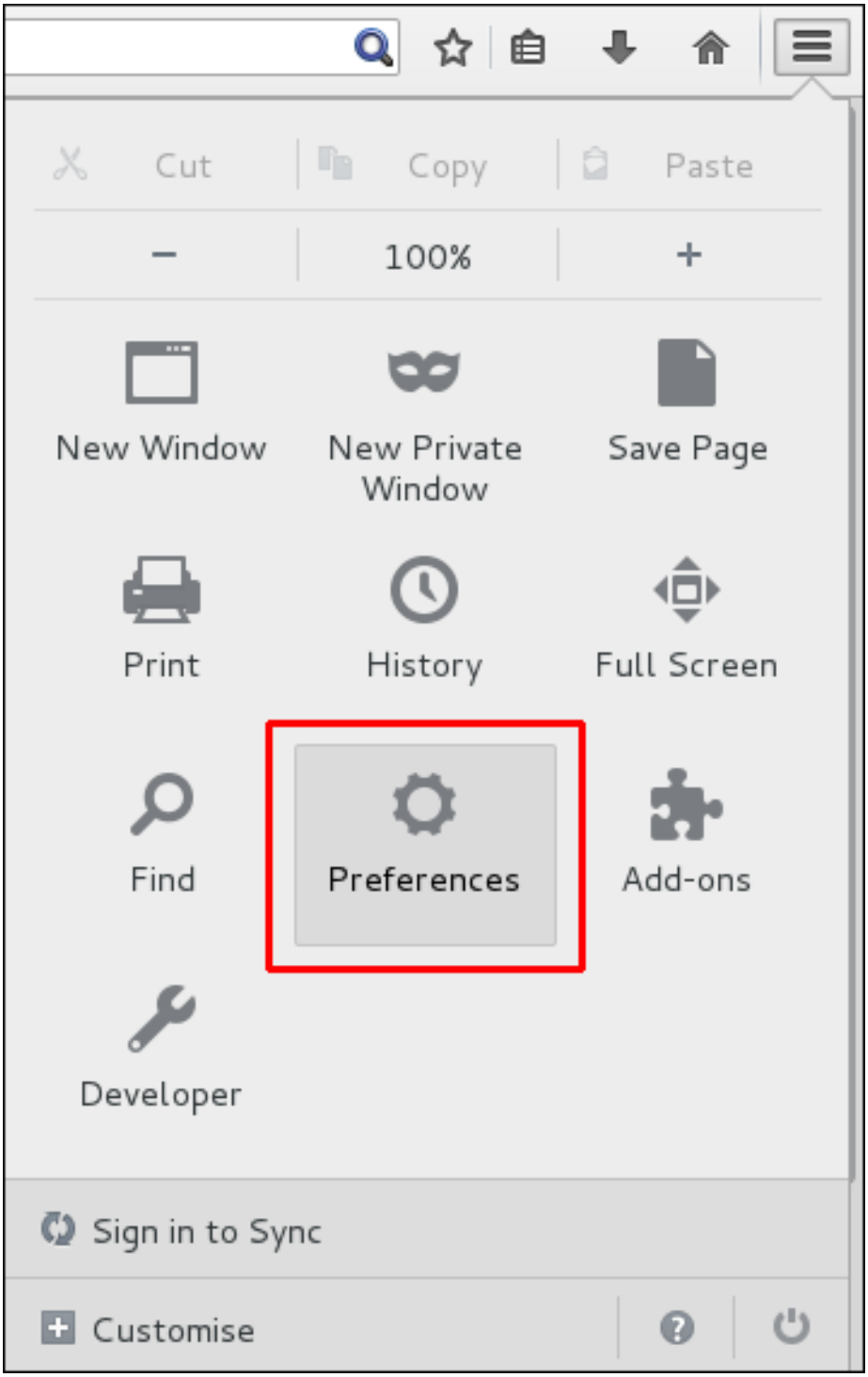


Figure 13.2. Firefox Preferences

2. Open the **Advanced** section and choose the **Certificates** tab.

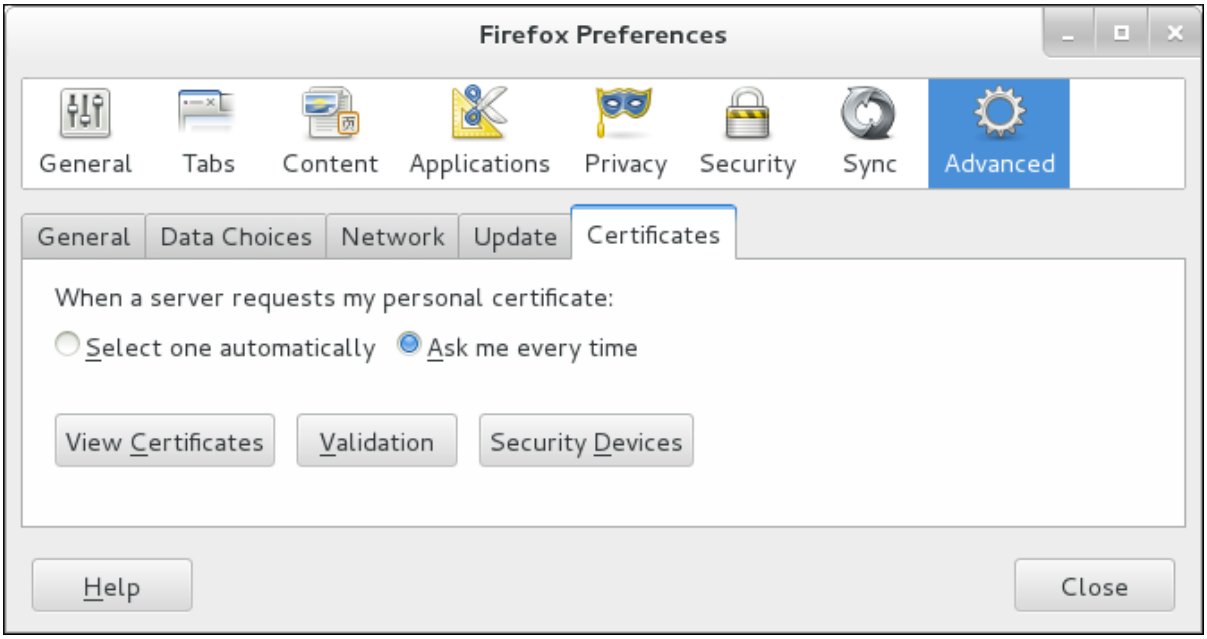


Figure 13.3. Certificates Tab in Firefox

3. Click **View Certificates** to open the **Certificate Manager**.

To import a CA certificate:

1. Download and save the CA certificate to your computer.
2. In the **Certificate Manager**, choose the **Authorities** tab and click **Import**.

