The Linux-PAM System Administrators' Guide

Andrew G. Morgan, morgan@linux.kernel.org DRAFT v0.71 1999/11/8

This manual documents what a system-administrator needs to know about the **Linux-PAM** library. It covers the correct syntax of the PAM configuration file and discusses strategies for maintaining a secure system.

Contents

1 Introduction

Linux-PAM (Pluggable Authentication Modules for Linux) is a suite of shared libraries that enable the local system administrator to choose how applications authenticate users.

In other words, without (rewriting and) recompiling a PAM-aware application, it is possible to switch between the authentication mechanism(s) it uses. Indeed, one may entirely upgrade the local authentication system without touching the applications themselves.

Historically an application that has required a given user to be authenticated, has had to be compiled to use a specific authentication mechanism. For example, in the case of traditional UN*X systems, the identity of the user is verified by the user entering a correct password. This password, after being prefixed by a two character "salt", is encrypted (with crypt(3)). The user is then authenticated if this encrypted password is identical to the second field of the user's entry in the system password database (the `/etc/passwd` file). On such systems, most if not all forms of privileges are granted based on this single authentication scheme. Privilege comes in the form of a personal user-identifier (`uid`) and membership of various groups. Services and applications are available based on the personal and group identity of the user. Traditionally, group membership has been assigned based on entries in the `/etc/group` file.

Unfortunately, increases in the speed of computers and the widespread introduction of network based computing, have made once secure authentication mechanisms, such as this, vulnerable to attack. In the light of such realities, new methods of authentication are continuously being developed.

It is the purpose of the **Linux-PAM** project to separate the development of privilege granting software from the development of secure and appropriate authentication schemes. This is accomplished by providing a library of functions that an application may use to request that a user be authenticated. This PAM library is configured locally with a system file, `/etc/pam.conf` (or a series of configuration files located in `/etc/pam.d/`) to authenticate a user request via the locally available authentication modules. The modules themselves will usually be located in the directory `/usr/lib/security` and take the form of dynamically loadable object files (see `dlopen(3)`).

2 Some comments on the text

Before proceeding to read the rest of this document, it should be noted that the text assumes that certain files are placed in certain directories. Where they have been specified, the conventions we adopt here for locating these files are those of the relevant RFC (RFC-86.0, see 8 (bibliography)). If you are using a distribution of Linux (or some other operating system) that supports PAM but chooses to distribute these files in a diferent way (Red Hat is one such distribution), you should be careful when copying examples directly from the text.

As an example of the above, where it is explicit, the text assumes that PAM loadable object files (the *modules*) are to be located in the following directory: `/usr/lib/security/`. However, Red Hat Linux, in agreement with the Linux File System Standard (the FSSTND), places these files in `/lib/security`. Please be careful to perform the necessary transcription when using the examples from the text.

3 Overview

For the uninitiated, we begin by considering an example. We take an application that grants some service to users; *login* is one such program. *Login* does two things, it first establishes that the requesting user is whom they claim to be and second provides them with the requested service: in the case of *login* the service is a command shell (*bash, tcsh, zsh, etc.*) running with the identity of the user.

Traditinally, the former step is achieved by the *login* application prompting the user for a password and then verifying that it agrees with that located on the system; hence verifying that the so far as the system is concerned the user is who they claim to be. This is the task that is delegated to **Linux-PAM**.

From the perspective of the application programmer (in this case the person that wrote the *login* application), **Linux-PAM** takes care of this authentication task – verifying the identity of the user.

The flexibility of **Linux-PAM** is that *you*, the system administrator, have the freedom to stipulate which authentication scheme is to be used. You have the freedom to set the scheme for any/all PAM-aware applications on your Linux system. That is, you can authenticate from anything as naive as *simple trust* (`pam_permit`) to something as paranoid as a combination of a retinal scan, a voice print and a one-time password!

