Red Hat Enterprise Linux 7 System-Level Authentication Guide

About System-Level Services for Authentication and Identity Management

Tomáš Čapek Aneta Petrová Ella Deon Ballard

Red Hat Enterprise Linux 7 System-Level Authentication Guide

About System-Level Services for Authentication and Identity Management

Tomáš Čapek
Red Hat Customer Content Services
tcapek@redhat.com

Aneta Petrová
Red Hat Customer Content Services
apetrova@redhat.com

Ella Deon Ballard
Red Hat Customer Content Services

Legal Notice

Abstract

This guide covers different applications and services available to configure authentication on local systems.

Table of Contents

Chapter 1. Introduction to System Authentication **3**
1.1. Confirming User Identities 3
1.2. As Part of Planning Single Sign-On 4
1.3. Available Services 4

Part I. System Logins **6**

Chapter 2. Configuring System Authentication **7**
2.1. Identity Management Tools for System Authentication 7
2.2. Using authconfig 7

Chapter 3. Selecting the Identity Store for Authentication with authconfig **13**
3.1. IPAv2 13
3.2. LDAP and FreeIPA 15
3.3. NIS 18
3.4. Winbind 20

Chapter 4. Configuring Authentication Mechanisms **24**
4.1. Configuring Local Authentication Using authconfig 24
4.2. Configuring System Passwords Using authconfig 26
4.3. Configuring Kerberos (with LDAP or NIS) Using authconfig 30
4.4. Smart Cards 33
4.5. One-Time Passwords 39
4.6. Configuring Fingerprints Using authconfig 39

Chapter 5. Managing Kickstart and Configuration Files Using authconfig **42**

Chapter 6. Enabling Custom Home Directories Using authconfig **43**

Part II. Identity and Authentication Stores **46**

Chapter 7. Using and Caching Credentials with SSSD **47**
7.1. The Basics of SSSD Configuration 47
7.2. SSSD and System Services 50
7.3. SSSD and Identity Providers (Domains) 67
7.4. Managing Local System Users in SSSD 113
7.5. Downgrading SSSD 117
7.6. Using NSCD with SSSD 118
7.7. Troubleshooting SSSD 118

Chapter 8. Using realmd to Connect to an Identity Domain **125**

Part III. Secure Applications **126**

Chapter 9. Using Pluggable Authentication Modules (PAM) **127**
9.1. About PAM 127
9.2. About PAM Configuration Files 127
9.3. PAM and Administrative Credential Caching 131
9.4. Restricting Domains for PAM services 133

Chapter 10. Using Kerberos **136**
10.1. About Kerberos 136
10.2. Configuring the Kerberos KDC 140
10.3. Configuring a Kerberos Client 145
10.4. Setting up a Kerberos Client for Smart Cards 147

10.5. Setting up Cross-Realm Kerberos Trusts 148

Chapter 11. Working with certmonger 153
11.1. certmonger and Certificate Authorities 153
11.2. Requesting a Certificate with certmonger 153
11.3. Storing Certificates in NSS Databases 154
11.4. Tracking Certificates with certmonger 155

Chapter 12. Configuring Applications for Single Sign-On 157
12.1. Configuring Firefox to Use Kerberos for Single Sign-On 157
12.2. Certificate Management in Firefox 159
12.3. Certificate Management in Email Clients 162

Appendix A. Revision History 166

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, username 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 via 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:
    - A user inserts a smart card into the card reader. Pluggable authentication modules (PAMs) on Red Hat Enterprise Linux detect the inserted smart card.
    - 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.
    - 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 screensavers, 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 backends 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 files-based authentication.

Other services and applications are also available, but these are common ones and form the core of this guide.

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 simple 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. **--test** prints the changes to stdout 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.

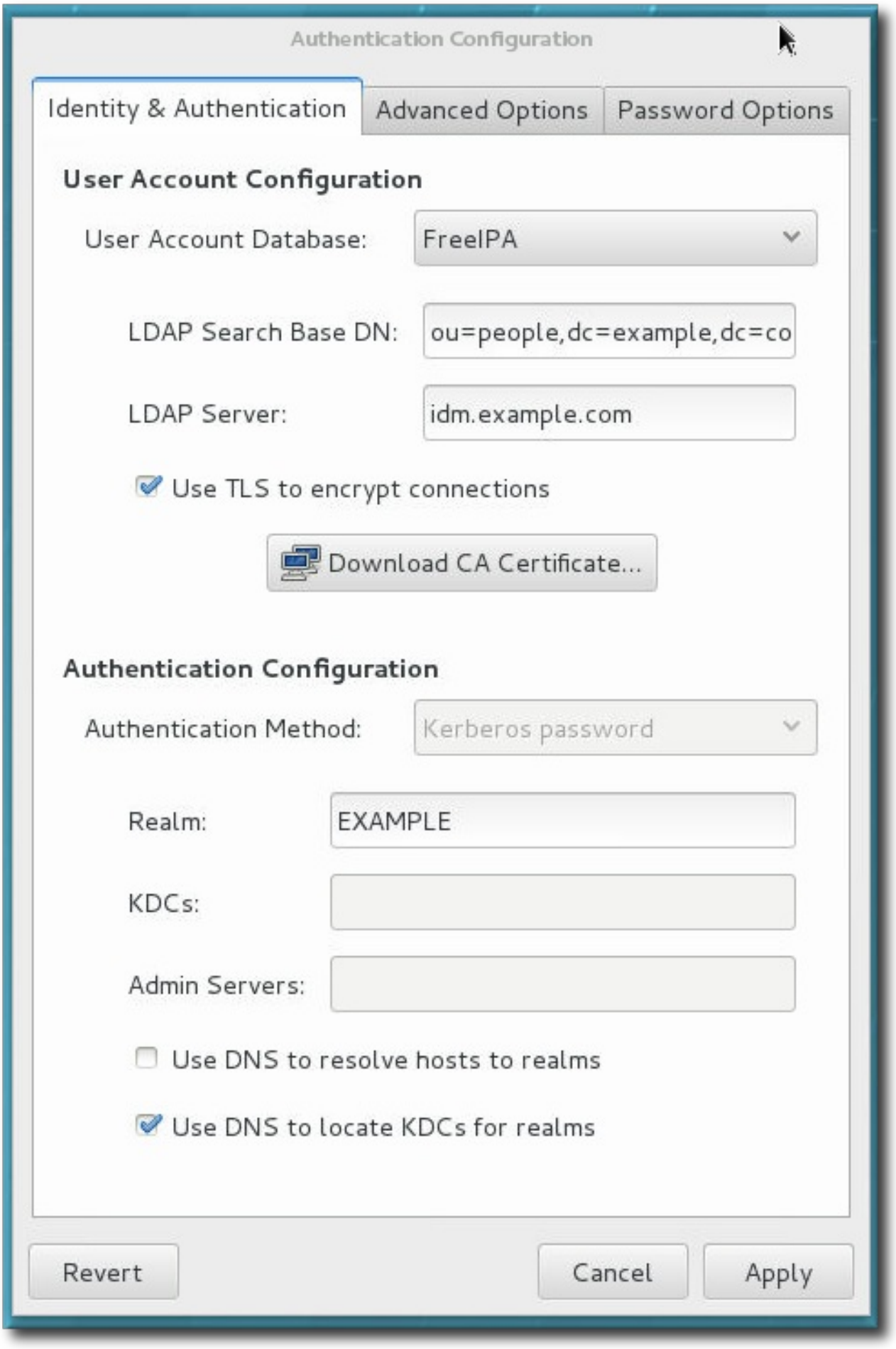


Figure 2.1. authconfig Window

2.2.4. Testing Authentication Settings

It is critical that authentication is fully and properly configured. Otherwise, the worst-case scenarios is that all users (even root) could be locked out of the system; in less severe situations, it is still possible for some users to be blocked or for the wrong identities to be used.

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/authconfigbackup20160701
```

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/authconfigbackup20160701
```

The **`authconfig`** command saves an automatic backup every time the configuration is altered. It is possible to restore the most recently-created automatic backup automatically by 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 backend. 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.

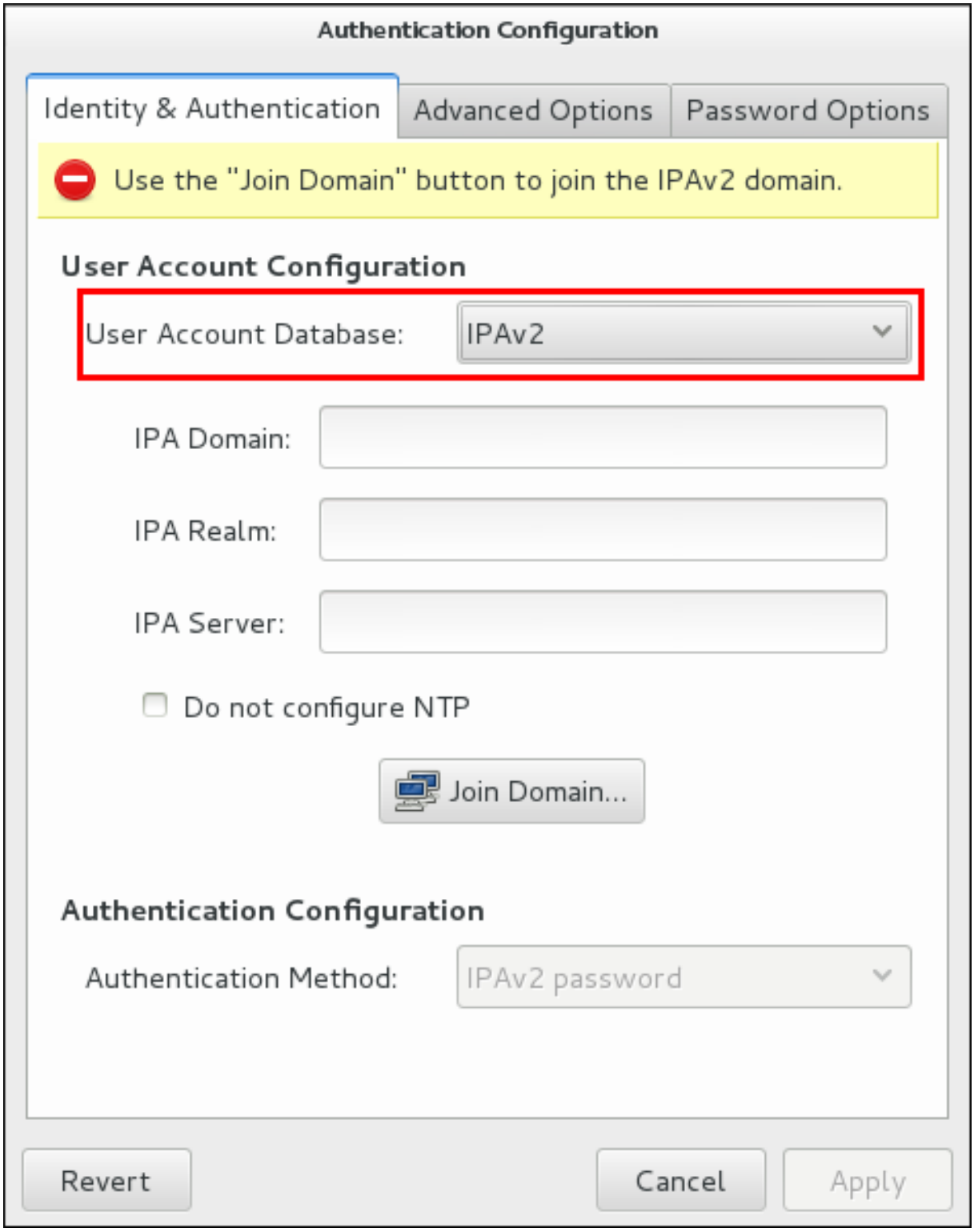


Figure 3.1. Authentication Configuration

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 hostname 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 username 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.

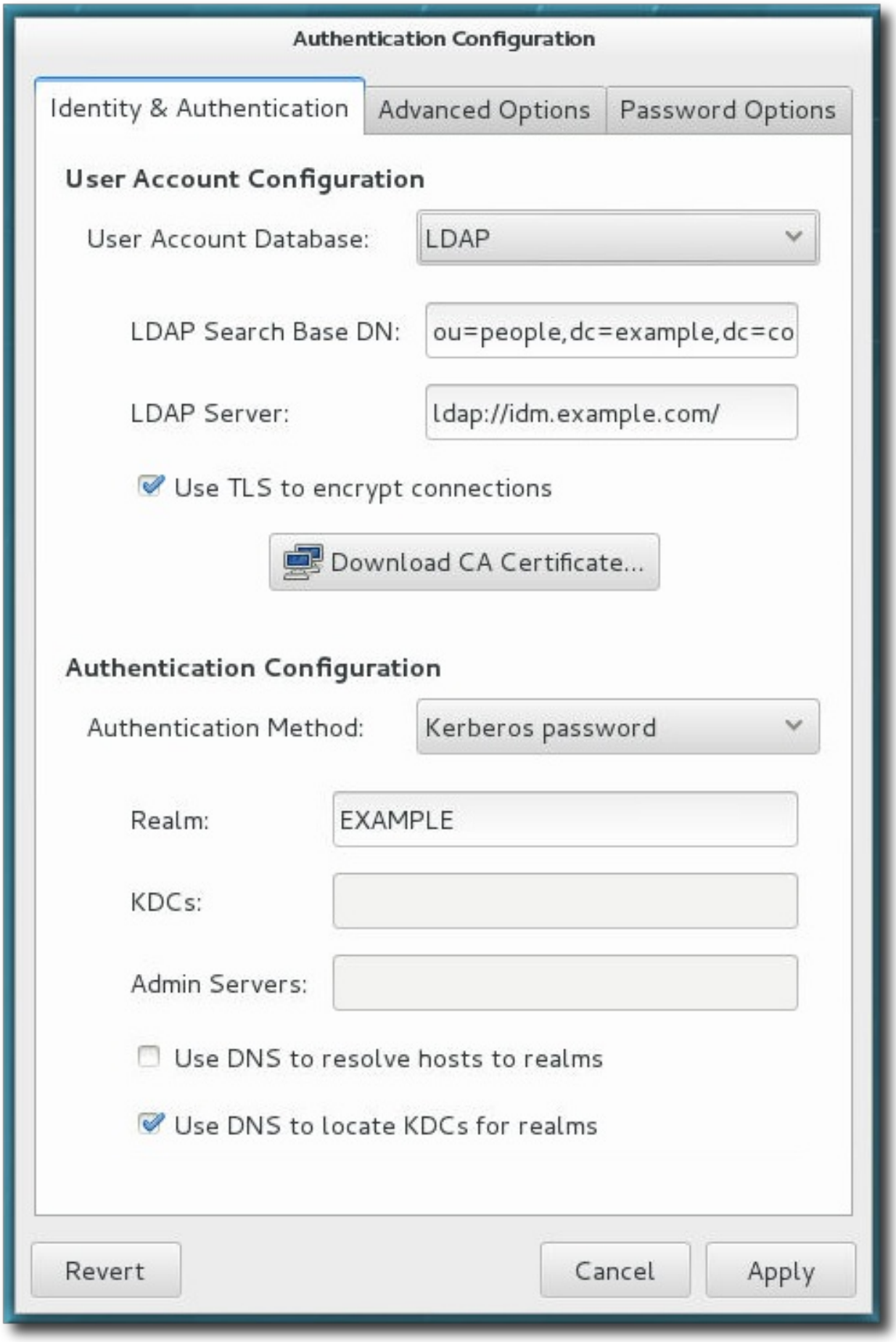
3.2. LDAP and FreeIPA

Both standard LDAP directories (such as OpenLDAP and Red Hat Directory Server) can be used as LDAP identity providers. Additionally, older IPA 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.

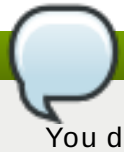
- **LDAP Search Base DN** gives the root suffix or *distinguished name* (DN) for the user directory. All of the user entries used for identity or authentication exist below this parent entry. For example, **ou=people,dc=example,dc=com**.

  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 non-secure standard port connection (URL starting with **ldap://**), you can use the **Use TLS to encrypt connections** checkbox to encrypt communication with the LDAP server using StartTLS. Selecting this checkbox also enables the **Download CA Certificate** button.



Note

You do not need to select the **Use TLS to encrypt connections** checkbox 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

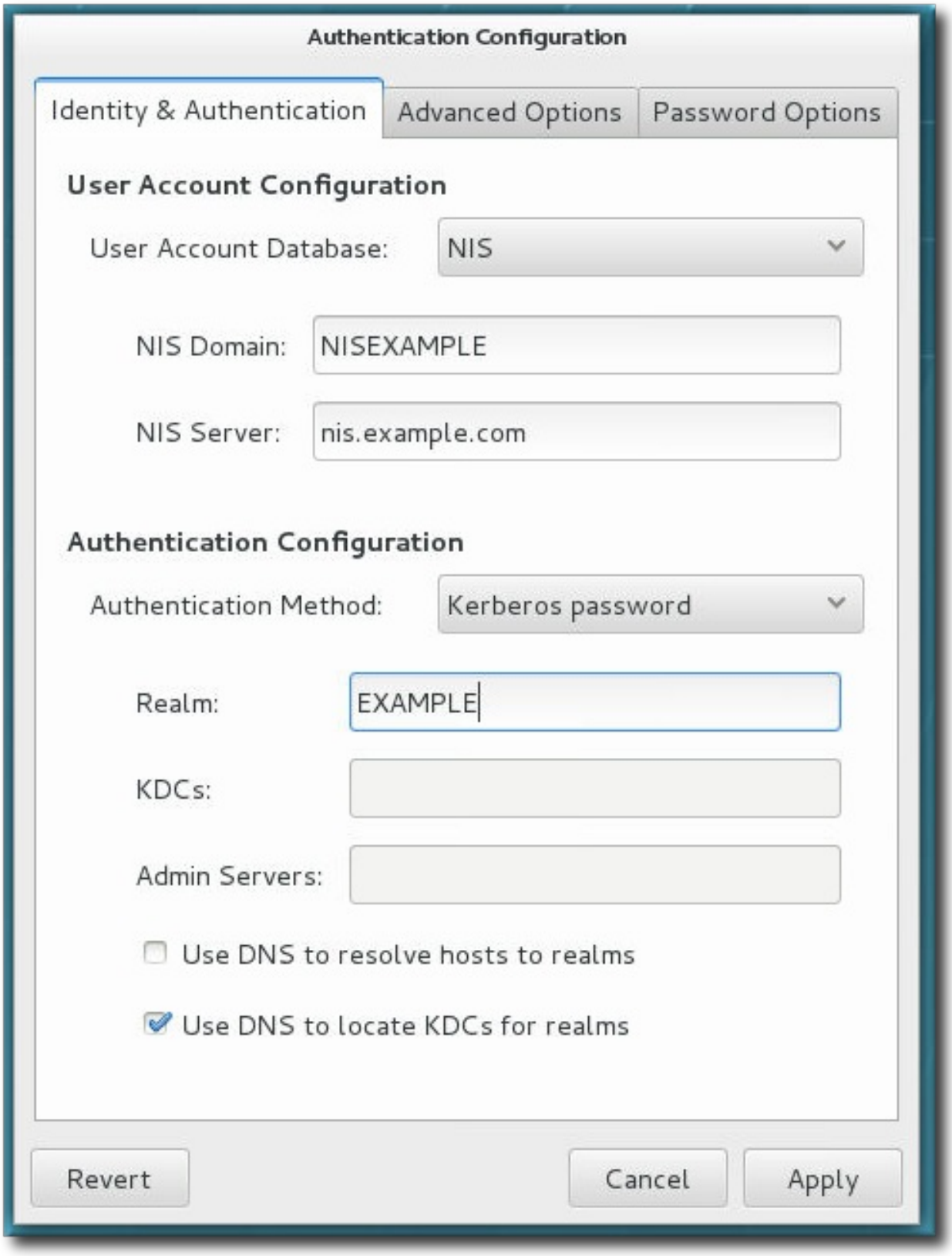
Important

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 hostname. 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 backend 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

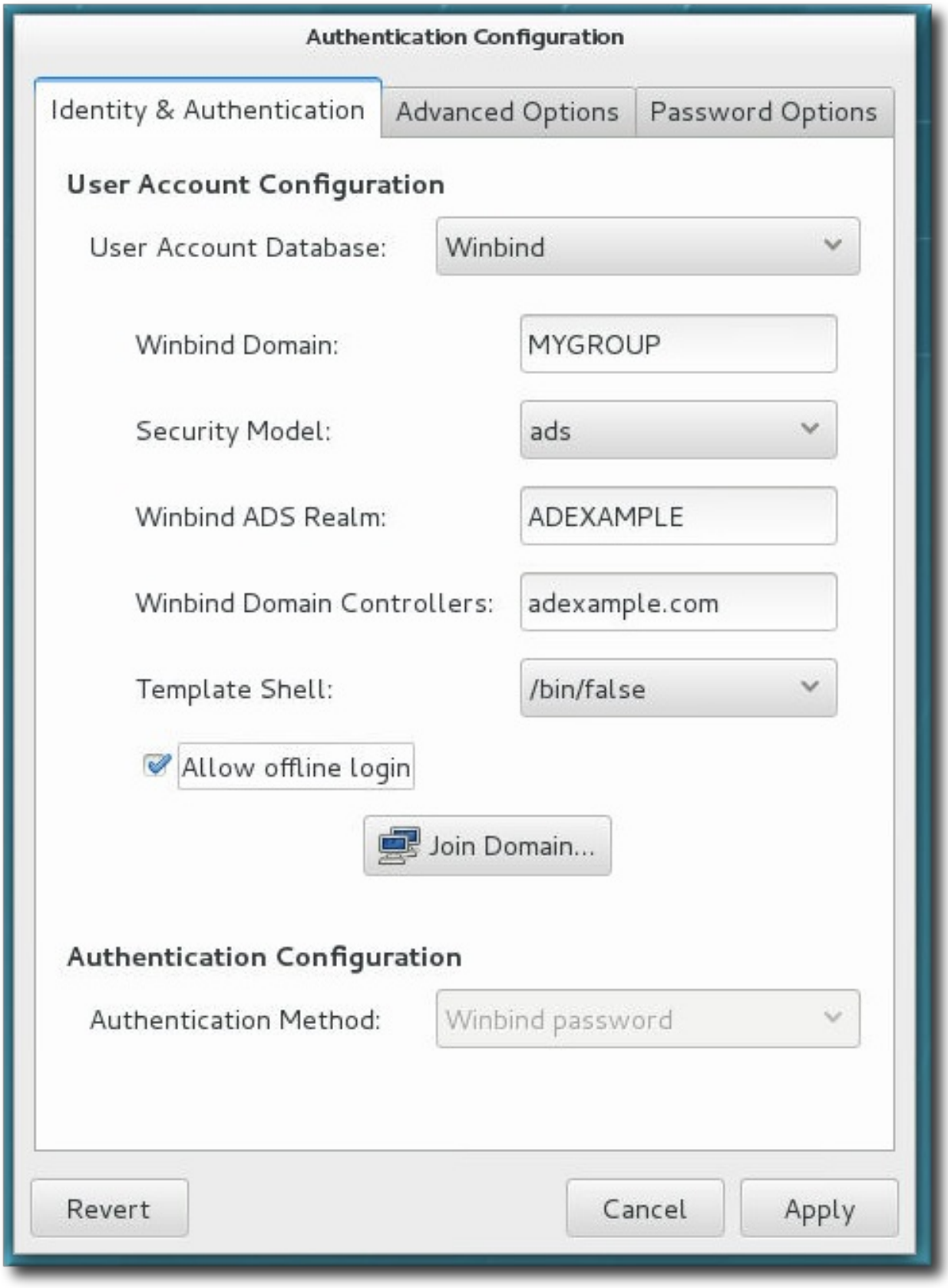
1. Install the **samba-winbind** package. This is required for Windows integration features in Samba services, but is not installed by default.

   ```
   [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 username/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 username/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 username and password. This mode does support encrypted passwords.

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

Note

When verifying that a given user exists in the Windows domain, always use Windows 2000-style 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
```

  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 hostname 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 hostnames.

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

When verifying that a given user exists in the Windows domain, always use Windows 2000-style 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 username, 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-based 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 backend. 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.

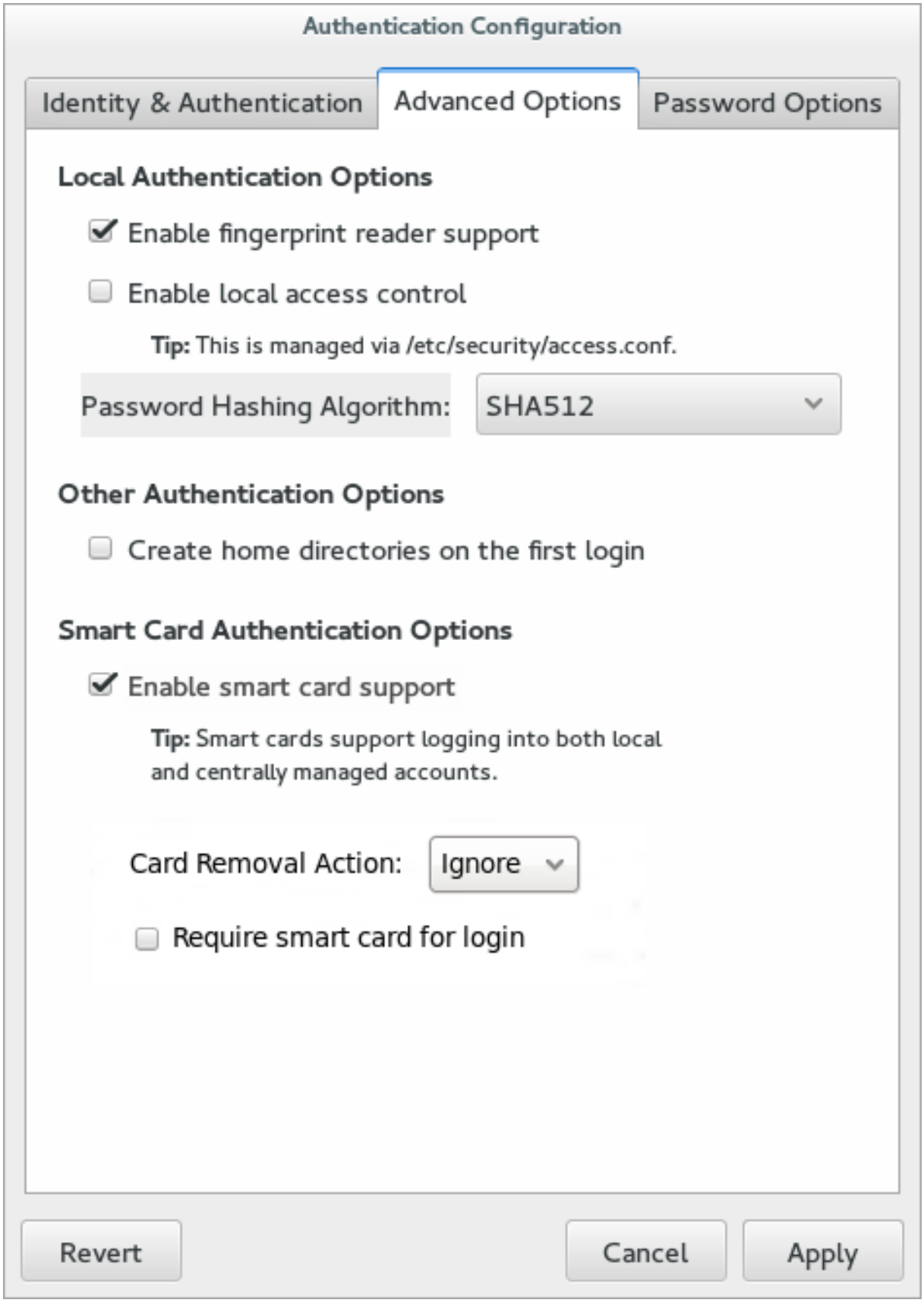


Figure 4.1. Local Accounts Fields

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.

