The Linux-PAM Application Developers' Guide

Andrew G. Morgan, morgan@parc.power.net DRAFT v0.54 1996/11/30

This manual documents what an application developer needs to know about the **Linux-PAM** library. It describes how an application might use the **Linux-PAM** library to authenticate users. In addition it contains a description of the funtions to be found in `libpam_misc` library, that can be used in general applications. Finally, it contains some comments on PAM related security issues for the application developer.

Contents

1 Introduction

1.1 Synopsis

For general applications that wish to use the services provided by **Linux-PAM** the following is a summary of the relevant linking information:

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

cc -o application .... -lpam
```

In addition to `libpam`, there is a library of miscellaneous functions that make the job of writing *PAM-aware* applications easier (this library is not covered in the DCE-RFC for PAM):

```
...
#include <security/pam_misc.h>
```

```
cc -o application .... -lpam -lpam_misc
```

1.2 Description

Linux-PAM (Pluggable Authentication Modules for Linux) is a library that enables the local system administrator to choose how individual applications authenticate users. For an overview of the **Linux-PAM** library see the **Linux-PAM** System Administrators' Guide.

It is the purpose of the **Linux-PAM** project to liberate the development of privilege granting software from the development of secure and appropriate authentication schemes. This is accomplished by providing a documented library of functions that an application may use for all forms of user authentication management. This library dynamically loads locally configured authentication modules that actually perform the authentication tasks.

From the perspective of an application developer the information contained in the local configuration of the PAM library should not be important. Indeed it is intended that an application treat the functions documented here as a "black box" that will deal with all aspects of user authentication. "All aspects" includes user verification, account management, session initialization/termination and also the resetting of passwords (*authentication tokens*).

2 The public interface to Linux-PAM

Firstly, the relevant include file for the **Linux-PAM** library is `<security/pam_appl.h>`. It contains the definitions for a number of functions. After listing these functions, we collect some guiding remarks for programmers.

2.1 What can be expected by the application

Here we document those functions in the **Linux-PAM** library that may be called from an application.

2.1.1 Initialization of Linux-PAM

```
extern int pam_start(const char *service_name, const char *user,
                     const struct pam_conv *pam_conversation,
                     pam_handle_t **pamh);
```

This is the first of the **Linux-PAM** functions that must be called by an application. It initializes the interface and reads the system configuration file, `/etc/pam.conf` (see the **Linux-PAM** System Administrators' Guide). Following a successful return (`PAM_SUCCESS`) the contents of `*pamh` is a handle that provides continuity for successive calls to the **Linux-PAM** library. The arguments expected by `pam_start` are as follows: the `service_name` of the program, the `user`name of the individual to be authenticated, a pointer to an application-supplied `pam_conv` structure and a pointer to a `pam_handle_t` *pointer*.

The `pam_conv` structure is discussed more fully in the section 2.2.1. The `pam_handle_t` is a *blind* structure and the application should not attempt to probe it directly for information. Instead the **Linux-PAM** library provides the functions `pam_set_item` and `pam_get_item`. These functions are documented below.

2.1.2 Termination of the library