To illustrate the flexibility you face, consider the following situation: a system administrator (parent) wishes to improve the mathematical ability of her users (children). She can configure their favorite "Shoot 'em up game" (PAM-aware of course) to authenticate them with a request for the product of a couple of random numbers less than 12. It is clear that if the game is any good they will soon learn their *multiplication tables*. As they mature, the authentication can be upgraded to include (long) division!

Linux-PAM deals with four separate types of (management) task. These are: *authentication management*; *account management*; *session management*; and *password management*. The association of the preferred management scheme with the behavior of an application is made with entries in the relevant **Linux-PAM** configuration file. The management functions are performed by *modules* specified in the configuration file. The syntax for this file is discussed in the section 4 (below).

Here is a figure that describes the overall organization of **Linux-PAM**.

```
+----------------+
| application: X |
+----------------+       /  +----------+     +================+
| authentication-[---->--\--] Linux-    |--<--| PAM config file|
|       +        [----<--/--]   PAM     |     |================|
|[conversation()][--+    \  |          |     | X auth .. a.so |
+----------------+  |    /  +-n--n-----+     | X auth .. b.so |
|                |  |       __|  |           |           _____/
|  service user  |  A      |     |           |____,-----'
|                |  |      V     A
+----------------+  +------|-----|---------+ -----+------+
                       +---u-----u----+    |      |      |
                       |   auth....   |--[ a ]--[ b ]--[ c ]
                       +--------------+
                       |   acct....   |--[ b ]--[ d ]
                       +--------------+
                       |   password   |--[ b ]--[ c ]
                       +--------------+
                       |   session    |--[ e ]--[ c ]
                       +--------------+
```

By way of explanation, the left of the figure represents the application; application X. Such an application interfaces with the **Linux-PAM** library and knows none of the specifics of its configured authentication method. The **Linux-PAM** library (in the center) consults the contents of the PAM configuration file and loads the modules that are appropriate for application-X. These modules fall into one of four management groups (lower-center) and are stacked in the order they appear in the configuaration file. These modules, when called by **Linux-PAM**, perform the various authentication tasks for the application. Textual information, required from/or offered to the user, can be exchanged through the use of the application-supplied *conversation* function.

3.1 Getting started

The following text was contributed by Seth Chaiklin:

```
To this point, we have described how PAM should work in an
ideal world, in which all applications are coded properly.
However, at the present time (October 1998), this is far
from the case.  Therefore, here are some practical considerations
in trying to use PAM in your system.

Why bother, is it really worth all the trouble?

If you running Linux as a single user system, or in an
environment where all the users are trusted, then there
is no real advantage for using PAM.
```

Ed: there is actually an advantage since you can *dummy down* the authentication to the point where you don't have any... Almost like Win95.