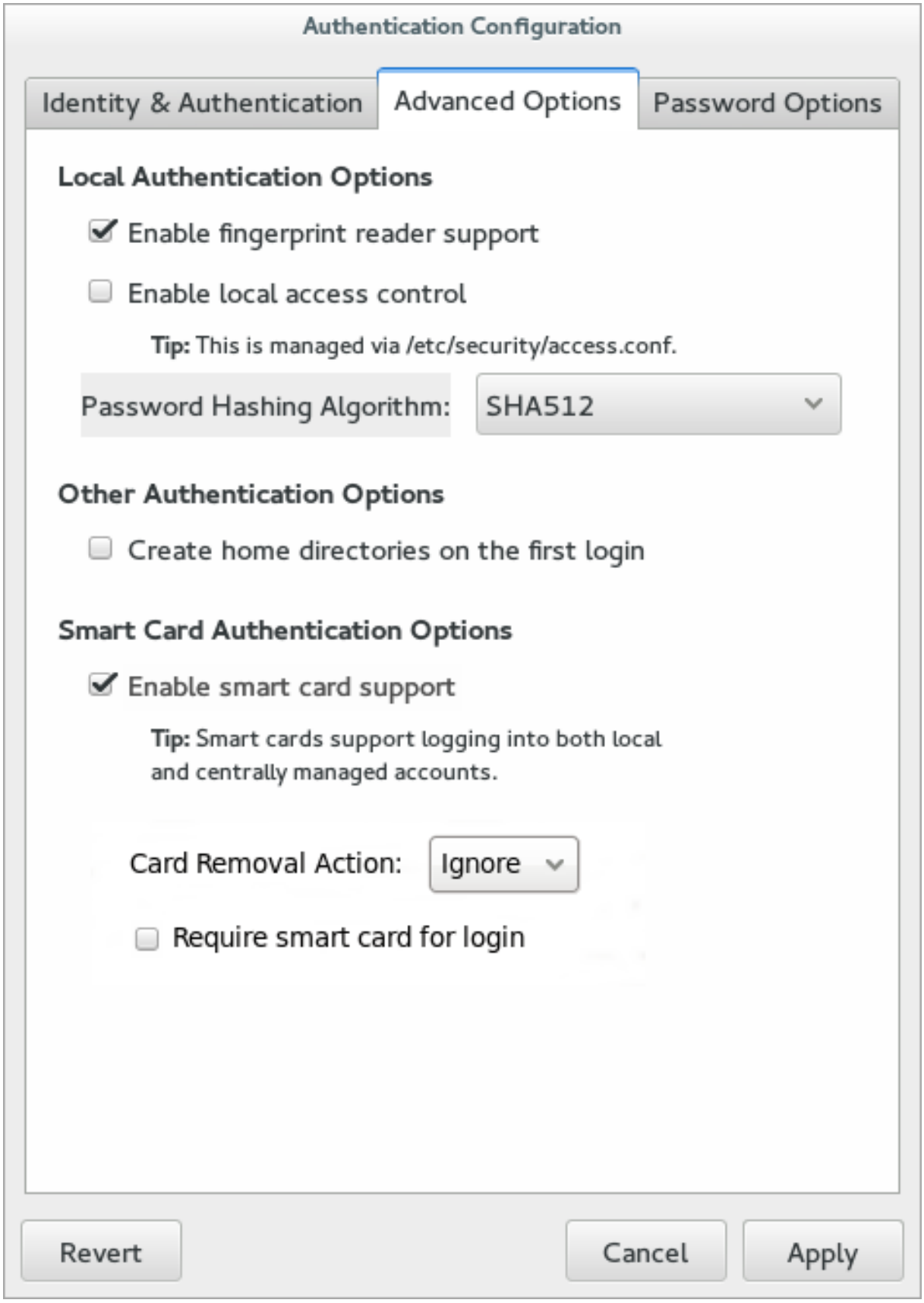
Important

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:

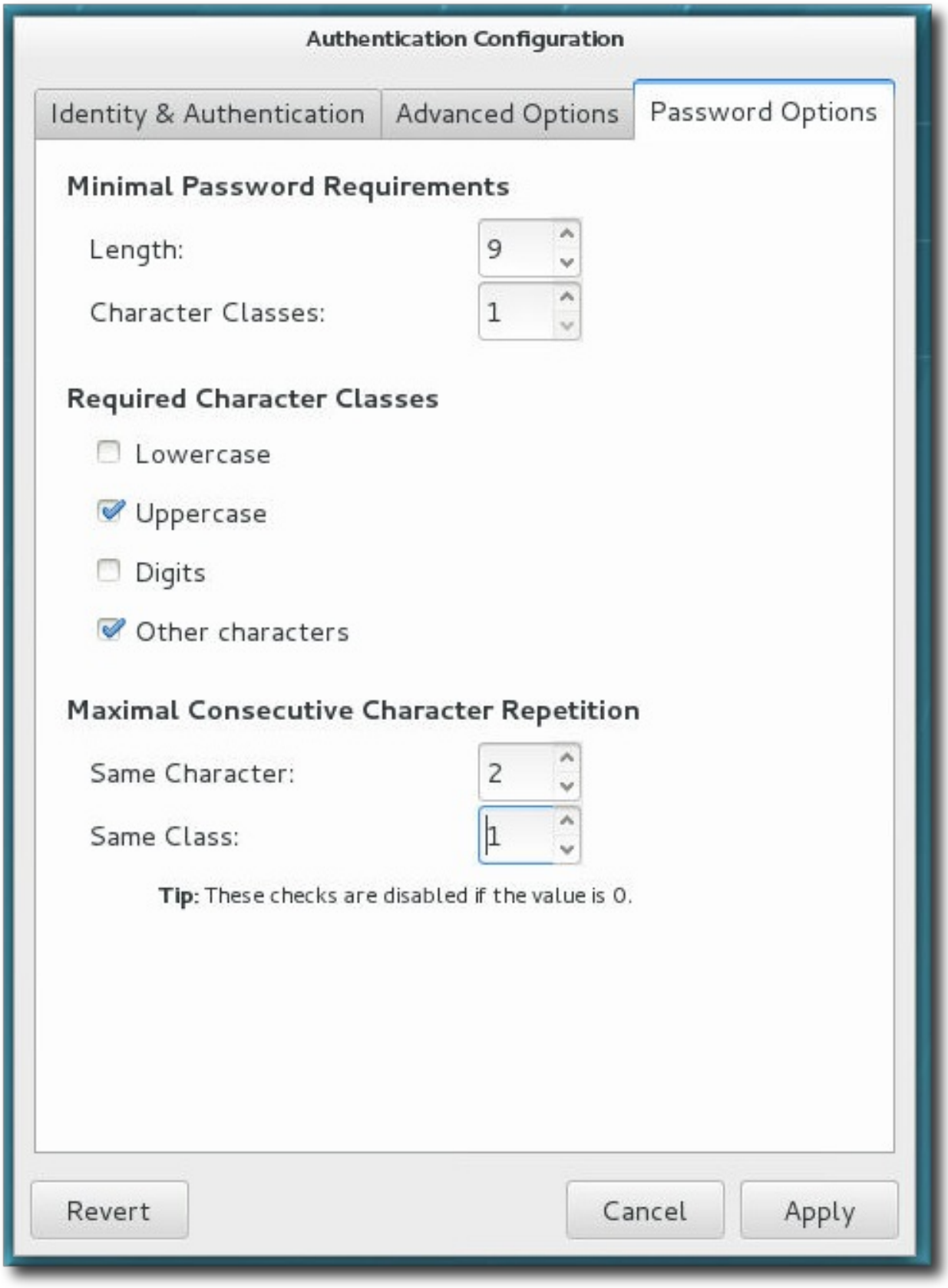
```
[root@server ~]# authconfig --passalgo=sha512 --update
```

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** checkbox 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 repeate 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.

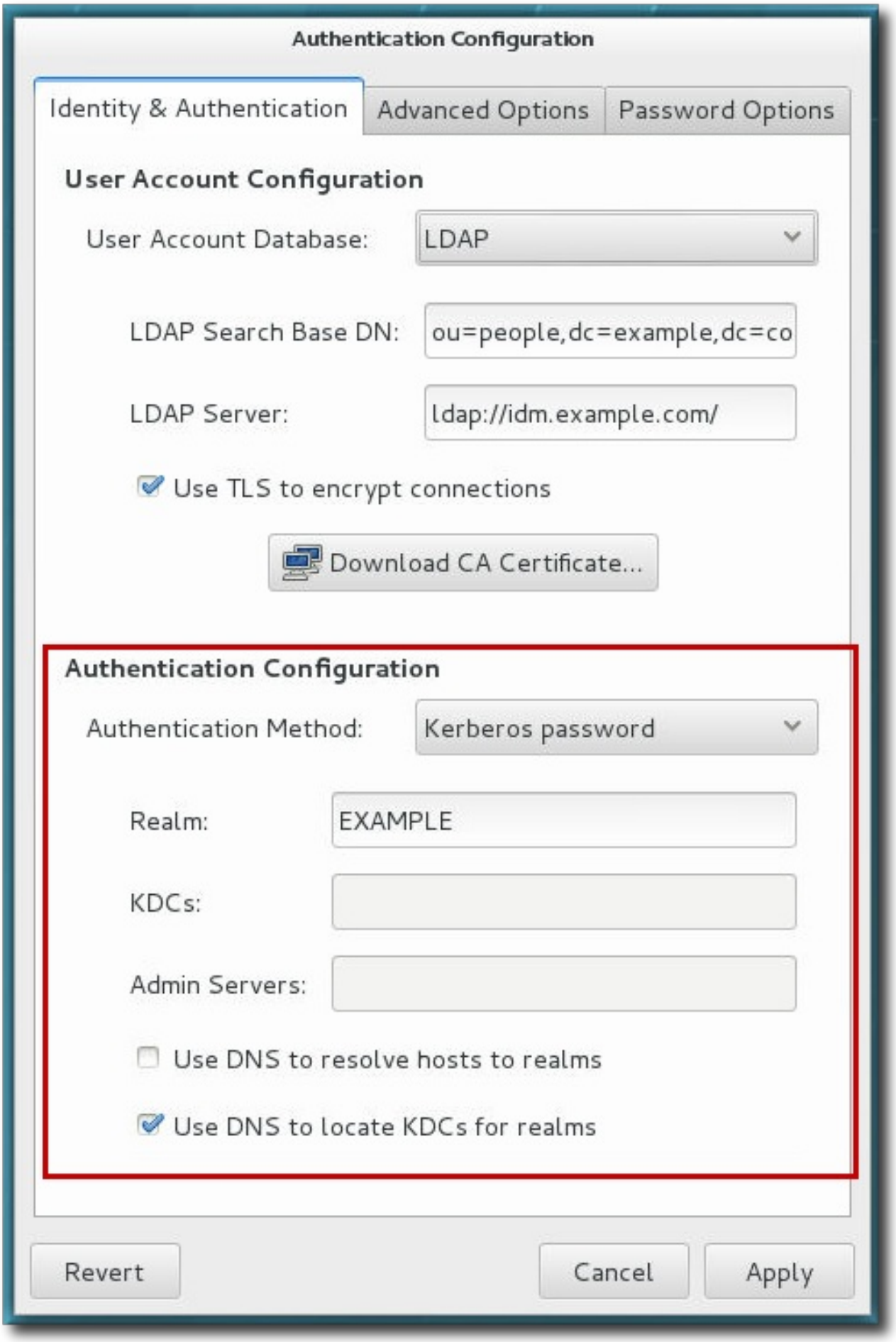


Figure 4.2. Kerberos Fields

- **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 hostname 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.

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.

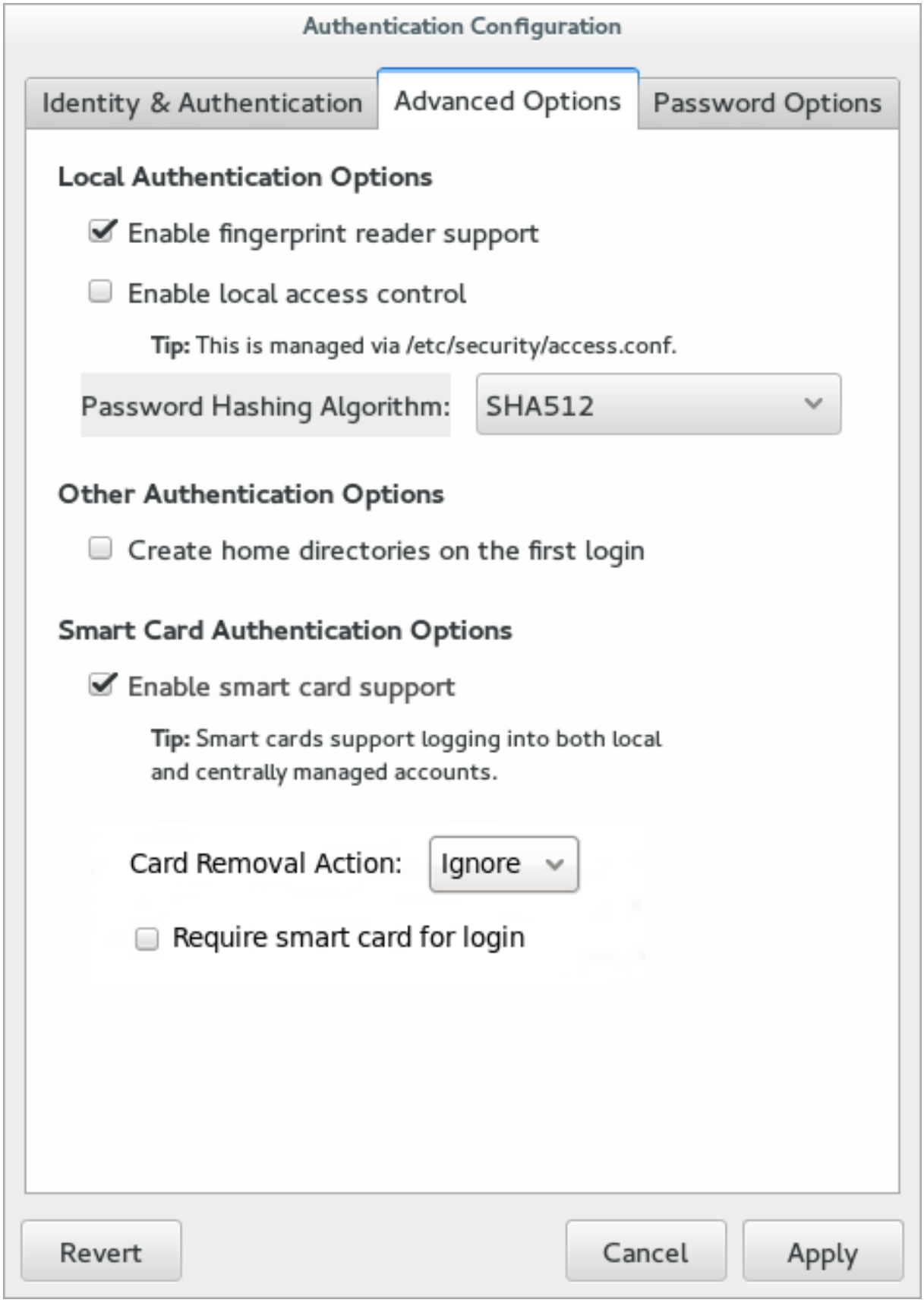


Figure 4.3. Smart Card Options

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
- coolkey
- ccid
- gdm
- authconfig
- authconfig-gtk
- krb5-libs
- krb5-workstation
- krb5-pkinit-openssl
- 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** checkbox.
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** checkbox 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.

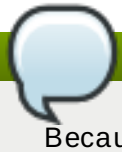
7. By default, the mechanisms to check whether a certificate has been revoked (Online Certificate Status Protocol, or OCSP, responses) are disabled. To validate whether a certificate has been revoked before its expiration period, enable OCSP checking by adding the **ocsp_on** option to the ***cert_policy***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

   a. Open the **pam_pkcs11.conf** file.

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

   b. 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/libcoolk
eypk11.so
```

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.

4.5. 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.

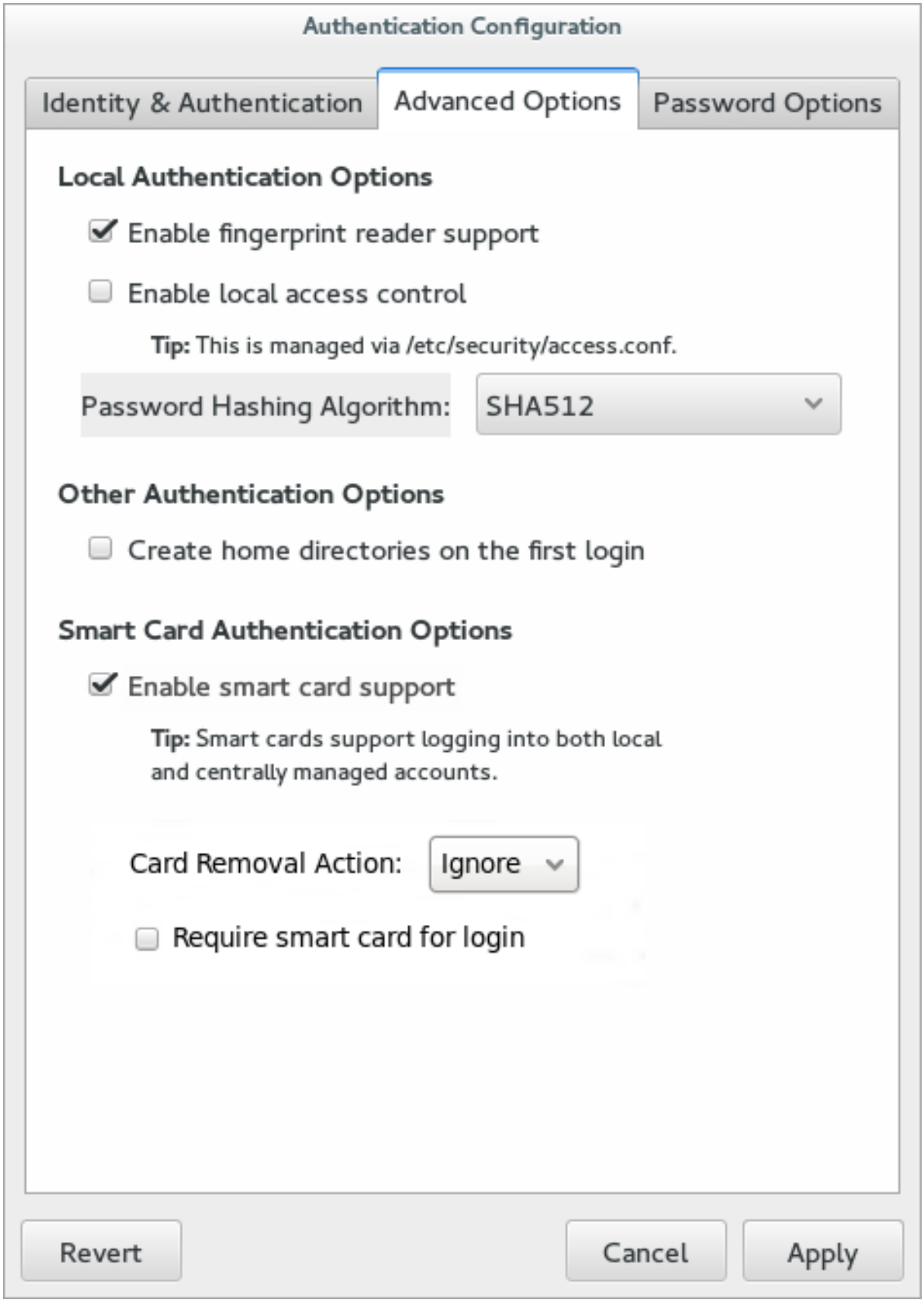


Figure 4.4. Fingerprint Options

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** prints the full configuration, with changes, to stdout 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. Re-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.

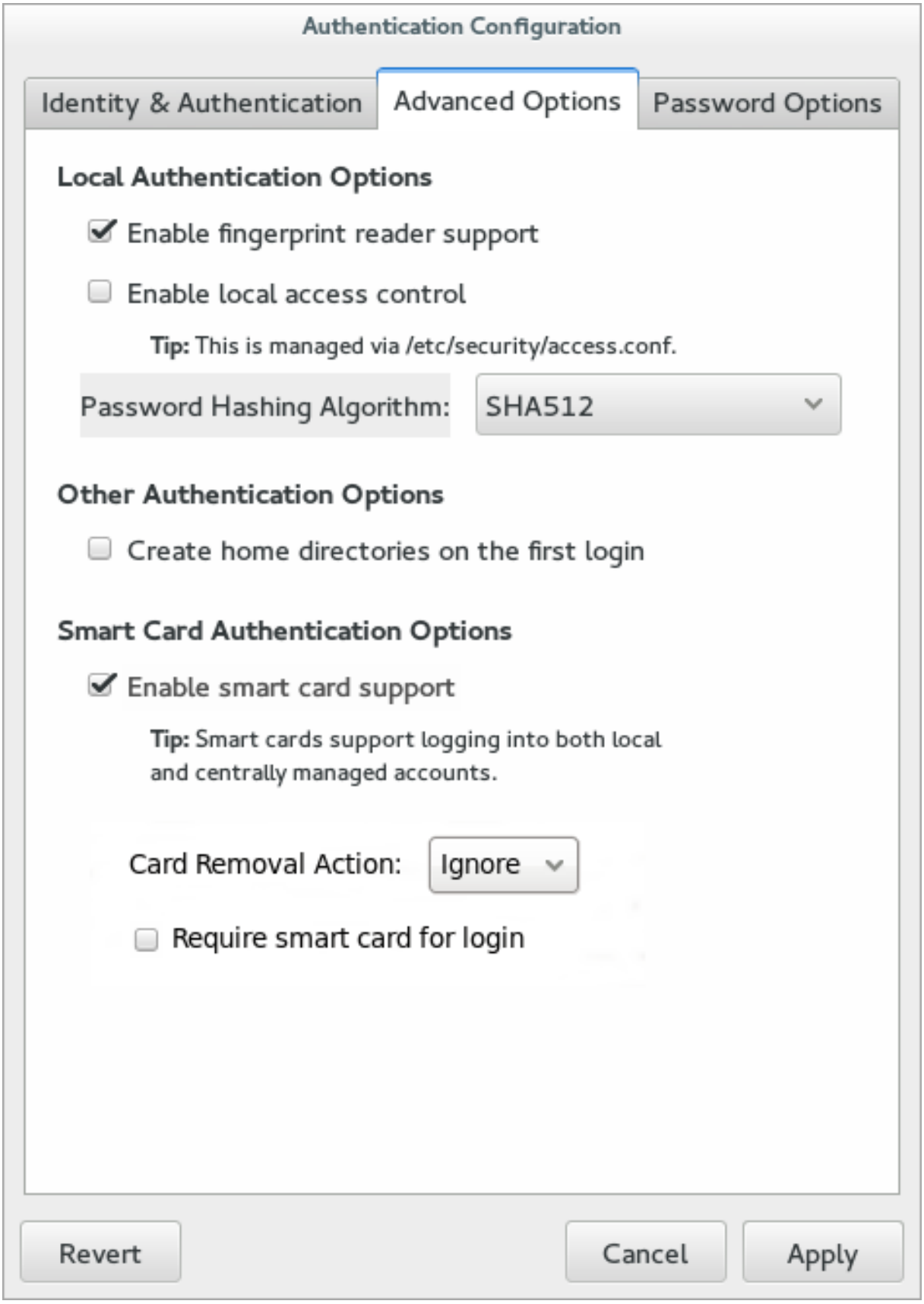


Figure 6.1. Home Directory Option

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. Using and Caching Credentials with SSSD

The System Security Services Daemon (SSSD) provides access to different identity and authentication providers. This service ties a local system to a larger backend system. That can be a simple LDAP directory, domains for Active Directory or IdM in Red Hat Enterprise Linux, or Kerberos realms.

SSSD configures a way to connect to an identity store to retrieve authentication information and then uses that to create a local cache of users and credentials. With some types of identity providers — including Active Directory — SSSD also pulls in authorization information.

SSSD is an intermediary between local clients and any configured data store. This relationship brings a number of benefits for administrators:

- *Reducing the load on identification/authentication servers.* Rather than having every client service attempt to contact the identification server directly, all of the local clients can contact SSSD which can connect to the identification server or check its cache.
- *Permitting offline authentication.* By default, SSSD keeps a cache of user identities (user name, UID, GID). You should also explicitly enable SSSD to keep a cache of credentials, that is, hashed passwords, that SSSD retrieves from remote services. Keeping these caches allows users to authenticate to resources successfully, even if the remote identification server is offline or the local machine is offline.
- *Using a single user account.* Remote users frequently have two (or even more) user accounts, such as one for their local system and one for the organizational system. This is necessary to connect to a virtual private network (VPN). Because SSSD supports caching and offline authentication, remote users can connect to network resources simply by authenticating to their local machine and then SSSD maintains their network credentials.

SSSD caches those users and credentials, so if the local system *or* the identity provider go offline, the user credentials are still available to services to verify.

7.1. The Basics of SSSD Configuration

SSSD is a local service which connects a system to a larger, external identity service. This is done by configuring *domains* in the SSSD configuration file. Each domain represents a different, external data source. Domains always represent an *identity provider* which supplies user information and, optionally, define other providers for different kinds of operations, such as authentication or password changes. (The identity provider can also be used for all operations, if all operations are performed within a single domain or server.)

Note

SSSD allows all user identities to be created and maintained in a separate, external identity source. For Windows integration, then the Active Directory domain can be used to manage user accounts (as it is with most environments). Local system users do not need to be created or synced with user accounts in Active Directory — SSSD uses those Windows identities and lets those Windows users access the local system and local services.

SSSD also defines which services on the system use SSSD for credentials caching and user accounts. These relate to foundational security services such as the Name Service Switch (NSS) and pluggable authentication modules (PAM), which are then used by higher-level applications.

7.1.1. Setting up the sssd.conf File

SSSD services and domains are configured in a **.conf** file. By default, this is **/etc/sssd/sssd.conf** — although that file must be created and configured manually, since SSSD is not configured after installation.

7.1.1.1. Creating the sssd.conf File

There are three parts of the SSSD configuration file:

- **[sssd]**, for general SSSD process and operational configuration; this basically lists the configured services, domains, and configuration parameters for each
- *[service_name]*, for configuration options for each supported system service, as described in Section 7.2, “SSSD and System Services”
- *[domain_type/DOMAIN_NAME]*, for configuration options for each configured identity provider

Important

While services are optional, at least one identity provider domain must be configured before the SSSD service can be started.

Example 7.1. Simple sssd.conf File