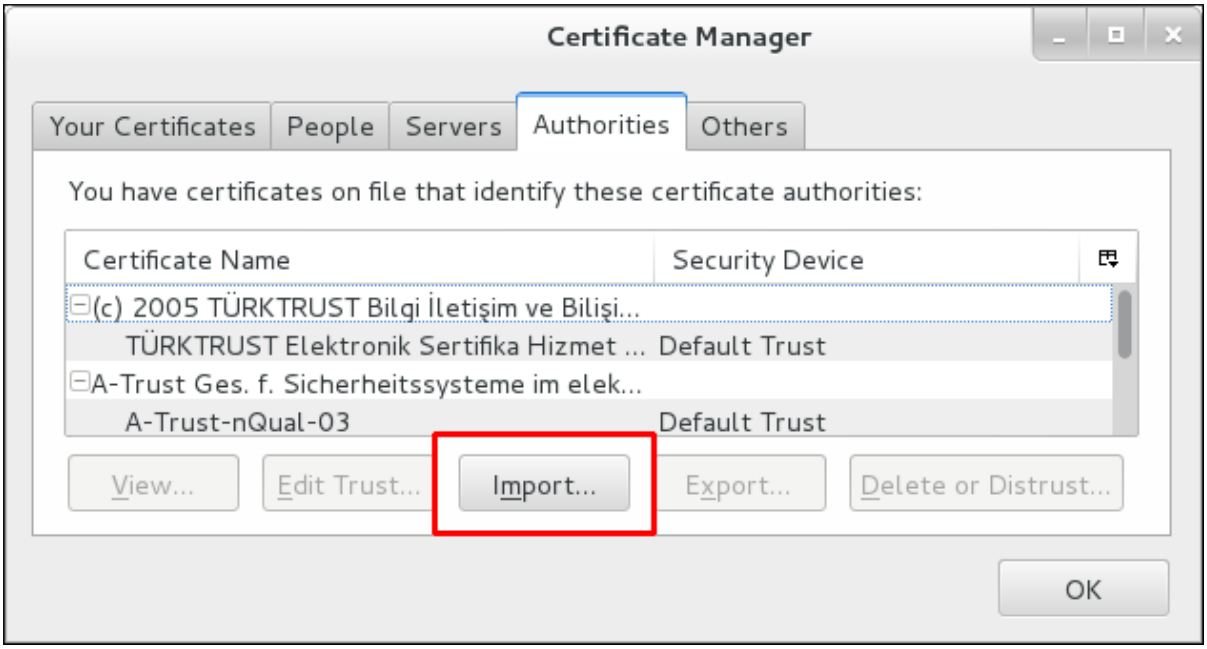


Figure 13.4. Importing the CA Certificate in Firefox

3. Select the downloaded CA certificate.

To set the certificate trust relationships:

1. In the **Certificate Manager**, under the **Authorities** tab, select the appropriate certificate and click **Edit Trust**.

2. Edit the certificate trust settings.

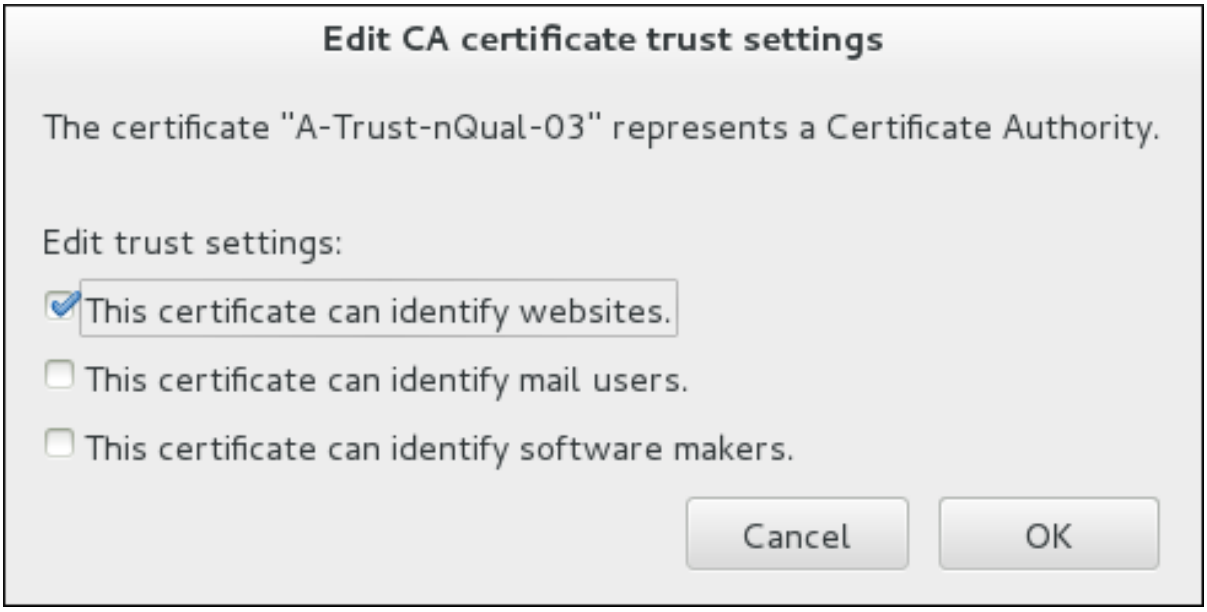


Figure 13.5. Editing the Certificate Trust Settings in Firefox

To use a personal certificate for authentication:

1. In the **`Certificate Manager`**, under the **`Your Certificates`** tab, click **`Import`**.

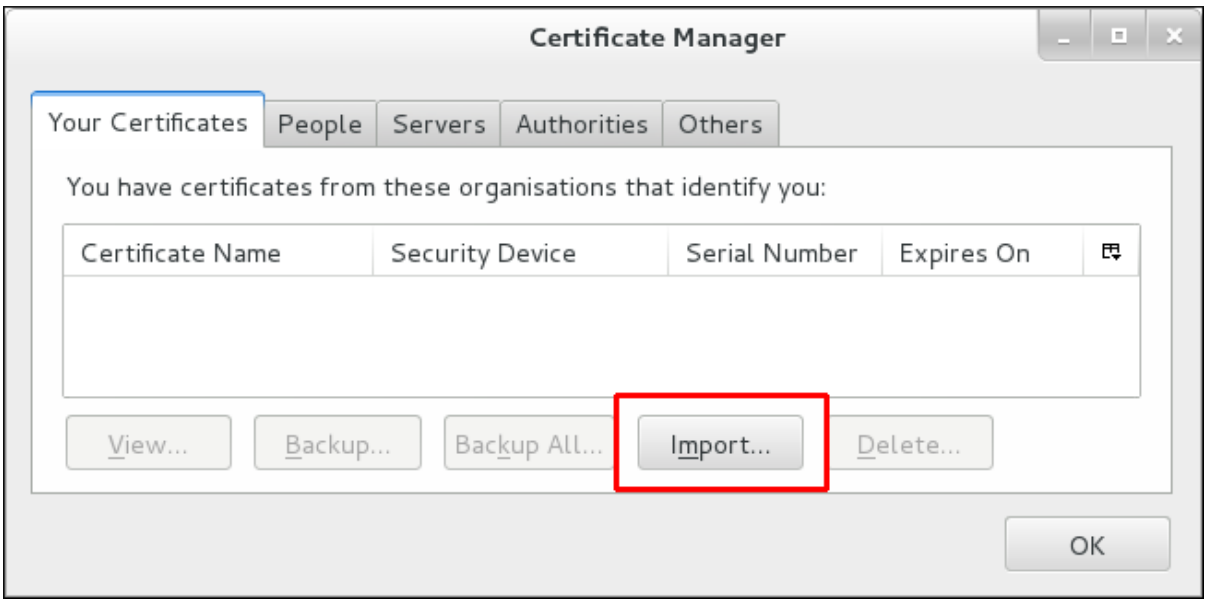


Figure 13.6. Importing a Personal Certificate for Authentication in Firefox

2. Select the required certificate from your computer.

13.3. CERTIFICATE MANAGEMENT IN EMAIL CLIENTS

The following example shows how to manage certificates in the Mozilla Thunderbird email client. It represents a procedure to set up certificates in email clients in general.

1. In Mozilla Thunderbird, open the Thunderbird main menu and select **`Preferences`** → **`Account Settings`**.

2. Select the **Security** item, and click **View Certificates** to open the **Certificate Manager**.

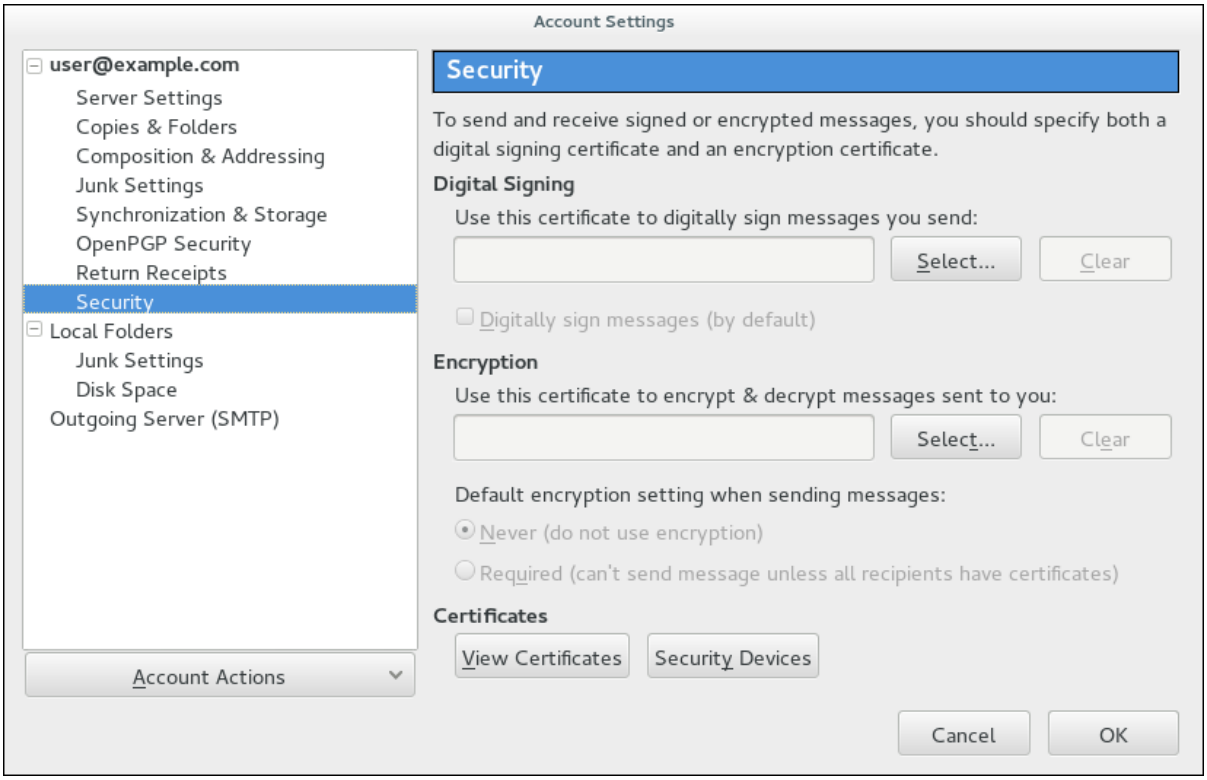


Figure 13.7. Account Settings in Thunderbird

To import a CA certificate:

1. Download and save the CA certificate to your computer.
2. In the **Certificate Manager**, choose the **Authorities** tab and click **Import**.

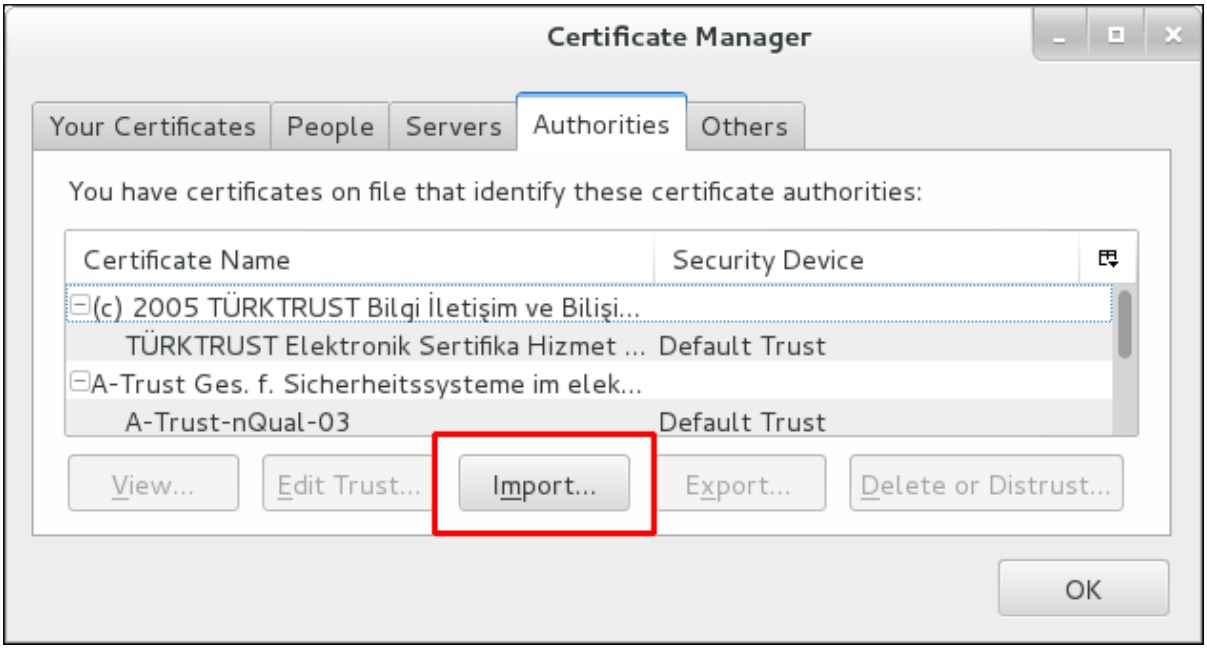


Figure 13.8. Importing the CA Certificate in Thunderbird

3. Select the downloaded CA certificate.

To set the certificate trust relationships:

1. In the **Certificate Manager**, under the **Authorities** tab, select the appropriate certificate and click **Edit Trust**.
2. Edit the certificate trust settings.

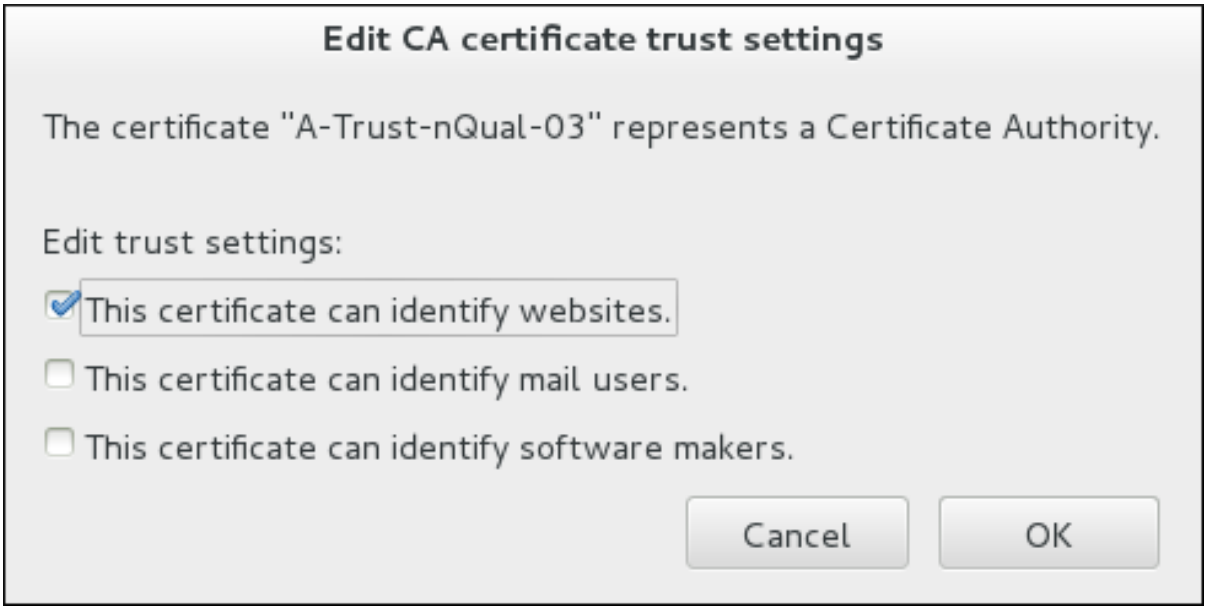


Figure 13.9. Editing the Certificate Trust Settings in Thunderbird

To use a personal certificate for authentication:

1. In the **Certificate Manager**, under the **Your Certificates** tab, click **Import**.

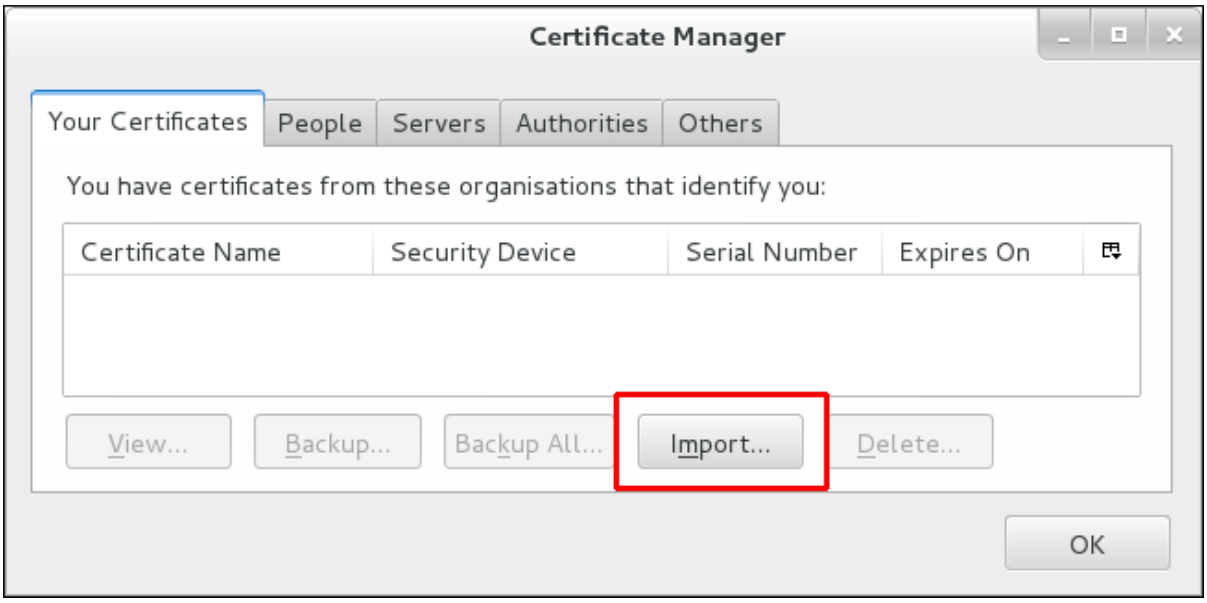


Figure 13.10. Importing a Personal Certificate for Authentication in Thunderbird

2. Select the required certificate from your computer.
3. Close the **Certificate Manager** and return to the **Security** item in **Account Settings**.
4. Under the **Digital Signing** section of the form, click **Select** to choose your personal certificate to use for signing messages.

5. Under **`Encryption`**, click **`Select`** to choose your personal certificate to encrypt and decrypt messages.

APPENDIX A. TROUBLESHOOTING

A.1. TROUBLESHOOTING SSSD

A.1.1. Setting Debug Logs for SSSD Domains

Each domain sets its own debug log level. Increasing the log level can provide more information about problems with SSSD or with the domain configuration.