In a networked environment, it is clear that you need to think a little more about how users etc., are authenticated:]

```
If you are running Linux as a server, where several different
services are being provided (e.g., WWW with areas restricted by
password control, PPP), then there can be some real and interesting
value for PAM.  In particular, through the use of modules, PAM can
enable a program to search through several different password
databases, even if that program is not explicitly coded for
that particular database.  Here are some examples of the possibilities
that this enables.

   o  Apache has a module that provides PAM services.  Now
   authentication
      to use particular directories can be conducted by PAM, which
      means that the range of modules that are available to PAM can
      be used, including RADIUS, NIS, NCP (which means that Novell
      password databases can be used).

   o  pppd has a PAMified version (available from RedHat)  Now it is
      possible to use a series of databases to authenticate ppp users.
      In addition to the normal Linux-based password databases (such
      as /etc/passwd and /etc/shadow), you can use PAM modules to
      authenticate against Novell password databases or NT-based
      password databases.

   o  The preceding two examples can be combined.  Imagaine that the
      persons in your office/department are already registered with a
      username and password in a Novell or NT LAN.  If you wanted to
      use this database on your Linux server (for PPP access, for
      web access, or even for normal shell access), you can use PAM
      to authenticate against this existing database, rather than
      maintain a separate database on both Linux and the LAN server.


Can I use PAM for any program that requires authentication?

Yes and no.   Yes, if you have access to the source code, and can
add the appropriate PAM functions.  No, if you do not have access
to the source code, and the binary does not have the PAM functions
included.

In other words, if a program is going to use PAM, then it has to
have PAM functions explicitly coded into the program.  If they
are not, then it is not possible to use PAM.

How can I tell whether a program has PAM coded into it or not?

A quick-and-dirty (but not always reliable) method is to ldd
<programname>
If libpam and libpam_misc are not among the libraries that the program
```

```
uses, then it is not going to work with PAM.  However, it is possible
that the libraries are included, but there are still problems, because
the PAM coding in the program does not work as it should.  So a
more reliable method is to make the follow tests.

In the /etc/pam.d directory, one needs to make a configuration file
for the program that one wants to run.  The exact name of the
configuration
file is hard-coded into the program.  Usually, it is the same name as
the
program, but not always.  For sake of illustration, let's assume that
the program is named "pamprog" and the name of the configuration file
is /etc/pam.d/pamprog.

In the /etc/pam.d/pamprog put the following two lines:

auth    required  pam_permit.so
auth    required  pam_warn.so


Now try to use pamprog.  The first line in the configuration file
says that all users are permitted.  The second line will write a
warning to your syslog file (depending on whether your syslog is
writing messages).  If this test succeeds, then you know that you
have a program that can understand pam, and you can start the more
interesting work of deciding how to stack modules in your
/etc/pam.d/pamprog file.
```

4 The Linux-PAM configuration file

Linux-PAM is designed to provide the system administrator with a great deal of flexibility in configuring the privilege granting applications of their system. The local configuration of those aspects of system security controlled by `Linux-PAM` is contained in one of two places: either the single system file, `/etc/pam.conf`; or the `/etc/pam.d/` directory. In this section we discuss the correct syntax of and generic options respected by entries to these files.

4.1 Configuration file syntax

The reader should note that the **Linux-PAM** specific tokens in this file are case *insensitive*. The module paths, however, are case sensitive since they indicate a file's *name* and reflect the case dependence of typical Linux file-systems. The case-sensitivity of the arguments to any given module is defined for each module in turn.

In addition to the lines described below, there are two *special* characters provided for the convenience of the system administrator: comments are preceded by a '`#`' and extend to the next end-of-line; also, module specification lines may be extended with a '\' escaped newline.

A general configuration line of the `/etc/pam.conf` file has the following form:

```
service-name   module-type   control-flag   module-path   arguments
```

Below, we explain the meaning of each of these tokens. The second (and more recently adopted) way of configuring **Linux-PAM** is via the contents of the `/etc/pam.d/` directory. Once we have explained the meaning of the above tokens, we will describe this method.

`service-name`

The name of the service associated with this entry. Frequently the service name is the conventional name of the given application. For example, '`ftpd`', '`rlogind`' and '`su`', *etc.* .

There is a special `service-name`, reserved for defining a default authentication mechanism. It has the name '`OTHER`' and may be specified in either lower or upper case characters. Note, when there is a module specified for a named service, the '`OTHER`' entries are ignored.

`module-type`

One of (currently) four types of module. The four types are as follows:

- `auth`; this module type provides two aspects of authenticating the user. Firstly, it establishes that the user is who they claim to be, by instructing the application to prompt the user for a password or other means of identification. Secondly, the module can grant `group` membership (independently of the `/etc/groups` file discussed above) or other privileges through its *credential* granting properties.
- `account`; this module performs non-authentication based account management. It is typically used to restrict/permit access to a service based on the time of day, currently available system resources (maximum number of users) or perhaps the location of the applicant user—'`root`' login only on the console.
- `session`; primarily, this module is associated with doing things that need to be done for the user before/after they can be given service. Such things include the logging of information concerning the opening/closing of some data exchange with a user, mounting directories, etc. .
- `password`; this last module type is required for updating the authentication token associated with the user. Typically, there is one module for each 'challenge/response' based authentication (`auth`) module-type.