```
[sssd]
domains = LOCAL
services = nss
config_file_version = 2

[nss]
filter_groups = root
filter_users = root

[domain/LOCAL]
id_provider = local
auth_provider = local
access_provider = permit
```

The **[sssd]** section has three important parameters:

- **domains** lists all of the domains, configured in the **sssd.conf**, which SSSD uses as identity providers. If a domain is not listed in the **domains** key, it is not used by SSSD, even if it has a configuration section.
- **services** lists all of the system services, configured in the **sssd.conf**, which use SSSD; when SSSD starts, the corresponding SSSD service is started for each configured system service. If a service is not listed in the **services** key, it is not used by SSSD, even if it has a configuration section.
- **config_file_version** sets the version of the configuration file to set file format expectations. This is version 2, for all recent SSSD versions.

Note

Even if a service or domain is configured in the **sssd.conf** file, SSSD does not interact with that service or domain unless it is listed in the **services** or **domains** parameters, respectively, in the **[sssd]** section.

Other configuration parameters are listed in the **sssd.conf** man page.

Each service and domain parameter is described in its respective configuration section in this chapter and in their man pages.

7.1.1.2. Using a Custom Configuration File

By default, the **sssd** process assumes that the configuration file is **/etc/sssd/sssd.conf**.

An alternative file can be passed to SSSD by using the **-c** option with the **sssd** command:

```
[root@server ~]# sssd -c /etc/sssd/customfile.conf --daemon
```

7.1.1.3. Additional Resources

While this chapter covers the basics of configuring services and domains in SSSD, this is not a comprehensive resource. Many other configuration options are available for each functional area in SSSD; check out the man page for the specific functional area to get a complete list of options.

Some of the common man pages are listed in Table 7.1, “A Sampling of SSSD Man Pages”. There is also a complete list of SSSD man pages in the "See Also" section of the **sssd(8)** man page.

Table 7.1. A Sampling of SSSD Man Pages

Functional Area	Man Page
General Configuration	sssd.conf(8)
sudo Services	sssd-sudo
LDAP Domains	sssd-ldap
Active Directory Domains	sssd-ad sssd-ldap
Identity Management (IdM or IPA) Domains	sssd-ipa sssd-ldap
Kerberos Authentication for Domains	sssd-krb5
OpenSSH Keys	sss_ssh_authorizedkeys sss_ssh_knownhostsproxy
Cache Maintenance	sss_cache (cleanup) sss_useradd, sss_usermod, sss_userdel, sss_seed (user cache entry management)

7.1.2. Starting and Stopping SSSD

7.1.2. Starting and Stopping SSSD

Important

Configure at least one domain before starting SSSD for the first time. See Section 7.3, “SSSD and Identity Providers (Domains)”.

To start or stop SSSD, use the **systemctl** utility:

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

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

By default, SSSD is not configured to start automatically. To configure SSSD to start automatically during system boot, run the **systemctl enable** command:

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

For more information on managing services using **systemctl**, see the "Managing System Services" section in the Red Hat Enterprise Linux 7 System Administrator's Guide.

7.2. SSSD and System Services

SSSD and its associated services are configured in the **/etc/sssd/sssd.conf** file. The **[sssd]** section also lists the services that are active and should be started when **sssd** starts within the **services** directive.

SSSD can provide credentials caches for several system services:

- A Name Service Switch (NSS) provider service that answers name service requests from the **sssd_nss** module. This is configured in the **[nss]** section of the SSSD configuration.

  This is described in Section 7.2.1, “Configuring Services: NSS”.

- A PAM provider service that manages a PAM conversation through the **sssd_pam** module. This is configured in the **[pam]** section of the configuration.

  This is described in Section 7.2.2, “Configuring Services: PAM”.

- An SSH provider service that defines how SSSD manages the **known_hosts** file and other key-related configuration. Using SSSD with OpenSSH is described in Section 7.2.5, “Configuring Services: OpenSSH and Cached Keys”.

- An **autofs** provider service that connects to an LDAP server to retrieve configured mount locations. You can set basic configuration in an **[autofs]** section in the configuration file; the options for **[autofs]** are described in the **sssd-ldap(5)** man page in the "Autofs Options" section. Other configuration options can be set as part of an LDAP identity provider in a **[domain/NAME]** section in the configuration file.

  This is described in Section 7.2.3, “Configuring Services: autofs”.

- A **sudo** provider service that connects to an LDAP server to retrieve configured **sudo** policies. You can set basic configuration in a **[sudo]** section in the configuration file; the options for **[sudo]** are described in the **sssd-ldap(5)** man page in the "Sudo Options" section. Other

configuration options can be set as part of an LDAP identity provider in a **[domain/NAME]** section in the configuration file. The **sssd-sudo(5)** man page also contains useful information on how to configure **sudo** to work with **sssd**.

This is described in Section 7.2.4, “Configuring Services: sudo”.

- A PAC responder service that defines how SSSD works with Kerberos to manage Active Directory users and groups. This is specifically part of managing Active Directory identity providers with domains.

7.2.1. Configuring Services: NSS

SSSD provides an NSS module, **sssd_nss**, which instructs the system to use SSSD to retrieve user information. The NSS configuration must include a reference to the SSSD module, and then the SSSD configuration sets how SSSD interacts with NSS.

7.2.1.1. About NSS Service Maps and SSSD

The Name Service Switch (NSS) provides a central configuration for services to look up a number of configuration and name resolution services. NSS provides one method of mapping system identities and services with configuration sources.

SSSD works with NSS as a provider services for several types of NSS maps:

- Passwords (**passwd**)
- User groups (**shadow**)
- Groups (**groups**)
- Netgroups (**netgroups**)
- Services (**services**)

7.2.1.2. Configuring NSS Services to Use SSSD

NSS can use multiple identity and configuration providers for any and all of its service maps. The default is to use system files for services; for SSSD to be included, the **nss_sss** module has to be included for the desired service type.

1. Use the Authentication Configuration tool to enable SSSD. This automatically configured the **nsswitch.conf** file to use SSSD as a provider.

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

   This automatically configures the password, shadow, group, and netgroups services maps to use the SSSD module:

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

   netgroup:   files sss
   ```

2. The services map is not enabled by default when SSSD is enabled with **authconfig**. To include that map, open the **nsswitch.conf** file and add the **sss** module to the **services** map:

```
[root@server ~]# vim /etc/nsswitch.conf

...
services: file sss
...
```

7.2.1.3. Configuring SSSD to Work with NSS

The options and configuration that SSSD uses to service NSS requests are configured in the SSSD configuration file, in the **[nss]** services section.

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

```
[root@server ~]# vim /etc/sssd/sssd.conf
```

2. Make sure that NSS is listed as one of the services that works with SSSD.

```
[sssd]
config_file_version = 2
sbus_timeout = 30
services = nss, pam
```

3. In the **[nss]** section, change any of the NSS parameters. These are listed in Table 7.2, "SSSD [nss] Configuration Parameters".

```
[nss]
filter_groups = root
filter_users = root
entry_cache_timeout = 300
entry_cache_nowait_percentage = 75
```

4. Restart SSSD.

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

Table 7.2. SSSD [nss] Configuration Parameters

Parameter	Value Format	Description

Parameter	Value Format	Description
entry_cache_nowait_percentage	integer	Specifies the time interval after which SSSD automatically begins refreshing the entry cache. The time interval is specified as a percentage of the **entry_cache_timeout** value for the domain. For example, if **entry_cache_timeout** is set to 300 seconds and **entry_cache_nowait_percentage** is set to 75, the entry cache will begin refreshing when a request comes in at 75% of the timeout interval, which is 225 seconds. The automatic refresh ensures that future requests will not be blocked by waiting for a cache update. The allowed values for this option are 0 to 99, which sets the percentage based on the **entry_cache_timeout** value. The default value is 50%. Setting the option to zero (**0**) disables the entry cache refresh.
entry_negative_timeout	integer	Specifies how long, in seconds, **sssd_nss** should cache *negative* cache hits. A negative cache hit is a query for an invalid database entries, including non-existent entries.
filter_users, filter_groups	string	Tells SSSD to exclude certain users from being fetched from the NSS database. This is particularly useful for system accounts such as **root**.
filter_users_in_groups	Boolean	Sets whether users listed in the **filter_users** list appear in group memberships when performing group lookups. If set to **FALSE**, group lookups return all users that are members of that group. If not specified, this value defaults to **true**, which filters the group member lists.

Parameter	Value Format	Description
override_homedir	string	Overrides the user's home directory. This can be an absolute value or a template. The template takes the following variables: **%u** login name **%U** UID number **%d** domain name **%f** fully qualified user name (user@domain) **%o** The original home directory retrieved from the identity provider. **%%** a literal '%' This option can also be set globally or per domain. For example: `override_homedir = /home/%u` This is not set by default; in that case, the value is retrieved from the LDAP directory (the identity provider).
fallback_homedir	string	Sets a default template for a user's home directory if one is not specified explicitly by the domain's data provider. The available values for this option are the same as for override_homedir. For example: `override_homedir = /home/%u` By default, this is not set and there is no substitute value if a user home directory is not set.
override_shell	string	Overrides the login shell for all users. This option can be specified globally in the NSS service section or per domain. By default, this is not set, and SSSD uses the value retrieved from the LDAP identity provider.

Parameter	Value Format	Description
allowed_shells	string	Restricts user shell to one of the listed values. The list is evaluate in strict order: 1. Any shell present in **/etc/shells**. 2. Next, a shell in the **allowed_shells** list which is specified in the **shell_fallback** parameter for SSSD. 3. If a shell is not in the **/etc/shells** or **allowed_shells** lists, then a nologin shell. 4. An empty string for shell is passed as-is to libc. The "/etc/shells" is only read on SSSD startup, which means that a restart of the SSSD is required in case a new shell is installed. By default, this is not set, and the user shell is used.
vetoed_shells	string	Replaces any instance of the listed shells with the shell specified in the **shell_fallback** parameter.
shell_fallback	string	The default shell to use if an allowed shell is not installed on the machine. The default value is **/bin/sh**.
default_shell	String	Sets the default shell to use if the provider does not return one during lookup. This option supersedes any other shell options if it takes effect and can be set either in the NSS section or per domain. By default, this is not set and returns NULL if no shell is specified. This relies on **libc** to supply a value, such as **/bin/sh**.
get_domains_timeout	Integer	Specifies the time in seconds for which the list of subdomains will be considered valid. The default is 60 seconds.
memcache_timeout	Integer	Specifies the time in seconds for which records in the in-memory cache will be valid. The default is 300 seconds.
debug_level	integer, 0 - 9	Sets a debug logging level.
reconnection_retries	integer	Sets the number of times services attempt to reconnect in case a data provider crashes or restarts. The default is 3.

7.2.2. 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 any changes can be reverted.

SSSD provides a PAM module, **sssd_pam**, which instructs the system to use SSSD to retrieve user information. The PAM configuration must include a reference to the SSSD module, and then the SSSD configuration sets how SSSD interacts with PAM.

To configure the PAM service:

1. Use **authconfig** to enable SSSD for system authentication.

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

This automatically updates the PAM configuration to reference all of the SSSD modules:

```
#%PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
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

account  required pam_unix.so
account  sufficient pam_localuser.so
account  sufficient pam_succeed_if.so uid < 500 quiet
account  [default=bad success=ok user_unknown=ignore] pam_sss.so
account  required pam_permit.so

password requisite pam_pwquality.so try_first_pass retry=3
password sufficient pam_unix.so sha512 shadow nullok try_first_pass
use_authtok
password sufficient pam_sss.so use_authtok
password required pam_deny.so

session  optional pam_keyinit.so revoke
session  required pam_limits.so
session  [success=1 default=ignore] pam_succeed_if.so service in
crond quiet use_uid
session  sufficient pam_sss.so
session  required pam_unix.so
```

These modules can be set to **include** statements, as necessary.

2. Open the **sssd.conf** file.

```
# vim /etc/sssd/sssd.conf
```

3. Make sure that PAM is listed as one of the services that works with SSSD.

```
[sssd]
config_file_version = 2
sbus_timeout = 30
services = nss, pam
```

4. In the **[pam]** section, change any of the PAM parameters. These are listed in Table 7.3, "SSSD [pam] Configuration Parameters".

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

5. Restart SSSD.

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

Table 7.3. SSSD [pam] Configuration Parameters

Parameter	Value Format	Description
offline_credentials_expiration	integer	Sets how long, in days, to allow cached logins if the authentication provider is offline. This value is measured from the last successful online login. If not specified, this defaults to zero (**0**), which is unlimited.
offline_failed_login_attempts	integer	Sets how many failed login attempts are allowed if the authentication provider is offline. If not specified, this defaults to zero (**0**), which is unlimited.
offline_failed_login_delay	integer	Sets how long to prevent login attempts if a user hits the failed login attempt limit. If set to zero (**0**), the user cannot authenticate while the provider is offline once he hits the failed attempt limit. Only a successful online authentication can re-enable offline authentication. If not specified, this defaults to five (**5**).
reconnection_retries	integer	Sets the number of times services attempt to reconnect in case a data provider crashes or restarts. The default is 3.

Parameter	Value Format	Description
pam_verbosity	integer, 0 - 3	Controls how detailed is the information displayed to the user during authentication. The higher the value, the more information SSSD displays. The default is **1**.
pam_id_timeout	integer	On a per-client-application basis, this option controls for how long – in seconds – SSSD caches the user identity information, to avoid excessive round-trips to the identity provider. The default is **5**.
pam_pwd_expiration_warning	integer	Sets the number of days before a password expires that the users should be warned about the impending expiration. The default is **0**, meaning that SSSD automatically displays the warning if it receives the information about the expiration time from the back-end server.

For more information about the SSSD [pam] configuration parameters in **sssd.conf**, see the sssd.conf(5) man page.

7.2.3. Configuring Services: autofs

7.2.3.1. About Automount, LDAP, and SSSD

Automount maps are commonly flat files, which define a relationship between a map, a mount directory, and a fileserver. (Automount is described in the Storage Administration Guide.)

For example, let's say that there is a fileserver called **nfs.example.com** which hosts the directory **pub**, and automount is configured to mount directories in the **/shares/** directory. So, the mount location is **/shares/pub**. All of the mounts are listed in the **auto.master** file, which identifies the different mount directories and the files which configure them. The **auto.shares** map identifies each file server and mount directory, which automount then mounts in the **/shares/** directory. The relationships could be viewed like this:

```
        auto.master
   _____|__________
   |               |
   |               |
/shares/        auto.shares
                   |
       |
       |
            nfs.example.com:pub
```

Every mount point, then, is defined in two different files (at a minimum): the **auto.master** and **auto.** *whatever* file, and those files have to be available to each local automount process.

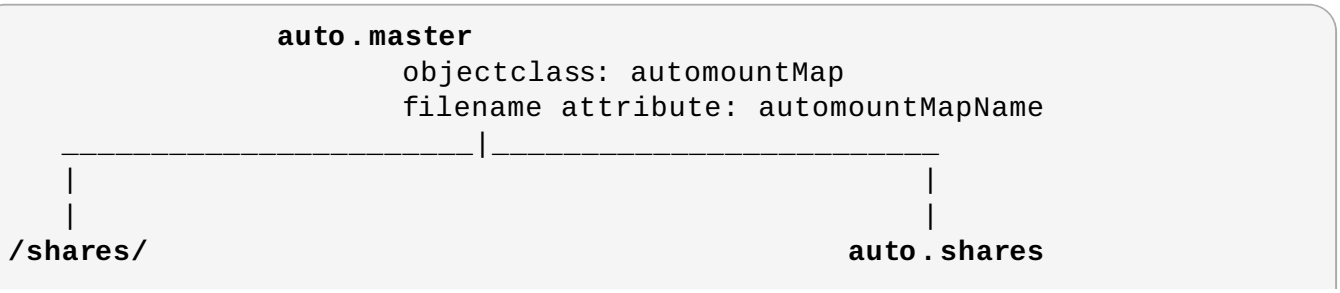
One way for administrators to manage that for large environments is to store the automount configuration in a central LDAP directory, and just configure each local system to point to that LDAP directory. That means that updates only need to be made in a single location, and any new maps are automatically recognized by local systems.

For automount-LDAP configuration, the automount files are stored as LDAP entries, which are then translated into the requisite automount files. Each element is then translated into an LDAP attribute.

The LDAP entries look like this:

```
# container entry
dn: cn=automount,dc=example,dc=com
objectClass: nsContainer
objectClass: top
cn: automount

# master map entry
dn: automountMapName=auto.master,cn=automount,dc=example,dc=com
objectClass: automountMap
objectClass: top
automountMapName: auto.master

# shares map entry
dn: automountMapName=auto.shares,cn=automount,dc=example,dc=com
objectClass: automountMap
objectClass: top
automountMapName: auto.shares

# shares mount point
dn:
automountKey=/shares,automountMapName=auto.master,cn=automount,dc=exampl
e,dc=com
objectClass: automount
objectClass: top
automountKey: /shares
automountInformation: auto.shares

# pub mount point
dn:
automountKey=pub,automountMapName=auto.shares,cn=automount,dc=example,dc
=com
objectClass: automount
objectClass: top
automountKey: pub
automountInformation: filer.example.com:/pub
description: pub
```

The schema elements, then, match up to the structure like this (with the RFC 2307 schema):



```
objectclass: automount                          objectclass: automountMap
mount point name attribute: automountKey        filename attribute:
automountMapName
map name attribute: automountInformation            |
                                        |
                                                    |
                                                nfs.example.com:pub
                                                objectclass: automount
                                                mount point name attribute:
automountKey
                                                fileserver attribute:
automountInformation
```

autofs uses those schema elements to derive the automount configuration. The **/etc/sysconfig/autofs** file identifies the LDAP server, directory location, and schema elements used for automount entities:

```
LDAP_URI=ldap://ldap.example.com
SEARCH_BASE="cn=automount,dc=example,dc=com"
MAP_OBJECT_CLASS="automountMap"
ENTRY_OBJECT_CLASS="automount"
MAP_ATTRIBUTE="automountMapName"
ENTRY_ATTRIBUTE="automountKey"
VALUE_ATTRIBUTE="automountInformation"
```

Rather than pointing the automount configuration to the LDAP directory, it can be configured to point to SSSD. SSSD, then, stores all of the information that automount needs, and as a user attempts to mount a directory, that information is cached into SSSD. This offers several advantages for configuration — such as failover, service discovery, and timeouts — as well as performance improvements by reducing the number of connections to the LDAP server. Most important, using SSSD allows all mount information to be cached, so that clients can still successfully mount directories *even if the LDAP server goes offline*.

7.2.3.2. Configuring autofs Services in SSSD

1. Make sure that the **autofs** package is installed.
2. Open the **sssd.conf** file.

   ```
   [root@server ~]# vim /etc/sssd/sssd.conf
   ```

3. Add the **autofs** service to the list of services that SSSD manages.

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

4. Create a new **[autofs]** service configuration section. This section can be left blank; there is only one configurable option, for timeouts for negative cache hits.

   This section is required, however, for SSSD to recognize the **autofs** service and supply the default configuration.

```
[autofs]
```

5. The automount information is read from a configured LDAP domain in the SSSD configuration, so an LDAP domain must be available. If no additional settings are made, then the configuration defaults to the RFC 2307 schema and the LDAP search base (**ldap_search_base**) for the automount information. This can be customized:

   - The directory type, **autofs_provider**; this defaults to the **id_provider** value; a value of *none* explicitly disables autofs for the domain.
   - The search base, **ldap_autofs_search_base**.
   - The object class to use to recognize map entries, **ldap_autofs_map_object_class**
   - The attribute to use to recognize map names, **ldap_autofs_map_name**
   - The object class to use to recognize mount point entries, **ldap_autofs_entry_object_class**
   - The attribute to use to recognize mount point names, **ldap_autofs_entry_key**
   - The attribute to use for additional configuration information for the mount point, **ldap_autofs_entry_value**

   For example:

```
[domain/LDAP]
...
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
```

6. Save and close the **sssd.conf** file.
7. Configure **autofs** to look for the automount map information in SSSD by editing the **nsswitch.conf** file and changing the location from **ldap** to **sss**:

```
[root@server ~]# vim /etc/nsswitch.conf

automount: files sss
```

8. Restart SSSD.

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

7.2.4. Configuring Services: sudo

7.2.4.1. About sudo, LDAP, and SSSD

sudo rules are defined in the **sudoers** file, which must be distributed separately to every machine to maintain consistency.

One way for administrators to manage that for large environments is to store the **sudo** configuration in a central LDAP directory, and just configure each local system to point to that LDAP directory. That means that updates only need to be made in a single location, and any new rules are automatically recognized by local systems.

For **sudo**-LDAP configuration, each **sudo** rule is stored as an LDAP entry, with each component of the **sudo** rule defined in an LDAP attribute.

The **sudoers** rule looks like this:

```
Defaults    env_keep+=SSH_AUTH_SOCK
...
%wheel        ALL=(ALL)       ALL
```

The LDAP entry looks like this:

```
# sudo defaults
dn: cn=defaults,ou=SUDOers,dc=example,dc=com
objectClass: top
objectClass: sudoRole
cn: defaults
description: Default sudoOptions go here
sudoOption: env_keep+=SSH_AUTH_SOCK

# sudo rule
dn: cn=%wheel,ou=SUDOers,dc=example,dc=com
objectClass: top
objectClass: sudoRole
cn: %wheel
sudoUser: %wheel
sudoHost: ALL
sudoCommand: ALL
```

Note

SSSD only caches **sudo** rules which apply to the local system, depending on the value of the ***sudoHost*** attribute. This can mean that the ***sudoHost*** value is set to ALL, uses a regular expression that matches the hostname, matches the systems netgroup, or matches the systems hostname, fully-qualified domain name, or IP address.

The **sudo** service can be configured to point to an LDAP server and to pull its rule configuration from those LDAP entries. Rather than pointing the **sudo** configuration to the LDAP directory, it can be configured to point to SSSD. SSSD, then, stores all of the information that **sudo** needs, and every time a user attempts a **sudo**-related operation, the latest **sudo** configuration can be pulled from the LDAP directory (through SSSD). SSSD, however, also caches all of the **sudo** rules, so that users can perform tasks, using that centralized LDAP configuration, *even if the LDAP server goes offline*.

7.2.4.2. Configuring sudo with SSSD

All of the SSSD **sudo** configuration options are listed in the **sssd-ldap(5)** man page.

To configure the **sudo** service:

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

```
[root@server ~]# vim /etc/sssd/sssd.conf
```

2. Add the **sudo** service to the list of services that SSSD manages.

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

3. Create a new **[sudo]** service configuration section. This section can be left blank; it is required for SSSD to recognize the **sudo** service and supply the default configuration.

```
[sudo]
```

4. The **sudo** information is read from a configured LDAP domain in the SSSD configuration, so an LDAP domain must be available. For an LDAP provider, these parameters are required:

   - The directory type, **sudo_provider**; this is always **ldap**.
   - The search base, **ldap_sudo_search_base**.
   - The URI for the LDAP server, **ldap_uri**.

   For example:

```
[domain/LDAP]
id_provider = ldap

sudo_provider = ldap
ldap_uri = ldap://example.com
ldap_sudo_search_base = ou=sudoers,dc=example,dc=com
```

   Setting IdM as the ID provider automatically enables the sudo provider, so it is not necessary to specify **sudo_provider = ipa** in the configuration file.

```
[domain/IDM]
id_provider = ipa
ipa_domain = example.com
ipa_server = ipa.example.com
```

5. Set the intervals to use to refresh the **sudo** rule cache.

   The cache *for a specific system user* is always checked and updated whenever that user performs a task. However, SSSD caches all rules which relate to the local system. That complete cache is updated in two ways:

   - Incrementally, meaning only changes to rules since the last full update (**ldap_sudo_smart_refresh_interval**, the time in seconds); the default is 15 minutes,

- Fully, which dumps the entire caches and pulls in all of the current rules on the LDAP server(**ldap_sudo_full_refresh_interval**, the time in seconds); the default is six hours.

These two refresh intervals are set separately. For example:

```
[domain/LDAP]
...
ldap_sudo_full_refresh_interval=86400
ldap_sudo_smart_refresh_interval=3600
```

Note

SSSD only caches **sudo** rules which apply to the local system. This can mean that the ***sudoHost*** value is set to ALL, uses a regular expression that matches the hostname, matches the systems netgroup, or matches the systems hostname, fully-qualified domain name, or IP address.

6. Optionally, set any values to change the schema used for **sudo** rules.

   Schema elements are set in the **ldap_sudorule_*** attributes. By default, all of the schema elements use the schema defined in sudoers.ldap; these defaults will be used in almost all deployments.

7. Save and close the **sssd.conf** file.

8. Configure **sudo** to look for rules configuration in SSSD by editing the **nsswitch.conf** file and adding the **sss** location:

```
[root@server ~]# vim /etc/nsswitch.conf

sudoers: files sss
```

9. Restart SSSD.

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

7.2.5. Configuring Services: OpenSSH and Cached Keys

OpenSSH creates secure, encrypted connections between two systems. One machine authenticates to another machine to allow access; the authentication can be of the machine itself for server connections or of a user on that machine.

This authentication is performed through *public-private key pairs* that identify the authenticating user or machine. The remote machine or user attempting to access the machine presents a key pair. The local machine then elects whether to trust that remote entity; if it is trusted, the public key for that remote machine is stored in the **known_hosts** file or for the remote user in **authorized_keys**. Whenever that remote machine or user attempts to authenticate again, the local system simply checks the **known_hosts** or **authorized_keys** file first to see if that remote entity is recognized and trusted. If it is, then access is granted.

The first problem comes in verifying those identities reliably.

The **known_hosts** file is a triplet of the machine name, its IP address, and its public key:

```
server.example.com,172.16.0.1 ssh-rsa
AbcdEfg1234ZYX098776/AbcdEfg1234ZYX098776/AbcdEfg1234ZYX098776=
```

The **known_hosts** file can quickly become outdated for a number of different reasons: systems using DHCP cycle through IP addresses, new keys can be re-issued periodically, or virtual machines or services can be brought online and removed. This changes the hostname, IP address, and key triplet.

Administrators have to clean and maintain a current **known_hosts** file to maintain security. (Or system users get in the habit of simply accepting any machine and key presented, which negates the security benefits of key-based security.)

Additionally, a problem for both machines and users is distributing keys in a scalable way. Machines can send their keys as part of establishing an encrypted session, but users have to supply their keys in advance. Simply propagating and then updating keys consistently is a difficult administrative task.

Lastly, SSH key and machine information are only maintained locally. There may be machines or users on the network which are recognized and trusted by some systems and not by others because the **known_hosts** file has not been updated uniformly.

The goal of SSSD is to server as a credentials cache. This includes working as a credentials cache for SSH public keys for machines and users. OpenSSH is configured to reference SSSD to check for cached keys; SSSD uses Red Hat Linux's Identity Management (IPA) domain as an identity, and Identity Management actually stores the public keys and host information.

Note

Only Linux machines enrolled, or joined, in the Identity Management domain can use SSSD as a key cache for OpenSSH. Other Unix machines and Windows machines must use the regular authentication mechanisms with the **known_hosts** file.

7.2.5.1. Configuring OpenSSH to Use SSSD for Host Keys

OpenSSH is configured in either a user-specific configuration file (**~/.ssh/config**) or a system-wide configuration file (**/etc/ssh/sshd_config**). The user file has precedence over the system settings and the first obtained value for a parameter is used.

In order to manage host keys, SSSD has a tool, **sss_ssh_knownhostsproxy**, which performs three operations:

1. Retrieves the public host key from the enrolled Linux system.
2. Stores the host key in a custom hosts file, **/var/lib/sss/pubconf/known_hosts**.
3. Establishes a connection with the host machine, either a socket (the default) or a secure connection.

