NAME

PAM − Pluggable Authentication Module

SYNOPSIS

#include <security/pam_appl.h>

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

DESCRIPTION

PAM

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

PAM API to provide a particular type of user authentication.

Interface Overview

The **PAM**
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

tion. An authentication transaction begins with a call to **pam_start(). pam_start()** allocates space, per-

quent 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. The calls to **pam_start(), pam_authenticate()** and **pam_setcred()** must be made with the same authentication handle returned by **pam_start().** This is necessary because during the call to **pam_authenticate(),** service modules may store module-specific 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 service s. 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 libraries, **/usr/lib/libpam.so.1** and **/usr/lib/libpam.a,** implement the framework inter-

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_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

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_REQUIRED 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_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

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_MAXRETRIES 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.

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_MAXRETRIES	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_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.

NOTES

The flag PAM_CHANGE_EXPIRED_AUTHTOK is typically used by a **login** application which has **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.

dance with the password service configured in the system through **pam.conf(4).**

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	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_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)

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, DCE, 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 new 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_NEWAUTHTOK** are available only to the module providers for security reasons.

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

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_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 − PAM Service Module APIs

SYNOPSIS

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

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

DESCRIPTION

PAM

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

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 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 of functions to change authentication tokens **(pam_sm_chauthtok(3))** and to manipulate authentication token attributes **(pam_sm_set_authtokattr(3))** and **(pam_sm_get_authtokattr(3)).** These back-end functions implement the functionality of **pam_chauthtok(3), pam_set_authtokattr(3),** and **pam_get_authtokattr(3),** respectively.

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.

Stateful Interface

tion. 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_DCE_AUTH_userid can be used as a name by the DCE module to store information about the state of user's authentication. Some modules use this technique to share data across two different module types.

cleanup function, when the application calls **pam_end()** to close the transaction.

Interaction with the User

tion 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 **pam_start(3)** for more information.

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.

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

One should also specify whether this module should come before or after some other module in the stack.

Though not required, it is recommended that the modules 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, DCE, 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_REQUIRED if the user has a null authentication token

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

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

pam_start( ),
and is referenced by the authentication handle, *pamh.*

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 service options passed in from the configuration file **pam.conf(4).** *argv* specifies the service options, which are interpreted and processed by the authentica-
options. If any
unknown option is passed in, the module should log the error and ignore the option.

NOTES

(success or failure) using the **pam_set_data(3)** function.

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

options.

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 rem ote name services to see if they are ready for updates, for example). If a pass-

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

application.

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

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 scheme options passed in from the configuration file **pam.conf(4).** *argv* specifies the scheme 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_set_cred()** 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 scheme options passed in from the configuration file **pam.conf(4).** *argv* specifies the scheme options, which are interpreted and processed by the authentica-

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.

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)

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 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

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. The memory used by *pam_message* should be allocated and freed by the PAM authentication service. 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* should be allocated by the application and freed by the PAM authentication service. The *pam_message* structure has the following entries:

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

The *pam_response* structure has the following entries:

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

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

appdata_ptr is an application data pointer which is passed by the application to the PAM service

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()*

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>

char ∗**pam_strerror(int** *errnum***);**

DESCRIPTION

pam_strerror( ) maps the 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.

ERRORS

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

SEE ALSO

pam(3)

NAME

pam.conf − configuration file for pluggable authentication modules

SYNOPSIS

/etc/pam.conf

DESCRIPTION

pam.conf is the configuration file for the Pluggable Authentication Module architecture, or **PAM.** A **PAM** module provides functionality for one or more of four possible services: *authentication, account management, session management,* and *password management.* An authentication service module pro-

provides functionality to determine if the current user's account is valid. This includes checking for

modules can be implemented as a shared library object which can be referenced in the **pam.conf** configuration file.

