NAME

PAM − PAM Service Module APIs

SYNOPSIS

#include <security/pam_appl.h>
#include <security/pam_modules.h>

cc [*flag* ...] *file* ... **−lpam** [*library* ...]

DESCRIPTION

PAM gives system administrators the flexibility of choosing any authentication service available on the system to perform authentication. The framework also allows new authentication service modules to be plugged in and made available without modifying the applications.

The PAM framework, **libpam,** consists of an interface library and multiple authentication service modules. The PAM interface library is the layer implementing the Application Programming Interface (API). The authentication service modules are a set of dynamically loadable objects invoked by the PAM API to provide a particular type of user authentication.

This manual page gives an overview of the PAM APIs for the service modules.

Interface Overview

The PAM service module interface consists of functions which can be grouped into four categories. The names for all the authentication library functions start with **pam_sm.** The only difference between the **pam_*()** interfaces and their corresponding **pam_sm_*()** interfaces is that all the **pam_sm_*()** interfaces require extra parameters to pass service specific options to the shared modules. They are otherwise identical.

The first category contains functions to authenticate an individual user (**pam_sm_authenticate**(3)) and to set the credentials of the user **(pam_sm_setcred**(3)). These back-end functions implement the functionality of **pam_authenticate**(3) and **pam_setcred**(3) respectively.

The second category contains functions to do account management (**pam_sm_acct_mgmt**(3)). This includes checking for password aging and access-hour restrictions. This back-end function implements the functionality of **pam_acct_mgmt**(3).

The third category contains functions to perform session management (**pam_sm_open_session**(3) and **pam_sm_close_session**(3)) after access to the system has been granted. These back-end functions implement the functionality of **pam_open_session**(3) and **pam_close_session**(3), respectively.

The fourth category consists a function to change authentication tokens (**pam_sm_chauthtok**(3)). This back-end function implements the functionality of **pam_chauthtok**(3).

Stateful Interface

A sequence of calls sharing a common set of state information is referred to as an authentication transaction. An authentication transaction begins with a call to **pam_start()**. **pam_start()** allocates space, performs various initialization activities, and assigns an authentication handle to be used for subsequent calls to the library. Note that the service modules do not get called or initialized when **pam_start()** is called. The modules are loaded and the symbols resolved upon first use of that function.

The PAM handle keeps certain information about the transaction that can be accessed through the **pam_get_item()** API. Though the modules can also use **pam_set_item()** to change any of the item information, it is recommended that nothing be changed except PAM_AUTHTOK and PAM_OLDAUTHTOK.

If the modules want to store any module specific state information then they can use the **pam_set_data**(3) function to store that information with the PAM handle. The data should be stored with a name which is unique across all modules and module types. For example, SUNW_PAM_UNIX_AUTH_userid can be used as a name by the UNIX module to store information about the state of user's authentication. Some modules use this technique to share data across two different module types.

Also, during the call to **pam_authenticate()**, the UNIX module may store the authentication status (success or reason for failure) in the handle, using a unique name such as SUNW_SECURE_RPC_DATA. This information is intended for use by **pam_setcred()**.

During the call to **pam_acct_mgmt()**, the account modules may store data in the handle to indicate which passwords have aged. This information is intended for use by **pam_chauthtok()**.

The module can also store a cleanup function associated with the data. The PAM framework calls this cleanup function, when the application calls **pam_end()** to close the transaction.

Interaction with the User

The PAM service modules do not communicate directly with the user; instead they rely on the application to perform all such interactions. The application passes a pointer to the function, **conv(),** along with any associated application data pointers, through the **pam_conv** structure when it initiates an authentication transaction (via a call to **pam_start()**). The service module will then use the function, **conv()**, to prompt the user for data, output error messages, and display text information. Refer to **pam_start**(3) for more information. The modules are responsible for the localization of all messages to the user.

CONVENTIONS

Though the PAM framework enforces no rules about the module's names, location, options and such, there are certain conventions that all module providers are expected to follow.

By convention, the modules should be located in the **/usr/lib/security** directory. Additional modules may be located in **/opt/<pkg>/lib.**

By convention, the modules are named **pam_<service_name>_<module_type>.so.1.** If the given module implements more than one module type (for example, **pam_unix.so.1** module), then the module_type suffix should be dropped.

For every such module, there should be a corresponding manual page in section 5 which should describe the *module_type* it supports, the functionality of the module, along with the options it supports. The dependencies should be clearly identified to the system administrator. For example, it should be made clear whether this module is a stand-alone module or depends upon the presence of some other module. One should also specify whether this module should come before or after some other module in the stack.