```
extern int pam_end(pam_handle_t *pamh, int pam_status);
```

This function is the last function an application should call in the **Linux-PAM** library. Upon return the handle `pamh` is no longer valid and all memory associated with it will be invalid (likely to cause a segmentation fault if accessed).

Under normal conditions the argument `pam_status` has the value PAM_SUCCESS, but in the event of an unsuccessful service application the approprite **Linux-PAM** error-return value should be used here. attempt its purpose is to be passed as an argument to the module specific function `cleanup()` (see the **Linux-PAM** Module Developers' Guide).

2.1.3 Setting PAM items

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

This function is used to (re)set the value of one of the following **item_type**s:

`PAM_SERVICE`

The service name

`PAM_USER`

The user name

`PAM_TTY`

The terminal name

`PAM_RHOST`

The remote host name

`PAM_CONV`

The conversation structure (see section 2.2.1)

`PAM_RUSER`

The remote user name

`PAM_USER_PROMPT`

The string used when prompting for a user's name.

For all `item_types`, other than `PAM_CONV`, `item` is a pointer to a <NUL> terminated character string. In the case of `PAM_CONV`, `item` points to an initialized `pam_conv` structure (see section 2.2.1).

A successful call to this function returns `PAM_SUCCESS`. However, the application should expect one of the following errors:

`PAM_PERM_DENIED`

An attempt was made to replace the conversation structure with a `NULL` value.

`PAM_BUF_ERR`

The function ran out of memory making a copy of the item.

`PAM_BAD_ITEM`

The application attempted to set an undefined item.

2.1.4 Getting PAM items

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

This function is used to obtain the value of the indicated `item_type`. Upon successful return, `*item` contains a pointer to the value of the corresponding item. Note, this is a pointer to the *actual* data and should *not* be `free()`'ed or over-written! A successful call is signaled by a return value of `PAM_SUCCESS`. If an attempt is made to get an undefined item, `PAM_BAD_ITEM` is returned.

2.1.5 Understanding errors