This tool has the format:

```
sss_ssh_knownhostsproxy [-d sssd_domain] [-p ssh_port] HOSTNAME
[PROXY_COMMAND]
```

Table 7.4. sss_ssh_knownhostsproxy Options

Short Argument	Long Argument	Description
	HOSTNAME	Gives the hostname of the host to check and connect to. In the OpenSSH configuration file, this can be a token, **%h**.
	PROXY_COMMAND	Passes a proxy command to use to connect to the SSH client. This is similar to running **ssh -o ProxyCommand=***value*. This option is used when running **sss_ssh_knownhostsproxy** from the command line or through another script, but is not necessary in the OpenSSH configuration file.
-d *sssd_domain*	--domain *sssd_domain*	Only searches for public keys in entries in the specified domain. If not given, SSSD searches for keys in all configured domains.
-p *port*	--port *port*	Uses this port to connect to the SSH client. By default, this is port 22.

To use this SSSD tool, add or edit two parameters to the **/etc/ssh/sshd_config** or **~/.ssh/config** file:

- Specify the command to use to connect to the SSH client (**ProxyCommand**). This is the **sss_ssh_knownhostsproxy**, with the desired arguments and hostname.
- Specify the location of the SSSD hosts file, rather than the default **known_hosts** file (**GlobalKnownHostsFile**). The SSSD hosts file is **/var/lib/sss/pubconf/known_hosts**.

For example, this looks for public keys in the SSSD domain and connects over whatever port and host are supplied:

```
ProxyCommand /usr/bin/sss_ssh_knownhostsproxy -p %p %h
GlobalKnownHostsFile /var/lib/sss/pubconf/known_hosts
```

7.2.5.2. Configuring OpenSSH to Use SSSD for User Keys

User keys are stored on a local system in the **authorized_keys** file for OpenSSH. As with hosts, SSSD can maintain and automatically update a separate cache of user public keys for OpenSSH to refer to. This is kept in the **.ssh/sss_authorized_keys** file.

OpenSSH is configured in either a user-specific configuration file (**~/.ssh/config**) or a system-wide configuration file (**/etc/ssh/sshd_config**). The user file has precedence over the system settings and the first obtained value for a parameter is used.

In order to manage user keys, SSSD has a tool, **sss_ssh_authorizedkeys**, which performs two operations:

1. Retrieves the user's public key from the user entries in the Identity Management (IPA) domain.

2. Stores the user key in a custom file, **.ssh/sss_authorized_keys**, in the standard authorized keys format.

This tool has the format:

```
sss_ssh_authorizedkeys [-d sssd_domain] USER
```

Table 7.5. sss_ssh_authorizedkeys Options

Short Argument	Long Argument	Description
	USER	Gives the username or account name for which to obtain the public key. In the OpenSSH configuration file, this can be represented by a token, **%u**.
-d *sssd_domain*	--domain *sssd_domain*	Only searches for public keys in entries in the specified domain. If not given, SSSD searches for keys in all configured domains.

To configure OpenSSH to use SSSD for user keys, use the authorized key command. Specify the command to run to retrieve user keys and the user under whose account it is run. For example:

```
AuthorizedKeysCommand /usr/bin/sss_ssh_authorizedkeys
AuthorizedKeysCommandUser nobody
```

7.3. SSSD and Identity Providers (Domains)

SSSD recognizes *domains*, which are entries within the SSSD configuration file associated with different, external data sources. Domains are a combination of an identity provider (for user information) and, optionally, other providers such as authentication (for authentication requests) and for other operations, such as password changes. (The identity provider can also be used for all operations, if all operations are performed within a single domain or server.)

SSSD works with different LDAP identity providers (including OpenLDAP, Red Hat Directory Server, and Microsoft Active Directory) and can use native LDAP authentication, Kerberos authentication, or provider-specific authentication protocols (such as Active Directory).

A domain configuration defines the *identity provider*, the *authentication provider*, and any specific configuration to access the information in those providers. There are several types of identity and authentication providers:

- LDAP, for general LDAP servers
- Active Directory (an extension of the LDAP provider type)
- Identity Management (an extension of the LDAP provider type)
- Local, for the local SSSD database
- Proxy
- Kerberos (authentication provider only)

The identity and authentication providers can be configured in different combinations in the domain

entry. The possible combinations are listed in Table 7.6, “Identity Store and Authentication Type Combinations”.

Table 7.6. Identity Store and Authentication Type Combinations

Identification Provider	Authentication Provider
Identity Management (LDAP)	Identity Management (LDAP)
Active Directory (LDAP)	Active Directory (LDAP)
Active Directory (LDAP)	Kerberos
LDAP	LDAP
LDAP	Kerberos
proxy	LDAP
proxy	Kerberos
proxy	proxy

Along with the domain entry itself, the domain name must be added to the list of domains that SSSD will query. For example:

```
[sssd]
domains = LOCAL,Name
...

[domain/Name]
id_provider = type
auth_provider = type
provider_specific = value
global = value
```

global attributes are available to any type of domain, such as cache and time out settings. Each identity and authentication provider has its own set of required and optional configuration parameters.

Table 7.7. General [domain] Configuration Parameters

Parameter	Value Format	Description

Parameter	Value Format	Description
id_provider	string	Specifies the data backend to use for this domain. The supported identity backends are: - ldap - ipa (Identity Management in Red Hat Enterprise Linux) - ad (Microsoft Active Directory) - proxy, for a legacy NSS provider, such as **nss_nis**. Using a proxy ID provider also requires specifying the legacy NSS library, which is set in the **proxy_lib_name** option. - local, the SSSD internal local provider
auth_provider	string	Sets the authentication provider used for the domain. The default value for this option is the value of **id_provider**. The supported authentication providers are ldap, ipa, ad, krb5 (Kerberos), proxy, and none.
min_id,max_id	integer	*Optional.* Specifies the UID and GID range for the domain. If a domain contains entries that are outside that range, they are ignored. The default value for **min_id** is **1**; the default value for **max_id** is **0**, which is unlimited. **Important** The default **min_id** value is the same for all types of identity provider. If LDAP directories are using UID numbers that start at one, it could cause conflicts with users in the local **/etc/passwd** file. To avoid these conflicts, set **min_id** to **1000** or higher as possible.

Parameter	Value Format	Description
cache_credentials	Boolean	*Optional.* Specifies whether to store user credentials in the local SSSD domain database cache. The default value for this parameter is **false**. Set this value to **true** for domains other than the LOCAL domain to enable offline authentication.
entry_cache_timeout	integer	*Optional.* Specifies how long, in seconds, SSSD should cache *positive* cache hits. A positive cache hit is a successful query.
use_fully_qualified_names	Boolean	*Optional.* Specifies whether requests to this domain require fully-qualified domain names. If set to **true**, all requests to this domain must use fully-qualified domain names. It also means that the output from the request displays the fully-qualified name. Restricting requests to fully-qualified user names allows SSSD to differentiate between domains with users with conflicting usernames. If ***use_fully_qualified_names*** is set to **false**, it is possible to use the fully-qualified name in the requests, but only the simplified version is displayed in the output. SSSD can only parse names based on the domain name, not the realm name. The same name can be used for both domains and realms, however.

7.3.1. Creating an LDAP Identity Provider

An LDAP domain simply means that SSSD uses an LDAP directory as the identity provider (and, optionally, also as an authentication provider). SSSD supports several major directory services:

- Red Hat Directory Server
- OpenLDAP
- Identity Management (IdM or IPA)
- Microsoft Active Directory 2008 R2

Note

All of the parameters available to a general LDAP identity provider are also available to Identity Management and Active Directory identity providers, which are subsets of the LDAP provider.

7.3.1.1. Parameters for Configuring an LDAP Domain

An LDAP directory can function as both an identity provider and an authentication provider. The configuration requires enough information to identify and connect to the user directory in the LDAP server, but the way that those connection parameters are defined is flexible.

Other options are available to provide more fine-grained control, like specifying a user account to use to connect to the LDAP server or using different LDAP servers for password operations. The most common options are listed in Table 7.8, “LDAP Domain Configuration Parameters”.

Note

Many other options are listed in the man page for LDAP domain configuration, **sssd-ldap(5)**.

Table 7.8. LDAP Domain Configuration Parameters

Parameter	Description
ldap_uri	Gives a comma-separated list of the URIs of the LDAP servers to which SSSD will connect. The list is given in order of preference, so the first server in the list is tried first. Listing additional servers provides failover protection. This can be detected from the DNS SRV records if it is not given.
ldap_search_base	Gives the base DN to use for performing LDAP user operations. **Important** If used incorrectly, ***ldap_search_base*** might cause SSSD lookups to fail. With an AD provider, setting ***ldap_search_base*** is not required. The AD provider automatically discovers all the necessary information. Red Hat recommends not to set the parameter in this situation and instead rely on what the AD provider discovers.

Parameter	Description
ldap_tls_reqcert	Specifies how to check for SSL server certificates in a TLS session. There are four options: - *never* disables requests for certificates. - *allow* requests a certificate, but proceeds normally even if no certificate is given or a bad certificate is given. - *try* requests a certificate and proceeds normally if no certificate is given, If a bad certificate is given, the session terminates. - *demand* and *hard* are the same option. This requires a valid certificate or the session is terminated. The default is *hard*.
ldap_tls_cacert	Gives the full path and file name to the file that contains the CA certificates for all of the CAs that SSSD recognizes. SSSD will accept any certificate issued by these CAs. This uses the OpenLDAP system defaults if it is not given explicitly.
ldap_referrals	Sets whether SSSD will use LDAP referrals, meaning forwarding queries from one LDAP database to another. SSSD supports database-level and subtree referrals. For referrals within the same LDAP server, SSSD will adjust the DN of the entry being queried. For referrals that go to different LDAP servers, SSSD does an exact match on the DN. Setting this value to **true** enables referrals; this is the default. Referrals can negatively impact overall performance because of the time spent attempting to trace referrals. Disabling referral checking can significantly improve performance.

Parameter	Description

ldap_schema	Sets what version of schema to use when searching for user entries. This can be **rfc2307**, **rfc2307bis**, **ad**, or **ipa**. The default is **rfc2307**.
	
	In RFC 2307, group objects use a multi-valued attribute, ***memberuid***, which lists the names of the users that belong to that group. In RFC 2307bis, group objects use the ***member*** attribute, which contains the full distinguished name (DN) of a user or group entry. RFC 2307bis allows nested groups using the ***member*** attribute. Because these different schema use different definitions for group membership, using the wrong LDAP schema with SSSD can affect both viewing and managing network resources, even if the appropriate permissions are in place.
	
	For example, with RFC 2307bis, all groups are returned when using nested groups or primary/secondary groups.
	
	`$ id`
	`uid=500(myserver)`
	`gid=500(myserver)`
	`groups=500(myserver),510(myotherg`
	`roup)`
	
	If SSSD is using RFC 2307 schema, only the primary group is returned.
	
	This setting only affects how SSSD determines the group members. It does not change the actual user data.

ldap_search_timeout	Sets the time, in seconds, that LDAP searches are allowed to run before they are canceled and cached results are returned. When an LDAP search times out, SSSD automatically switches to offline mode.
ldap_rfc2307_fallback_to_local_users	Sets whether to check the local system users (**/etc/passwd**) if an LDAP group member is not found in the LDAP directory. This allows local system users to be added to LDAP groups. If this is set to false (the default), then any local user is deleted when running **id** with an LDAP provider, because SSSD uses only the LDAP user accounts for identities.

Parameter	Description
ldap_network_timeout	Sets the time, in seconds, SSSD attempts to poll an LDAP server after a connection attempt fails. The default is six seconds.
ldap_opt_timeout	Sets the time, in seconds, to wait before aborting synchronous LDAP operations if no response is received from the server. This option also controls the timeout when communicating with the KDC in case of a SASL bind. The default is five seconds.

7.3.1.2. Configuring an LDAP Identity Provider

The LDAP configuration is very flexible, depending on your specific environment and the SSSD behavior. These are some common examples of an LDAP domain, but the SSSD configuration is not limited to these examples.

Note

Along with creating the domain entry, add the new domain to the list of domains for SSSD to query in the **sssd.conf** file. For example:

```
domains = LOCAL,LDAP1,AD,PROXYNIS
```

Example 7.2. A Basic LDAP Domain Configuration

An LDAP domain requires three things:

- An LDAP server
- The search base
- A way to establish a secure connection

The last item depends on the LDAP environment. SSSD requires a secure connection since it handles sensitive information. This connection can be a dedicated TLS/SSL connection or it can use Start TLS.

Using a dedicated TLS/SSL connection simply uses an LDAPS connection to connect to the server and is therefore set as part of the **ldap_uri** option:

```
# An LDAP domain
[domain/LDAP]
cache_credentials = true

id_provider = ldap
auth_provider = ldap

ldap_uri = ldaps://ldap.example.com:636
ldap_search_base = dc=example,dc=com
```

Using Start TLS requires a way to input the certificate information to establish a secure connection dynamically over an insecure port. This is done using the **ldap_id_use_start_tls** option to use Start TLS and then **ldap_tls_cacert** to identify the CA certificate which issued the SSL server certificates.

```
# An LDAP domain
[domain/LDAP]
cache_credentials = true

id_provider = ldap
auth_provider = ldap

ldap_uri = ldap://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
```

7.3.2. Creating an Identity Management (IdM) Identity Provider

The Identity Management (IdM or IPA) identity provider is an extension of a generic LDAP provider. All of the configuration options for an LDAP provider are available to the IdM provider, as well as some additional parameters which allow SSSD to work as a client of the IdM domain and extend IdM functionality.

Identity Management can work as a provider for identities, authentication, access control rules, and passwords, all of the ***_provider** parameters for a domain. Additionally, Identity Management has configuration options within its own domain to manage SELinux policies, automount information, and host-based access control. All of those features in IdM domains can be tied to SSSD configuraiton, allowing those security-related policies to be applied and cached for system users.

Example 7.3. Basic IdM Provider

An IdM provider, like an LDAP provider, can be set to serve several different services, including identity, authentication, and access control

For IdM servers, there are two additional settings which are very useful (although not required):

- With the **id_provider = ipa** setting, use **ldap_schema = ipa**. The **rfc2307** default schema value is used only for **id_provider = ldap**.
- Set SSSD to update the Identity Management domain's DNS server with the IP address of this client when the client first connects to the IdM domain.

```
[sssd]
domains = local,example.com
...

[domain/example.com]
id_provider = ipa
ipa_server = ipaserver.example.com
ipa_hostname = ipa1.example.com
auth_provider = ipa
access_provider = ipa
```

```
chpass_provider = ipa

# set which schema to use
ldap_schema = ipa

# automatically update IdM DNS records
dyndns_update = true
```

Identity Management defines and maintains security policies and identities for users across a Linux domain. This includes access control policies, SELinux policies, and other rules. Some of these elements in the IdM domain interact directly with SSSD, using SSSD as an IdM client — and those features can be managed in the IdM domain entry in **sssd.conf**.

Most of the configuration parameters relate to setting schema elements (which is not relevant in most deployments because IdM uses a fixed schema) and never need to be changed. In fact, none of the features in IdM require client-side settings. But there may be circumstances where tweaking the behavior is helpful.

Example 7.4. IdM Provider with SELinux

IdM can define SELinux user policies for system users, so it can work as an SELinux provider for SSSD. This is set in the **selinux_provider** parameter. The provider defaults to the **id_provider** value, so this is not necessary to set explicitly to *support* SELinux rules. However, it can be useful to explicitly *disable* SELinux support for the IdM provider in SSSD.

```
selinux_provider = ipa
```

Example 7.5. IdM Provider with Host-Base Access Control

IdM can define host-based access controls, restricting access to services or entire systems based on what host a user is using to connect or attempting to connect to. This rules can be evaluated and enforced by SSSD as part of the access provider behavior.

For host-based access controls to be in effect, the Identity Management server must be the access provider, at a minimum.

There are two options which can be set for how SSSD evaluates host-based access control rules:

- SSSD can evaluate what machine (source host) the user is using to connect to the IdM resource; this is disabled by default, so that only the target host part of the rule is evaluated.
- SSSD can refresh the host-based access control rules in its cache at a specified interval.

For example:

```
access_provider = ipa
ipa_hbac_refresh = 120

# check for source machine rules; disabled by default
```

Example 7.6. Identity Management with Cross-Realm Kerberos Trusts

Identity Management (IdM or IPA) can be configured with trusted relationships between Active Directory DNS domains and Kerberos realms. This allows Active Directory users to access services and hosts on Linux systems.

There are two configuration settings in SSSD that are used with cross-realm trusts:

- A service that adds required data to Kerberos tickets
- A setting to support subdomains

Kerberos Ticket Data

Microsoft uses a special authorization structure called *privileged access certificates* or MS-PAC. A PAC is embedded in a Kerberos ticket as a way of identifying the entity to other Windows clients and servers in the Windows domain.

SSSD has a special PAC service which generates the additional data for Kerberos tickets. When using an Active Directory domain, it may be necessary to include the PAC data for Windows users. In that case, enable the **pac** service in SSSD:

```
[sssd]
services = nss, pam, pac
...
```

Windows Subdomains

Normally, a domain entry in SSSD corresponds directly to a single identity provider. However, with IdM cross-realm trusts, the IdM domain can trust another domain, so that the domains are transparent to each other. SSSD can follow that trusted relationship, so that if an IdM domain is configured, any Windows domain is also automatically searched and supported by SSSD — without having to be configured in a domain section in SSSD.

This is configured by adding the **subdomains_provider** parameter to the IdM domain section. This is actually an optional parameter; if a subdomain is discovered, then SSSD defaults to using the **ipa** provider type. However, this parameter can also be used to disable subdomain fetches by setting a value of **none**.

```
[domain/IDM]
...
subdomains_provider = ipa
get_domains_timeout = 300
```

7.3.3. Creating an Active Directory Identity Provider

The most basic type of domain is an LDAP domain. Any LDAPv3 directory server can be configured as an LDAP identity provider for an SSSD domain. Some specialty LDAP services have additional, specific configuration, which can either simplify service-specific configuration or supply service-specific functionality. One of those identity provider types is for Active Directory.

As shown in Example 7.1, “Simple sssd.conf File”, the SSSD configuration file has three major sections: the first configures the SSSD service (**[sssd]**), the second configures system services which will use SSSD as an identity cache (such as **[nss]** and **[pam]**), and the third section configures the identity domains (**[domain/NAME]**).

By default, only an identity provider really needs to be configured — the identity provider is used for the authentication, access (authorization), and password providers if no other types or servers are identified. Active Directory can be configured as any kind of provider using the **ad** option.

```
[domain/ADEXAMPLE]
id_provider = ad
auth_provider = ad
access_provider = ad
chpass_provider = ad

ad_server = dc1.example.com
ad_hostname = client.example.com
ad_domain = example.com
```

The connection information is required to identify what Active Directory server to use.

In addition to that basic configuration, the Active Directory identity provider can be configured specifically for the Active Directory environment and specific features, such as how to use POSIX attributes or mapping for Windows SIDs on the local system, failover servers, and account information such as home directories.

All of the LDAP domain parameters are available to the Active Directory provider, as well as Active Directory-specific configuration parameters. The complete lists are available in the sssd-ldap and sssd-ad man pages.

There are a number of options in the generic LDAP provider configuration which can be used to configure an Active Directory provider. Using the **ad** value is a short-cut which automatically pulls in the parameters and values to configure a given provider for Active Directory.

For example, the shortcut for an access provider is:

```
access_provider = ad
```

Using generic LDAP parameters, that configuration expands to:

```
access_provider = ldap
ldap_access_order = expire
ldap_account_expire_policy = ad
```

Those settings are all set implicitly by using the **ad** provider type.

7.3.3.1. About Active Directory Identities on the Local System

Active Directory can replicate user entries and attributes from its local directory into a *global catalog*, which makes the information available to other domains within the forest. SSSD checks this global catalog for information about users and groups, so information is not limited to a single Active Directory domain or subdomain — SSSD, too, has access to all user data for all domains within the topology.

SSSD, then, can be used by applications which need to query the Active Directory global catalog for user or group information.

There are inherent structural differences between how Windows and Linux handle system users and in the user schemas used in Active Directory and standard LDAPv3 directory services. When using an Active Directory identity provider with SSSD to manage system users, it is necessary to reconcile the Active Directory-style user to the new SSSD user. There are two ways to do this:

- Using ID mapping on SSSD to create a map between Active Directory security IDs (SIDs) and the generated UIDs on Linux.

  ID mapping is the simplest option for most environments because it requires no additional packages or configuration on Active Directory.

- Using Services for Unix to insert POSIX attributes on Windows user and group entries, and then having those attributes pulled into PAM/NSS.

  This requires more configuration and information within the Active Directory environment, but it gives more administrative control over the specific UID/GID values (and other POSIX attributes).

7.3.3.1.1. About Security ID Mapping

7.3.3.1.1.1. The Mechanism of ID Mapping

Linux/Unix systems use a local user ID number and group ID number to identify users on the system. These UID:GID numbers are a simple integer, such as *501:501*. These numbers are simple because they are always created and administered locally, even for systems which are part of a larger Linux/Unix domain.

Microsoft Windows and Active Directory use a different user ID structure to identify users, groups, and machines. Each ID is constructed of different segments that identify the security version, the issuing authority type, the machine, and the identity itself. For example:

```
S-1-5-21-3623811015-3361044348-30300820-1013
```

The third through sixth blocks are the machine identifier:

```
S-1-5-21-3623811015-3361044348-30300820-1013
```

The last block is the *relative identifier* (RID) which identifies the specific entity:

```
S-1-5-21-3623811015-3361044348-30300820-1013
```

A range of possible ID numbers are always assigned to SSSD. (This is a local range, so it is the same for every machine.)

```
|_____________________________|
|                             |
minimum ID                    max ID
```

This range is divided into 10,000 sections (by default), with each section allocated 200,000 IDs.

```
| slice 1 | slice 2 |   ...   |
|_________|_________|_________|
|         |         |         |
minimum ID                    max ID
```

When a new Active Directory domain is detected, the SID is hashed. Then, SSSD takes the modulus of the hash and the number of available sections to determine which ID section to assign to the Active Directory domain. This is a reliably consistent means of assigning ID sections, so the same ID range is assigned to the same Active Directory domain on most client machines.

```
| Active  | Active  |         |
```

```
|Directory|Directory|          |
|domain 1 |domain 2 |   ...    |
|         |         |          |
| slice 1 | slice 2 |   ...    |
|_________|_________|__________|
|         |         |          |
minimum ID                    max ID
```

Note

While the method of assigning ID sections is consistent, **ID mapping is based on the order that an Active Directory domain is encountered on a client machine** — so it may not result in consistent ID range assignments on all Linux client machines. If consistency is required, then consider disabling ID mapping and using explicit POSIX attributes.

7.3.3.1.1.2. ID Mapping Parameters

ID mapping is enabled in two parameters, one to enable the mapping and one to load the appropriate Active Directory user schema:

```
ldap_id_mapping = True
ldap_schema = ad
```

Note

When ID mapping is enabled, the ***uidNumber*** and ***gidNumber*** attributes are ignored. This prevents any manually-assigned values. If *any* values must be manually assigned, then *all* values must be manually assigned, and ID mapping should be disabled.

7.3.3.1.1.3. Mapping Users

When an Active Directory user attempts to log into a local system service for the first time, an entry for that user is created in the SSSD cache. The remote user is set up much like a system user:

- A system UID is created for the user based on his SID and the ID range for that domain.
- A GID is created for the user, which is identical to the UID.
- A private group is created for the user.
- A home directory is created, based on the home directory format in the **sssd.conf** file.
- A shell is created, according to the system defaults or the setting in the **sssd.conf** file.
- If the user belongs to any groups in the Active Directory domain, then, using the SID, SSSD adds the user to those groups on the Linux system.

7.3.3.1.2. About SSSD and POSIX Attributes

Active Directory can be configured to create and store POSIX attributes such as ***uidNumber***, ***gidNumber***, ***unixHomeDirectory***, and ***loginShell***. As with all user attributes, these are originally stored in the local domain, but they can be replicated to the global catalog — and once

they are in the global catalog, they are available to SSSD and any application which uses SSSD for its identity information.

Important

When SSSD uses the POSIX attributes directly, they must be published to the Active Directory global catalog. SSSD queries the global catalog for user information.

When POSIX attributes are already defined in Active Directory, then it is not recommended to use the SID/UID mapping as described in Section 7.3.3.1.1, “About Security ID Mapping”. The UID and GID numbers are already defined, and mapping creates new, different numbers. The best solution in that situation is to use the UID and GID numbers as defined in Active Directory and then apply that to the local Linux accounts managed by SSSD.

To use existing POSIX attributes, two things must be configured:

- The POSIX attributes must be published to Active Directory's global catalog.
- ID mapping (**`ldap_id_mapping`** in the Active Directory domain entry) must be disabled in SSSD.

```
ldap_id_mapping = False
```

7.3.3.1.3. Active Directory Users and Range Retrieval Searches

Microsoft Active Directory has an attribute, ***`MaxValRange`***, which sets a limit on how many values for a multi-valued attribute will be returned. This is the *range retrieval* search extension. Essentially, this runs multiple mini-searches, each returning a subset of the results within a given range, until all matches are returned.

For example, when doing a search for the ***`member`*** attribute, each entry could have multiple values, and there can be multiple entries with that attribute. If there are 2000 matching results (or more), then ***`MaxValRange`*** limits how many are displayed at once; this is the value range. The given attribute then has an additional flag set, showing which range in the set the result is in:

```
attribute:range=low-high:value
```

For example, results 100 to 500 in a search:

```
member;range=99-499: cn=John Smith...
```

This is described in the Microsoft documentation at http://msdn.microsoft.com/en-us/library/cc223242.aspx.

SSSD supports range retrievals with Active Directory providers as part of user and group management, without any additional configuration.

However, some LDAP provider attributes which are available to configure searches — such as **`ldap_user_search_base`** — are not performant with range retrievals. Be cautious when configuring search bases in the Active Directory provider domain and consider what searches may trigger a range retrieval.

7.3.3.1.4. Linux Clients and Active Directory DNS Sites

SSSD connects a local Linux system to a larger Active Directory environment. This requires that SSSD have an awareness of possible configurations within the Active Directory forest and work with them so that the Linux client is cleanly integrated.

Active Directory forests can be very large, with numerous different domain controllers, domains and subdomains, and physical sites. To increase client performance, Active Directory uses specially-named DNS records to identify domain controllers within the same domain but at different physical locations. Clients connect to the closest domain controller.

Note

For information on DNS, see the Linux Domain Identity, Authentication, and Policy Guide. For information on how DNS and Active Directory work together in Red Hat Identity Management, see the Windows Integration Guide.

Active Directory extends normal DNS SRV records to identify a specific physical location or site for its domain controllers. Clients (such as SSSD) can determine which domain controllers to use based on their own site configuration.

SSSD can determine which domain controller to use by querying the Active Directory domain first for its site configuration, and then for the domain controller DNS records.