To change the log level, set the ***debug_level*** parameter for each section in the **sssd.conf** file for which to produce extra logs. For example:

```
[domain/LDAP]
cache_credentials = true
debug_level = 9
```

Table A.1. Debug Log Levels

Level	Description
0	Fatal failures. Anything that would prevent SSSD from starting up or causes it to cease running.
1	Critical failures. An error that does not kill the SSSD, but one that indicates that at least one major feature is not going to work properly.
2	Serious failures. An error announcing that a particular request or operation has failed.
3	Minor failures. These are the errors that would percolate down to cause the operation failure of 2.
4	Configuration settings.
5	Function data.
6	Trace messages for operation functions.
7	Trace messages for internal control functions.
8	Contents of function-internal variables that may be interesting.

Level	Description
9	Extremely low-level tracing information.

To change the debug level while SSSD is running, use the **sss_debuglevel** utility, which is part of the **sssd-tools** package. For more information about how it works, see the sss_debuglevel man page.

A.1.2. Checking SSSD Log Files

SSSD uses a number of log files to report information about its operation, located in the **/var/log/sssd/** directory. SSSD produces a log file for each domain, as well as an **sssd_pam.log** and an **sssd_nss.log** file.

Additionally, the **/var/log/secure** file logs authentication failures and the reason for the failure.

A.1.3. Problems with SSSD Configuration

Q: SSSD fails to start

A: SSSD requires that the configuration file be properly set up, with all the required entries, before the daemon will start.

SSSD requires at least one properly configured domain before the service will start. Without a domain, attempting to start SSSD returns an error that no domains are configured:

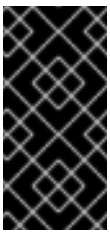
```
# sssd -d4 -i

[sssd] [ldb] (3): server_sort:Unable to register control with
rootdse!
[sssd] [confdb_get_domains] (0): No domains configured, fatal
error!
[sssd] [get_monitor_config] (0): No domains configured.
```

Edit the **/etc/sssd/sssd.conf** file and create at least one domain.

SSSD also requires at least one available service provider before it will start. If the problem is with the service provider configuration, the error message indicates that there are no services configured:

```
[sssd] [get_monitor_config] (0): No services configured!
```

Edit the **/etc/sssd/sssd.conf** file and configure at least one service provider.



IMPORTANT

SSSD requires that service providers be configured as a comma-separated list in a single ***services*** entry in the **/etc/sssd/sssd.conf** file. If services are listed in multiple entries, only the last entry is recognized by SSSD.

SSSD also requires the ownership and permissions of the **/etc/sssd/sssd.conf** to be set correctly. If the ownership or permissions are set incorrectly, attempt to start SSSD returns these error messages:

```
[sssd] [confdb_ldif_from_ini_file] (0x0020): Permission check on
config file failed.
[sssd] [confdb_init_db] (0x0020): Cannot convert INI to LDIF
[1]: [Operation not permitted]
[sssd] [confdb_setup] (0x0010): ConfDB initialization has failed
[1]: Operation not permitted
[sssd] [load_configuration] (0x0010): Unable to setup ConfDB
[1]: Operation not permitted
[sssd] [main] (0x0020): Cannot read config file
/etc/sssd/sssd.conf. Please check that the file is accessible
only by the owner and owned by root.root.
```

Set the correct ownership and permissions of the **/etc/sssd/sssd.conf** file:

```
# chmod 600 /etc/sssd/sssd.conf
# chown root:root /etc/sssd/sssd.conf
```

Q: I do not see any groups with `id` or group members with `getent group`.

A: This may be due to an incorrect **ldap_schema** setting in the **[domain/DOMAINNAME]** section of **sssd.conf**.

SSSD supports RFC 2307 and RFC 2307bis schema types. By default, SSSD uses the more common RFC 2307 schema.

The difference between RFC 2307 and RFC 2307bis is the way which group membership is stored in the LDAP server. In an RFC 2307 server, group members are stored as the multi-valued ***memberuid*** attribute, which contains the name of the users that are members. In an RFC2307bis server, group members are stored as the multi-valued ***member*** or ***uniqueMember*** attribute which contains the DN of the user or group that is a member of this group. RFC2307bis allows nested groups to be maintained as well.

If group lookups are not returning any information:

1. Set **ldap_schema** to **rfc2307bis**.
2. Delete **/var/lib/sss/db/cache_DOMAINNAME.ldb**.
3. Restarting SSSD.

If that does not work, add this line to **sssd.conf**:

```
ldap_group_name = uniqueMember
```

Then delete the cache and restart SSSD again.

Q: Authentication fails against LDAP.

A: To perform authentication, SSSD requires that the communication channel be encrypted. This means that if **sssd.conf** is configured to connect over a standard protocol (**ldap://**), it attempts to encrypt the communication channel with Start TLS. If **sssd.conf** is configured to connect over a secure protocol (**ldaps://**), then SSSD uses SSL.

This means that the LDAP server must be configured to run in SSL or TLS. TLS must be enabled for the standard LDAP port (389) or SSL enabled on the secure LDAPS port (636). With either SSL or TLS, the LDAP server must also be configured with a valid certificate trust.

An invalid certificate trust is one of the most common issues with authenticating against LDAP. If the client does not have proper trust of the LDAP server certificate, it is unable to validate the connection, and SSSD refuses to send the password. The LDAP protocol requires that the password be sent in plain text to the LDAP server. Sending the password in plain text over an unencrypted connection is a security problem.

If the certificate is not trusted, a **syslog** message is written, indicating that TLS encryption could not be started. The certificate configuration can be tested by checking if the LDAP server is accessible apart from SSSD. For example, this tests an anonymous bind over a TLS connection to **test.example.com**:

```
$ ldapsearch -x -ZZ -h test.example.com -b dc=example,dc=com
```

If the certificate trust is not properly configured, the test fails with this error:

```
ldap_start_tls: Connect error (-11) additional info: TLS error -
8179:Unknown code ___f 13
```

To trust the certificate:

1. Obtain a copy of the public CA certificate for the certificate authority used to sign the LDAP server certificate and save it to the local system.
2. Add a line to the **sssd.conf** file that points to the CA certificate on the filesystem.

   ```
   ldap_tls_cacert = /path/to/cacert
   ```

3. If the LDAP server uses a self-signed certificate, remove the **ldap_tls_reqcert** line from the **sssd.conf** file.

   This parameter directs SSSD to trust any certificate issued by the CA certificate, which is a security risk with a self-signed CA certificate.

Q: Connecting to LDAP servers on non-standard ports fail.

A: When running SELinux in enforcing mode, the client's SELinux policy has to be modified to connect to the LDAP server over the non-standard port. For example:

```
# semanage port -a -t ldap_port_t -p tcp 1389
```

Q: NSS fails to return user information

A: This usually means that SSSD cannot connect to the NSS service.