```
extern const char *pam_strerror(int errnum);
```

This function returns some text describing the **Linux-PAM** error associated with the argument `errnum`. If the error is not recognized "`Unknown Linux-PAM error`" is returned.

2.1.6 Authenticating the user

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

This function serves as an interface to the authentication mechanisms of the loaded modules. The single *optional* flag, which may be logically OR'd with `PAM_SILENT`, takes the following value,

`PAM_DISALLOW_NULL_AUTHTOK`

Instruct the authentication modules to return `PAM_AUTH_ERR` if the user does not have a registered authorization token—it is set to `NULL` in the system database.

The value returned by this function is one of the following:

PAM_AUTH_ERR

The user was not authenticated

PAM_CRED_INSUFFICIENT

For some reason the application does not have sufficient credentials to authenticate the user.

PAM_AUTHINFO_UNAVAIL

The modules were not able to access the authentication information. This might be due to a network or hardware failure etc.

PAM_USER_UNKNOWN

The supplied username is not known to the authentication service

PAM_MAXTRIES

One or more of the authentication modules has reached its limit of tries authenticating the user. Do not try again.

If one or more of the authentication modules fails to load, for whatever reason, this function will return PAM_ABORT.

2.1.7 Setting user credentials

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

This function is used to set the module-specific credentials of the user. It is usually called after the user has been authenticated. This function simply calls the pam_sm_setcred functions of each of the loaded modules. Valid flags, any one of which, may be logically OR'd with PAM_SILENT, are:

PAM_CRED_ESTABLISH

Set the credentials for the authentication service,

PAM_CRED_DELETE

Delete the credentials associated with the authentication service,

PAM_CRED_REINITIALIZE

Reinitialize the user credentials, and

PAM_CRED_REFRESH

Extend the lifetime of the user credentials.

A successful return is signalled with PAM_SUCCESS. Errors that are especially relevant to this function are the following:

PAM_CRED_UNAVAIL

A module cannot retrieve the user's credentials.

`PAM_CRED_EXPIRED`

The user's credentials have expired.

`PAM_USER_UNKNOWN`

The user is not known to an authentication module.

`PAM_CRED_ERR`

A module was unable to set the credentials of the user.

2.1.8 Account management

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

This function is typically called after the user has been authenticated. It establishes whether the user's account is healthy. That is to say, whether the user's account is still active and whether the user is permitted to gain access to the system at this time. Valid flags, any one of which, may be logically OR'd with `PAM_SILENT`, and are the same as those applicable to the `flags` argument of `pam_authenticate`.

This function simply calls the corresponding functions of each of the loaded modules, as instructed by the configuration file, `/etc/pam.conf`.

The normal response from this function is `PAM_SUCCESS`, however, specific failures are indicated by the following error returns:

`PAM_AUTHTOKEN_REQD`

The user **is** valid but their authentication token has *expired*. The correct response to this return-value is to require that the user satisfies the `pam_chauthtok()` function before obtaining service. It may not be possible for some applications to do this. In such cases, the user should be denied access until such time as they can update their password.

`PAM_ACCT_EXPIRED`

The user is no longer permitted access to the system.

`PAM_AUTH_ERR`

There was an authentication error.

`PAM_PERM_DENIED`

The user is not permitted to gain access at this time.

`PAM_USER_UNKNOWN`

The user is not known to a module's account management component.

2.1.9 Updating authentication tokens

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

This function is used to change the authentication token for a given user (as indicated by the state associated with the handle, `pamh`). The following is a valid but optional flag which may be logically OR'd with `PAM_SILENT`,

`PAM_CHANGE_EXPIRED_AUTHTOK`

This argument indicates to the modules that the users authentication token (password) should only be changed if it has expired.

Note, if this argument is not passed, the application requires that *all* authentication tokens are to be changed. `PAM_SUCCESS` is the only successful return value, valid error-returns are:

`PAM_AUTHTOK_ERR`

A module was unable to obtain the new authentication token.

`PAM_AUTHTOK_RECOVERY_ERR`

A module was unable to obtain the old authentication token.

`PAM_AUTHTOK_LOCK_BUSY`

One or more of the modules was unable to change the authentication token since it is currently locked.

`PAM_AUTHTOK_DISABLE_AGING`

Authentication token aging has been disabled for at least one of the modules.

`PAM_PERM_DENIED`

Permission denied.

`PAM_TRY_AGAIN`

Not all of the modules were in a position to update the authentication token(s). In such a case none of the user's authentication tokens are updated.

`PAM_USER_UNKNOWN`

The user is not known to the authentication token changing service.

2.1.10 Session initialization

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

This function is used to indicate that an authenticated session has begun. It is used to inform the module that the user is currently in a session. It should be possible for the **Linux-PAM** library to open a session and close the same session (see section 2.1.11) from different applications.

Currently, this function simply calls each of the corresponding functions of the loaded modules. The only valid flag is `PAM_SILENT` and this is, of course, *optional*.

If any of the *required* loaded modules are unable to open a session for the user, this function will return `PAM_SESSION_ERR`.

2.1.11 Terminating sessions

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

This function is used to indicate that an authenticated session has ended. It is used to inform the module that the user is exiting a session. It should be possible for the **Linux-PAM** library to open a session and close the same session from different applications.

Currently, this function simply calls each of the corresponding functions of the loaded modules. The only valid flag is `PAM_SILENT` and this is, of course, *optional*.

If any of the *required* loaded modules are unable to close a session for the user, this function will return `PAM_SESSION_ERR`.

2.1.12 Setting PAM environment variables

```
extern int pam_putenv(pam_handle_t *pamh, const char *name_value);
```

Warning, the environment support in **Linux-PAM** *is based solely on a six line email from the developers at Sun. Its interface is likely to be generally correct, however, the details are likely to be changed as more information becomes available.*

This function attempts to (re)set a **Linux-PAM** environment variable. The `name_value` argument is a single `NUL` terminated string of one of the following forms:

"`NAME=value of variable`"

In this case the environment variable of the given `NAME` is set to the indicated value: "`value of variable`". If this variable is already known, it is overwritten. Otherwise it is added to the **Linux-PAM** environment.

"`NAME=`"

This function sets the variable to an empty value. It is listed separately to indicate that this is the correct way to achieve such a setting.

"`NAME`"

Without an '`=`' the `pam_putenv()` function will delete the correspoding variable from the **Linux-PAM** environment.

Success is indicated with a return value of `PAM_SUCCESS`. Failure is indicated by one of the following returns:

`PAM_PERM_DENIED`

name given is a `NULL` pointer

`PAM_BAD_ITEM`

variable requested (for deletion) is not currently set

`PAM_ABORT`

the **Linux-PAM** handle, `pamh`, is corrupt

`PAM_BUF_ERR`

failed to allocate memory when attempting update

2.1.13 Getting a PAM environment variable