Simplified PAM.CONF configuration file

The **pam.conf** file contains a listing of services. Each service is paired with a corresponding service module. When a service is requested, its associated module is invoked. Each entry has the following format:

<*service_name*> <*module_type*> <*control_flag*> <*module_path*> <*options*>

Below is an example of the **pam.conf** configuration file with support for authentication, account, and session management modules.

```
login    auth      required   /usr/lib/security/pam_unix.so.1   debug
login    session   required   /usr/lib/security/pam_unix.so.1
login    account   required   /usr/lib/security/pam_unix.so.1
ftp      auth      required   /usr/lib/security/pam_dce.so.1    debug
ftp      session   required   /usr/lib/security/pam_unix.so.1
telnet   session   required   /usr/lib/security/pam_unix.so.1
OTHER    auth      required   /usr/lib/security/pam_unix.so.1
OTHER    passwd    required   /usr/lib/security/pam_unix.so.1
```

The *service_name* denotes the service (for example, **login, passwd,** or **rlogin).** The keyword, *OTHER,* indicates the module all other applications which have not been specified should use. The *OTHER* keyword can also be used if all services of the same *module_type* have the same requirements. In the example above, since all of the services use the same session module, they could have been replace by a single *OTHER* line.

module_type denotes the service module type: authentication (*auth*), account management (*account*), session management (*session*), or password management (*password*).

The *control_flag* field determines the behavior of stacking, and will be discussed in more detail below.

The *module_path* field specifies the pathname to a shared library object which implements the service functionality.

The *options* field is used by the **PAM** framework layer to pass module specific options to the modules. It is up to the module to parse and interpret the options. This field can be used by the modules to turn on debugging or to pass any module specific parameters such as a TIMEOUT value. It can also be used manual pages. For example, **pam_unix(5)** lists the options accepted by the UNIX module.

Integrating Multiple Authentication Services With Stacking

When a service_name of the same *module_type* is defined more than once, the service is said to be *stacked.* Each module referenced in the *module_path* for that service is then processed in the order that it occurs in the configuration file. The *control_flag* field specifies the continuation and failure semantics of the modules, and may be *required, optional,* or *sufficient.*

The PAM framework processes each service module in the stack. If all *required* modules in the stack succeed, then success is returned *(optional* and *sufficient* error values are ignored). If one or more *required* modules fail, then the error value from the first *required* module that failed is returned.

If none of the service modules in the stack are designated as *required*, then the PAM framework requires that at least one *optional* or *sufficient* module succeed. If all fail then the error value from the first service module in the stack is returned.

The only exception to the above is caused by the *sufficient* flag. If a service module that is designated as *sufficient* succeeds, then the **PAM** framework immediately returns success to the application (all subsequent services modules, even *required* ones, in the stack are ignored), given that all prior *required* modules had also succeeded. If a prior *required* module failed, then the error value from that module is returned.

All service modules can be *required, optional,* or *sufficient* except the password modules. All password modules must be designated as *required.*

If a module does not exist or can not be opened, then the **pam.conf** entry is ignored and an error will be logged through **syslog(3)** at the LOG_CRIT level.

Below is a sample configuration file that stacks the **login** and **rlogin** services.

```
login     auth      required      /usr/lib/security/pam_unix.so.1      debug
login     auth      optional      /usr/lib/security/pam_dce.so.1
rlogin    auth      sufficient    /usr/lib/security/pam_rhosts_auth.so.1
rlogin    auth      required      /usr/lib/security/pam_unix.so.1
```

In the case of **login,** the user is authenticated by the UNIX and DCE authentication modules. The *required* keyword for *control_flag* requires that the user be allowed to login only if the user is authenticated by the UNIX service module. DCE authentication is optional by virtue of the *optional* keyword in the *control_flag* field. The user can still log in even if DCE authentication fails.

In the case of **rlogin,** the *sufficient* keyword for *control_flag* specifies that if the *rhosts* authentication check succeeds, then PAM should return success to **rlogin** and **rlogin** should not prompt the user for a password. The UNIX authentication module (the next module in the stack) will only be invoked if the *rhosts* check fails. This gives the system administrator the flexibility to determine if *rhosts* alone is sufficient enough to authenticate a remote user.

NOTES

If an error is found in an entry due to invalid *service_name, module_type,* or *control_flag,* then the entry is ignored. If there are no valid entries for the given *module_type,* the **PAM** framework returns an error to the application.