Ensure that the NSS service is running:

```
# service sssd status
Redirecting to /bin/systemctl status sssd.service
  sssd.service - System Security Services Daemon
   Loaded: loaded (/usr/lib/systemd/system/sssd.service;
enabled)
   Active: active (running) since Wed 2015-01-14 10:17:26 CET;
1min 30s ago
   Process: 683 ExecStart=/usr/sbin/sssd -D -f (code=exited,
status=0/SUCCESS)
 Main PID: 745 (sssd)
   CGroup: /system.slice/sssd.service
           ├─745 /usr/sbin/sssd -D -f
     ├─746 /usr/libexec/sssd/sssd_be --domain default --debug-
to-files...
     ├─804 /usr/libexec/sssd/sssd_nss --debug-to-files
     └─805 /usr/libexec/sssd/sssd_pam --debug-to-files
```

NSS service is running when SSSD is in the **`Active: active (running)`** state and when the output includes **`sssd_nss`**.

If NSS is running, make sure that the provider is properly configured in the **`[nss]`** section of the **`/etc/sssd/sssd.conf`** file. Especially check the ***`filter_users`*** and ***`filter_groups`*** attributes.

Make sure that NSS is included in the list of services that SSSD uses.

Check the configuration in the **`/etc/nsswitch.conf`** file. For more information, see the section called “Configure NSS Services to Use SSSD”.

Q: NSS returns incorrect user information

A: If searches are returning the incorrect user information, check that there are not conflicting user names in separate domains. When there are multiple domains, set the ***`use_fully_qualified_domains`*** attribute to **`true`** in the **`/etc/sssd/sssd.conf`** file. This differentiates between different users in different domains with the same name.

Q: Setting the password for the local SSSD user prompts twice for the password

A: When attempting to change a local SSSD user's password, it may prompt for the password twice:

```
[root@clientF11 tmp]# passwd user1000
Changing password for user user1000.
New password:
Retype new password:
New Password:
Reenter new Password:
passwd: all authentication tokens updated successfully.
```

This is the result of an incorrect PAM configuration. Ensure that the ***`use_authtok`*** option is correctly configured in your **`/etc/pam.d/system-auth`** file. For examples of the correct configuration, see Section 7.5.2, “Configuring Services: PAM”.

Q: An Active Directory identity provider is properly configured in my `sssd.conf` file, but SSSD fails to connect to it, with GSS-API errors.

A: SSSD can only connect with an Active Directory provider using its host name. If the host name is not given, the SSSD client cannot resolve the IP address to the host, and authentication fails.

For example, with this configuration:

```
[domain/ADEXAMPLE]
debug_level = 0xFFF0
id_provider = ad
ad_server = 172.16.0.1
ad_domain = example.com
krb5_canonicalize = False
```

The SSSD client returns this GSS-API failure, and the authentication request fails:

```
(Fri Jul 27 18:27:44 2012) [sssd[be[ADTEST]]] [sasl_bind_send]
(0x0020): ldap_sasl_bind failed (-2)[Local error]
(Fri Jul 27 18:27:44 2012) [sssd[be[ADTEST]]] [sasl_bind_send]
(0x0080): Extended failure message: [SASL(-1): generic failure: GSSAPI
Error: Unspecified GSS failure.  Minor code may provide more
information (Cannot determine realm for numeric host address)]
```

To avoid this error, set the ***`ad_server`*** to the name of the Active Directory host, or use the **`_srv_`** keyword to use the DNS service discovery, as described in Section 7.4.3, “Configuring DNS Service Discovery”.

Q: I configured SSSD for central authentication, but now several of my applications (such as Firefox or Adobe) will not start.

A: Even on 64-bit systems, 32-bit applications require a 32-bit version of SSSD client libraries to use to access the password and identity cache. If a 32-bit version of SSSD is not available, but the system is configured to use the SSSD cache, then 32-bit applications can fail to start.

For example, Firefox can fail with permission denied errors:

```
Failed to contact configuration server. See
http://www.gnome.org/projects/gconf/
for information. (Details -  1: IOR file '/tmp/gconfd-
somebody/lock/ior'
not opened successfully, no gconfd located: Permission denied 2: IOR
file '/tmp/gconfd-somebody/lock/ior' not opened successfully, no
gconfd
located: Permission denied)
```

For Adobe Reader, the error shows that the current system user is not recognized:

```
[jsmith@server ~]$ acroread
(acroread:12739): GLib-WARNING **: getpwuid_r(): failed due to unknown
user id (366)
```

Other applications may show similar user or permissions errors.

Q: SSSD is showing an automount location that I removed.

A: The SSSD cache for the automount location persists even if the location is subsequently changed or removed. To update the autofs information in SSSD:

1. Remove the autofs cache, as described in Section A.1.4, “A User Cannot Log In After UID or GID Changed”.
2. Restart SSSD:

```
# systemctl restart sssd
```

A.1.4. A User Cannot Log In After UID or GID Changed

After a user or group ID changed, SSSD prevents a user from logging in.

What this means:
SSSD recognizes users and groups based on user IDs (UID) and group IDs (GID). When the UID or GID of an existing user or group changes, SSSD does not recognize the user or group.

To fix the problem:
Clear the SSSD cache using the **sss_cache** utility:

1. Make sure the sssd-tools is installed.
2. To clear the SSSD cache for a specific user and leave the rest of the cache records intact:

```
# sss_cache --user user_name
```

To clear the cache for an entire domain:

```
# sss_cache --domain domain_name
```

The utility invalidates records in the SSSD cache for a user, group, or domain. After this, SSSD retrieves the records from the identity provider to refresh the cache.

For details on **sss_cache**, see the sss_cache(8) man page.

A.1.5. SSSD Control and Status Utility

SSSD provides the **sssctl** utility to obtain status information, manage data files during troubleshooting, and other SSSD-related tasks.

1. To use **sssctl**, install the sssd-tools package:

```
[root@server ~]# yum install sssd-tools
```

2. Some options of **sssctl** use the SSSD InfoPipe responder. To enable it, add **ifp** to the

services option of your **/etc/sssd/sssd.conf**:

```
[sssd]
services = nss, sudo, pam, ssh, ifp
```

3. Restart SSSD:

```
[root@server ~]# systemctl restart sssd.service
```

A.1.5.1. SSSD Configuration Validation

The **sssctl config-check** command performs a static analysis of the SSSD configuration files. This enables you to validate the **/etc/sssd/sssd.conf** file and **/etc/sssd/conf.d/*** files to receive a report before restarting SSSD.

The command performs the following checks on SSSD configuration files:

Permissions

The owner and group owner must be set to **root:root** and the permissions to **600**.

File names

File names in **/etc/sssd/conf.d/** must use the suffix **.conf** and not start with a period (hidden files).

Typographical errors

Typographical errors are checked in section and option names. Note that values are not checked.