```
extern const char *pam_getenv(pam_handle_t *pamh, const char *name);
```

Warning, the environment support in **Linux-PAM** *is based solely on a six-line email from the developers at Sun. Its interface is likely to be generally correct, however, the details are likely to be changed as more information becomes available.*

Obtain the value of the indicated **Linux-PAM** environment variable. On error, internal failure or the unavailability of the given variable (unspecified), this function simply returns `NULL`.

2.1.14 Getting the PAM environment

```
extern const char * const *pam_getenvlist(pam_handle_t *pamh);
```

Warning, the environment support in **Linux-PAM** *is based solely on a six line email from the developers at Sun. Its interface is likely to be generally correct, however, the details are likely to be changed as more information becomes available.*

This function returns a pointer to the complete `Linux-PAM` environment. It is a pointer to a *read-only* list of *read-only* environment variables. It should be noted that this memory will become invalid after a call to `pam_end()` (see the section 2.1.2). If application wishes to make use of this list after such a call, it should first make a copy of all the set variables. (A function that performs such a transcription is to be found in `libpam_misc`.)

2.2 What is expected of an application

2.2.1 The conversation function

An application must provide a “conversation function”. It is used for direct communication between a loaded module and the application and will typically provide a means for the module to prompt the user for a password etc. . The structure, `pam_conv`, is defined by including `<security/pam_appl.h>`; to be,

```
struct pam_conv {
    int (*conv)(int num_msg,
        const struct pam_message **msg,
        struct pam_response **resp,
        void *appdata_ptr);
    void *appdata_ptr;
};
```

It is initialized by the application before it is passed to the library. The *contents* of this structure are attached to the `*pamh` handle. The point of this argument is to provide a mechanism for any loaded module to interact directly with the application program. This is why it is called a *conversation* structure.

When a module calls the referenced `conv()` function, the argument `*appdata_ptr` is set to the second element of this structure.

The other arguments of a call to `conv()` concern the information exchanged by module and application. That is to say, `num_msg` holds the length of the array of pointers, `msg`. After a successful return, the pointer `*resp` points to an array of `pam_response` structures, holding the application supplied text. Note, `*resp` is an `struct pam_response` array and *not* an array of pointers.

The message (from the module to the application) passing structure is defined by `<security/pam_appl.h>` as:

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

Valid choices for `msg_style` are:

`PAM_PROMPT_ECHO_OFF`

Obtain a string without echoing any text

`PAM_PROMPT_ECHO_ON`

Obtain a string whilst echoing text

`PAM_ERROR_MSG`

Display an error

`PAM_TEXT_INFO`

Display some text.

The point of having an array of messages is that it becomes possible to pass a number of things to the application in a single call from the module. It can also be convenient for the application that related things come at once: a windows based application can then present a single form with many messages/prompts on at once.

The response (from the application to the module) passing structure is defined by including `<security/pam_appl.h>` as:

```
struct pam_response {
    char *resp;
    int resp_retcode;
};
```

Currently, there are no definitions for `resp_retcode` values; the normal value is `0`. The length of the `*resp` array is known in advance by the module, since it is dependent on the number of prompts sent in the `msg` array. The memory for this array and its string elements should be *dynamically* allocated with one of the `malloc()` library of functions. *It will be* `free()` *'d by the module.*

The maximum length of the `pam_msg.msg` and `pam_response.resp` character strings is `PAM_MAX_MSG_SIZE`.

`PAM_SUCCESS` is the expected return value of this function. However, should an error occur the application should not set `*resp` but simply return `PAM_CONV_ERR`.

Note, if an application wishes to use two conversation functions, it should activate the second with a call to `pam_set_item()`.

2.3 Programming notes

Note, all of the authentication service function calls accept the token `PAM_SILENT`, which instructs the modules to not send messages to the application. This token can be logically OR'd with any one of the permitted tokens specific to the individual function calls. `PAM_SILENT` does not override the prompting of the user for passwords etc., it only stops informative messages from being generated.

3 Security issues of Linux-PAM

A poorly (or maliciously) written application can defeat any **Linux-PAM** module's authentication mechanisms by simply ignoring it's return values. It is the applications task and responsibility to grant privileges and access to services. The **Linux-PAM** library simply assumes the responsibility of *authenticating* the user; ascertaining that the user *is* who they say they are. Care should be taken to anticipate all of the documented behavior of the **Linux-PAM** library functions. A failure to do this will most certainly lead to a future security breach.

In general writers of authorization-granting applications should assume that each module is likely to call any or *all* 'libc' functions. For 'libc' functions that return pointers to static/dynamically allocated structures (ie. the library allocates the memory and the user is not expected to '`free()`' it) any module call to this function is likely to corrupt a pointer previously obtained by the application. The application programmer should either re-call such a 'libc' function after any call to the **Linux-PAM** library, or copy the structure contents to some safe area of memory before passing control to the **Linux-PAM** library.

When picking the *service name* that corresponds to the first entry in the `/etc/pam.conf` file, the application programmer should avoid the temptation of choosing, `argv[0]`. Since a user can always establish the service that uses the weakest authentication service. By symbolically linking the targeted application to a link of that name, the user assumes power to choose the authentication method for any application.

Care should be taken to ensure that the conv() function is robust. Such a function is provided in the library libpam_misc (see 4).

4 A library of miscellaneous helper functions

To aid the work of the application developer a library of miscellaneous functions is provided. It is called libpam_misc, and contains functions for allocating memory (securely), a text based conversation function, and routines for enhancing the standard PAM-environment variable support.

4.1 Requirements

The functions, structures and macros made available by this library can be defined by including <security/pam_misc.h>. It should be noted that this library is specific to **Linux-PAM** and is not referred to in the defining DCE-RFC (see 7) below.

4.2 Functions supplied

4.2.1 Safe string duplication

```
extern char *xstrdup(const char *s)
```

Return a duplicate copy of the NUL terminated string, s. NULL is returned if there is insufficient memory available for the duplicate or if s=NULL.

4.2.2 A text based conversation function

```
extern int misc_conv(int num_msg, const struct pam_message **msgm,
                     struct pam_response **response, void *appdata_ptr);
```

This is a function that will prompt the user with the appropriate comments and obtain the appropriate inputs as directed by authentication modules.

4.2.3 Transcribing an environment to that of Linux-PAM

```
extern int pam_misc_paste_env(pam_handle_t *pamh,
                              const char * const * user_env);
```

This function takes the supplied list of environment pointers and *uploads* its contents to the **Linux-PAM** environment. Success is indicated by PAM_SUCCESS.

4.2.4 Saving the Linux-PAM environment for later use

```
extern char **pam_misc_copy_env(pam_handle_t *pamh);
```

This function returns a pointer to a list of environment variables that are a direct copy of the **Linux-PAM** environment. The memory associated with these variables are the responsibility of the application and should be liberated with a call to `pam_misc_drop_env()`.

4.2.5 Liberating a locally saved environment

```
extern char **pam_misc_drop_env(char **env);
```

This function is defined to complement the `pam_misc_copy_env()` function. It liberates the memory associated with `env`, *overwriting* with `0` all memory before `free()`ing it.

4.2.6 BSD like Linux-PAM environment variable setting

```
extern int pam_misc_setenv(pam_handle_t *pamh, const char *name,
                           const char *value, int readonly);
```

This function performs a task equivalent to `pam_putenv()`, its syntax is, however, more like the BSD style function; `setenv()`. The `name` and `value` are concatenated with an "=" to form a `name_value` and passed to `pam_putenv()`. If, however, the **Linux-PAM** variable is already set, the replacement will only be applied if the last argument, `readonly`, is zero.

5 An example application

To get a flavor of the way a `Linux-PAM` application is written we include the following example. It prompts the user for their password and indicates whether their account is valid on the standard output, its return code also indicates the success (`0` for success; `1` for failure).

