NAME

PAM − Pluggable Authentication Module

SYNOPSIS

#include <security/pam_appl.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.

Interface Overview

The PAM library interface consists of functions which can be grouped into five categories. The names for all the authentication library functions start with **pam_.**

The first category contains functions for establishing and terminating an authentication activity (**pam_start**(3) and (**pam_end**(3)), functions to maintain module specific data (**pam_[sg]et_data**(3)), functions to maintain state information (**pam_[sg]et_item**(3)), and a function to return error status information (**pam_strerror**(3)).

The second category contains functions to authenticate an individual user (**pam_authenticate**(3)) and to set the credentials of the user (**pam_setcred**(3)).

The third category contains functions to do account management (**pam_acct_mgmt**(3)). This includes checking for password aging and access-hour restrictions.

The fourth category contains functions to perform session management (**(pam_open_session**(3) and (**pam_close_session**(3)) after access to the system has been granted.

The fifth category consists of functions to change authentication tokens **pam_chauthtok**(3)). An authentication token is the object used to verify the identity of the user. In UNIX, an authentication token is a user’s password.

All the **pam_*()** interfaces are implemented through the library **libpam.** For each of the categories listed above, excluding the first category **pam_start()**, **pam_end()**, **pam_[sg]et_data()**, **pam_[sg]et_item()**, and **pam_strerror()**) there exists a dynamically loadable shared module that provides the appropriate service layer functionality upon demand. The functional entry points in the service layer start with the **pam_sm_** prefix. The only difference between the **pam_sm_*()** interfaces and their corresponding **pam_** interfaces is that all the **pam_sm_*()** interfaces require extra parameters to pass service specific options to the shared modules. Please refer to **pam_sm**(3) for an overview of the PAM service module APIs.

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 a PAM authentication handle to be used for subsequent calls to the library.

After initiating an authentication transaction, applications can invoke **pam_authenticate()** to authenticate a particular user, and **pam_acct_mgmt()** to perform system entry management (the application may want to determine if the user's password has expired).

If the user has been successfully authenticated, applications call **pam_setcred()** to set any user credentials associated with the authentication service. Within one authentication transaction (between **pam_start()** and **pam_end()**), all calls to the PAM interface should be made with the same authentication handle returned by **pam_start()**. This is necessary because certain service modules may store module-specific data in the handle that is intended for use by other modules. For example, during the call to **pam_authenticate()**, service modules may store data in the handle that is intended for use by **pam_setcred()**.

To perform session management, applications call **pam_open_session()**. For example, the system may want to store the total time for the session. The function **pam_close_session()** closes the current session.

When necessary, applications can call **pam_get_item()** and **pam_set_item()** to access and update specific authentication information. Such information may include the current username.

To terminate an authentication transaction, the application simply calls **pam_end()**, which frees previously allocated space used to store authentication information.

Application - Authentication Service Interactive Interface

The authentication service in PAM does not communicate directly with the user; instead it relies 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 a *pam_conv* structure to the authentication service when it initiates an authentication transaction (via a call to **pam_start()**). The service 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.

Stacking Multiple Schemes

The PAM architecture enables authentication by multiple authentication services through *stacking.* System entry applications, such as **login**(1), stack multiple service modules to authenticate users with multiple authentication services. The order in which authentication service modules are stacked is specified in the configuration file, **pam.conf**(4). A system administrator determines this ordering, and also determines whether the same password can be used for all authentication services.

Administrative Interface

The authentication library, **/usr/lib/libpam.so.1**, implements the framework interface. Various authentication services are implemented by their own loadable modules whose paths are specified through the **pam.conf**(4) file.

RETURN VALUES

The PAM functions may return one of the following generic values, or one of the values defined in the specific man pages:

PAM_SUCCESS	Successful function return
PAM_OPEN_ERR	**dlopen()** failure when dynamically loading a service module
PAM_SYMBOL_ERR	Symbol not found
PAM_SERVICE_ERR	Error in service module
PAM_SYSTEM_ERR	System error
PAM_BUF_ERR	Memory buffer error
PAM_CONV_ERR	Conversation failure
PAM_PERM_DENIED	Permission denied

SEE ALSO

pam_authenticate(3), **pam_open_session**(3), **pam_chauthtok**(3), **pam_set_item**(3), **pam_setcred**(3), **pam_sm**(3), **pam_start**(3), **pam_strerror**(3), **pam.conf**(4)

WARNING

Please note that all the PAM APIs and the data structures are subject to change without notice.