`control-flag`

The control-flag is used to indicate how the PAM library will react to the success or failure of the module it is associated with. Since modules can be *stacked* (modules of the same type execute in series, one after another), the control-flags determine the relative importance of each module. The application is not made aware of the individual success or failure of modules listed in the '`/etc/pam.conf`' file. Instead, it receives a summary *success* or *fail* response from the **Linux-PAM** library. The order of execution of these modules is that of the entries in the `/etc/pam.conf` file; earlier entries are executed before later ones. As of Linux-PAM v0.60, this *control-flag* can be defined with one of two syntaxes.

The simpler (and historical) syntax for the control-flag is a single keyword defined to indicate the severity of concern associated with the success or failure of a specific module. There are four such keywords: `required`, `requisite`, `sufficient` and `optional`.

The Linux-PAM library interprets these keywords in the following manner:

- `required`; this indicates that the success of the module is required for the `module-type` facility to succeed. Failure of this module will not be apparent to the user until all of the remaining modules (of the same `module-type`) have been executed.
- `requisite`; like `required`, however, in the case that such a module returns a failure, control is directly returned to the application. The return value is that associated with the *first* `required` or `requisite` module to fail. Note, this flag can be used to protect against the possibility of a user getting the opportunity to enter a password over an unsafe medium. It is conceivable that such behavior might inform an attacker of valid accounts on a system. This possibility should be weighed against the not insignificant concerns of exposing a sensitive password in a hostile environment.
- `sufficient`; the success of this module is deemed '*sufficient*' to satisfy the **Linux-PAM** library that this module-type has succeeded in its purpose. In the event that no previous `required` module has failed, no more '*stacked*' modules of this type are invoked. (Note, in this case subsequent `required` modules are **not** invoked.). A failure of this module is not deemed as fatal to satisfying the application that this `module-type` has succeeded.
- `optional`; as its name suggests, this `control-flag` marks the module as not being critical to the success or failure of the user's application for service. In general, **Linux-PAM** ignores such a module when determining if the module stack will succeed or fail. However, in the absence of any definite successes or failures of previous or subsequent stacked modules this module will determine the nature of the response to the application. One example of this latter case, is when the other modules return something like `PAM_IGNORE`.

The more elaborate (newer) syntax is much more specific and gives the administrator a great deal of control over how the user is authenticated. This form of the control flag is delimeted with square brackets and consists of a series of `value=action` tokens:

```
[value1=action1 value2=action2 ...]
```

Here, `valueI` is one of the following *return values*: `success`; `open_err`; `symbol_err`; `service_err`; `system_err`; `buf_err`; `perm_denied`; `auth_err`; `cred_insufficient`; `authinfo_unavail`; `user_unknown`; `maxtries`; `new_authtok_reqd`; `acct_expired`; `session_err`; `cred_unavail`; `cred_expired`; `cred_err`; `no_module_data`; `conv_err`; `authtok_err`; `authtok_recover_err`; `authtok_lock_busy`; `authtok_disable_aging`; `try_again`; `ignore`; `abort`; `authtok_expired`; `module_unknown`; `bad_item`; `conv_again`; `incomplete`; and `default`. The last of these (`default`) can be used to set the action for those return values that are not explicitly defined.

The `actionI` can be a positive integer or one of the following tokens: `ignore`; `ok`; `done`; `bad`; `die`; and `reset`. A positive integer, `J`, when specified as the action, can be used to indicate that the next J modules of the current type will be skipped. In this way, the administrator can develop a moderately sophisticated stack of modules with a number of different paths of execution. Which path is taken can be determined by the reactions of individual modules.