1. SSSD attempts to connect to the Active Directory domain and looks up any available domain controller through normal DNS discovery.
2. It retrieves a list of primary and fallback servers.
3. SSSD sends a special CLDAP ping to any domain controller. The ping is really an LDAP search which looks for the DNS domain, domain SID, and version:

   ```
   (&(&(DnsDomain=ad.domain)(DomainSid=S-1-5-21-1111-2222-3333))
   (NtVer=0x01000016))
   ```

   This is used to retrieve the information about the client's site (if one is configured).
4. If a site is configured for the client, then the reply contains extended DNS SRV records for the primary server, containing the site name (*site-name._sites.*):

   ```
   _service._protocol.site-name._sites.domain.name
   ```

   The backup server record is also sent, as a standard SRV record:

   ```
   _service._protocol.domain.name
   ```

   If no site is configured, then a standard SRV record is sent for all primary and backup servers.

7.3.3.2. Configuring an Active Directory Domain with ID Mapping

When configuring an Active Directory domain, the simplest configuration is to use the **ad** value for all providers (identity, access, password). Also, load the native Active Directory schema for user and group entries, rather than using the default RFC 2307.

Other configuration is available in the general LDAP provider configuration (sssd-ldap) and Active Directory-specific configuration (sssd-ad). This includes setting LDAP filters for a specific user or group subtree, filters for authentication, and values for some account settings. Some additional configuration is covered in Section 7.3.3.5, “Additional Configuration Examples”.

1. Make sure that both the Active Directory and Linux systems have a properly configured environment.
   - Name resolution must be properly configured, particularly if service discovery is used with SSSD.
   - The clocks on both systems must be in sync for Kerberos to work properly.
2. Set up the Linux system as an Active Directory client and enroll it within the Active Directory domain. This is done by configuring the Kerberos and Samba services on the Linux system.
   a. Set up Kerberos to use the Active Directory Kerberos realm.
      a. Open the Kerberos client configuration file.

```
[root@server ~]# vim /etc/krb5.conf
```

      b. Configure the **[logging]** and **[libdefaults]** sections so that they connect to the Active Directory realm.

```
[logging]
 default = FILE:/var/log/krb5libs.log

[libdefaults]
 default_realm = EXAMPLE.COM
 dns_lookup_realm = true
 dns_lookup_kdc = true
 ticket_lifetime = 24h
 renew_lifetime = 7d
 rdns = false
 forwardable = true
```

      If autodiscovery is not used with SSSD, then also configure the **[realms]** and **[domain_realm]** sections to explicitly define the Active Directory server.

   b. Configure the Samba server to connect to the Active directory server.
      a. Open the Samba configuration file.

```
[root@server ~]# vim /etc/samba/smb.conf
```

      b. Set the Active Directory domain information in the **[global]** section.

```
[global]
   workgroup = EXAMPLE
   client signing = yes
   client use spnego = yes
   kerberos method = secrets and keytab
```

```
    log file = /var/log/samba/%m.log
    password server = AD.EXAMPLE.COM
    realm = EXAMPLE.COM
    security = ads
```

c. Add the Linux machine to the Active Directory domain.

a. Obtain Kerberos credentials for a Windows administrative user.

```
[root@server ~]# kinit Administrator
```

b. Add the machine to the domain using the **net** command.

```
[root@server ~]# net ads join -k
Joined 'server' to dns domain 'example.com'
```

This creates a new keytab file, **/etc/krb5.keytab**.

List the keys for the system and check that the host principal is there.

```
[root@server ~]# klist -k
```

3. If necessary, install the **oddjob-mkhomedir** package to allow SSSD to create home directories for Active Directory users.

```
[root@server ~]# yum install oddjob-mkhomedir
```

4. Use **authconfig** to enable SSSD for system authentication. Use the **--enablemkhomedir** to enable SSSD to create home directories.

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

5. Open the SSSD configuration file.

```
[root@rhel-server ~]# vim /etc/sssd/sssd.conf
```

6. Configure the Active Directory domain.

a. In the **[sssd]** section, add the Active Directory domain to the list of active domains. This is the name of the domain entry that is set in *[domain/NAME]* in the SSSD configuration file.

Also, add **pac** to the list of services; this enables SSSD to set and use MS-PAC information on tickets used to communicate with the Active Directory domain.

```
[sssd]
config_file_version = 2
domains = ad.example.com
services = nss, pam, pac
```

b. Create a new domain section at the bottom of the file for the Active Directory domain. This section has the format *domain/NAME*, such as **domain/ad.example.com**. For each provider, set the value to **ad**, and give the connection information for the specific Active Directory instance to connect to.

```
[domain/ad.example.com]
id_provider = ad
ad_server = adserver.example.com
ad_hostname = client.example.com
auth_provider = ad
chpass_provider = ad
access_provider = ad
```

c. Enable credentials caching; this allows users to log into the local system using cached information, even if the Active Directory domain is unavailable.

```
cache_credentials = true
```

d. Configure access controls.

```
ldap_access_order = expire
ldap_account_expire_policy = ad
```

7. Restart the SSH service to load the new PAM configuration.

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

8. Start the SSSD service.

```
[root@rhel-server ~]# systemctl start sssd.service
```

7.3.3.3. Configuring an Active Directory Domain with POSIX Attributes

To use Active Directory-defined POSIX attributes in SSSD, those attributes must be replicated to the global catalog. That requires additional configuration on the Active Directory domain. Additionally, ID mapping must be **disabled** in SSSD, so the POSIX attributes are used from Active Directory rather than creating new settings locally.

Other configuration is available in the general LDAP provider configuration (sssd-ldap) and Active Directory-specific configuration (sssd-ad). This includes setting LDAP filters for a specific user or group subtree, filters for authentication, and values for some account settings. Some additional configuration is covered in Section 7.3.3.5, “Additional Configuration Examples”.

1. Make sure that both the Active Directory and Linux systems have a properly configured environment.
   - Name resolution must be properly configured, particularly if service discovery is used with SSSD.
   - The clocks on both systems must be in sync for Kerberos to work properly.
2. In the Active Directory domain, set the POSIX attributes to be replicated to the global catalog.

a. Install *Identity Management for UNIX Components* on all primary and child domain controllers. Full details are available in the Microsoft documentation at http://technet.microsoft.com/en-us/library/cc731178.aspx.

   This allows the POSIX attributes and related schema to be available to user accounts. These attributes are available in the **UNIX Attributes** tab in the entry's **Properties** menu.

b. Install the Active Directory Schema Snap-in to add attributes to be replicated to the global catalog. This is described in the Microsoft documentation at http://technet.microsoft.com/en-us/library/cc755885%28v=ws.10%29.aspx.

c. The full details for replicating schema are in the Microsoft documentation at http://support.microsoft.com/kb/248717.

   For the relevant POSIX attributes (***uidNumber***, ***gidNumber***, ***unixHomeDirectory***, and ***loginShell***), open the **Properties** menu, select the **Replicate this attribute to the Global Catalog** checkbox, and then click **OK**.

3. Set up the Linux system as an Active Directory client and enroll it within the Active Directory domain. This is done by configuring the Kerberos and Samba services on the Linux system.

   a. Set up Kerberos to use the Active Directory Kerberos realm.

      a. Open the Kerberos client configuration file.

```
[root@server ~]# vim /etc/krb5.conf
```

      b. Configure the **[logging]** and **[libdefaults]** sections so that they connect to the Active Directory realm.

```
[logging]
 default = FILE:/var/log/krb5libs.log

[libdefaults]
 default_realm = EXAMPLE.COM
 dns_lookup_realm = true
 dns_lookup_kdc = true
 ticket_lifetime = 24h
 renew_lifetime = 7d
 rdns = false
 forwardable = true
```

      If autodiscovery is not used with SSSD, then also configure the **[realms]** and **[domain_realm]** sections to explicitly define the Active Directory server.

   b. Configure the Samba server to connect to the Active directory server.

      a. Open the Samba configuration file.

```
[root@server ~]# vim /etc/samba/smb.conf
```

      b. Set the Active Directory domain information in the **[global]** section.

```
[global]
   workgroup = EXAMPLE
   client signing = yes
```

```
    client use spnego = yes
    kerberos method = secrets and keytab
    log file = /var/log/samba/%m.log
    password server = AD.EXAMPLE.COM
    realm = EXAMPLE.COM
    security = ads
```

c. Add the Linux machine to the Active Directory domain.

a. Obtain Kerberos credentials for a Windows administrative user.

```
[root@server ~]# kinit Administrator
```

b. Add the machine to the domain using the **net** command.

```
[root@server ~]# net ads join -k
Joined 'server' to dns domain 'example.com'
```

This creates a new keytab file, **/etc/krb5.keytab**.

c. List the keys for the system and check that the host principal is there.

```
[root@server ~]# klist -ke
```

d. Test that users can search the global catalog, using an **ldapsearch**.

```
[root@server ~]# ldapsearch -H
ldap://server.ad.example.com:3268 -Y GSSAPI -N -b
"dc=ad,dc=example,dc=com" "(&(objectClass=user)
(sAMAccountName=aduser))"
```

4. Install the **sssd-ad** package.

```
[root@server ~]# yum install sssd-ad
```

5. Open the SSSD configuration file.

```
[root@rhel-server ~]# vim /etc/sssd/sssd.conf
```

6. Configure the Active Directory domain.

a. In the **[sssd]** section, add the Active Directory domain to the list of active domains. This is the name of the domain entry that is set in *[domain/NAME]* in the SSSD configuration file.

Also, add **pac** to the list of services; this enables SSSD to set and use MS-PAC information on tickets used to communicate with the Active Directory domain.

```
[sssd]
config_file_version = 2
domains = ad.example.com
services = nss, pam, pac
```

b. Create a new domain section at the bottom of the file for the Active Directory domain. This section has the format *domain/NAME*, such as **domain/ad.example.com**. For each provider, set the value to **ad**, and give the connection information for the specific Active Directory instance to connect to.

```
[domain/ad.example.com]
id_provider = ad
ad_server = adserver.example.com
ad_hostname = client.example.com
auth_provider = ad
chpass_provider = ad
access_provider = ad
```

c. **Disable** ID mapping. This tells SSSD to search the global catalog for POSIX attributes, rather than creating UID:GID numbers based on the Windows SID.

```
# disabling ID mapping
ldap_id_mapping = False
```

d. If home directory and a login shell are set in the user accounts, then comment out these lines to configure SSSD to use the POSIX attributes rather than creating the attributes based on the template.

```
# Comment out if the users have the shell and home dir set on
the AD side
#default_shell = /bin/bash
#fallback_homedir = /home/%d/%u
```

e. Microsoft Active Directory allows each account to have two Kerberos principals. If the host principal for the domain (such as *client.ad.example.com@AD.EXAMPLE.COM*) is not available, then uncomment the **ldap_sasl_authid** line and set the host principal to use.

```
# Uncomment and adjust if the default principal
SHORTNAME$@REALM is not available
# ldap_sasl_authid =
host/client.ad.example.com@AD.EXAMPLE.COM
```

f. Set whether to use short names or fully-qualified user names for Active Directory users. In complex topologies, using fully-qualified names may be necessary for disambiguation.

```
# Comment out if you prefer to user shortnames.
use_fully_qualified_names = True
```

g. Enable credentials caching; this allows users to log into the local system using cached information, even if the Active Directory domain is unavailable.

```
cache_credentials = true
```

h. Configure access controls.

```
ldap_access_order = expire
ldap_account_expire_policy = ad
```

7. Set the file permissions and owner for the SSSD configuration file.

```
[root@server ~]# chown root:root /etc/sssd/sssd.conf
[root@server ~]# chmod 0600 /etc/sssd/sssd.conf
[root@server ~]# restorecon /etc/sssd/sssd.conf
```

8. Start the SSSD service.

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

9. If necessary, install the **oddjob-mkhomedir** package to allow SSSD to create home directories for Active Directory users.

```
[root@server ~]# yum install oddjob-mkhomedir
```

10. Use **authconfig** to enable SSSD for system authentication. Use the **--enablemkhomedir** to enable SSSD to create home directories.

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

11. Restart the SSH service to load the new PAM configuration.

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

Using **authconfig** in the above procedure automatically configured the NSS and PAM configuration files to use SSSD as their identity source.

For example, the **nsswitch.conf** file has SSSD (**sss**) added as a source for user, group, and service information.

```
passwd:         files sss
shadow:         files sss
group:          files sss
...
services:       files sss
...
netgroup:       files sss
```

The different **pam.d** files add a line for the **pam_sss.so** module beneath every **pam_unix.so** line in the **/etc/pam.d/system-auth** and **/etc/pam.d/password-auth** files.

```
auth         sufficient    pam_sss.so use_first_pass
...
account      [default=bad success=ok user_unknown=ignore] pam_sss.so
...
password     sufficient    pam_sss.so use_authtok
...
session      optional      pam_mkhomedir.so
session      optional      pam_sss.so
```

7.3.3.4. Configuring Active Directory as an LDAP Domain

While Active Directory can be configured as a type-specific identity provider, it can also be configured as a pure LDAP identity provider with a Kerberos authentication provider.

1. It is recommended that SSSD connect to the Active Directory server using SASL, which means that the local host must have a service keytab *for the Windows domain* on the Linux host.

   This keytab can be created using Samba.

   a. Configure the **/etc/krb5.conf** file to use the Active Directory realm.

```
[logging]
 default = FILE:/var/log/krb5libs.log

[libdefaults]
 default_realm = AD.EXAMPLE.COM
 dns_lookup_realm = true
 dns_lookup_kdc = true
 ticket_lifetime = 24h
 renew_lifetime = 7d
 rdns = false
 forwardable = true

[realms]
# Define only if DNS lookups are not working
# AD.EXAMPLE.COM = {
#  kdc = server.ad.example.com
#  admin_server = server.ad.example.com
# }

[domain_realm]
# Define only if DNS lookups are not working
# .ad.example.com = AD.EXAMPLE.COM
# ad.example.com = AD.EXAMPLE.COM
```

   b. Set the Samba configuration file, **/etc/samba/smb.conf**, to point to the Windows Kerberos realm.

```
[global]
   workgroup = EXAMPLE
   client signing = yes
   client use spnego = yes
   kerberos method = secrets and keytab
   log file = /var/log/samba/%m.log
   password server = AD.EXAMPLE.COM
   realm = EXAMPLE.COM
   security = ads
```

   c. Then, run the **net ads** command to log in as an administrator principal. This administrator account must have sufficient rights to add a machine to the Windows domain, but it does not require domain administrator privileges.

```
[root@server ~]# net ads join -U Administrator
```

d. Run **net ads** again to add the host machine to the domain. This can be done with the host principal (*host/FQDN*) or, optionally, with the NFS service (*nfs/FQDN*).

```
[root@server ~]# net ads join createupn="host/rhel-
server.example.com@AD.EXAMPLE.COM" -U Administrator
```

2. Make sure that the Services for Unix package is installed on the Windows server.
3. Set up the Windows domain which will be used with SSSD.
   a. On the Windows machine, open **Server Manager**.
   b. Create the Active Directory Domain Services role.
   c. Create a new domain, such as **ad.example.com**.
   d. Add the Identity Management for UNIX service to the Active Directory Domain Services role. Use the Unix NIS domain as the domain name in the configuration.
4. On the Active Directory server, create a group for the Linux users.
   a. Open **Administrative Tools** and select **Active Directory Users and Computers**.
   b. Select the Active Directory domain, **ad.example.com**.
   c. In the **Users** tab, right-click and select **Create a New Group**.
   d. Name the new group **unixusers**, and save.
   e. Double-click the **unixusers** group entry, and open the **Users** tab.
   f. Open the **Unix Attributes** tab.
   g. Set the NIS domain to the NIS domain that was configured for **ad.example.com** and, optionally, set a group ID (GID) number.
5. Configure a user to be part of the Unix group.
   a. Open **Administrative Tools** and select **Active Directory Users and Computers**.
   b. Select the Active Directory domain, **ad.example.com**.
   c. In the **Users** tab, right-click and select **Create a New User**.
   d. Name the new user **aduser**, and make sure that the **User must change password at next logon** and **Lock account** checkboxes are *not* selected.

      Then save the user.
   e. Double-click the **aduser** user entry, and open the **Unix Attributes** tab. Make sure that the Unix configuration matches that of the Active Directory domain and the **unixgroup** group:
      - The NIS domain, as created for the Active Directory domain
      - The UID
      - The login shell, to **/bin/bash**

- The home directory, to **/home/aduser**
- The primary group name, to **unixusers**

Note

Password lookups on large directories can take several seconds per request. The initial user lookup is a call to the LDAP server. Unindexed searches are much more resource-intensive, and therefore take longer, than indexed searches because the server checks every entry in the directory for a match. To speed up user lookups, index the attributes that are searched for by SSSD:

- uid
- uidNumber
- gidNumber
- gecos

For information on ignoring group members, see Section 7.3.4.5, “Ignoring Group Members”.

6. On the Linux system, configure the SSSD domain.

```
[root@rhel-server ~]# vim /etc/sssd/sssd.conf
```

For a complete list of LDAP provider parameters, see the **sssd-ldap(5)** man pages.

Example 7.7. An Active Directory 2008 R2 Domain with Services for Unix

```
[sssd]
config_file_version = 2
domains = ad.example.com
services = nss, pam

...

[domain/ad.example.com]
cache_credentials = true

# for performance
ldap_referrals = false

id_provider = ldap
auth_provider = krb5
chpass_provider = krb5
access_provider = ldap

ldap_schema = rfc2307bis

ldap_sasl_mech = GSSAPI
ldap_sasl_authid = host/rhel-server.example.com@AD.EXAMPLE.COM

#provide the schema for services for unix
ldap_schema = rfc2307bis
```

```
ldap_user_search_base = ou=user accounts,dc=ad,dc=example,dc=com
ldap_user_object_class = user
ldap_user_home_directory = unixHomeDirectory
ldap_user_principal = userPrincipalName

# optional - set schema mapping
# parameters are listed in sssd-ldap
ldap_user_object_class = user
ldap_user_name = sAMAccountName

ldap_group_search_base = ou=groups,dc=ad,dc=example,dc=com
ldap_group_object_class = group

ldap_access_order = expire
ldap_account_expire_policy = ad

krb5_realm = AD-REALM.EXAMPLE.COM
# required
krb5_canonicalize = false
```

7. Restart SSSD.

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

7.3.3.5. Additional Configuration Examples

7.3.3.5.1. Account Settings

With Linux users, certain system preferences are set by default for new users. For example, the **pam_oddjob_mkhomedir.so** library automatically creates home directories in a defined location.

These system preferences either may not be set in the Windows user accounts or may be set to something incompatible with a Linux system. There are two such areas:

- The user home directory
- A default user shell

7.3.3.5.1.1. Setting a User Home Directory

Red Hat Enterprise Linux has a PAM library (**pam_oddjob_mkhomedir.so**) which automatically creates user directories when a user first logs in. This includes Active Directory users, when they first log into a Linux system.

With SSSD, the format of the user directory is retrieved from the identity provider. If the identity provider has a home directory format that is different than the format for the Linux system or if it does not supply a value, then SSSD can be configured to create a home directory using a template specified in its configuration. The template can be set globally in the NSS service section or per domain.

There are two possible parameters:

- **fallback_homedir**, which supplies a template if the identity provider does not supply one

- **override_homedir**, which sets a template to use regardless of what information is set in the identity provider

For more information on the parameters, see Table 7.2, “SSSD [nss] Configuration Parameters”.

```
[nss]
fallback_homedir = /home/%u

...

[domain/ADEXAMPLE]
id_provider = ad
ad_server = adserver.example.com
ad_hostname = client.example.com
auth_provider = ad
...
override_homedir = /home/%d/%u
```

7.3.3.5.1.2. Setting a User Shell

By default, SSSD attempts to retrieve information about user shells from the identity provider. In both Active Directory and LDAPv3 schema, this is defined in the ***loginShell*** attribute. However, this is an optional attribute, so it may not be defined for every user. For Active Directory users, the defined login shell may not be allowed on the Linux system.

There are a number of ways to handle shells in the SSSD configuration:

- Setting a fallback value if no shells are supplied (**shell_fallback**)
- Setting lists of allowed or blacklisted shells (**allowed_shells** and **vetoed_shells**)
- Setting a default value (**default_shell**)
- Setting a value to use, even if another value is given in the identity provider (**override_shell**)

Note

allowed_shells, **vetoed_shells**, and **shell_fallback** can only be set as global settings, not per domain. However, these parameters do not affect local system users, only external users retrieved through SSSD identity providers. Using a general setting, such as **/bin/bash**, is good for most external users.

Default values can be set per domain, while some values (such as the white and blacklists for shells) must be set globally, in the NSS service configuration. For example:

```
[nss]
shell_fallback = /bin/sh
allowed_shells =  /bin/sh,/bin/zsh,/bin/bash
vetoed_shells =  /bin/ksh
...

[domain/ADEXAMPLE]
id_provider = ad
ad_server = adserver.example.com
```

```
ad_hostname = client.example.com
auth_provider = ad
...
default_shell = /bin/zsh
```

7.3.3.5.2. Enabling Dynamic DNS Updates

IdM and Active Directory allow clients to refresh their DNS records automatically. It is also possible to actively maintain the DNS records of the clients to make sure they are updated, including timing out (aging) and removing (scavenging) inactive records.

SSSD allows the Linux system to imitate a Windows client by refreshing its DNS record, which also prevents its record from being marked inactive and removed from the DNS record. When dynamic DNS updates are enabled, then the client's DNS record is refreshed at several times:

- When the identity provider comes online (always)
- When the Linux system reboots (always)
- Periodically (optional configuration)

Note

This can be set to the same interval as the DHCP lease, which means that the Linux client is renewed after the lease is renewed.

DNS updates are sent to the Active Directory server using Kerberos/GSSAPI for DNS (GSS-TSIG); this means that only secure connections need to be enabled.

The dynamic DNS configuration is set for each domain. For example:

```
[domain/ad.example.com]
id_provider = ad
ad_server = adserver.example.com
ad_hostname = client.example.com
auth_provider = ad
chpass_provider = ad
access_provider = ad

ldap_schema = ad

dyndns_update = true
dyndns_refresh_interval = 43200
dyndns_update_ptr = true
dyndns_ttl = 3600
```

Table 7.9. Options for Dynamic DNS Updates

Option	Description	Format

Option	Description	Format
dyndns_update	Sets whether to update the DNS server dynamically with the client IP address. This requires secure updates. This must be set to true for any other dynamic DNS setting to be enabled. The default is true.	boolean
dyndns_ttl	Sets a time-to-live for the client's DNS record. The default is 3600 seconds.	integer
dyndns_refresh_interval	Sets a frequency to perform an automatic DNS update, in addition to the update when the provider comes online. The default is 86400 seconds (24 hours).	integer
dyndns_update_ptr	Sets whether to update the PTR record when the client updates its DNS records. The default is true.	boolean
dyndns_iface	Chooses the interface the IP address of which is used for the dynamic DNS updates. This setting can be used only if **dyndns_update** is true, and it is optional. By default, the IP address of the AD LDAP connection is used.	string
dyndns_force_tcp	Specifies whether the **nsupdate** utility defaults to using the TCP protocol for communicating with the DNS server. By default, this is set to **false**, meaning **nsupdate** chooses the protocol to be used.	boolean

7.3.3.5.3. Using a Filter with Access Controls

There is an Active Directory access provider, which means that Active Directory is used as the source for authorization information. This is actually a shortcut, that combines several generic LDAP parameters into a single configuration parameter. Setting the Active Directory provider:

```
access_provider = ad
```

This is the same as setting several different LDAP parameters, including setting the access order to check for account expirations.

```
access_provider = ldap
ldap_access_order = expire
ldap_account_expire_policy = ad
```

There is an additional option to identify which user accounts to grant access to based on an LDAP filter. First, accounts must match the filter, and then they must pass the expiration check (implicit in the **access_provider = ad** setting).

For example, this sets that only users which belong to the administrators group and have a ***unixHomeDirectory*** attribute match the access control check:

```
access_provider = ad
ad_access_filter = (&(memberOf=cn=admins,ou=groups,dc=example,dc=com)
(unixHomeDirectory=*))
```

The access control check requires a secure connection (SASL with a GSS-API mechanism). Configuring the same functionality using the generic LDAP parameters requires defining that SASL/GSS-API connection, the filter, and the expiration checks.

```
access_provider = ldap
ldap_access_order = filter, expire
ldap_account_expire_policy = ad
ldap_access_filter = (&(memberOf=cn=admins,ou=groups,dc=example,dc=com)
(unixHomeDirectory=*))
ldap_sasl_mech = GSSAPI
ldap_sasl_authid = CLIENT_SHORTNAME$@EXAMPLE.COM
ldap_schema = ad
```

7.3.4. Setting Additional Identity Provider Options

7.3.4.1. Setting Username Formats

One of the primary actions that SSSD performs is mapping a local system user to an identity in the remote identity provider. SSSD uses a combination of the username and the domain backend name to create the login identity.

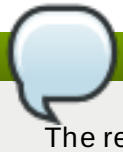
As long as they belong to different domains, SSSD can recognize different users with the same username. For example, SSSD can successfully authenticate both **jsmith** in the **ldap.example.com** domain and **jsmith** in the **ldap.otherexample.com** domain.

The name format used to construct full username is (optionally) defined universally in the **[sssd]** section of the configuration and can then be defined individually in each domain section.

Usernames for different services — LDAP, Samba, Active Directory, Identity Management, even the local system — all have different formats. The expression that SSSD uses to identify username/domain name sets must be able to interpret names in different formats. This expression is set in the **re_expression** parameter.

In the global default, this filter constructs a name in the form *name@domain*:

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



Note

The regular expression format is Python syntax.

The domain part may be supplied automatically, based on the domain name of the identity provider. Therefore, a user can log in as **jsmith** and if the user belongs to the LOCAL domain (for example), then his username is interpreted by SSSD as **jsmith@LOCAL**.

However, other identity providers may have other formats. Samba, for example, has a very strict format so that username must match the form *DOMAIN\username*. For Samba, then, the regular expression must be:

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

Some providers, such as Active Directory, support multiple different name formats. Active Directory and Identity Management, for example, support three different formats by default:

- *username*
- *username@domain.name*
- *DOMAIN\username*

The default value for Active Directory and Identity Management providers, then, is a more complex filter that allows all three name formats:

```
(((?P<domain>[^\\]+)\\(?P<name>.+$))|((?P<name>[^@]+)@(?P<domain>.+$))|
(^(?P<name>[^@\\]+)$))
```

Note

Requesting information with the fully-qualified name, such as **jsmith@ldap.example.com**, always returns the proper user account. If there are multiple users with the same username in different domains, specifying only the username returns the user for whichever domain comes first in the lookup order.

While **re_expression** is the most important method for setting username formats, there are two other options which are useful for other applications.

Default Domain Name Value

The first sets a default domain name to be used with all users, **default_domain_suffix**. (This is a global setting, available in the **[sssd]** section only.) There may be a case where multiple domains are configured but only one stores user data and the others are used for host or service identities. Setting a default domain name allows users to log in with only their username, not specifying the domain name (which would be required for users outside the primary domain).

```
[sssd]
...
default_domain_suffix = USERS.EXAMPLE.COM
```

Full Name Format for Output

The other parameter is related to **re_expression**, only instead of defining how to *interpret* a username, it defines how to *print* an identified name. The **full_name_format** parameter sets how the username and domain name (once determined) are displayed.

SSSD always returns usernames within a subdomain as fully-qualified. The default format is printed as **username@domain**. The **full_name_format** parameter sets the format in **printf** format, so the default is represented as:

```
full_name_format = %1$s@%2$s
```

The username is argument 1, the domain is argument 2, and $s means that the value is a string.