NAME

pam_start, pam_end − authentication transaction routines for PAM

SYNOPSIS

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

#include <security/pam_appl.h>

int pam_start(const char **service*, **const char** **user*, **const struct pam_conv** **pam_conv*, **pam_handle_t** ***pamh***);**

int pam_end(pam_handle_t **pamh*, **int** *status***);**

DESCRIPTION

pam_start() is called to initiate an authentication transaction. **pam_start()** takes as arguments the name of the current service, *service,* the name of the user to be authenticated, *user,* the address of the conversation structure, *pam_conv,* and the address of a variable to be assigned the authentication handle, *pamh.* Upon successful completion, *pamh* will refer to a PAM handle for use with subsequent calls to the authentication library.

The pam_conv structure, *pam_conv,* contains the address of the conversation function provided by the application. The underlying PAM service module invokes this function to output information to and retrieve input from the user. The *pam_conv* structure has the following entries:

```
struct pam_conv {
        int     (*conv)();          /* Conversation function */
        void    *appdata_ptr;       /* Application data */
};
where
        int conv(int num_msg,
                const struct pam_message **msg, struct pam_response **resp,
                void *appdata_ptr);
```

The function **conv()** is called by a service module to hold a PAM conversation with the application or user. For window applications, the application can create a new pop-up window to be used by the interaction.

The parameter *num_msg* is the number of messages associated with the call. The parameter *msg* is a pointer to an array of length *num_msg* of the *pam_message* structure.

The structure *pam_message* is used to pass prompt, error message, or any text information from the authentication service to the application or user. It is the responsibility of the PAM service modules to localize the messages. The memory used by *pam_message* has to be allocated and freed by the PAM modules. The *pam_message* structure has the following entries:

```
struct pam_message{
        int     msg_style;
        char    *msg;
};
```

The message style, *msg_style*, can be set to one of the following values:

PAM_PROMPT_ECHO_OFF	prompt user, disabling echoing of response
PAM_PROMPT_ECHO_ON	prompt user, enabling echoing of response
PAM_ERROR_MSG	print error message
PAM_TEXT_INFO	print general text information

The maximum size of the message and the response string is PAM_MAX_MSG_SIZE defined in <**security/pam.appl.h**>.

The structure *pam_response* is used by the authentication service to get the user's response back from the application or user. The storage used by *pam_response* has to be allocated by the application and freed by the PAM modules. The *pam_response* structure has the following entries:

```
struct pam_response{
        char    *resp;
        int     resp_retcode;       /* currently not used, should be set to 0 */
};
```

It is the responsibility of the conversation function to strip off newline characters for PAM_PROMPT_ECHO_OFF and PAM_PROMPT_ECHO_ON message styles, and to add newline characters (if appropriate) for PAM_ERROR_MSG and PAM_TEXT_INFO message styles.

appdata_ptr is an application data pointer which is passed by the application to the PAM service modules. Since the PAM modules pass it back through the conversation function, the applications can use this pointer to point to any application-specific data.

pam_end() is called to terminate the authentication transaction identified by *pamh* and to free any storage area allocated by the authentication module. The argument, *status*, is passed to the *cleanup(|)* function stored within the pam handle, and is used to determine what module specific state must be purged. A cleanup function is attached to the handle by the underlying PAM modules through a call to **pam_set_item**(3) to free module specific data.

RETURN VALUES

Refer to **pam**(3) for information on error related return values.