- `ignore` - when used with a stack of modules, the module's return status will not contribute to the return code the application obtains.
- `bad` - this action indicates that the return code should be thought of as indicative of the module failing. If this module is the first in the stack to fail, its status value will be used for that of the whole stack.

- `die` - equivalent to `bad` with the side effect of terminating the module stack and PAM immediately returning to the application.
- `ok` - this tells **PAM** that the administrator thinks this return code should contribute directly to the return code of the full stack of modules. In other words, if the former state of the stack would lead to a return of `PAM_SUCCESS`, the module's return code will override this value. Note, if the former state of the stack holds some value that is indicative of a modules failure, this 'ok' value will not be used to override that value.
- `done` - equivalent to `ok` with the side effect of terminating the module stack and PAM immediately returning to the application.
- `reset` - clear all memory of the state of the module stack and start again with the next stacked module.

Just to get a feel for the power of this new syntax, here is a taste of what you can do with it. With **Linux-PAM-0.63**, the notion of client plug-in agents was introduced. This is something that makes it possible for PAM to support machine-machine authentication using the transport protocol inherent to the client/server application. With the "`[ ... value=action ... ]`" control syntax, it is possible for an application to be configured to support binary prompts with compliant clients, but to gracefully fall over into an alternative authentication mode for older, legacy, applications. Flexible eh?

`module-path`

The path-name of the dynamically loadable object file; *the pluggable module* itself. If the first character of the module path is '/', it is assumed to be a complete path. If this is not the case, the given module path is appended to the default module path: `/usr/lib/security` (but see the notes 2 (above)).

`args`

The `args` are a list of tokens that are passed to the module when it is invoked. Much like arguments to a typical Linux shell command. Generally, valid arguments are optional and are specific to any given module. Invalid arguments are ignored by a module, however, when encountering an invalid argument, the module is required to write an error to `syslog(3)`. For a list of *generic* options see the next section.

Any line in (one of) the confiuration file(s), that is not formatted correctly, will generally tend (erring on the side of caution) to make the authentication process fail. A corresponding error is written to the system log files with a call to `syslog(3)`.

4.2 Directory based configuration

More flexible than the single configuration file, as of version 0.56, it is possible to configure `libpam` via the contents of the `/etc/pam.d/` directory. In this case the directory is filled with files each of which has a filename equal to a service-name (in lower-case): it is the personal configuration file for the named service.

Linux-PAM can be compiled in one of two modes. The preferred mode uses either `/etc/pam.d/` or `/etc/pam.conf` configuration but not both. That is to say, if there is a `/etc/pam.d/` directory then libpam only uses the files contained in this directory. However, in the absence of the `/etc/pam.d/` directory the `/etc/pam.conf` file is used. The other mode (and the one currently supported by Red Hat 4.2 and higher) is to use both `/etc/pam.d/` and `/etc/pam.conf` in sequence. In this mode, entries in `/etc/pam.d/` override those of `/etc/pam.conf`.

The syntax of each file in `/etc/pam.d/` is similar to that of the `/etc/pam.conf` file and is made up of lines of the following form:

```
module-type   control-flag   module-path   arguments
```

The only difference being that the `service-name` is not present. The service-name is of course the name of the given configuration file. For example, `/etc/pam.d/login` contains the configuration for the *login* service.

This method of configuration has a number of advantages over the single file approach. We list them here to assist the reader in deciding which scheme to adopt:

- A lower chance of misconfiguring an application. There is one less field to mis-type when editing the configuration files by hand.
- Easier to maintain. One application may be reconfigured without risk of interfering with other applications on the system.
- It is possible to symbolically link different services configuration files to a single file. This makes it easier to keep the system policy for access consistent across different applications. (It should be noted, to conserve space, it is equally possible to *hard* link a number of configuration files. However, care should be taken when administering this arrangement as editing a hard linked file is likely to break the link.)
- A potential for quicker configuration file parsing. Only the relevant entries are parsed when a service gets bound to its modules.
- It is possible to limit read access to individual **Linux-PAM** configuration files using the file protections of the filesystem.
- Package management becomes simpler. Every time a new application is installed, it can be accompanied by an `/etc/pam.d/`*xxxxxx* file.