EXAMPLES

The following is a sample pam.conf configuration file. Lines that begin with the # symbol are treated as comments, and therefore ignored.

```
#
# PAM configuration
#
# Authentication management for login service is stacked.
# Both UNIX and DCE authentication functions are invoked.
login     auth      required      /usr/lib/security/pam_unix.so.1
login     auth      required      /usr/lib/security/pam_dce.so.1     try_first_pass
#
# Authentication management for rlogin service is stacked.
# If the rhost check succeeds, do not continue
rlogin    auth      sufficient    /usr/lib/security/pam_rhosts_auth.so.1
rlogin    auth      required      /usr/lib/security/pam_unix.so.1
#
# Other services use UNIX authentication
OTHER     auth      required      /usr/lib/security/pam_unix.so.1
#
# Account management for login service is stacked.
# UNIX account management is required; DCE account management is optional
login     account   required      /usr/lib/security/pam_unix.so.1
login     account   optional      /usr/lib/security/pam_dce.so.1
OTHER     account   required      /usr/lib/security/pam_unix.so.1
#
# Session management
OTHER     session   required      /usr/lib/security/pam_unix.so.1
#
# Password management
OTHER     password  required      /usr/lib/security/pam_unix.so.1
```

Utilities and files

A list of utilities that are known to use **PAM** include: **login, passwd, su, dtlogin, rlogind, rshd, telnetd, ftpd, rpc.rexecd, uucpd, init, sac,** and **ttymon.**

modules. The convention, however, is the following:

/usr/lib/security/pam_<service_name>_<module_name>.so.x
implements various function of specific authentication services.

/etc/pam.conf
configuration file

/usr/lib/libpam.so
implements the PAM framework library

SEE ALSO

dtlogin(1), **init**(1) **in.ftpd**(1M), **in.rexecd**(1M), **in.rshd**(1M), **in.rlogind**(1M), **in.telnetd**(1M), **in.uucpd**(1), **login**(1), **passwd**(1), **sac**(1M), **su**(1M), **ttymon**(1M), **pam**(3)

NAME

pam_dce − authentication, account, and password management PAM functions for DCE

SYNOPSIS

/usr/lib/security/pam_dce.so

DESCRIPTION

The DCE PAM module allows for integration of DCE into the system entry services (such as **login)** through the **pam.conf(4)** file. The DCE service module for PAM consists of three components: *authentication, account management,* and *password management.* All three components are supported through the same loadable library, */usr/lib/security/pam_dce.so.*

DCE Authentication Module

The DCE authentication component provides functions to verify the identity of a user **(pam_sm_authenticate(3)),** and to initialize DCE credentials **(pam_sm_setcred(3)).** The DCE implementation of the **pam_sm_authenticate(3)** function compares the user entered password with the password from the DCE Security Service (i.e., the DCE Registry). If the passwords match, the user is

cation module:

debug	Syslog debugging information at the LOG_DEBUG level.
nowarn	Turns off warning messages about not being able to acquire DCE credentials.
use_first_pass	Use the user's initial password (entered when the user is authenticated to the first authentication module in the stack) to authenticate with DCE. If the passwords do not match, or if this is the first authentication module in the stack, quit and do not prompt the user for a password. It is recommended that this option only be used if the DCE authentication module is designated as *optional* in the **pam.conf** configuration file.
try_first_pass	Use the user's initial password (entered when the user is authenticated to the first authentication module in the PAM stack) to authenticate with DCE. If the passwords do not match, or if this is the first authentication module in the stack, prompt the user for a password.
use_mapped_pass	This option will attempt to retrieve the "onc_dce_passwd" attribute from the user's XFN context and use the user's initial password to decrypt the DCE password stored in this attribute. If the DCE password stored in this attribute is incorrect, then quit and do not prompt the user for a password. It is recommended that this option only be used if the DCE authentication module is designated as *optional.*
try_mapped_pass	Similar to **use_mapped_pass** with two exceptions. If the DCE password is incorrect, prompt for it. Once a valid DCE password is entered, encrypt it using the initial (first) password and store the encrypted value in the "onc_dce_passwd" attribute in the user's XFN context.

Note that both **use_mapped_pass** and **try_mapped_pass** require that the UNIX authentication module (see **pam_unix(5))** occurs before the DCE authentication module. This to ensure that the user's secure RPC credentials (for XFN/NIS+) have already been acquired.