SEE ALSO

pam_authenticate(3), **pam_set_item**(3), **pam_acct_mgmt**(3), **pam_open_session**(3), **pam_setcred**(3), **pam_chauthtok**(3), **pam_strerror**(3), **pam**(3)

NAME

pam_strerror − get PAM error message string

SYNOPSIS

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

#include <security/pam_appl.h>

const char *pam_strerror(pam_handle_t*pamh***, int** *errnum***);**

DESCRIPTION

pam_strerror() maps the PAM error number in *errnum* to a PAM error message string, and returns a pointer to that string. The application should not free or modify the string returned.

The *pamh* arguement is the PAM handle obtained by a prior call to **pam_start()**. If **pam_start()** returns an error, a **NULL** PAM handle should be passed.

ERRORS

pam_strerror() returns **NULL** if *errnum* is out-of-range.

SEE ALSO

pam(3), **pam_start**(3)

NAME

pam_set_data, pam_get_data − PAM routines to maintain module specific state

SYNOPSIS

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

#include <security/pam_appl.h>

int pam_set_data(pam_handle_t **pamh*, **const char** **module_data_name*, **const void** **data*, **void** *(*cleanup)(pam_handle_t *pamh, void *data, int pam_end_status))***);**

int pam_get_data(const pam_handle_t **pamh*, **const char** **module_data_name*, **void** ***data***);**

DESCRIPTION

pam_set_data() and **pam_set_data()** allow PAM service modules to access and update module specific information as needed. These functions should not be used by applications.

pam_set_data() stores module specific data within the PAM handle, *pamh.* The *module_data_name* argument uniquely identifies the data, and the *data* argument represents the actual data. *module_data_name* should be unique across all services (UNIX, etc).

The *cleanup* function is used to free any memory used by the *data* after it is no longer needed, and is invoked by **pam_end()**. The *cleanup* function takes as its arguments a pointer to the PAM handle, *pamh,* a pointer to the actual data, *data,* and a status code, *pam_end_status.* The status code determines exactly what state information needs to be purged, and is therefore specific to each module.

If **pam_set_data()** is called and module data already exists under the same *module_data_name* (from a prior call to **pam_set_data()**), then the existing *data* is replaced by the new *data,* and the existing *cleanup* function is replaced by the new *cleanup* function.

pam_get_data() retrieves module specific data stored in the PAM handle, *pamh,* identified by the unique name, *module_data_name.* The *data* argument is assigned the address of the requested data.

RETURN VALUES

In addition to the return values listed in **pam**(3), the following value may also be returned:

PAM_NO_MODULE_DATA	No module specific data is present

SEE ALSO

pam(3), **pam_end**(3)

NAME

pam_set_item, pam_get_item − authentication information routines for PAM

SYNOPSIS

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

#include <security/pam_appl.h>

int pam_set_item(pam_handle_t **pamh*, **int** *item_type*, **const void** **item*);

int pam_get_item(const pam_handle_t **pamh*, **int** *item_type*, **void** ***item*);

DESCRIPTION

pam_get_item() and **pam_set_item()** allow applications and PAM service modules to access and update PAM information as needed. The information is specified by *item_type,* and can be one of the following:

PAM_SERVICE	The service name
PAM_USER	The user name
PAM_AUTHTOK	The user authentication token
PAM_OLDAUTHTOK	The old user authentication token
PAM_TTY	The tty name
PAM_RHOST	The remote host name
PAM_RUSER	The remote user name
PAM_CONV	The pam_conv structure

The *item_type* PAM_AUTHTOK and PAM_OLDAUTHTOK are available only to the module providers for security reasons. The authentication module, account module, and session management module should treat PAM_AUTHTOK as the current authentication token, and should ignore PAM_OLDAUTHTOK. The password management module should treat PAM_OLDAUTHTOK as the current authentication token and PAM_AUTHTOK as the new authentication token.

pam_set_item() is passed the authentication handle, *pamh,* returned by **pam_start()**, a pointer to the object, *item,* and its type, *item_type.* If successful, **pam_set_item()** copies the item to an internal storage area allocated by the authentication module and returns PAM_SUCCESS. An item that had been previously set will be overwritten by the new value.

pam_get_item() is passed the authentication handle, *pamh,* returned by **pam_start()**, an *item_type*, and the address of the pointer, *item*, which is assigned the address of the requested object. The object data is valid until modified by a subsequent call to **pam_set_item()** for the same *item_type*, or unless it is modified by any of the underlying service modules. If the item has not been previously set, **pam_get_item()** returns a NULL pointer. An *item* retrieved by **pam_get_item()** should not be modified or freed. The item will be released by **pam_end()**.