4.3 Generic optional arguments

The following are optional arguments which are likely to be understood by any module. Arguments (including these) are in general *optional*.

`debug`

Use the `syslog(3)` call to log debugging information to the system log files.

`no_warn`

Instruct module to not give warning messages to the application.

`use_first_pass`

The module should not prompt the user for a password. Instead, it should obtain the previously typed password (from the preceding `auth` module), and use that. If that doesn't work, then the user will not be authenticated. (This option is intended for `auth` and `password` modules only).

try_first_pass

The module should attempt authentication with the previously typed password (from the preceding `auth` module). If that doesn't work, then the user is prompted for a password. (This option is intended for `auth` modules only).

use_mapped_pass

This argument is not currently supported by any of the modules in the **Linux-PAM** distribution because of possible consequences associated with U.S. encryption exporting restrictions. Within the U.S., module developers are, of course, free to implement it (as are developers in other countries). For compatibility reasons we describe its use as suggested in the **DCE-RFC 86.0**, see section 8 (bibliography) for a pointer to this document.

The `use_mapped_pass` argument instructs the module to take the clear text authentication token entered by a previous module (that requests such a token) and use it to generate an encryption/decryption key with which to safely store/retrieve the authentication token required for this module. In this way the user can enter a single authentication token and be quietly authenticated by a number of stacked modules. Obviously a convenient feature that necessarily requires some reliably strong encryption to make it secure. This argument is intended for the `auth` and `password` module types only.

expose_account

In general the leakage of some information about user accounts is not a secure policy for modules to adopt. Sometimes information such as users names or home directories, or preferred shell, can be used to attack a user's account. In some circumstances, however, this sort of information is not deemed a threat: displaying a user's full name when asking them for a password in a secured environment could also be called being 'friendly'. The `expose_account` argument is a standard module argument to encourage a module to be less discrete about account information as it is deemed appropriate by the local administrator.

4.4 Example configuration file entries

In this section, we give some examples of entries that can be present in the **Linux-PAM** configuration file. As a first attempt at configuring your system you could do worse than to implement these.

4.4.1 Default policy

If a system is to be considered secure, it had better have a reasonably secure '`OTHER`' entry. The following is a paranoid setting (which is not a bad place to start!):

```
#
# default; deny access
#
OTHER   auth     required       /usr/lib/security/pam_deny.so
OTHER   account  required       /usr/lib/security/pam_deny.so
OTHER   password required       /usr/lib/security/pam_deny.so
OTHER   session  required       /usr/lib/security/pam_deny.so
```

Whilst fundamentally a secure default, this is not very sympathetic to a misconfigured system. For example, such a system is vulnerable to locking everyone out should the rest of the file become badly written.

The module `pam_deny` (documented in a later section) is not very sophisticated. For example, it logs no information when it is invoked so unless the users of a system contact the administrator when failing to execute a service application, the administrator may go for a long while in ignorance of the fact that his system is misconfigured.

The addition of the following line before those in the above example would provide a suitable warning to the administrator.

```
#
# default; wake up! This application is not configured
#
OTHER   auth     required       /usr/lib/security/pam_warn.so
OTHER   password required       /usr/lib/security/pam_warn.so
```

Having two "`OTHER auth`" lines is an example of stacking.

On a system that uses the `/etc/pam.d/` configuration, the corresponding default setup would be achieved with the following file:

```
#
# default configuration: /etc/pam.d/other
#
auth     required       /usr/lib/security/pam_warn.so
auth     required       /usr/lib/security/pam_deny.so
account  required       /usr/lib/security/pam_deny.so
password required       /usr/lib/security/pam_warn.so
password required       /usr/lib/security/pam_deny.so
session  required       /usr/lib/security/pam_deny.so
```

This is the only explicit example we give for an `/etc/pam.d/` file. In general, it should be clear how to transpose the remaining examples to this configuration scheme.