Apart from the **%1$s** and **%2$s** expansions, the **%3$s** expansion is also supported. It expands into the domain flat name and is mostly used for AD domains, either directly configured or discovered using IdM trusts.

The format of the fully-qualified username is configurable. However, in some possible name configurations, SSSD could strip the domain component of the name, which can cause authentication errors. Because of this, if you set the full_name_format to a non-standard value, a warning will prompt you to change it to a more standard format.

7.3.4.2. Enabling Offline Authentication

User identities are always cached, as well as information about the domain services. However, user *credentials* are not cached by default. This means that SSSD always checks with the backend identity provider for authentication requests. If the identity provider is offline or unavailable, there is no way to process those authentication requests, so user authentication could fail.

It is possible to enable *offline credentials caching*, which stores credentials (after successful login) as part of the user account in the SSSD cache. Therefore, even if an identity provider is unavailable, users can still authenticate, using their stored credentials. Offline credentials caching is primarily configured in each individual domain entry, but there are some optional settings that can be set in the PAM service section, because credentials caching interacts with the local PAM service as well as the remote domain.

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

There are optional parameters that set when those credentials expire. Expiration is useful because it can prevent a user with a potentially outdated account or credentials from accessing local services indefinitely.

The credentials expiration itself is set in the PAM service, which processes authentication requests for the system.

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

[pam]
offline_credentials_expiration = 3
...

[domain/EXAMPLE]
cache_credentials = true
...
```

offline_credentials_expiration sets the number of days after a successful login that a single credentials entry for a user is preserved in cache. Setting this to zero (0) means that entries are kept forever. For more information about this setting, see Table 7.3, "SSSD [pam] Configuration Parameters".

While not related to the credentials cache specifically, each domain has configuration options on when individual user and service caches expire:

- **account_cache_expiration** sets the number of days after a successful login that the entire user account entry is removed from the SSSD cache. This must be equal to or longer than the individual offline credentials cache expiration period.
- **entry_cache_timeout** sets a validity period, in seconds, for all entries stored in the cache before SSSD requests updated information from the identity provider. There are also individual cache timeout parameters for group, service, netgroup, sudo, and autofs entries; these are listed in the **sssd.conf** man page. The default time is 5400 seconds (90 minutes).

For example:

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

[pam]
offline_credentials_expiration = 3
...

[domain/EXAMPLE]
cache_credentials = true
account_cache_expiration = 7
entry_cache_timeout = 14400
...
```

7.3.4.3. Setting Password Expirations

Password policies generally set an expiration time, when passwords expire and must be replaced. Those password expiration policies are evaluated by server-side, through the identity provider, and then a warning can be processed and displayed in SSSD through its PAM service.

There are two potential configuration areas for password warnings:

- A global default for all domains on how far in advance of the password expiration to display a warning. This is set for the PAM service.
- Per-domain settings on how far in advance of the password expiration to display a warning.

  When using a domain-level password expiration warning, an authentication provider (**auth_provider**) must also be configured for the domain.

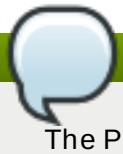
For example:

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

[pam]
pam_pwd_expiration_warning = 3
```

```
...

[domain/EXAMPLE]
id_provider = ipa
auth_provider = ipa
pwd_expiration_warning = 7
```

The password expiration warning must be sent from the server to SSSD for the warning to be displayed. If no password warning is sent from the server, no message is displayed through SSSD, even if the password expiration time is within the period set in SSSD.

If the password expiration warning is not set in SSSD or is set to zero (0), then the SSSD password warning filter is not applied and the server-side password warning is automatically displayed.



Note

The PAM or domain password expirations essentially override (or ignore) the password warning settings on the backend identity provider — as long as the password warning is sent from the server.

For example, a backend identity provider has the warning set at 28 days, but the PAM service in SSSD has it set to seven days. The provider sends the warning to SSSD starting at 28 days, but the warning is not displayed locally until seven days, according to the password expiration set in the SSSD configuration.

Note

A similar parameter is available when using Kerberos authentication providers to cache Kerberos credentials, **krb5_store_password_if_offline**.

7.3.4.4. LDAP Groups with Local System Users

LDAP identity providers (LDAP or IdM) can use RFC 2307 or RC2307bis schema. The Active Directory LDAP provider uses Active Directory-specific schema, which is compatible with RFC 2307bis. By using these schema elements, SSSD can manage local users within LDAP groups.

When a new LDAP group is created, a local user can be added as a member, with the ***memberUID*** attribute value set to the local user ID.

On the local system, the local user is included in the group members when using **getent group**:

```
[root@server ~]# getent group example
example:x:3:jsmith,bjensen,landerson,mreynolds
```

This queries the LDAP directory for the group information. Once that membership is processed, the user is added to the system configuration in **/etc/passwd**.

All of that — querying the LDAP group, creating the local user — is done through NSS (**nss_ldap**), outside SSSD.

Authentication operations and identity tools like **id**, however, go through SSSD, and there is no record of the local user in the LDAP identity provider configured for SSSD. There are two ways that SSSD can handle local user:

- It can delete the user from the local **passwd** file as if it were a remnant of a deleted local account.
- It can query the local user list (**passwd**) as a fallback if a user in a group is not found in LDAP, and then add that user to its cache as if it were an LDAP user.

This behavior is configured in the ***ldap_rfc2307_fallback_to_local_users*** parameter for the identity provider domain. By default, this is false, meaning that only users which exist in the LDAP provider are recognized, and a local user is deleted if it is added to an LDAP group. This can be set to true, which queries the local system users as a fallback if an LDAP group member is not found in the LDAP directory.

7.3.4.5. Ignoring Group Members

When looking up information about an LDAP group, all of the members for that group are returned, by default. For large groups or for nested groups, this can take a long time to process. The membership lists themselves are not actually used when evaluating whether a user belongs to a group — most services use something like **getent group** to determine if a user belongs to a group rather than checking the UID in the members list.

To improve overall performance, especially for identity lookups, it is possible to disable the group membership lookup. This essentially returns an empty group to SSSD to cache.

This is set per domain entry in the **ignore_group_members** parameter:

```
[domain\ad.example.com]
id_provider = ad
ad_server = adserver.example.com
ad_hostname = client.example.com
...
ignore_group_members = true
```

7.3.4.6. Using DNS Service Discovery

DNS service discovery, defined in RFC 2782, allows applications to check the SRV records in a given domain for certain services of a certain type; it then returns any servers discovered of that type.

With SSSD, the identity and authentication providers can either be explicitly defined (by IP address or hostname) or they can be discovered dynamically, using service discovery. If no provider server is listed — for example, if **id_provider = ldap** is set without a corresponding **ldap_uri** parameter — then discovery is automatically used.

The DNS discovery query has this format:

```
_service._protocol.domain
```

For example, a scan for an LDAP server using TCP in the **example.com** domain looks like this:

```
_ldap._tcp.example.com
```

Note

For every service with which to use service discovery, add a special DNS record to the DNS server:

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

For SSSD, the service type is LDAP by default, and almost all services use TCP (except for Kerberos, which starts with UDP). For service discovery to be enabled, the only thing that is required is the domain name. The default is to use the domain portion of the machine hostname, but another domain can be specified (using the **dns_discovery_domain** parameter).

So, by default, no additional configuration needs to be made for service discovery — with one exception. The password change provider has service discovery disabled by default, and it must be explicitly enabled by setting a service type.

```
[domain/EXAMPLE]
...
chpass_provider = ldap
ldap_chpass_dns_service_name = ldap
```

While no configuration is necessary, it is possible for service discovery to be customized by using a different DNS domain (**dns_discovery_domain**) or by setting a different service type to scan for. For example:

```
[domain/EXAMPLE]
id _provider = ldap

dns_discovery_domain = corp.example.com
ldap_dns_service_name = ldap

chpass_provider = krb5
ldap_chpass_dns_service_name = kerberos
```

Lastly, service discovery is never used with backup servers; it is only used for the primary server for a provider. What this means is that discovery can be used initially to locate a server, and then SSSD can fall back to using a backup server. To use discovery for the primary server, use **_srv_** as the primary server value, and then list the backup servers. For example:

```
[domain/EXAMPLE]
id _provider = ldap
ldap_uri = _srv_
ldap_backup_uri = ldap://ldap2.example.com

auth_provider = krb5
krb5_server = _srv_
krb5_backup_server = kdc2.example.com

chpass_provider = krb5
ldap_chpass_dns_service_name = kerberos
ldap_chpass_uri = _srv_
ldap_chpass_backup_uri = kdc2.example.com
```

Note

Service discovery cannot be used with backup servers, only primary servers.

If a DNS lookup fails to return an IPv4 address for a hostname, SSSD attempts to look up an IPv6 address before returning a failure. This only ensures that the asynchronous resolver identifies the correct address.

The hostname resolution behavior is configured in the ***lookup_family_order*** option in the **sssd.conf** configuration file.

7.3.4.7. Using IP Addresses in Certificate Subject Names (LDAP Only)

Using an IP address in the **ldap_uri** option instead of the server name may cause the TLS/SSL connection to fail. TLS/SSL certificates contain the server name, not the IP address. However, the *subject alternative name* field in the certificate can be used to include the IP address of the server, which allows a successful secure connection using an IP address.

1. Convert an existing certificate into a certificate request. The certificate signing request (CSR) must be signed with the private key of the LDAP server for which the certificate was issued; using the **-signkey** option, pass the PEM file that contains the private key.

   ```
   openssl x509 -x509toreq -in old_cert.pem -out req.pem -signkey
   old_cert.pem
   ```

2. Edit the **/etc/pki/tls/openssl.cnf** configuration file to include the server's IP address under the ***[v3_ca]*** section:

   ```
   subjectAltName = IP:192.0.2.1
   ```

3. Use the generated CSR to generate a new certificate with the specified IP address.

   To generate a self-signed certificate, use the **-signkey** option to sign the certificate with the PEM file containing the old LDAP server certificate and the corresponding private key:

   ```
   openssl x509 -req -in req.pem -out new_cert.pem -extfile
   ./openssl.cnf -extensions v3_ca -signkey old_cert.pem
   ```

   To generate a certificate signed by a certificate authority (CA), use the **-signkey** option to sign the certificate with a PEM file containing the CA private key:

   ```
   openssl x509 -req -in req.pem -out new_cert.pem -extfile
   ./openssl.cnf -extensions v3_ca -signkey key.pem
   ```

   The **-extensions** option sets which extensions to use with the certificate. For this, it should be **v3_ca** to load the appropriate section.

4. Copy the private key block from the **old_cert.pem** file into the **new_cert.pem** file to keep all relevant information in one file.

When creating a certificate through the **certutil** utility provided by the *nss-utils* package, note that **certutil** supports DNS subject alternative names as well as IP address subject alternative names for certificate creation.

7.3.4.8. Configuring Different Types of Access Control

SSSD provides a rudimentary access control for domain configuration, allowing either simple user allow/deny lists or using the LDAP backend itself.

7.3.4.8.1. Using the Simple Access Provider

The *Simple Access Provider* allows or denies access based on a list of usernames or groups.

The Simple Access Provider is a way to restrict access to certain, specific machines. For example, if a company uses laptops, the Simple Access Provider can be used to restrict access to only a specific user or a specific group, even if a different user authenticated successfully against the same authentication provider.

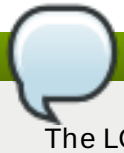
The most common options are **simple_allow_users** and **simple_allow_groups**, which grant access explicitly to specific users (either the given users or group members) and deny access to everyone else. It is also possible to create deny lists (which deny access only to explicit people and implicitly allow everyone else access).

The Simple Access Provider adheres to the following four rules to determine which users should or should not be granted access:

- If both the allow and deny lists are empty, access is granted.
- If any list is provided, allow rules are evaluated first, and then deny rules. Practically, this means that deny rules supersede allow rules.
- If an allowed list is provided, then all users are denied access unless they are in the list.
- If only deny lists are provided, then all users are allowed access unless they are in the list.

This example grants access to two users and anyone who belongs to the IT group; implicitly, all other users are denied:

```
[domain/example.com]
access_provider = simple
simple_allow_users = jsmith,bjensen
simple_allow_groups = itgroup
```



Note

The LOCAL domain in SSSD does not support **simple** as an access provider.

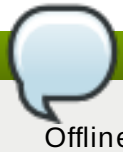
Other options are listed in the **sssd-simple** man page, but these are rarely used.

7.3.4.8.2. Using the LDAP Access Filter

An LDAP, Active Directory, or Identity Management server can provide access control rules for a domain. The associated filter option (**ldap_access_filter**) specifies which users are granted access to the specified host. The user filter must be used or all users are denied access.

For example:

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



Note

Offline caching for LDAP access providers is limited to determining whether the user's last online login attempt was successful. Users that were granted access during their last login will continue to be granted access while offline.

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.

7.3.4.9. Configuring Primary Server and Backup Servers

Identity and authentication providers for a domain can be configured for automatic failover. SSSD attempts to connect to the specified, primary server first. If that server cannot be reached, then SSSD then goes through the listed backup servers, in order.

Note

SSSD tries to connect to the primary server every 30 seconds, until the connection can be re-established, and then switches from the backup to the primary.

All of the major service areas have optional settings for primary and backup servers [1].

Table 7.10. Primary and Secondary Server Parameters

Service Area	Primary Server Attribute	Backup Server Attribute
LDAP identity provider	ldap_uri	ldap_backup_uri
Active Directory identity provider	ad_server	ad_backup_server
Identity Management (IdM or IPA) identity provider	ipa_server	ipa_backup_server
Kerberos authentication provider	krb5_server	krb5_backup_server
Kerberos password change provider	krb5_kpasswd	krb5_backup_kpasswd
Password change provider	ldap_chpass_uri	ldap_chpass_backup_uri

Primary and backup servers are given in comma-separated lists. Servers from the primary-server list are the first-choice servers; SSSD searches the backup servers only when it fails to reach any of the primary servers. List both primary and backup servers in order of preference; the first server listed is tried first. Service discovery using _**srv**_ is supported only for the primary servers.

```
[domain/EXAMPLE]
id_provider = ad
ad_server = ad.example.com, ad1.example.com
ad_backup_server = ad-backup.example.com, ad-backup1.example.com
```

For more information about the failover mechanism, see the sssd-ldap(5) man page.

7.3.5. Creating a Proxy Identity Provider

A proxy with SSSD is just a relay, an intermediary configuration. SSSD connects to its proxy service, and then that proxy loads the specified libraries. This allows SSSD to use some resources that it otherwise would not be able to use. For example, SSSD only supports LDAP and Kerberos as authentication providers, but using a proxy allows SSSD to use alternative authentication methods like a fingerprint scanner or smart card.

Table 7.11. Proxy Domain Configuration Parameters

Parameter	Description
proxy_pam_target	Specifies the target to which PAM must proxy as an authentication provider. The target is a PAM service – a file containing PAM stack information located in the default **/etc/pam.d/** directory. This is used to proxy an authentication provider.  **Important** Ensure that the proxy PAM stack does *not* recursively include **pam_sss.so**.
proxy_lib_name	Specifies which existing NSS library to proxy identity requests through. This is used to proxy an identity provider.

Example 7.8. Proxy Identity and Kerberos Authentication

The proxy library is loaded using the **proxy_lib_name** parameter. This library can be anything as long as it is compatible with the given authentication service. For a Kerberos authentication provider, it must be a Kerberos-compatible library, like NIS.

```
[domain/PROXY_KRB5]
auth_provider = krb5
krb5_server = kdc.example.com
krb5_realm = EXAMPLE.COM
```

```
id_provider = proxy
proxy_lib_name = nis
cache_credentials = true
```

Example 7.9. LDAP Identity and Proxy Authentication

The **proxy_pam_target** specifies a PAM service. For example, this uses a PAM fingerprint module with LDAP:

```
[domain/LDAP_PROXY]
id_provider = ldap
ldap_uri = ldap://example.com
ldap_search_base = dc=example,dc=com

auth_provider = proxy
proxy_pam_target = sssdpamproxy
cache_credentials = true
```

After the SSSD domain is configured, make sure that the specified PAM files are configured. In this example, the target is **sssdpamproxy**, so create a **/etc/pam.d/sssdpamproxy** file and load the PAM/LDAP modules:

```
auth          required      pam_frprint.so
account       required      pam_frprint.so
password      required      pam_frprint.so
session       required      pam_frprint.so
```

Example 7.10. Proxy Identity and Authentication

SSSD can have a domain with both identity and authentication proxies. The only configuration given then are the proxy settings, **proxy_pam_target** for the authentication PAM module and **proxy_lib_name** for the service, like NIS or LDAP.

This example illustrates a possible configuration, but this is not a realistic configuration. If LDAP is used for identity and authentication, then both the identity and authentication providers should be set to the LDAP configuration, not a proxy.

```
[domain/PROXY_PROXY]
auth_provider = proxy
id_provider = proxy
proxy_lib_name = ldap
proxy_pam_target = sssdproxyldap
cache_credentials = true
```

Once the SSSD domain is added, then update the system settings to configure the proxy service:

1. Create a **/etc/pam.d/sssdproxyldap** file which requires the **pam_ldap.so** module:

   ```
   auth          required      pam_ldap.so
   account       required      pam_ldap.so
   password      required      pam_ldap.so
   session       required      pam_ldap.so
   ```

2. Make sure the **nss-pam-ldap** package is installed.

```
[root@server ~]# yum install nss-pam-ldap
```

3. Edit the **/etc/nslcd.conf** file, the configuration file for the LDAP name service daemon, to contain the information for the LDAP directory:

```
uid nslcd
gid ldap
uri ldaps://ldap.example.com:636
base dc=example,dc=com
ssl on
tls_cacertdir /etc/openldap/cacerts
```

7.3.6. Configuring Kerberos Authentication with an Identity Provider

Both LDAP and proxy identity providers can use a separate Kerberos domain to supply authentication. Configuring a Kerberos authentication provider requires the *key distribution center* (KDC) and the Kerberos domain. All of the principal names must be available in the specified identity provider; if they are not, SSSD constructs the principals using the format *username@REALM*.

Note

Kerberos can only provide authentication; it cannot provide an identity database.

SSSD assumes that the Kerberos KDC is also a Kerberos kadmin server. However, production environments commonly have multiple, read-only replicas of the KDC and only a single kadmin server. Use the **krb5_kpasswd** option to specify where the password changing service is running or if it is running on a non-default port. If the **krb5_kpasswd** option is not defined, SSSD tries to use the Kerberos KDC to change the password.

The basic Kerberos configuration options are listed in Table 7.12, “Kerberos Authentication Configuration Parameters”. The **sssd-krb5(5)** man page has more information about Kerberos configuration options.

Example 7.11. Basic Kerberos Authentication

```
# A domain with identities provided by LDAP and authentication by
Kerberos
[domain/KRBDOMAIN]
id_provider = ldap
chpass_provider = krb5
ldap_uri = ldap://ldap.example.com
ldap_search_base = dc=example,dc=com
ldap-tls_reqcert = demand
ldap_tls_cacert = /etc/pki/tls/certs/ca-bundle.crt

auth_provider = krb5
krb5_server = kdc.example.com
krb5_backup_server = kerberos.example.com
```

```
krb5_realm = EXAMPLE.COM
krb5_kpasswd = kerberos.admin.example.com
krb5_auth_timeout = 15
krb5_use_kdcinfo = true
```

Example 7.12. Setting Kerberos Ticket Renewal Options

The Kerberos authentication provider, among other tasks, requests ticket granting tickets (TGT) for users and services. These tickets are used to generate other tickets dynamically for specific services, as accessed by the ticket principal (the user).

The TGT initially granted to the user principal is valid only for the lifetime of the ticket (by default, whatever is configured in the configured KDC). After that, the ticket cannot be renewed or extended. However, not renewing tickets can cause problems with some services when they try to access a service in the middle of operations and their ticket has expired.

Kerberos tickets are not renewable by default, but ticket renewal can be enabled using the **krb5_renewable_lifetime** and **krb5_renew_interval** parameters.

The lifetime for a ticket is set in SSSD with the **krb5_lifetime** parameter. This specifies how long a single ticket is valid, and overrides any values in the KDC.

Ticket renewal itself is enabled in the **krb5_renewable_lifetime** parameter, which sets the maximum lifetime of the ticket, counting all renewals.

For example, the ticket lifetime is set at one hour and the renewable lifetime is set at 24 hours:

```
krb5_lifetime = 1h
krb5_renewable_lifetime = 1d
```

This means that the ticket expires every hour and can be renewed continually up to one day.

The lifetime and renewable lifetime values can be in seconds (s), minutes (m), hours (h), or days (d).

The other option — which must also be set for ticket renewal — is the **krb5_renew_interval** parameter, which sets how frequently SSSD checks to see if the ticket needs to be renewed. At half of the ticket lifetime (whatever that setting is), the ticket is renewed automatically. (This value is always in seconds.)

```
krb5_lifetime = 1h
krb5_renewable_lifetime = 1d
krb5_renew_interval = 60s
```

Note

If the **krb5_renewable_lifetime** value is not set or the **krb5_renew_interval** parameter is not set or is set to zero (0), then ticket renewal is disabled. Both **krb5_renewable_lifetime** and **krb5_renew_interval** are required for ticket renewal to be enabled.

Table 7.12. Kerberos Authentication Configuration Parameters

Parameter	Description
chpass_provider	Specifies which service to use for password change operations. This is assumed to be the same as the authentication provider. To use Kerberos, set this to *krb5*.
krb5_server	Gives the primary Kerberos server, by IP address or hostnames, to which SSSD will connect.
krb5_backup_server	Gives a comma-separated list of IP addresses or hostnames of Kerberos servers to which SSSD will connect if the primary server is not available. The list is given in order of preference, so the first server in the list is tried first. After an hour, SSSD will attempt to reconnect to the primary service specified in the ***krb5_server*** parameter. When using service discovery for KDC or kpasswd servers, SSSD first searches for DNS entries that specify UDP as the connection protocol, and then falls back to TCP.
krb5_realm	Identies the Kerberos realm served by the KDC.
krb5_lifetime	Requests a Kerberos ticket with the specified lifetime in seconds (s), minutes (m), hours (h) or days (d).
krb5_renewable_lifetime	Requests a renewable Kerberos ticket with a total lifetime that is specified in seconds (s), minutes (m), hours (h) or days (d).
krb5_renew_interval	Sets the time, in seconds, for SSSD to check if tickets should be renewed. Tickets are renewed automatically once they exceed half their lifetime. If this option is missing or set to zero, then automatic ticket renewal is disabled.
krb5_store_password_if_offline	Sets whether to store user passwords if the Kerberos authentication provider is offline, and then to use that cache to request tickets when the provider is back online. The default is **false**, which does not store passwords.
krb5_kpasswd	Lists alternate Kerberos kadmin servers to use if the change password service is not running on the KDC.

Parameter	Description
krb5_ccname_template	Gives the directory to use to store the user's credential cache. This can be templatized, and the following tokens are supported: - *%u*, the user's login name - *%U*, the user's login UID - *%p*, the user's principal name - *%r*, the realm name - *%h*, the user's home directory - *%d*, the value of the **krb5ccache_dir** parameter - *%P*, the process ID of the SSSD client. - *%%*, a literal percent sign (%) - *XXXXXX*, a string at the end of the template which instructs SSSD to create a unique filename safely For example: `krb5_ccname_template = FILE:%d/krb5cc_%U_XXXXXX`
krb5_ccachedir	Specifies the directory to store credential caches. This can be templatized, using the same tokens as **krb5_ccname_template**, except for **%d** and **%P**. If **%u**, **%U**, **%p**, or **%h** are used, then SSSD creates a private directory for each user; otherwise, it creates a public directory.
krb5_auth_timeout	Gives the time, in seconds, before an online authentication or change password request is aborted. If possible, the authentication request is continued offline. The default is 15 seconds.
krb5_use_kdcinfo	Sets whether to create Kerberos information files used by the Kerberos locator plug-in. This is set to **true** by default. If it is set to **false**, then the files are not created by SSSD, and the Kerberos options must be set manually in the **krb5.conf** file.

7.4. Managing Local System Users in SSSD

7.4.1. Installing SSSD Utilities

Additional tools to handle the SSSD cache, user entries, and group entries are contained in the **sssd-tools** package. This package is not required, but it is useful to install to help administer user accounts.

```
[root@server ~]# yum install sssd-tools
```

7.4.2. SSSD and UID and GID Numbers

When a user is created — using system tools such as **useradd** or through an application such as Red Hat Identity Management or other client tools — the user is automatically assigned a user ID number and a group ID number.

When the user logs into a system or service, SSSD caches that username with the associated UID/GID numbers. The UID number is then used as the identifying key for the user. If a user with the same name but a different UID attempts to log into the system, then SSSD treats it as two different users with a name collision.

What this means is that SSSD does not recognize UID number changes. It interprets it as a different and new user, not an existing user with a different UID number. If an existing user changes the UID number, that user is prevented from logging into SSSD and associated services and domains. This also has an impact on any client applications which use SSSD for identity information; the user with the conflict will not be found or accessible to those applications.

Important

UID/GID changes are not supported in SSSD.

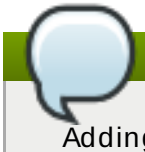
If a user for some reason has a changed UID/GID number, then the SSSD cache must be cleared for that user before that user can log in again. For example:

```
[root@server ~]# sss_cache -u jsmith
```

Cleaning the SSSD cache is covered in Section 7.4.5.1, “Purging the SSSD Cache”.

7.4.3. Creating Local System Users

There can be times when it is useful to seed users into the SSSD database rather than waiting for users to login and be added.



Note

Adding user accounts manually requires the **sssd-tools** package to be installed.

When creating new system users, it is possible to create a user within the SSSD local identity provider domain. This can be useful simply for creating new system users, for troubleshooting SSSD configuration, or for creating specialized or nested groups.

New users can be added using the **sss_useradd** command.

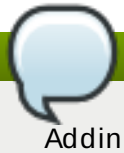
At its most basic, the **sss_useradd** command only requires the new username.

```
[root@server ~]# sss_useradd jsmith
```

There are other options (listed in the **sss_useradd(8)** man page) which can be used to set attributes on the account, like the UID and GID, the home directory, or groups which the user belongs to.