RETURN VALUES

Upon success **pam_get_item()** returns PAM_SUCCESS; otherwise it returns an error code. Refer to **pam**(3) for information on error related return values.

SEE ALSO

pam_start(3), **pam_authenticate**(3), **pam_acct_mgmt**(3), **pam_open_session**(3), **pam_setcred**(3), **pam_chauthtok**(3), **pam**(3)

NAME

pam_authenticate − perform authentication within the PAM framework

SYNOPSIS

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

#include <security/pam_appl.h>

int pam_authenticate(pam_handle_t **pamh*, **int** *flags***);**

DESCRIPTION

pam_authenticate() is called to authenticate the current user. The user is usually required to enter a password or similar authentication token depending upon the authentication service configured within the system. The user in question should have been specified by a prior call to **pam_start()** or **pam_set_item()**. The following flags may be set in the *flags* field:

PAM_SILENT
: 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

NOTES

In the case of authentication failures due to an incorrect username or password, it is the responsibility of the application to retry **pam_authenticate()** and to maintain the retry count. An authentication service module may implement an internal retry count and return an error PAM_MAXTRIES if the module does not want the application to retry.

If the PAM framework can not load the authentication module, then it will return PAM_ABORT. This indicates a serious failure and that the application should not attempt to retry the authentication.

For security reasons, the location of authentication failures is hidden from the user. Thus, if several authentication services are stacked and a single service fails, **pam_authenticate()** requires that the user re-authenticate to all the services.

A null authentication token in the authentication database will result in successful authentication unless PAM_DISALLOW_NULL_AUTHTOK was specified. In such cases, there will not be any prompting for the user to enter an authentication token.

RETURN VALUES

Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in **pam**(3), the following values may be returned:

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 the underlying

	authentication module
PAM_MAXTRIES	An authentication service has maintained a retry count which has been reached. No further retries should be attempted.

SEE ALSO

pam(3), **pam_start**(3), **pam_open_session**(3), **pam_setcred**(3)

NAME

pam_setcred − modify/delete user credentials for an authentication service

SYNOPSIS

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

#include <security/pam_appl.h>

int pam_setcred(pam_handle_t **pamh*, **int** *flags***);**

DESCRIPTION

pam_setcred() is used to establish, modify, or delete user credentials. **pam_setcred()** is typically called after the user has been authenticated and after a session has been opened (refer to **pam_authenticate**(3), **pam_acct_mgmt**(3), and **pam_open_session**(3)).

The user is specified by a prior call to **pam_start()** or **pam_set_item(),** and is referenced by 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 an authentication service
PAM_CRED_DELETE	Delete user credentials associated with an authentication service
PAM_CRED_REINITIALIZE	Reinitialize user credentials
PAM_CRED_REFRESH	Extend lifetime of user credentials
PAM_SILENT	Authentication service should not generate any messages

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

RETURN VALUES

Upon success, **pam_setcred()** returns PAM_SUCCESS. In addition to the error return values described in **pam**(3), the following values may be returned upon error:

PAM_CRED_UNAVAIL	Underlying authentication service can not retrieve user credentials unavailable
PAM_CRED_EXPIRED	User credentials expired
PAM_USER_UNKNOWN	User unknown to underlying authentication service
PAM_CRED_ERR	Failure setting user credentials

SEE ALSO

pam(3), **pam_start**(3), **pam_authenticate**(3), **pam_acct_mgmt**(3), **pam_open_session**(3)

NAME

pam_acct_mgmt − perform PAM account validation procedures

SYNOPSIS

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

#include <security/pam_appl.h>

int pam_acct_mgmt(pam_handle_t **pamh*, **int** *flags***);**

DESCRIPTION

The function **pam_acct_mgmt()** is called to determine if the current user's account is valid. This includes checking for password and account expiration, as well as verifying access hour restrictions. This function is typically called after the user has been authenticated with **pam_authenticate**(3).