By convention, the modules should support the following options:

debug
: Syslog debugging information at LOG_DEBUG level. Be careful as to not log any sensitive information such as passwords.

nowarn
: Turn off warning messages such as "password is about to expire"

In addition, it is recommended that the auth and the password module support the following options:

use_first_pass
: Instead of prompting the user for the password, use the user's initial password (entered when the user was authenticated to the first authentication module in the stack) for authentication. If the passwords do not match, or if no password has been entered, return failure and do not prompt the user for a password. Support for this scheme allows the user to type only one password for multiple schemes.

try_first_pass
: Instead of prompting the user for the password, use the user's initial password (entered when the user was authenticated to the first authentication module in the stack) for authentication. If the passwords do not match, or if no password has been entered, prompt the user for a password after identifying which type of password (ie. UNIX, etc.) is being requested. Support for this scheme allows the user to try to use only one password for multiple schemes, and type multiple passwords only if necessary.

If an unsupported option is passed to the modules, it should syslog the error at LOG_ERR level.

The permission bits on the service module should be set such that it is not writable by either "group" or "other". The PAM framework will not load the module if the above permission rules are not followed.

ERROR LOGGING

If there are any errors, the modules should log them using **syslog**(3) at the LOG_ERR level.

RETURN VALUES

The PAM service module functions may return any of the PAM error numbers specified in the specific man pages. It can also return a PAM_IGNORE error number to mean that the PAM framework should ignore this module regardless of whether it is required, optional or sufficient. This error number is normally returned when the module does not want to deal with the given user at all.

SEE ALSO

pam(3), **pam_start**(3), **pam_set_item**(3), **pam_authenticate**(3), **pam_open_session**(3), **pam_setcred**(3), **pam_chauthtok**(3), **pam_strerror**(3), **pam_sm_authenticate**(3), **pam_sm_open_session**(3), **pam_sm_setcred**(3), **pam_sm_chauthtok**(3), **pam.conf**(4)

NAME

pam_sm_acct_mgmt − Service provider implementation for pam_acct_mgmt

SYNOPSIS

cc [*flag* ...] *file* ... **−lpam** [*library* ...]

#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_acct_mgmt(pam_handle_t **pamh*, **int** *flags*, **int** *argc*, **const char** ***argv*);

DESCRIPTION

In response to a call to **pam_acct_mgmt**(3), the PAM framework calls **pam_sm_acct_mgmt()** from the modules listed in the **pam.conf**(4) file. The account management provider supplies the back-end functionality for this interface function. The applications should not call this API directly.

The function, **pam_sm_acct_mgmt()**, determines whether the current user's account and password are valid. This includes checking for password and account expiration, valid log-in times, etc. The user in question is specified by a prior call to **pam_start()**, and is referenced by the authentication handle, *pamh,* which is passed as the first argument to **pam_sm_acct_mgmt().** The following flags may be set in the *flags* field:

PAM_SILENT	The account management service should not generate any messages
PAM_DISALLOW_NULL_AUTHTOK	The account management service should return PAM_AUTHTOKEN_REQD if the user has a null authentication token

The *argc* argument represents the number of module options passed in from the configuration file **pam.conf**(4). *argv* specifies the module options, which are interpreted and processed by the account management service. Please refer to the specific module man pages for the various available *options.* If an unknown option is passed to the module, an error should be logged through **syslog**(3) and the option ignored.

If an account management module determines that the user password has aged or expired, it should save this information as state in the authentication handle, *pamh,* using **pam_set_data()**. **pam_chauthok()** uses this information to determine which passwords have expired.

RETURN VALUES

If there are no restrictions to logging in, PAM_SUCCESS is returned. The following error values may also be returned upon error:

PAM_USER_UNKNOWN	User not known to underlying authentication module
PAM_AUTHTOKEN_REQD	New authentication token required
PAM_ACCT_EXPIRED	User account has expired
PAM_PERM_DENIED	User denied access to account at this time
PAM_IGNORE	Ignore underlying account module regardless of whether the control flag is *required, optional*

or *sufficient*

SEE ALSO **pam**(3), **pam_acct_mgmt**(3), **syslog**(3), **pam.conf**(4)

NAME

pam_sm_authenticate − Service provider implementation for pam_authenticate

SYNOPSIS

cc [*flag* ...] *file* ... **−lpam** [*library* ...]

#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_authenticate(pam_handle_t **pamh*, **int** *flags*, **int** *argc*, **const char** ***argv***);**

DESCRIPTION

In response to a call to **pam_authenticate**(3), the PAM framework calls **pam_sm_authenticate()** from the modules listed in the **pam.conf**(4) file. The authentication provider supplies the back-end functionality for this interface function.

The function, **pam_sm_authenticate()**, is called to verify the identity of the current user. The user is usually required to enter a password or similar authentication token depending upon the authentication scheme configured within the system. The user in question is specified by a prior call to **pam_start()**, and is referenced by the authentication handle, *pamh.*

If the user is unknown to the authentication service, the service module should mask this error and continue to prompt the user for a password. It should then return the error, PAM_USER_UNKNOWN.