Options

Options must be placed in the correct sections.

To verify the configuration, run:

```
# sssctl config-check
Issues identified by validators: 3
[rule/allowed_sections]: Section [paM] is not allowed. Check for typos.
[rule/allowed_domain_options]: Attribute 'offline_timeoutX' is not allowed
in section 'domain/idm.example.com'. Check for typos.
[rule/allowed_sudo_options]: Attribute 'homedir_substring' is not allowed
in section 'sudo'. Check for typos.

Messages generated during configuration merging: 2
File /etc/sssd/conf.d/wrong-file-permissions.conf did not pass access
check. Skipping.
File configuration.conf.disabled did not match provided patterns.
Skipping.

Used configuration snippet files: 1
/etc/sssd/conf.d/sample.conf
```

A.1.5.2. Domain Information

The domain status displays a list of domains SSSD accesses, and enables you to retrieve their status.

1. List all domains available within SSSD, including trusted domains:

```
[root@server ~]# sssctl domain-list
idm.example.com
ad.example.com
```

2. Retrieve the status of the domain *idm.example.com*:

```
[root@server ~]# sssctl domain-status idm.example.com
Online status: Online
```

A.1.5.3. Cached Entries Information

The **sssctl** command enables you to retrieve information about a cached entry, to investigate and debug cache-related authentication problems.

To query cache information for the user account **admin**, run:

```
[root@server ~]# sssctl user-show admin
Name: admin
Cache entry creation date: 07/10/16 16:09:18
Cache entry last update time: 07/14/16 10:13:32
Cache entry expiration time: 07/14/16 11:43:32
Initgroups expiration time: Expired
Cached in InfoPipe: No
```

To query the cache information for a group or netgroup, use:

```
[root@server ~]# sssctl group-show groupname
[root@server ~]# sssctl netgroup-show netgroupname
```

A.1.5.4. Truncating the Log Files

When you debug a problem, you can use **sssctl logs-remove** to truncate all SSSD log files in the **/var/log/sssd/** directory to capture only newly created entries.

```
[root@server ~]# sssctl logs-remove
Truncating log files...
```

A.1.5.5. Removing the SSSD Cache

To remove the SSSD cache database files, the **sssctl** command provides the **remove-cache** option. Before the databases are removed, the command creates automatically a backup.

Use the following command to back up all local data and remove the SSSD cache:

```
[root@server ~]# sssctl cache-remove
SSSD must not be running. Stop SSSD now? (yes/no) [yes]
Creating backup of local data...
```

```
Removing cache files...
SSSD needs to be running. Start SSSD now? (yes/no) [yes]
```

NOTE

The backup stores only local data, such as local overrides, in the **/var/lib/sss/backup/** directory.

To automatically import the backed-up content, run **sssctl restore-local-data**.

A.1.5.6. Obtaining Information about an LDAP Group Takes Long

Operations that involve looking up information about an LDAP group take very long, especially in case of large groups or nested groups.

What this means:
By default, LDAP group information lookups return all members for the group. For operations that involve large groups or nested groups, returning all members makes the process longer.

To fix the problem:
The membership lists returned in group lookups are not used when evaluating whether a user belongs to a group. To improve performance, especially for identity lookups, disable the group membership lookup:

1. Open the **/etc/sssd/sssd.conf** file.
2. In the **[domain]** section, set the ***ignore_group_members*** option to **true**.

```
[domain/domain_name]
[... file truncated ...]
ignore_group_members = true
```

A.2. TROUBLESHOOTING SUDO WITH SSSD AND SUDO DEBUGGING LOGS

A.2.1. SSSD and sudo Debug Logging

The debug logging feature enables you to log additional information about SSSD and sudo.

The sudo Debug Log File
To enable sudo debugging:

1. Add the following lines to **/etc/sudo.conf**:

```
Debug sudo /var/log/sudo_debug.log all@debug
Debug sudoers.so /var/log/sudo_debug.log all@debug
```

2. Run the **sudo** command as the user you want to debug.

The **/var/log/sudo_debug.log** file is created automatically and provides detailed information to answer questions like:

- What information is available about the user and the environment when running the **sudo** command?

```
sudo[22259] settings: debug_flags=all@debug
sudo[22259] settings: run_shell=true
sudo[22259] settings: progname=sudo
sudo[22259] settings: network_addrs=192.0.2.1/255.255.255.0
fe80::250:56ff:feb9:7d6/ffff:ffff:ffff:ffff::
sudo[22259] user_info: user=user_name
sudo[22259] user_info: pid=22259
sudo[22259] user_info: ppid=22172
sudo[22259] user_info: pgid=22259
sudo[22259] user_info: tcpgid=22259
sudo[22259] user_info: sid=22172
sudo[22259] user_info: uid=10000
sudo[22259] user_info: euid=0
sudo[22259] user_info: gid=554801393
sudo[22259] user_info: egid=554801393
sudo[22259] user_info:
groups=498,6004,6005,7001,106501,554800513,554801107,554801108,55480
1393,554801503,554802131,554802244,554807670
sudo[22259] user_info: cwd=/
sudo[22259] user_info: tty=/dev/pts/1
sudo[22259] user_info: host=client
sudo[22259] user_info: lines=31
sudo[22259] user_info: cols=237
```

- What data sources are used to fetch sudo rules?

```
sudo[22259] <- sudo_parseln @ ./fileops.c:178 := sudoers: files sss
```

- SSSD plug-in starts with this line:

```
sudo[22259] <- sudo_sss_open @ ./sssd.c:305 := 0
```

- How many rules did SSSD return?

```
sudo[22259] Received 3 rule(s)
```

- Does a rule match or not?

```
sudo[22259] sssd/ldap sudoHost 'ALL' ... MATCH!
sudo[22259] <- user_in_group @ ./pwutil.c:1010 := false
```

The SSSD Debug Log Files

To enable SSSD debugging:

1. Add the **debug_level** option to the **[sudo]** and **[domain/*domain_name*]** sections of your **/etc/sssd/sssd.conf** file:

```
[domain/domain_name]
debug_level = 0x3ff0
...
```

```
[sudo]
debug_level = 0x3ff0
```

2. Restart SSSD:

```
# systemctl restart sssd
```

3. Run the **sudo** command to write the debug information to the log files.

The following log files are created:

The domain log file: `/var/log/sssd/sssd_`*`domain_name`*`.log`

This log file helps you to answer questions like:

- How many rules did SSSD return?

```
[sdap_sudo_refresh_load_done] (0x0400): Received 4-rules rules
```

- What sudo rules did SSSD download from the server?

```
[sssd[be[LDAP.PB]]] [sysdb_save_sudorule] (0x0400): Adding sudo
rule demo-name
```

- Are the matching rules stored in the cache?

```
[sdap_sudo_refresh_load_done] (0x0400): Sudoers is successfully
stored in cache
```