The *pamh* argument is an authentication handle obtained by a prior call to **pam_start()**. 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

RETURN VALUES

Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in **pam**(3), the following values may be returned:

PAM_USER_UNKNOWN
: User not known to underlying account management module

PAM_AUTH_ERR
: Authentication failure

PAM_AUTHTOKEN_REQD
: New authentication token required. This is normally returned if the machine security policies require that the password should be changed because the password is NULL or it has aged.

PAM_ACCT_EXPIRED
: User account has expired

SEE ALSO

pam(3), **pam_start**(3), **pam_authenticate**(3)

NAME

pam_chauthtok − perform passord related functions within the PAM framework

SYNOPSIS

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

#include <security/pam_appl.h>

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

DESCRIPTION

pam_chauthtok() is called to change 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, all password services should update their passwords.

Upon successful completion of the call, the authentication token of the user will be changed in accordance with the password service configured in the system through **pam.conf**(4).

NOTES

The flag PAM_CHANGE_EXPIRED_AUTHTOK is typically used by a **login** application which has determined that the user's password has aged or expired. Before allowing the user to login, the **login** application may invoke **pam_chauthtok()** with this flag to allow the user to update the password. Typically applications such as **passwd**(1) should not use this flag.

pam_chauthtok() performs a preliminary check before attempting to update passwords. This check is performed for each password module in the stack as listed in **pam.conf**(4). The check may include pinging remote name services to determine if they are available. If **pam_chauthtok()** returns PAM_TRY_AGAIN, then the check has failed, and passwords are not updated.

RETURN VALUES

Upon successful completion, PAM_SUCCESS is returned. In addition to the error return values described in **pam**(3), the following values may be returned:

PAM_PERM_DENIED	No permission
PAM_AUTHTOK_ERR	Authentication token manipulation error
PAM_AUTHTOK_RECOVERY_ERR	Authentication information 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_start**(3), **pam_authenticate**(3)

NAME

pam_open_session, pam_close_session − perform PAM session creation and termination operations

SYNOPSIS

cc [*flag* ...] *file* ... −**lpam** [*library* ...]
#include <security/pam_appl.h>

int pam_open_session(pam_handle_t **pamh*, **int** *flags***);**

int pam_close_session(pam_handle_t **pamh*, **int** *flags***);**

DESCRIPTION

pam_open_session() is called after a user has been successfully authenticated (refer to **pam_authenticate**(3) and **pam_acct_mgmt**(3)) and is used to notify the session modules that a new session has been initiated. All programs that use the **pam**(3) library should invoke **pam_open_session()** when beginning a new session. Upon termination of this activity, **pam_close_session()** should be invoked to inform **pam**(3) that the session has terminated.

The *pamh* argument is an authentication handle obtained by a prior call to **pam_start().** The following flag may be set in the *flags* field for **pam_open_session()** and **pam_close_session()**:

PAM_SILENT   The session service should not generate any messages

NOTES

In many instances, the **pam_open_session()** and **pam_close_session()** calls may be made by different processes. For example, in UNIX the **login** process opens a session, while the **init** process closes the session. In this case, UTMP/WTMP entries may be used to link the call to **pam_close_session()** with an earlier call to **pam_open_session()**. This is possible because UTMP/WTMP entries are uniquely identified by a combination of attributes, including the user login name and device name, which are accessible through the PAM handle, *pamh.* The call to **pam_open_session()** should precede UTMP/WTMP entry management and the call to **pam_close_session()** should follow UTMP/WTMP exit management.

RETURN VALUES

Upon successful completion, PAM_SUCCESS is returned. In addition to the return values defined in **pam**(3), the following value may be returned on error:

PAM_SESSION_ERR   Can not make/remove an entry for the specified session

SEE ALSO

pam(3), **pam_start**(3), **pam_authenticate**(3), **pam_acct_mgmt**(3), **getutxent**(3C)