```
[root@server ~]# sss_useradd --UID 501 --home /home/jsmith --groups
admin,dev-group jsmith
```

7.4.4. Seeding Users into the SSSD Cache During Kickstart



Note

Adding user accounts manually requires the **sssd-tools** package to be installed.

With SSSD, users in a remote domain are not available in a local system until that identity is retrieved from the identity provider. However, some network interfaces are not available until a user has logged in — which is not possible if the user identity is somewhere over the network. In that case, it is possible to seed the SSSD cache with that user identity, associated with the appropriate domain, so that the user can log in locally and activate the appropriate interfaces.

This is done using the **sss_seed** utility:

```
sss_seed --domain EXAMPLE.COM --username testuser --password-file
/tmp/sssd-pwd.txt
```

This utility requires options that identify, at a minimum, the username, domain name, and password.

- **--domain** gives the domain name from the SSSD configuration. This domain must already exist in the SSSD configuration.
- **--username** for the short name of the user account.
- **--password-file** for the path and name of a file containing a temporary password for the seed entry. If the user account already exists in the SSSD cache, then the temporary password in this file overwrites the stored password in the SSSD cache.

Additional account configuration options are listed in the **sss_seed(8)** man page.

This would almost always be run as part of a kickstart or automated setup, so it would be part of a larger set of scripts, which would also enable SSSD, set up an SSSD domain, and create the password file. For example:

```
function make_sssd {
cat <<- _EOF_
[sssd]
domains = LOCAL
services = nss,pam

[nss]

[pam]

[domain/LOCAL]
id_provider = local
auth_provider = local
access_provider = permit

_EOF_
}

make_sssd >> /etc/sssd/sssd.conf

authconfig --enablesssd --enablesssdauth --update
```

```
function make_pwdfile {
cat <<1 _EOF_
password
_EOF_
}

make_pwdfile >> /tmp/sssd-pwd.txt

sss_seed --domain EXAMPLE.COM --username testuser --password-file
/tmp/sssd-pwd.txt
```

7.4.5. Managing the SSSD Cache

SSSD can define multiple domains of the same type and different types of domain. SSSD maintains a separate database file for each domain, meaning each domain has its own cache. These cache files are stored in the **/var/lib/sss/db/** directory.

7.4.5.1. Purging the SSSD Cache

As LDAP updates are made to the identity provider for the domains, it can be necessary to clear the cache to reload the new information quickly.

The cache purge utility, **sss_cache**, invalidates records in the SSSD cache for a user, a domain, or a group. Invalidating the current records forces the cache to retrieve the updated records from the identity provider, so changes can be realized quickly.

Note

This utility is included with SSSD in the **sssd** package.

Most commonly, this is used to clear the cache and update the records for an entire domain:

Example 7.13. Purging Domain Records

```
[root@server ~]# sss_cache -d LDAP1
```

If the administrator knows that a specific record (user, group, or netgroup) has been updated, then **sss_cache** can purge the records for that specific account, and leave the rest of the cache intact.

Example 7.14. Purging a User Record

```
[root@server ~]# sss_cache -u jsmith
```

Table 7.13. sss_cache Options

Short Argument	Long Argument	Description

Short Argument	Long Argument	Description
-d *name*	--domain *name*	Invalidates cache entries for users, groups, and other entries only within the specified domain.
-G	--groups	Invalidates all group records. If **-g** is also used, **-G** takes precedence and **-g** is ignored.
-g *name*	--group *name*	Invalidates the cache entry for the specified group.
-N	--netgroups	Invalidates cache entries for all netgroup cache records. If **-n** is also used, **-N** takes precedence and **-n** is ignored.
-n *name*	--netgroup *name*	Invalidates the cache entry for the specified netgroup.
-U	--users	Invalidates cache entries for all user records. If the **-u** option is also used, **-U** takes precedence and **-u** is ignored.
-u *name*	--user *name*	Invalidates the cache entry for the specified user.

7.4.5.2. Deleting Domain Cache Files

All cache files are named for the domain. For example, for a domain named **exampleldap**, the cache file is named **cache_exampleldap.ldb**.

Be careful when you delete a cache file. This operation has significant effects:

- Deleting the cache file deletes all user data, both identification and cached credentials. Consequently, do not delete a cache file unless the system is online and can authenticate with a username against the domain's servers. Without a credentials cache, offline authentication will fail.
- If the configuration is changed to reference a different identity provider, SSSD will recognize users from both providers until the cached entries from the original provider time out.

  It is possible to avoid this by purging the cache, but the better option is to use a different domain name for the new provider. When SSSD is restarted, it creates a new cache file with the new name and the old file is ignored.

7.5. 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
```

7.6. 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.

7.7. Troubleshooting SSSD

- Section 7.7.1, “Setting Debug Logs for SSSD Domains”
- Section 7.7.2, “Checking SSSD Log Files”
- Section 7.7.3, “Problems with SSSD Configuration”

7.7.1. 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 7.14. 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 doesn't 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.
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.

7.7.2. 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.

7.7.3. 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
```

```
[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.

Q:

I don't 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 doesn't 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 Section 7.2.1.2, “Configuring 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 usernames 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 7.2.2, “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 hostname. If the hostname 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 7.3.4.6, “Using 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 7.4.5.1, “Purging the SSSD Cache”.
2. Restart SSSD, as in Section 7.1.2, “Starting and Stopping SSSD”.

[1] Most services default to the identity provider server if a specific server for that service is not set.

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.

Part III. Secure Applications

Chapter 9. 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.

9.1. 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.

9.1.1. 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.

9.1.2. 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 mix-and-match, as well as test, authentication methods for different programs without recompiling them.

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

9.2. 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.

9.2.1. 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.

9.2.2. Annotated PAM Configuration Example

Example 9.1, “Simple PAM Configuration” is a sample PAM application configuration file:

Example 9.1. 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.

9.3. 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 9.1. The Authentication Icon

9.3.1. 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.

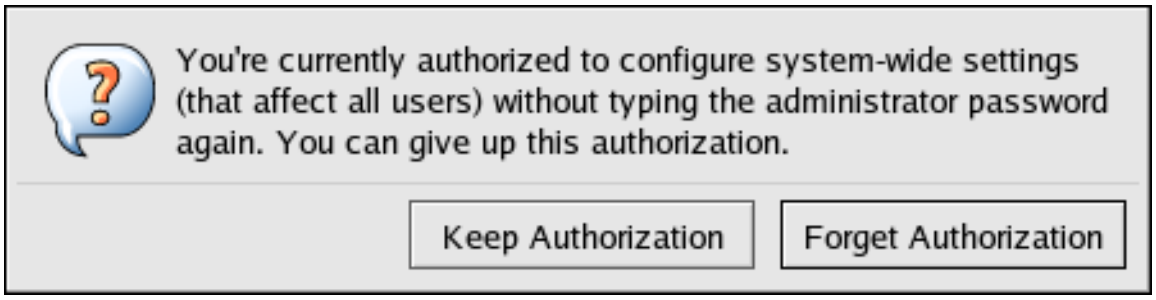


Figure 9.2. Dismiss Authentication Dialog

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**.

9.3.2. 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/**.

9.4. 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 9.2. 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 10. 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 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.

10.1. 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.

10.1.1. 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 usernames 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 10.1, “Kerberos Authentication, in Steps”, 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.

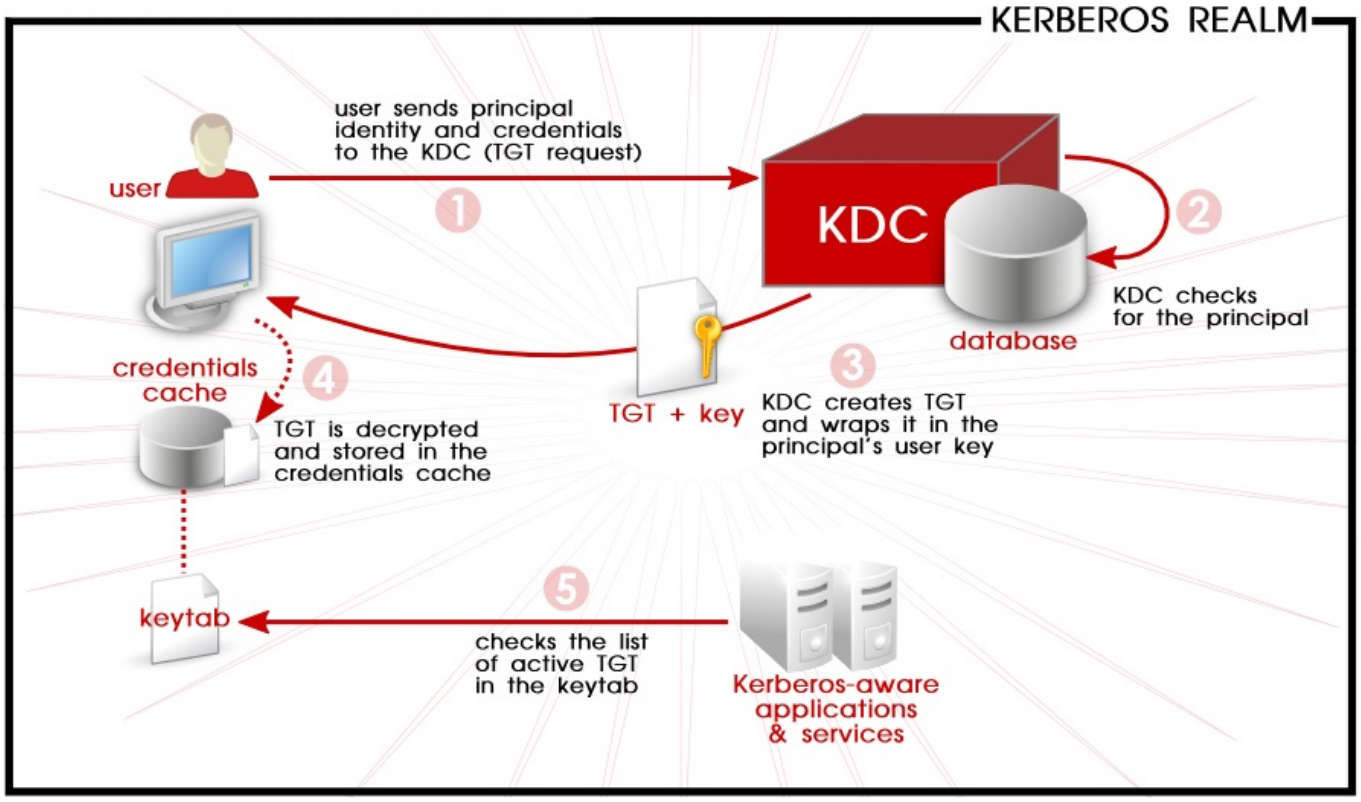


Figure 10.1. Kerberos Authentication, in Steps

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 ten to twenty-four 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 re-enter their password 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.

10.1.2. 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
```

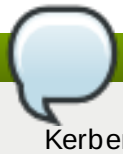
The configuration specifies two mappings. The first mapping specifies that any system in the example.com DNS domain belongs to the *EXAMPLE.COM* realm. The second specifies that a system with the exact name example.com is also in the realm. The distinction between a domain and a specific host is marked 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"
```

10.1.3. 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.

10.1.4. 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*, or *kerberized*. 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.
- Kerberos is an all-or-nothing solution. To secure a network with Kerberos, one must either use Kerberos-aware versions of *all* client/server applications that transmit passwords unencrypted, or not use that client/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.

10.1.5. Additional Resources for Kerberos

Kerberos can be a complex service to implement, with a lot of flexibility in how it is deployed. Table 10.1, “External Kerberos Documentation” and Table 10.2, “Important Kerberos Manpages” list of a few of the most important or most useful sources for more information on using Kerberos.

Table 10.1. 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" webpage from MIT	http://web.mit.edu/kerberos/www/
The Kerberos Frequently Asked Questions (FAQ)	http://www.cmf.nrl.navy.mil/CCS/people/kenh/kerberos-faq.html

Documentation	Location
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
A how-to article for kerberizing a network.	http://www.ornl.gov/~jar/HowToKerb.html
Kerberos Network Design Manual is a thorough overview of the Kerberos system.	http://www.networkcomputing.com/netdesign/kerb1.html

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

Table 10.2. Important Kerberos Manpages

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.

10.2. 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.

10.2.1. 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 hostnames 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
```

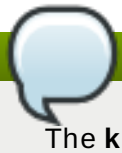
The **create** command creates the database that stores keys for the Kerberos realm. The **-s** argument creates a *stash* file in which the master server key is stored. If no stash file is present from which to read the key, the Kerberos server (**krb5kdc**) prompts the user for the master server password (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 username (not the Kerberos server). If that username 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
```

10.2.2. 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-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.

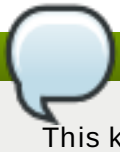
3. Start **kadmin.local** from a root shell on the master KDC.

   a. 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
```

b. 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.

a. 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
```

b. 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. Double-check 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.

10.3. 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, Kerberized 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 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
```

3. To use kerberized **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.

    a. Using **kadmin**, add a host principal for the workstation on the KDC. The instance in this case is the hostname 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
```

    b. 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 kerberized network services, install the *krb5-server* package and start the services. The kerberized services are listed in Table 10.3, “Common Kerberized Services”.

Table 10.3. Common Kerberized Services

Service Name	Usage Information

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.
rsh and rlogin	Enable **klogin**, **eklogin**, and **kshell**.
Telnet	Enable **krb5-telnet**.
FTP	Create and extract a key for the principal with a root of **ftp**. Be certain to set the instance to the fully qualified hostname of the FTP server, then enable **gssftp**.
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.

10.4. 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-openssl
[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/libcoolk
eypk11.so
```

10.5. 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-Linux) environments, particularly if the Linux environment is not using a more structured domain configuration like Identity Management.

10.5.1. 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.

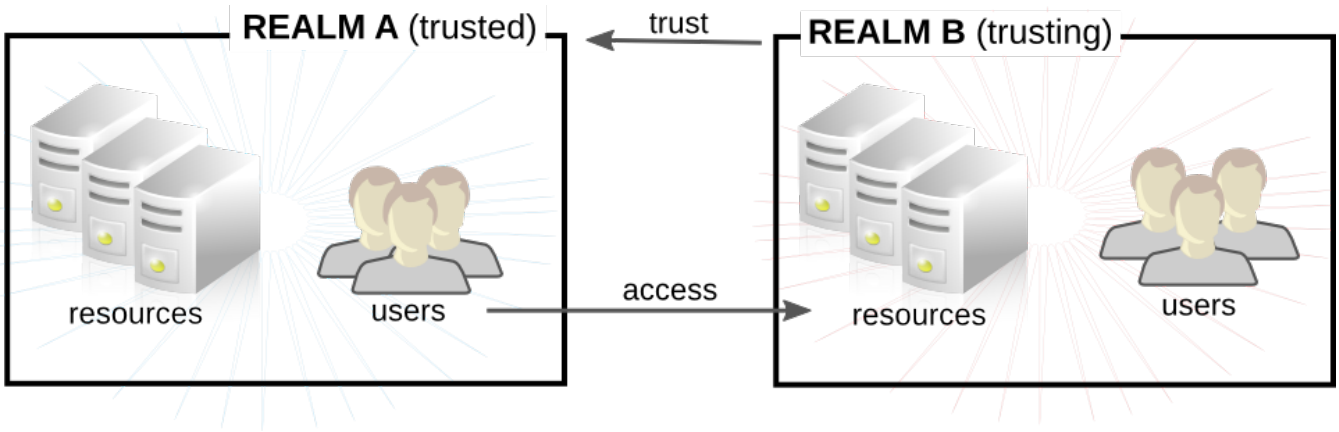


Figure 10.2. Basic Trust

In Figure 10.2, “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 implicity trusts Realm C, as well. The trust flows along realms; this is a *transitive* trust.

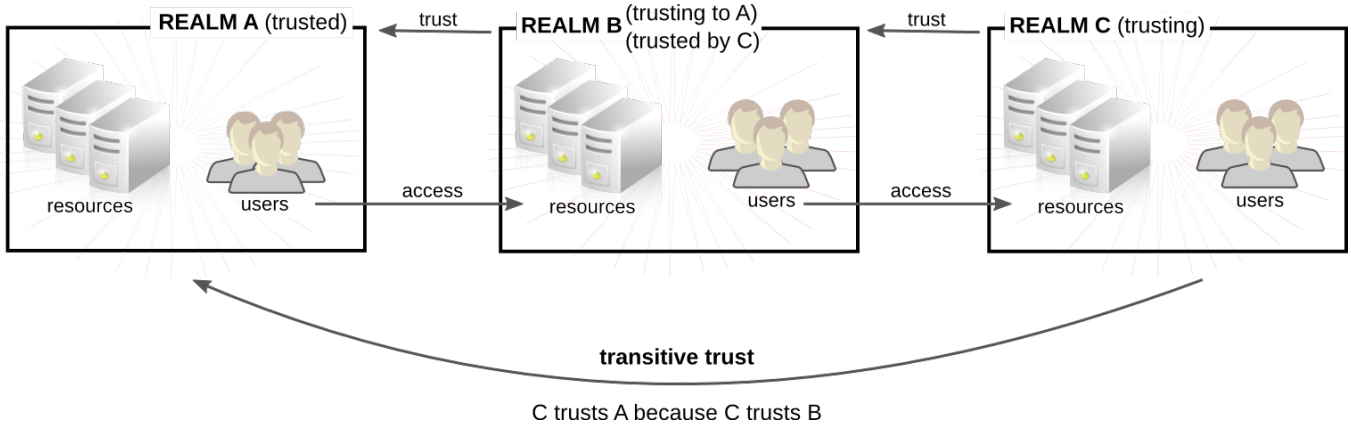


Figure 10.3. Transitive Trust

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 hostname 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 10.4, “Trusts in the Same Domain”, A shares a key with EXAMPLE.COM, and EXAMPLE.COM shares a key with B.

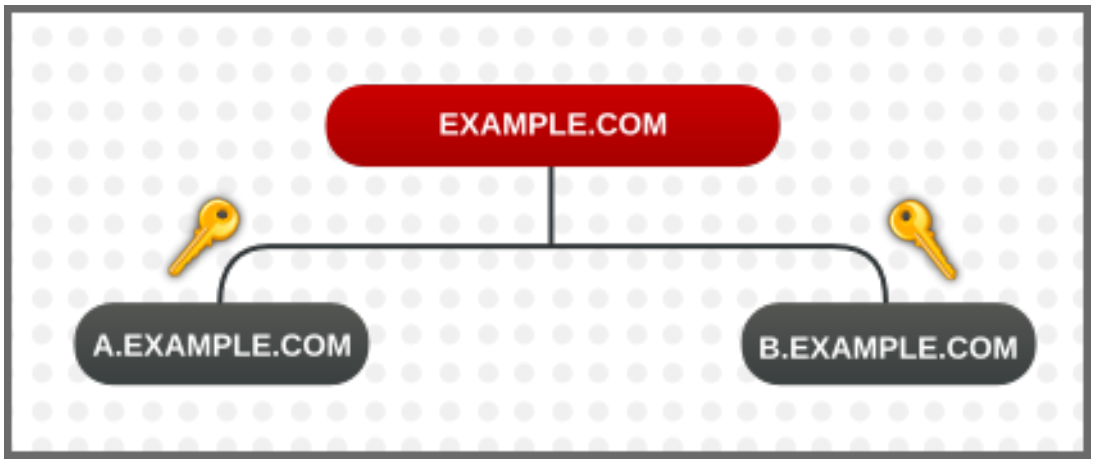


Figure 10.4. Trusts in the Same Domain

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.

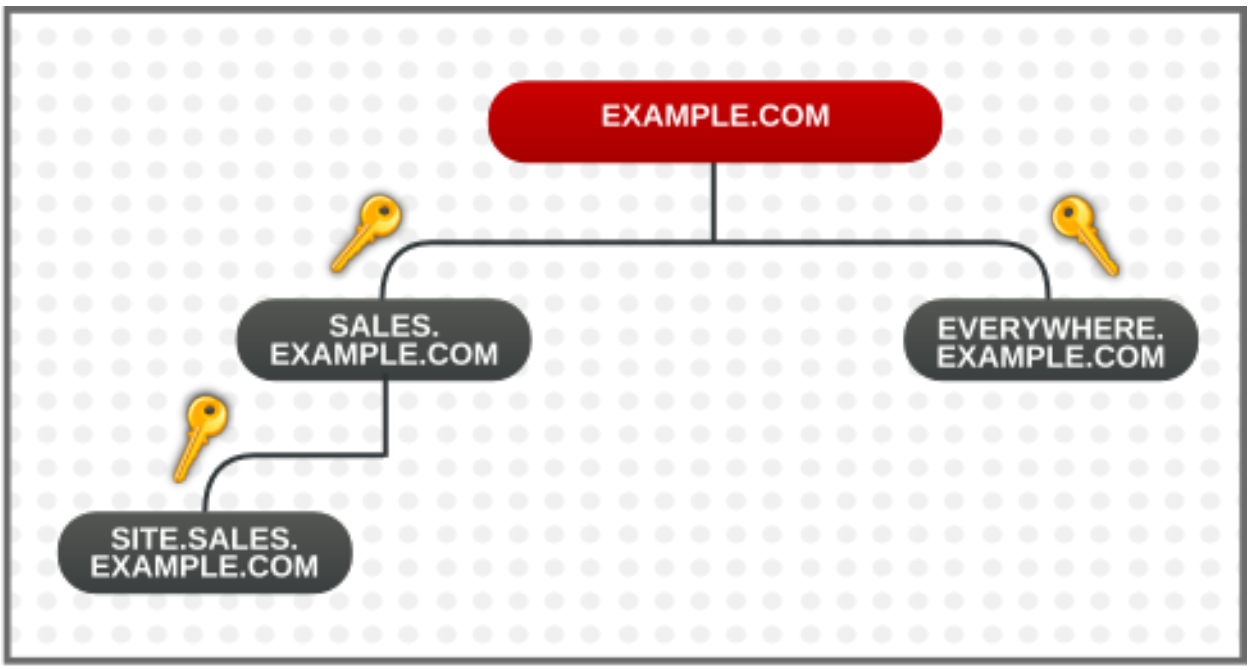


Figure 10.5. Child/Parent Trusts in the Same Domain

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.

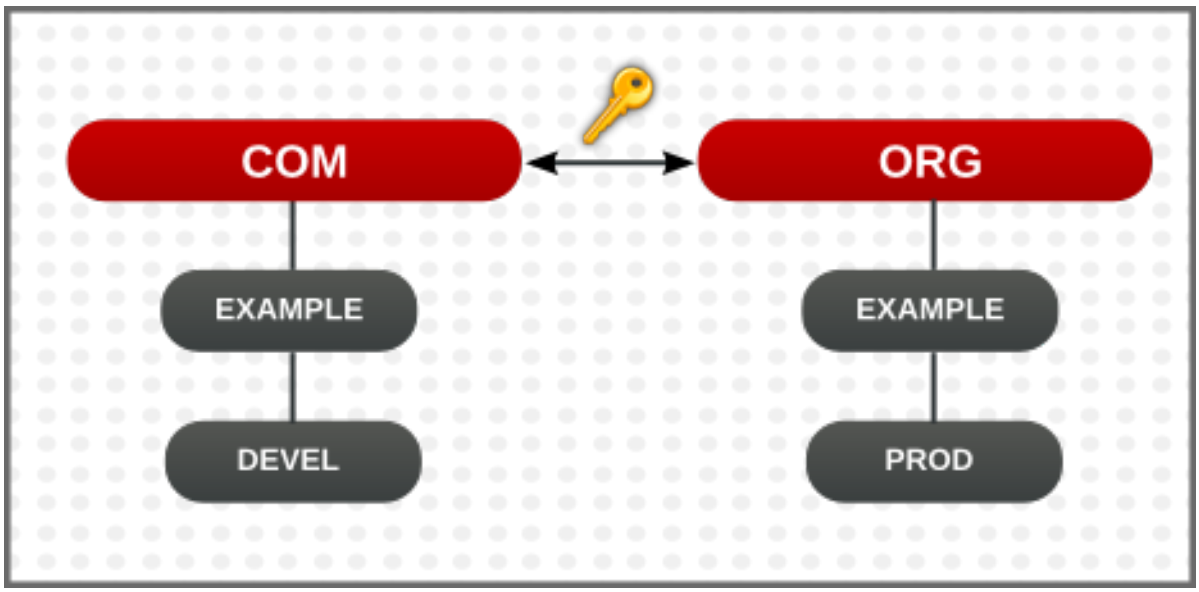


Figure 10.6. Trusts in Different Domains

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.

10.5.2. 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 bi-directional 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 11. 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 and revoke 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. As part of managing certificates, 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**.

11.1. 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://git.fedorahosted.org/cgit/certmonger.git/tree/doc/submit.txt.

11.2. Requesting a 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 desired 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 an ID for the request; **certmonger** uses this nickname to refer to the combination of storage locations and request options, and the nickname is also displayed in the output of the **getcert list** command. If you do not specify this option, **certmonger** assigns an automatically-generated nickname.

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 11.1. 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
```

11.3. 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
```

11.4. 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 an ID for the tracking request; **certmonger** uses this nickname to refer to the combination of storage locations and request options, and the nickname is also displayed in the output of the **getcert list** command. If you do not specify this option, **certmonger** assigns an automatically-generated nickname.

```
[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 12. 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.

12.1. 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.

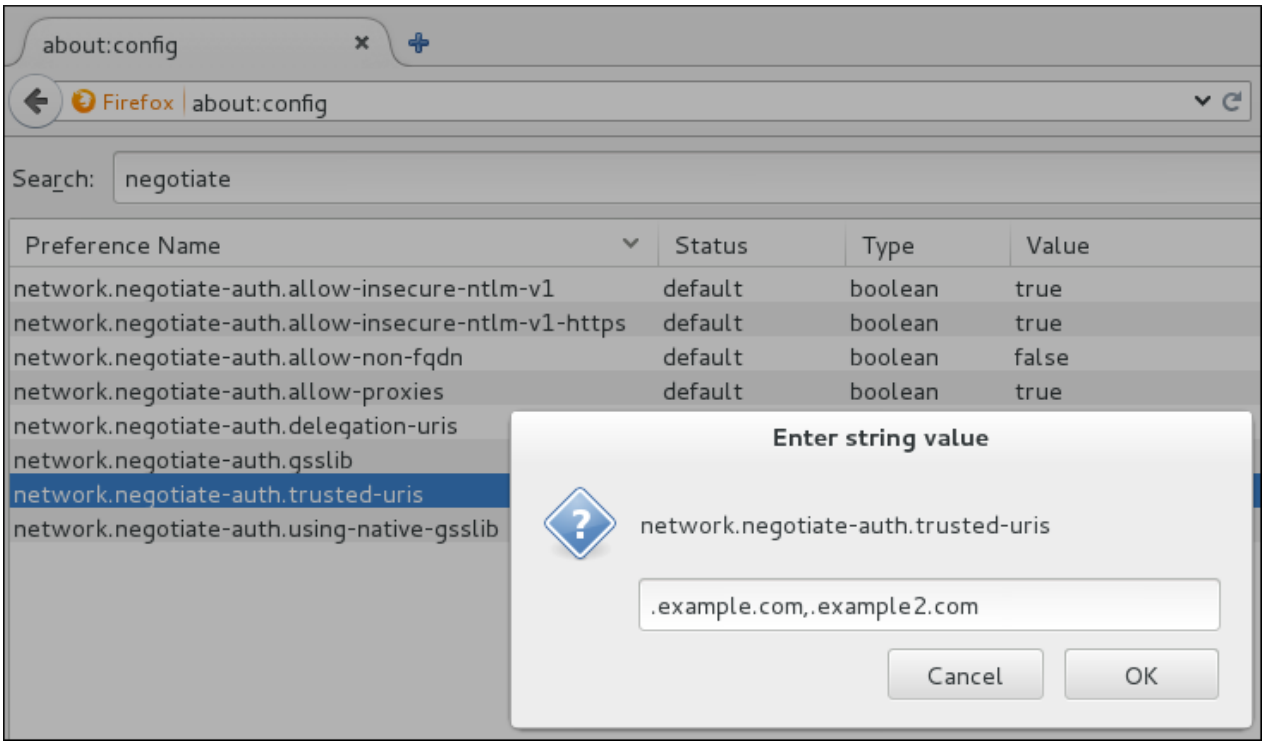


Figure 12.1. Manual Firefox Configuration

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 Kerberized server to act as the user.

Note

For information about configuring Firefox to use Kerberos in Identity Management, refer to the corresponding section in the Linux Domain Identity, Authentication, and Policy Guide.

12.1.1. Firefox Configuration for Kerberos Troubleshooting

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.