The following flag may be passed in to **pam_sm_authenticate():**

PAM_SILENT
: The authentication service should not generate any messages

PAM_DISALLOW_NULL_AUTHTOK
: The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token

The *argc* argument represents the number of module options passed in from the configuration file **pam.conf**(4). *argv* specifies the module options, which are interpreted and processed by the authentication service. Please refer to the specific module man pages for the various available *options.* If any unknown option is passed in, the module should log the error and ignore the option.

Before returning, **pam_sm_authenticate()** should call **pam_get_item()** and retrieve PAM_AUTHTOK. If it has not been set before (ie. the value is NULL), **pam_sm_authenticate()** should set it to the password entered by the user using **pam_set_item()**.

An authentication module may save the authentication status (success or reason for failure) as state in the authentication handle using **pam_set_data**(3). This information is intended for use by **pam_setcred()**.

NOTES

Modules should not retry the authentication in the event of a failure. Applications handle authentication retries and maintain the retry count. To limit the number of retries, the module can return a PAM_MAXTRIES error.

RETURN VALUES

Upon successful completion, PAM_SUCCESS must be returned. In addition, the following values may be returned:

PAM_MAXTRIES	Maximum number of authentication attempts exceeded
PAM_AUTH_ERR	Authentication failure
PAM_CRED_INSUFFICIENT	Can not access authentication data due to insufficient credentials
PAM_AUTHINFO_UNAVAIL	Underlying authentication service can not retrieve authentication information
PAM_USER_UNKNOWN	User not known to underlying authentication module
PAM_IGNORE	Ignore underlying authentication module regardless of whether the control flag is *required*, *optional* or *sufficient*

SEE ALSO

pam(3), **pam_authenticate**(3), **pam.conf**(4)

NAME

pam_sm_chauthtok − Service provider implementation for pam_chauthtok

SYNOPSIS

cc [*flag* ...] *file* ... **−lpam** [*library* ...]

#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_chauthtok(pam_handle_t **pamh*, **const int** *flags***);**

DESCRIPTION

In response to a call to **pam_chauthtok**(3) the PAM framework calls **pam_sm_chauthtok**(3) from the modules listed in the **pam.conf**(4) file. The password management provider supplies the back-end functionality for this interface function.

pam_sm_chauthtok() changes the authentication token associated with a particular user referenced by the authentication handle, *pamh.*

The following flag may be passed in to **pam_chauthtok()**:

PAM_SILENT
: The password service should not generate any messages

PAM_CHANGE_EXPIRED_AUTHTOK
: The password service should only update those passwords that have aged. If this flag is not passed, the password service should update all passwords.

PAM_PRELIM_CHECK
: The password service should only perform preliminary checks. No passwords should be updated.

PAM_UPDATE_AUTHTOK
: The password service should update passwords

Note that PAM_PRELIM_CHECK and PAM_UPDATE_AUTHTOK can not be set at the same time.

Upon successful completion of the call, the authentication token of the user will be ready for change or will be changed (depending upon the flag) in accordance with the authentication scheme configured within the system.

The *argc* argument represents the number of module options passed in from the configuration file **pam.conf**(4). *argv* specifies the module options, which are interpreted and processed by the password management service. Please refer to the specific module man pages for the various available *options.*

It is the responsibility of **pam_sm_chauthtok()** to determine if the new password meets certain strength requirements. **pam_sm_chauthtok()** may continue to re-prompt the user (for a limited number of times) for a new password until the password entered meets the strength requirements.

Before returning, **pam_sm_chauthtok()** should call **pam_get_item()** and retrieve both PAM_AUTHTOK and PAM_OLDAUTHTOK. If both are NULL, **pam_sm_chauthtok()** should set them to the new and old passwords as entered by the user.

NOTES

The PAM framework invokes the password services twice. The first time the modules are invoked with the flag, PAM_PRELIM_CHECK. During this stage, the password modules should only perform preliminary checks (ping remote name services to see if they are ready for updates, for example). If a password module detects a transient error (remote name service temporarily down, for example) it should return PAM_TRY_AGAIN to the PAM framework, which will immediately return the error back to the application. If all password modules pass the preliminary check, the PAM framework invokes the password services again with the flag, PAM_UPDATE_AUTHTOK. During this stage, each password module should proceed to update the appropriate password. Any error will again be reported back to application.

If a service module receives the flag, PAM_CHANGE_EXPIRED_AUTHTOK, it should check whether the password has aged or expired. If the password has aged or expired, then the service module should proceed to update the password. If the status indicates that the password has not yet aged/expired, then the password module should return PAM_IGNORE.

If a user's password has aged or expired, a PAM account module could save this information as state in the authentication handle, pamh, using **pam_set_data()**. The related password management module could retreive this information using **pam_get_data()** to determine whether or not it should prompt the user to update the password for this particular module.