If the user's DCE account has expired then an error will be returned from **pam_sm_authenticate(3).**

In order to authenticate the user to the DCE security service **pam_sm_authenticate(3)** calls **sec_login_setup_identity(3)** followed by **sec_login_valid_and_cert_ident(3).**

pam_sm_setcred(3) calls the function **sec_login_set_context(3)** to set the DCE credentials after they have been validated by **pam_sm_authenticate.**

DCE Account Management Module

The DCE account management component provides a function to perform account management **(pam_sm_acct_mgmt(3)).** The DCE implementation of the **pam_sm_acct_mgmt(3)** function retrieves the user's account and password expiration information from the DCE Security Server and verifies that the user's account and password have not expired. The following options may be passed to the DCE account module:

allow_expired_passwd	Allows a user with an expired password to login.
debug	Syslog debugging information at the LOG_DEBUG level.
nowarn	Turns off the warning messages displayed when a user's account and/or password are going to expire.

pam_sm_acct_mgmt(3) calls the function **sec_login_inquire_net_info(3)** to retrieve information about when a user's account and/or password is going to expire.

DCE Password Management Module

The DCE password management component provides a function to change passwords **(pam_sm_chauthtok(3)).** The following option may be passwd to the DCE account module:

debug	Syslog debugging information at the LOG_DEBUG level.

SEE ALSO

pam(3), **sec_login_setup_identity**(3), **sec_login_valid_and_cert_ident**(3), **sec_login_set_context**(3), **sec_login_inquire_net_info**(3), **pam.conf**(4), **pam_unix**(5)

NAME

pam_dial_auth − authentication management PAM module for dialups

SYNOPSIS

/usr/lib/security/pam_dial_auth.so

DESCRIPTION

The dialup PAM module, */usr/lib/security/pam_dial_auth.so*, authenticates a user according to the /etc/dialups and /etc/d_passwd files. Only **pam_sm_authenticate()** is implemented within this module. **pam_sm_setcred()** is a null function. */usr/lib/security/pam_dial_auth.so* is designed to be stacked immediately below the */usr/lib/security/pam_unix.so* module for the login service.

pam_sm_authenticate() performs authentication only if both the /etc/dialups and /etc/d_passwd files

user's shell is compared against entries in the /etc/d_passwd file. If there is a matching entry, the user is prompted for a password which is validated against the entry in the /etc/d_passwd file. If the passwords match, the user is authenticated. The following option may be passed in to this service module:

debug	**syslog(3)** debugging information at LOG_DEBUG level

SEE ALSO

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

NAME

pam_rhosts_auth − authentication management PAM modules using ruserok()

SYNOPSIS

/usr/lib/security/pam_rhosts_auth.so

DESCRIPTION

The rhosts PAM module, */usr/lib/security/pam_rhosts_auth.so*, authenticates a user via the **rlogin** authentication protocol. Only **pam_sm_authenticate()** is implemented within this module. **pam_sm_authenticate()** uses the ruserok(3) library funtion to authenticate the **rlogin** or **rsh** user. **pam_sm_setcred()** is a null function.

/usr/lib/security/pam_rhosts_auth.so is designed to be stacked on top of the */usr/lib/security/pam_unix.so* module for both the **rlogin** and **rsh** services. This module is normally configured as *sufficient* so that subsequent authentication is performed only on failure of **pam_sm_authenticate().** The following option may be passed in to this service module:

debug	**syslog(3)** debugging information at LOG_DEBUG level

SEE ALSO

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

NAME

pam_test − authentication management functions for testing PAM

SYNOPSIS

/usr/lib/security/pam_test.so

DESCRIPTION

The TEST service module for PAM is divided into four components: The *authentication, account management, password management,* and *session management.* The test is a shared object that is dynamically loaded to provide the necessary functionality.

TEST Authentication Component

The TEST authentication module, typically */usr/lib/security/pam_test.so,* provides functions to test the PAM framework functionality using the **pam_sm_authenticate(3)** call. The TEST module implementation of the **pam_sm_authenticate(3)** function compares the user entered password with the password set in the **pam.conf(4)** file, or the string "test" if a default test password has not been set. The following options may be passed in to the TEST Authentication module:

debug	Syslog debugging information at the LOG_DEBUG level.
passwd=foobar	Sets the password to be "foobar".
first_pass_good	The first password is always good when used with the use_first_pass or try_first_pass option.
first_pass_bad	The first password is always bad when used with the use_first_pass or try_first_pass option.
always_fail	Always returns PAM_AUTH_ERR.
always_succeed	Always returns PAM_SUCCESS.
always_ignore	Always returns PAM_IGNORE.
use_first_pass	Use the user’s initial password (entered when the user is authenticated to the first authentication module in the stack) to authenticate with the TEST module. If the passwords do not match, or if this is the first authentication module in the stack, quit and do not prompt the user for a password. It is recommended that this option only be used if the TEST authentication module is designated as *optional* in the **pam.conf** configuration file.
try_first_pass	Use the user’s initial password (entered when the user is authenticated to the first authentication module in the stack) to authenticate with the TEST module. If the passwords do not match, or if this is the first authentication module in the stack, prompt the user for a password.