12.2. Certificate Management in Firefox

To manage certificates in Firefox, open the **Certificate Manager**.

1. In Mozilla Firefox, open the Firefox menu and click **Preferences**.

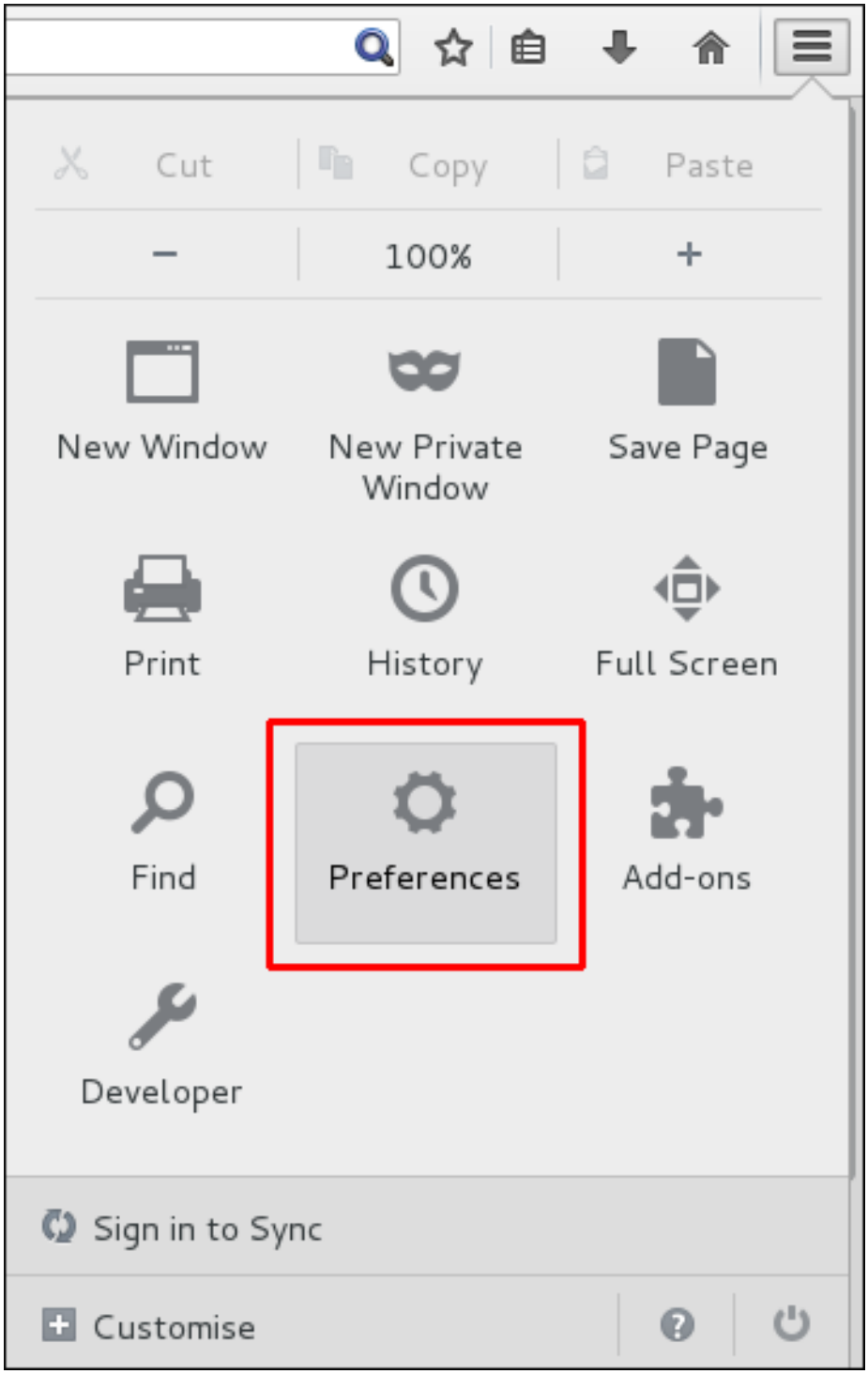


Figure 12.2. Firefox Preferences

2. Open the **Advanced** section and choose the **Certificates** tab.

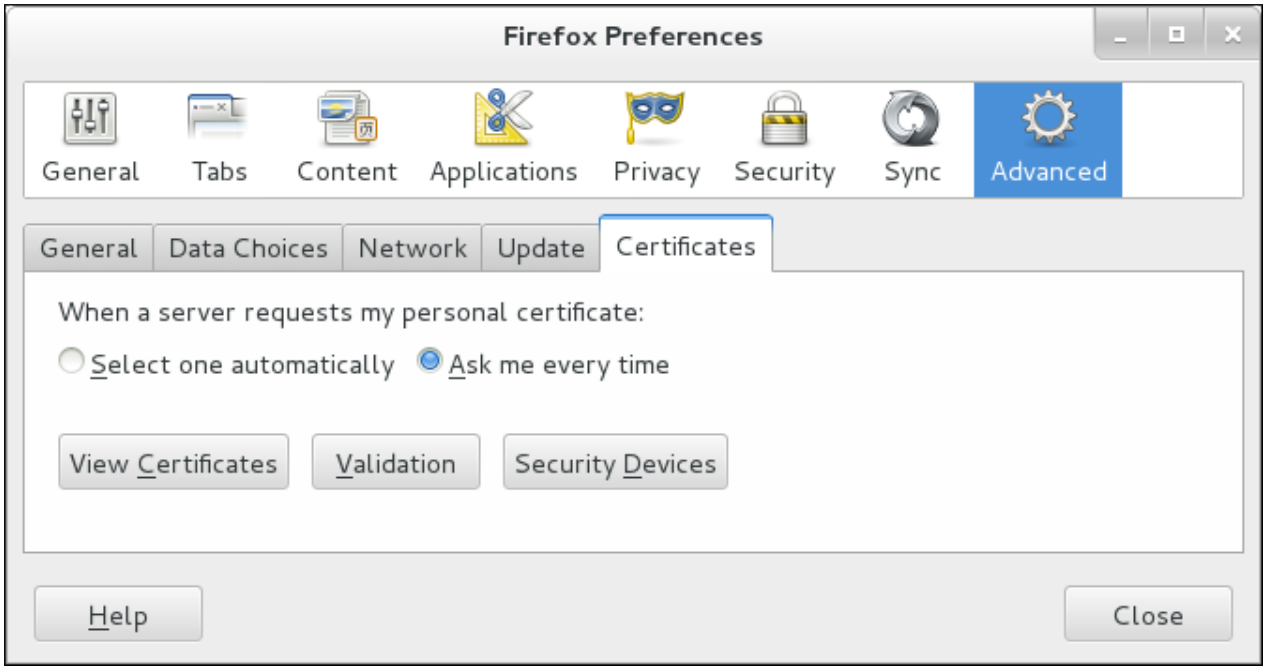


Figure 12.3. Certificates Tab in Firefox

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`**.

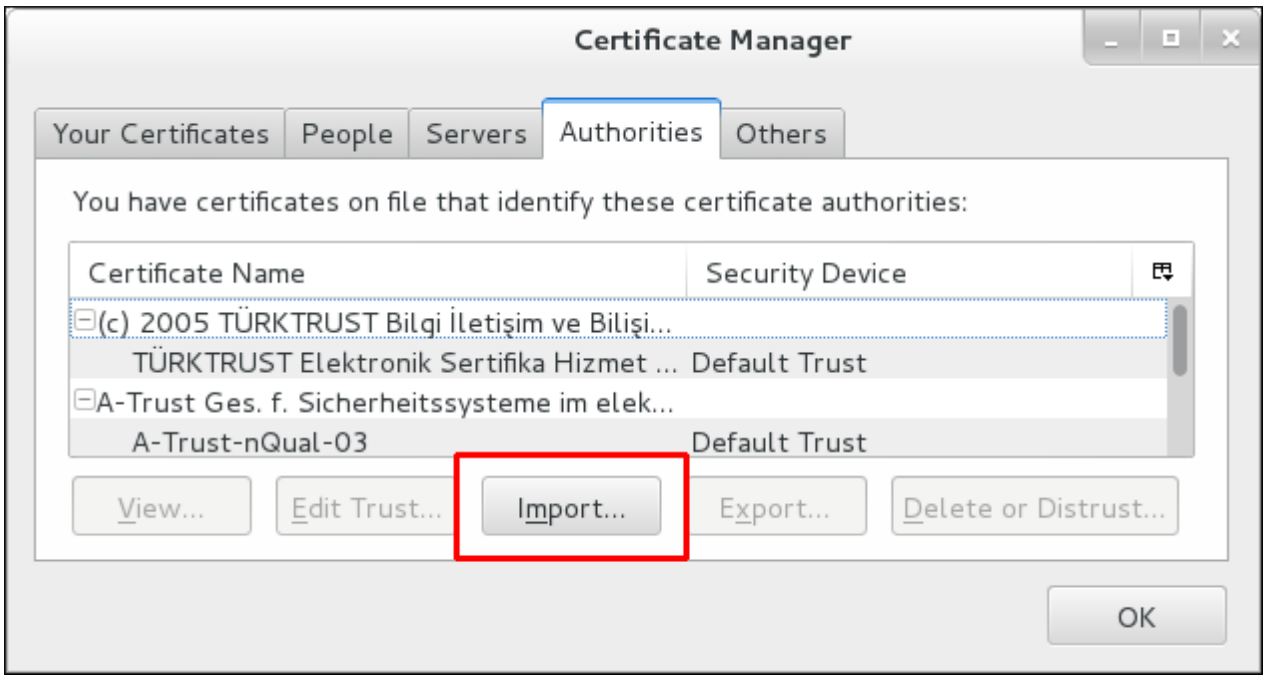


Figure 12.4. Importing the CA Certificate in Firefox

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.

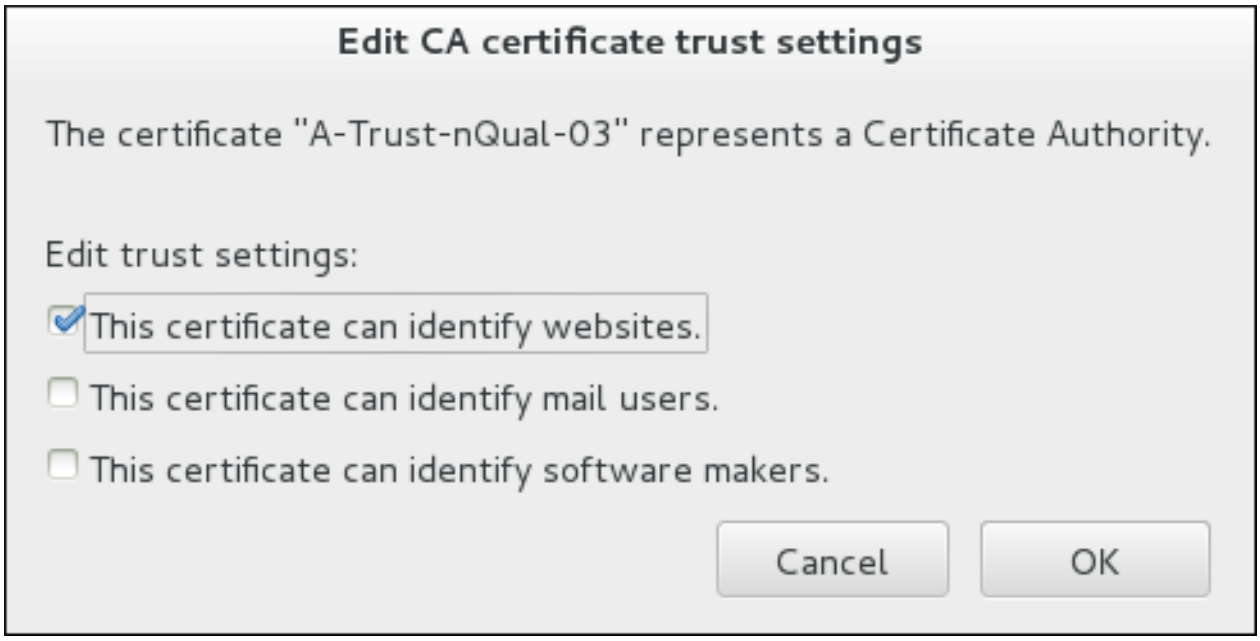


Figure 12.5. Editing the Certificate Trust Settings in Firefox

To use a personal certificate for authentication:

1. In the **Certificate Manager**, under the **Your Certificates** tab, click **Import**.

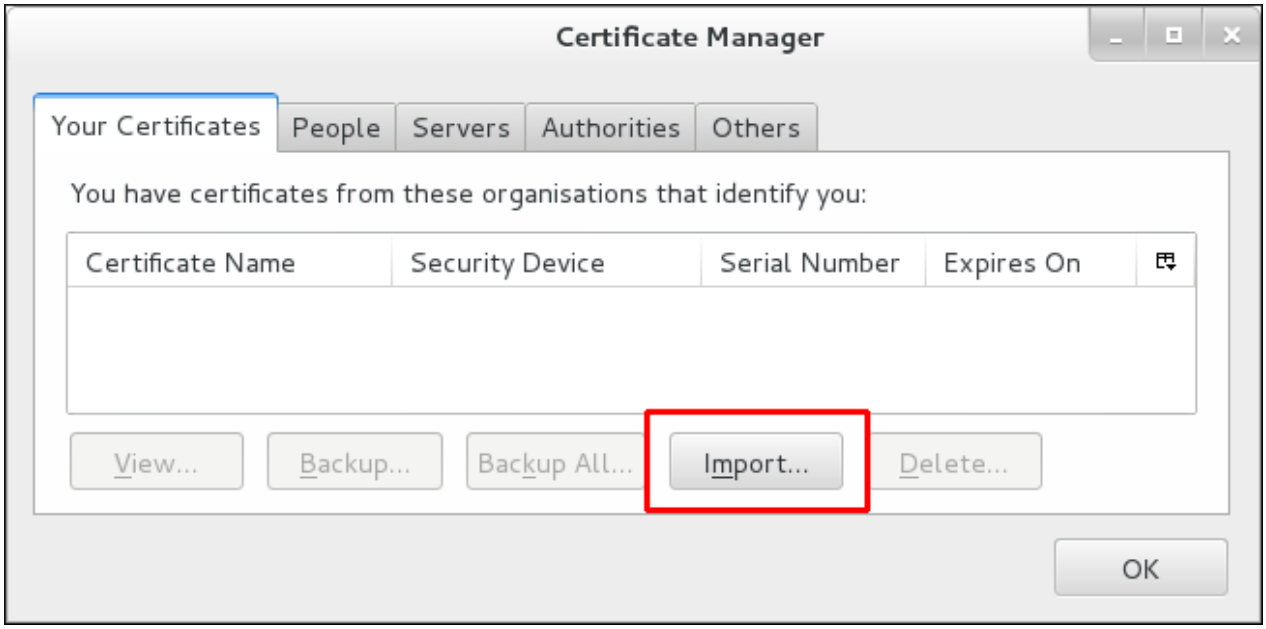


Figure 12.6. Importing a Personal Certificate for Authentication in Firefox

2. Select the required certificate from your computer.

12.3. 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**.

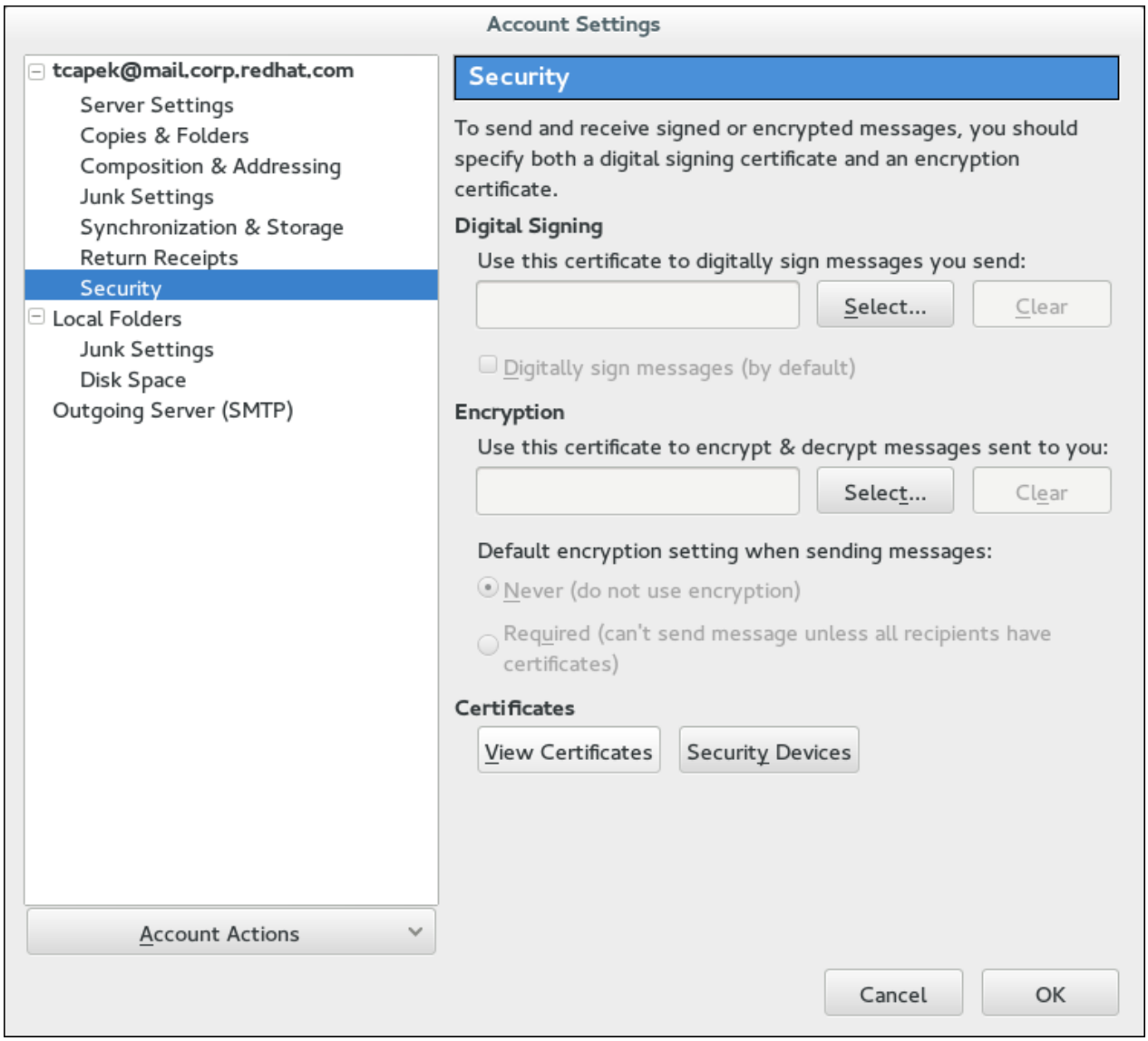


Figure 12.7. Account Settings in Thunderbird

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**.

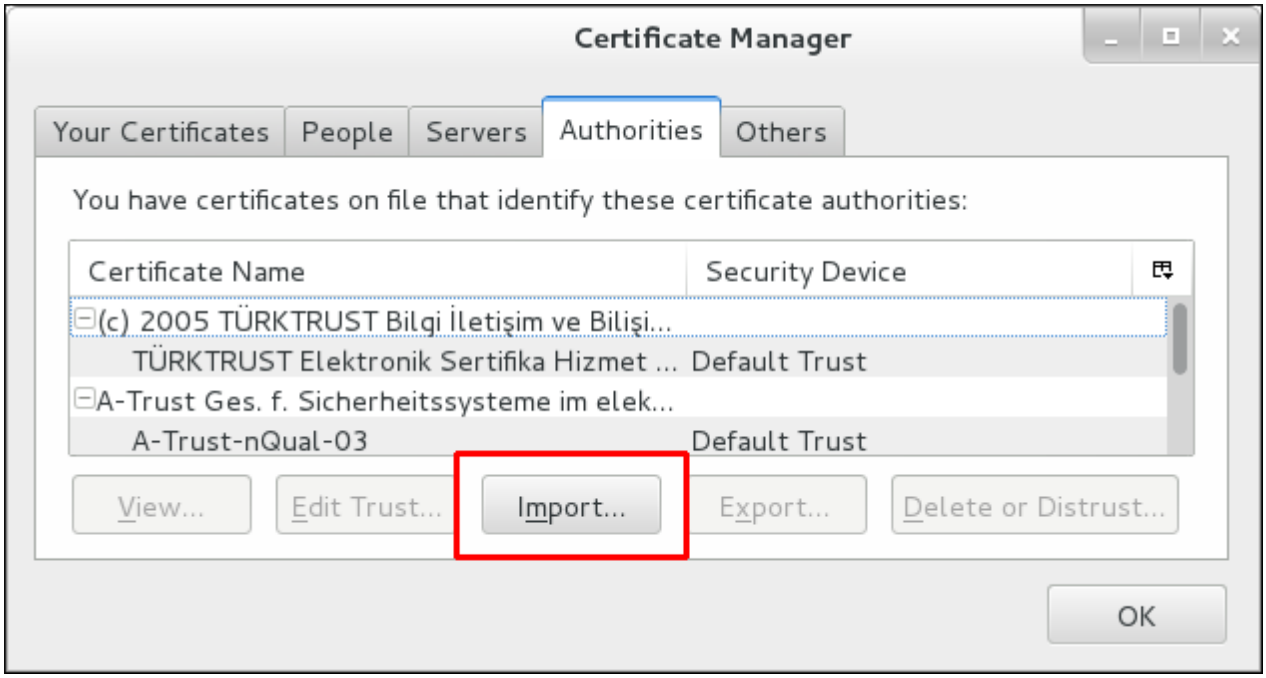


Figure 12.8. Importing the CA Certificate in Thunderbird

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.

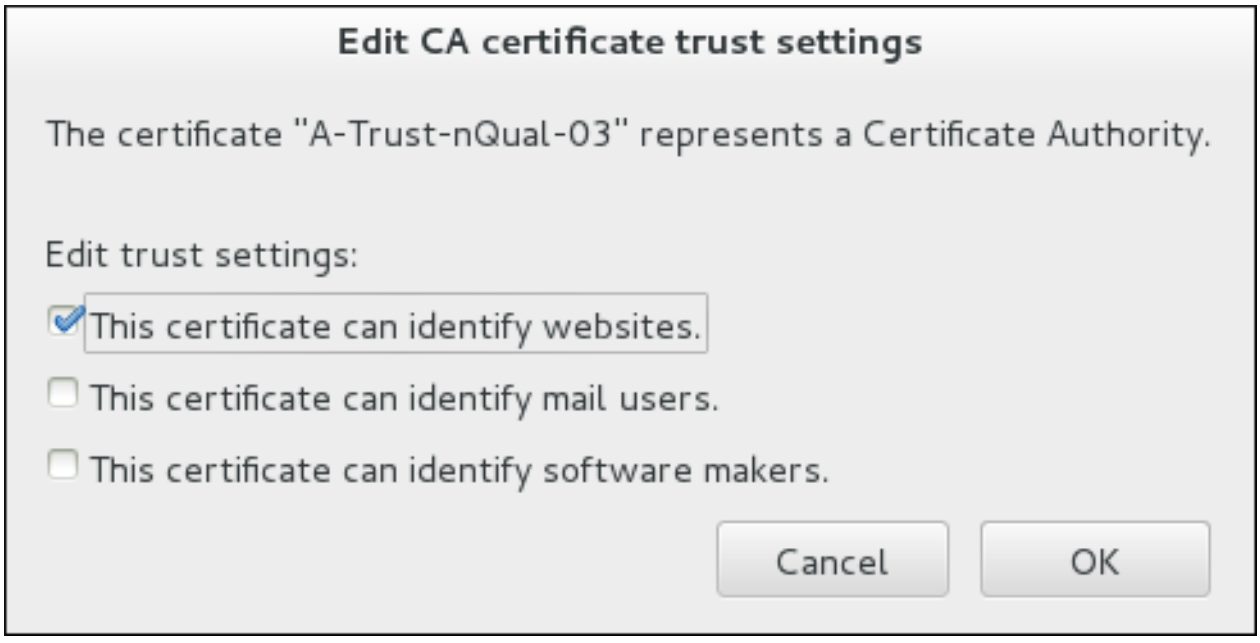


Figure 12.9. Editing the Certificate Trust Settings in Thunderbird

To use a personal certificate for authentication:

1. In the **Certificate Manager**, under the **Your Certificates** tab, click **Import**.

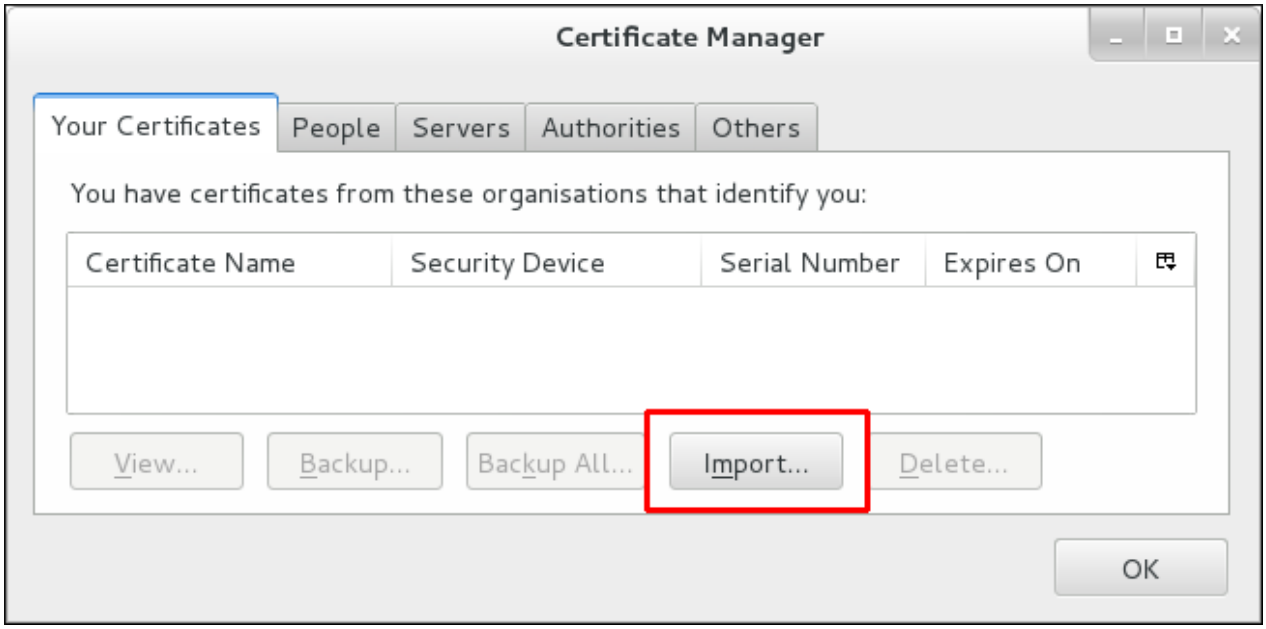


Figure 12.10. Importing a Personal Certificate for Authentication in Thunderbird

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. Revision History

Note that revision numbers relate to the edition of this manual, not to version numbers of Red Hat Enterprise Linux.

Revision 7.0-11 **Thu Mar 03 2016** **Aneta Petrová**
Async update: added restricting domains for PAM services.

Revision 7.0-10 **Tue Feb 09 2016** **Aneta Petrová**
Async update: 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.