- What filter was used to download the rules from the server?

```
[sdap_get_generic_ext_step] (0x0400): calling ldap_search_ext with
[(&(objectClass=sudoRole)(|(!(sudoHost=*))(sudoHost=ALL)
(sudoHost=client.example.com)(sudoHost=client)(sudoHost=192.0.2.1)
(sudoHost=192.0.2.0/24)
(sudoHost=2620:52:0:224e:21a:4aff:fe23:1394)
(sudoHost=2620:52:0:224e::/64)(sudoHost=fe80::21a:4aff:fe23:1394)
(sudoHost=fe80::/64)(sudoHost=+*)(|(sudoHost=*\\*)(sudoHost=*?*)
(sudoHost=*\2A*)(sudoHost=*[*]*))))][dc=example,dc=com]
```

Use this filter to look up the rules in the IdM database:

```
# ldapsearch -x -D "cn=Directory Manager" -W -H
ldap://server.example.com -b dc=example,dc=com '(&
(objectClass=sudoRole)...)'
```

The sudo responder log file: `/var/log/sssd/sssd_sudo.log`

This log file helps you to answer questions like:

- How many rules did SSSD return?

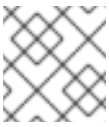
```
[sssd[sudo]] [sudosrv_get_sudorules_from_cache] (0x0400):
Returning 4-rules rules for [user@idm.example.com]
```

- What filter was applied for searching the cache of SSSD?

```
[sudosrv_get_sudorules_query_cache] (0x0200): Searching sysdb with
[(&(objectClass=sudoRule)(|(sudoUser=ALL)(sudoUser=user)
(sudoUser=#10001)(sudoUser=%group-1)(sudoUser=%user)
(sudoUser=+*)))]
```

- How do I look up the rules returned from the SSSD cache? Use the following filter to look up the rules:

```
# ldbsearch -H /var/lib/sss/db/cache_domain_name.ldb -b cn=sysdb
'(&(objectClass=sudoRule)...)'
```



NOTE

The **ldbsearch** utility is included in the ldb-tools package.

A.3. TROUBLESHOOTING FIREFOX KERBEROS CONFIGURATION

If Kerberos authentication is not working, turn on verbose logging for the authentication process.

1. Close all instances of Firefox.
2. In a command prompt, export values for the **NSPR_LOG_*** variables:

```
export NSPR_LOG_MODULES=negotiateauth:5
export NSPR_LOG_FILE=/tmp/moz.log
```

3. Restart Firefox *from that shell*, and visit the website where Kerberos authentication is failing.
4. Check the **/tmp/moz.log** file for error messages with *nsNegotiateAuth* in the message.

There are several common errors that occur with Kerberos authentication.

No credentials found

```
-1208550944[90039d0]: entering nsNegotiateAuth::GetNextToken()
-1208550944[90039d0]: gss_init_sec_context() failed: Miscellaneous
failure
No credentials cache found
```

This means that no Kerberos tickets are available (meaning that they expired or were not generated). To fix this, run **kinit** to generate the Kerberos ticket, and then open the website again.

Server not found in Kerberos database

```
-1208994096[8d683d8]: entering nsAuthGSSAPI::GetNextToken()
-1208994096[8d683d8]: gss_init_sec_context() failed: Miscellaneous
failure
Server not found in Kerberos database
```

This means that the browser is unable to contact the KDC. This is usually a Kerberos configuration problem. The correct entries must be in the **[domain_realm]** section of the **/etc/krb5.conf** file to identify the domain. For example:

```
.example.com = EXAMPLE.COM
example.com = EXAMPLE.COM
```

No errors are present in the log

An HTTP proxy server could be stripping off the HTTP headers required for Kerberos authentication. Try to connect to the site using HTTPS, which allows the request to pass through unmodified.

APPENDIX B. REVISION HISTORY

Note that revision numbers relate to the edition of this manual, not to version numbers of Red Hat Enterprise Linux.

Revision 7.0-27 **Mon Jun 25 2018** **Filip Hanzelka**
Updated *Working with certmonger*.

Revision 7.0-26 **Tue Apr 10 2018** **Filip Hanzelka**
Preparing document for 7.5 GA publication.

Revision 7.0-25 **Mon Mar 19 2018** **Lucie Maňásková**
Minor updates.

Revision 7.0-24 **Mon Feb 12 2018** **Aneta Šteflová Petrová**
Minor fixes and updates.

Revision 7.0-23 **Mon Jan 29 2018** **Aneta Šteflová Petrová**
Minor fixes.

Revision 7.0-22 **Mon Dec 4 2017** **Aneta Šteflová Petrová**
Updated *Smart cards*.

Revision 7.0-21 **Mon Nov 20 2017** **Aneta Šteflová Petrová**
Minor fixes.

Revision 7.0-20 **Mon Nov 6 2017** **Aneta Šteflová Petrová**
Minor fixes.

Revision 7.0-19 **Mon Aug 14 2017** **Aneta Šteflová Petrová**
Updated sections that referred to the coolkey package.

Revision 7.0-18 **Tue Jul 18 2017** **Aneta Šteflová Petrová**
Document version for 7.4 GA publication.

Revision 7.0-17 **Mon Mar 27 2017** **Aneta Šteflová Petrová**
Fixed broken links.

Revision 7.0-16 **Mon Feb 27 2017** **Aneta Šteflová Petrová**
Updated Kerberos KDC proxy. Other minor updates.

Revision 7.0-15 **Wed Dec 7 2016** **Aneta Šteflová Petrová**
Added SSSD client-side views. Other minor updates.

Revision 7.0-14 **Tue Oct 18 2016** **Aneta Šteflová Petrová**
Version for 7.3 GA publication.

Revision 7.0-13 **Wed Jul 27 2016** **Marc Muehlfeld**
Added Kerberos over HTTP (kdcproxy), requesting a certificate through SCEP, other minor updates.

Revision 7.0-11 **Thu Mar 03 2016** **Aneta Petrová**
Added restricting domains for PAM services.

Revision 7.0-10 **Tue Feb 09 2016** **Aneta Petrová**
Split authconfig chapter into smaller chapters, other minor updates.

Revision 7.0-9 **Thu Nov 12 2015** **Aneta Petrová**

Version for 7.2 GA release.

Revision 7.0-8 **Fri Mar 13 2015** **Tomáš Čapek**
Async update with last-minute edits for 7.1.

Revision 7.0-6 **Wed Feb 25 2015** **Tomáš Čapek**
Version for 7.1 GA release.

Revision 7.0-4 **Fri Dec 05 2014** **Tomáš Čapek**
Rebuild to update the sort order on the splash page.

Revision 7.0-1 **July 16, 2014** **Ella Deon Ballard**
Initial draft for RHEL 7.0.