RETURN VALUES

Upon successful completion, PAM_SUCCESS must be returned. The following values may also be returned:

PAM_PERM_DENIED	No permission
PAM_AUTHTOK_ERR	Authentication token manipulation error
PAM_AUTHTOK_RECOVERY_ERR	Old authentication token cannot be recovered
PAM_AUTHTOK_LOCK_BUSY	Authentication token lock busy
PAM_AUTHTOK_DISABLE_AGING	Authentication token aging disabled
PAM_USER_UNKNOWN	User unknown to password service
PAM_TRY_AGAIN	Preliminary check by password service failed

SEE ALSO

pam(3), **pam_chauthtok**(3), **pam.conf**(4)

NAME

pam_sm_open_session, pam_sm_close_session − Service provider implementation for pam_open_session and pam_close_session respectively

SYNOPSIS

cc [*flag* ...] *file* ... **−lpam** [*library* ...]

#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_open_session(pam_handle_t **pamh*, **int** *flags*, **int** *argc*, **const char** ***argv***);**

int pam_sm_close_session(pam_handle_t **pamh*, **int** *flags*, **int** *argc*, **const char** ***argv***);**

DESCRIPTION

In response to a call to pam_open_session**(3)** and **pam_close_session**(3), the PAM framework calls **pam_sm_open_session( )** and **pam_sm_close_session( )**, respectively from the modules listed in the **pam.conf**(4) file. The session management provider supplies the back-end functionality for this interface function.

pam_sm_open_session( ) is called to initiate session management. **pam_sm_close_session( )** is invoked when a session has terminated. The argument *pamh* is an authentication handle. The following flag may be set in the *flags* field:

PAM_SILENT	Session service should not generate any messages

The *argc* argument represents the number of module options passed in from the configuration file **pam.conf**(4). *argv* specifies the module options, which are interpreted and processed by the session management service. If an unknown option is passed in, an error should be logged through **syslog**(3) and the option ignored.

RETURN VALUES

Upon successful completion, PAM_SUCCESS should be returned. The following values may also be returned upon error:

PAM_SESSION_ERR	Can not make/remove an entry for the specified session
PAM_IGNORE	Ignore underlying session module regardless of whether the control flag is *required, optional* or *sufficient*

SEE ALSO

pam(3), **pam_open_session**(3), **syslog**(3), **pam.conf**(4)

NAME

pam_sm_setcred − Service provider implementation for pam_setcred

SYNOPSIS

cc [*flag* ...] *file* ... **−lpam** [*library* ...]

#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_setcred(pam_handle_t **pamh*, **int** *flags*, **int** *argc*, **const char** ***argv***);**

DESCRIPTION

In response to a call to **pam_set_cred**(3), the PAM framework calls **pam_sm_setcred()** from the modules listed in the **pam.conf**(4) file. The authentication provider supplies the back-end functionality for this interface function.

pam_sm_setcred() is called to set the credentials of the current user associated with the authentication handle, *pamh.* The following flags may be set in the *flags* field. Note that the first four flags are mutually exclusive:

PAM_CRED_ESTABLISH	Set user credentials for the authentication service
PAM_CRED_DELETE	Delete user credentials associated with the authentication service
PAM_CRED_REINITIALIZE	Reinitialize user credentials
PAM_CRED_REFRESH	Extend lifetime of user credentials
PAM_SILENT	Authentication service should not generate messages

If none of these flags are set, PAM_CRED_ESTABLISH is used as the default.

The *argc* argument represents the number of module options passed in from the configuration file **pam.conf**(4). *argv* specifies the module options, which are interpreted and processed by the authentication service. If an unknown option is passed to the module, an error should be logged and the option ignored.

If the PAM_SILENT flag is not set, then **pam_sm_setcred()** should print any failure status from the corresponding **pam_sm_authenticate()** function using the conversation function.

The authentication status (success or reason for failure) is saved as module-specific state in the authentication handle by the authentication module. The status should be retreived using **pam_get_data()**, and used to determine if user credentials should be set.

NOTES

pam_sm_setcred() is passed the same module options that are used by **pam_sm_authenticate()**.

RETURN VALUES

Upon successful completion, PAM_SUCCESS should be returned. The following values may also be returned upon error:

PAM_CRED_UNAVAIL	Underlying authentication service can not retrieve user credentials

PAM_CRED_EXPIRED	User credentials have expired
PAM_USER_UNKNOWN	User unknown to the authentication service
PAM_CRED_ERR	Failure in setting user credentials
PAM_IGNORE	Ignore underlying authentication module regardless of whether the control flag is *required*, *optional* or *sufficient*

SEE ALSO

pam(3), **pam_authenticate**(3), **pam_setcred**(3), **pam_sm_authenticate**(3), **pam.conf**(4)