```
/*
  This program was contributed by Shane Watts
  [modifications by AGM]

  You need to add the following (or equivalent) to the /etc/pam.conf file.
  # check authorization
  check_user   auth       required     /usr/lib/security/pam_unix_auth.so
  check_user   account    required     /usr/lib/security/pam_unix_acct.so
 */

#include <security/pam_appl.h>
#include <security/pam_misc.h>
#include <stdio.h>
```

```
static struct pam_conv conv = {
    misc_conv,
    NULL
};

int main(int argc, char *argv[])
{
    pam_handle_t *pamh=NULL;
    int retval;
    const char *user="nobody";

    if(argc == 2) {
        user = argv[1];
    }

    if(argc > 2) {
        fprintf(stderr, "Usage: check_user [username]\n");
        exit(1);
    }

    retval = pam_start("check_user", user, \&conv, \&pamh);

    if (retval == PAM_SUCCESS)
        retval = pam_authenticate(pamh, 0);    /* is user really user? */

    if (retval == PAM_SUCCESS)
        retval = pam_acct_mgmt(pamh, 0);       /* permitted access? */

    /* This is where we have been authorized or not. */

    if (retval == PAM_SUCCESS) {
        fprintf(stdout, "Authenticated\n");
    } else {
        fprintf(stdout, "Not Authenticated\n");
    }

    if (pam_end(pamh,retval) != PAM_SUCCESS) {     /* close Linux-PAM */
        pamh = NULL;
        fprintf(stderr, "check_user: failed to release authenticator\n");
        exit(1);
    }

    return ( retval == PAM_SUCCESS ? 0:1 );       /* indicate success */
}
```

6 Files

`/usr/include/security/pam_appl.h`

header file for **Linux-PAM** applications interface

`/usr/include/security/pam_misc.h`

header file for useful library functions for making applications easier to write

`/usr/lib/libpam.so.*`

the shared library providing applications with access to **Linux-PAM**.

`/etc/pam.conf`

the **Linux-PAM** configuration file.

`/usr/lib/security/pam_*.so`

the primary location for **Linux-PAM** dynamically loadable object files; the modules.

7 See also

- The **Linux-PAM** System Administrators' Guide.
- The **Linux-PAM** Module Writers' Guide.
- The V. Samar and R. Schemers (SunSoft), "UNIFIED LOGIN WITH PLUGGABLE AUTHENTICATION MODULES", Open Software Foundation Request For Comments 86.0, October 1995.

8 Notes

I intend to put development comments here... like "at the moment this isn't actually supported". At release time what ever is in this section will be placed in the Bugs section below! :)

- `pam_strerror()` should be internationalized....
- Note, the `resp_retcode` of struct `pam_message`, has no purpose at the moment. Ideas/suggestions welcome!
- more security issues are required....

9 Author/acknowledgments

This document was written by Andrew G. Morgan (morgan@parc.power.net) with many contributions from Marc Ewing, Michael K. Johnson, Al Longyear, Marek Michalkiewicz, Theodore Ts'o, Jeff Uphoff, Joseph S. D. Yao and Alex O. Yuriev.

Thanks are also due to Sun Microsystems, especially to Vipin Samar and Charlie Lai for their advice. At an early stage in the development of **Linux-PAM**, Sun graciously made the documentation for their implementation of PAM available. This act greatly accelerated the development of **Linux-PAM**.

10 Bugs/omissions

This manual is hopelessly unfinished. Only a partial list of people is credited for all the good work they have done.