The TEST module **pam_sm_setcred(3)** function always returns PAM_SUCCESS.

TEST Account Management Component

The TEST Account Management Component **(pam_sm_acct_mgmt(3))** always returns PAM_SUCCESS.

TEST Password Management Component

The TEST Password Management Component function **(pam_sm_chauthtok(3)),** always returns PAM_SUCCESS.

TEST Session Management Component

The TEST Session Management Component functions **(pam_sm_open_session(3), pam_sm_close_session(3))** always return PAM_SUCCESS.

SEE ALSO

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

NAME

pam_unix − authentication, account, session, and password management PAM modules for UNIX

SYNOPSIS

/usr/lib/security/pam_unix.so

DESCRIPTION

The UNIX service module for PAM, */usr/lib/security/pam_unix.so,* provides functionality for all four PAM modules: The *authentication* module, the *account management* module, the *session management* module, and the *password management* module. The pam_unix.so module is a shared object that can be dynamically loaded to provide the necessary functionality upon demand. Its path is specified in the **PAM** configuration file.

Unix Authentication Module

The UNIX authentication component provides functions to verify the identity of a user, **(pam_sm_authenticate())** and to set user specific credentials **(pam_sm_setcred()). pam_sm_authenticate()** compares the user entered password with the password from UNIX password database. If the passwords match, the user is authenticated. The following options may be passed to the UNIX service module:

debug	**syslog(3)** debugging information at LOG_DEBUG level
nowarn	turn off warning messages
use_first_pass	It compares the password in the password database with the user's initial password (entered when the user authenticated to the first authentication module in the stack). If the passwords do not match, or if no password has been entered, quit and do not prompt the user for a password. This option should only be used if the authentication service is designated as *optional* in the **pam.conf** configuration file.
try_first_pass	It compares the password in the password database with the user's initial password (entered when the user authenticated to the first authentication module in the stack). If the passwords do not match, or if no password has been entered, prompt the user for a password.

The **pam_sm_setcred()** function sets user specific credentials. For UNIX, this is a NULL function.

Unix Account Management Module

The UNIX account management component provides a function to perform account management **(pam_sm_acct_mgmt()).** The **pam_sm_acct_mgmt()** function retrieves the user's password entry from

following option may be passed in to the UNIX service module:

debug	**syslog(3)** debugging information at LOG_DEBUG level
nowarn	turn off warning messages

Unix Session Management Module

The UNIX session management component provides functions to initiate **(pam_sm_open_session())** and terminate **(pam_sm_close_session())** UNIX sessions. Currently for UNIX, these functions are empty. The following option may be passed in to the UNIX service module:

debug	**syslog(3)** debugging information at LOG_DEBUG level
nowarn	turn off warning messages

Unix Password Management Module

The UNIX password management component provides a function to change passwords **(pam_sm_chauthtok())** in the UNIX password database.

This module must be *required* in **pam.conf.** It can not be *optional* or *sufficient.*

The following option may be passed in to the UNIX service module:

debug	**syslog(3)** debugging information at LOG_DEBUG level
nowarn	turn off warning messages
use_first_pass	It compares the password in the password database with the user's old password (entered to the first password module in the stack). If the passwords do not match, or if no password has been entered, quit and do not prompt the user for the old password. It also attempts to use the new password (entered to the first password module in the stack) as the new password for this module. If the new password fails, quit and do not prompt the user for a new password.
try_first_pass	It compares the password in the password database with the user's old password (entered to the first password module in the stack). If the passwords do not match, or if no password has been entered, prompt the user for the old password. It also attempts to use the new password (entered to the first password module in the stack) as the new password for this module. If the new password fails, prompt the user for a new password.

SEE ALSO

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