On a less sensitive computer, one on which the system administrator wishes to remain ignorant of much of the power of `Linux-PAM`, the following selection of lines (in `/etc/pam.conf`) is likely to mimic the historically familiar Linux setup.

```
#
# default; standard UNIX access
#
OTHER   auth     required       /usr/lib/security/pam_unix_auth.so
OTHER   account  required       /usr/lib/security/pam_unix_acct.so
OTHER   password required       /usr/lib/security/pam_unix_passwd.so
OTHER   session  required       /usr/lib/security/pam_unix_session.so
```

In general this will provide a starting place for most applications. Unfortunately, most is not all. One application that might require additional lines is *ftpd* if you wish to enable *anonymous-ftp*.

To enable anonymous-ftp, the following lines might be used to replace the default (`OTHER`) ones. (***WARNING*** as of 1996/12/28 this does not work correctly with any ftpd. Consequently, this description may be subject to change or the application will be fixed.)

```
#
```

```
# ftpd; add ftp-specifics. These lines enable anonymous ftp over
#       standard UNIX access (the listfile entry blocks access to
#       users listed in /etc/ftpusers)
#
ftpd    auth    sufficient  /usr/lib/security/pam_ftp.so
ftpd    auth    required    /usr/lib/security/pam_unix_auth.so use_first_pass
ftpd    auth    required    /usr/lib/security/pam_listfile.so \
                        onerr=succeed item=user sense=deny file=/etc/ftpusers
```

Note, the second line is necessary since the default entries are ignored by a service application (here *ftpd*) if there are *any* entries in `/etc/pam.conf` for that specified service. Again, this is an example of authentication module stacking. Note the use of the `sufficient` control-flag. It says that "if this module authenticates the user, ignore the subsequent `auth` modules". Also note the use of the "`use_first_pass`" module-argument, this instructs the UNIX authentication module that it is not to prompt for a password but rely one already having been obtained by the ftp module.

5 Security issues of Linux-PAM

This section will discuss good practices for using Linux-PAM in a secure manner. *It is currently sadly lacking...suggestions are welcome!*

5.1 If something goes wrong

Linux-PAM has the potential to seriously change the security of your system. You can choose to have no security or absolute security (no access permitted). In general, **Linux-PAM** errs towards the latter. Any number of configuration errors can dissable access to your system partially, or completely.

The most dramatic problem that is likely to be encountered when configuring **Linux-PAM** is that of *deleting* the configuration file(s): `/etc/pam.d/*` and/or `/etc/pam.conf`. This will lock you out of your own system!

To recover, your best bet is to reboot the system in single user mode and set about correcting things from there. The following has been *adapted* from a life-saving email on the subject from David Wood:

NOTE: The following assumes that booting to single-user mode does not prompt for a password. However, on Debian and other distributions, this is not true. Init will instead start `sulogin`, which asks for the root password. The program `sulogin` does not use PAM, so a broken PAM setup will not break this program.

```
> What the hell do I do now?

OK, don't panic. The first thing you have to realize is that
this happens to 50% of users who ever do anything with PAM.
It happened here, not once, not twice, but three times, all
different, and in the end, the solution was the same every
time.

First, I hope you installed LILO with a delay. If you can,
reboot, hit shift or tab or something and type:
```

```
    LILO boot: linux single

(Replace 'linux' with 'name-of-your-normal-linux-image').
This will let you in without logging in.  Ever wondered how
easy it is to break into a linux machine from the console?
Now you know.

If you can't do that, then get yourself a bootkernel floppy
and a root disk a-la slackware's rescue.gz.  (Red Hat's
installation disks can be used in this mode too.)

In either case, the point is to get back your root prompt.

Second, I'm going to assume that you haven't completely
nuked your pam installation - just your configuration files.
Here's how you make your configs nice again:

    cd /etc
    mv pam.conf pam.conf.orig
    mv pam.d pam.d.orig
    mkdir pam.d
    cd pam.d

and then use vi to create a file called "other" in this
directory.  It should contain the following four lines:

    auth     required       pam_unix_auth.so
    account  required       pam_unix_acct.so
    password required       pam_unix_passwd.so
    session  required       pam_unix_session.so

Now you have the simplest possible PAM configuration that
will work the way you're used to.  Everything should
magically start to work again.  Try it out by hitting ALT-F2
and logging in on another virtual console.  If it doesn't
work, you have bigger problems, or you've mistyped
something.  One of the wonders of this system (seriously,
perhaps) is that if you mistype anything in the conf files,
you usually get no error reporting of any kind on the
console - just some entries in the log file.  So look there!
(Try 'tail /var/log/messages'.)

From here you can go back and get a real configuration
going, hopefully after you've tested it first on a machine
you don't care about screwing up.  :/

Some pointers (to make everything "right" with Red Hat...):
```

```
Install the newest pam, pamconfig, and pwdb from the
redhat current directory, and do it all on the same
command line with rpm...

    rpm -Uvh [maybe --force too] pam-* pamconfig-* pwdb-*

Then make sure you install (or reinstall) the newest
version of libc, util-linux, wuftp, and NetKit. For
kicks you might try installing the newest versions of
the affected x apps, like xlock, but I haven't gotten
those to work at all yet.
```

5.2 Avoid having a weak 'other' configuration

It is not a good thing to have a weak default (`OTHER`) entry. This service is the default configuration for all PAM aware applications and if it is weak, your system is likely to be vulnerable to attack.

Here is a sample "other" configuration file. The *pam_deny* module will deny access and the *pam_warn* module will send a syslog message to `auth.notice`:

```
#
# The PAM configuration file for the `other' service
#
auth      required   pam_deny.so
auth      required   pam_warn.so
account   required   pam_deny.so
account   required   pam_warn.so
password  required   pam_deny.so
password  required   pam_warn.so
session   required   pam_deny.so
session   required   pam_warn.so
```

6 A reference guide for available modules

Here, we collect together some descriptions of the various modules available for **Linux-PAM**. In general these modules should be freely available. Where this is not the case, it will be indicated.

Also please note the comments contained in the section 2 (on text conventions above) when copying the examples listed below.

6.1 The access module

6.1.1 Synopsis

Module Name:

`pam_access`

Author[s]:

Alexei Nogin <alexei@nogin.dnttm.ru>

Maintainer:

Author

Management groups provided:

account

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Requires a configuration file `/etc/security/access.conf`

Network aware:

Through `PAM_TTY` if set, otherwise attempts getting tty name of the stdin file descriptor with `ttyname()`. Standard gethostname(), `yp_get_default_domain()`, `gethostbyname()` calls. **NIS** is used for netgroup support.

6.1.2 Overview of module

Provides logdaemon style login access control.

6.1.3 Account component

Recognized arguments:

Description:

This module provides logdaemon style login access control based on login names and on host (or domain) names, internet addresses (or network numbers), or on terminal line names in case of non-networked logins. Diagnostics are reported through `syslog(3)`. Wietse Venema's `login_access.c` from *logdaemon-5.6* is used with several changes by A. Nogin.

Examples/suggested usage:

Use of module is recommended, for example, on administrative machines such as **NIS** servers and mail servers where you need several accounts active but don't want them all to have login capability.

For `/etc/pam.d` style configurations where your modules live in `/lib/security`, start by adding the following line to `/etc/pam.d/login`, `/etc/pam.d/rlogin`, `/etc/pam.d/rsh` and `/etc/pam.d/ftp`:

```
account  required       /lib/security/pam_access.so
```

Note that use of this module is not effective unless your system ignores `.rhosts` files. See the the pam_rhosts_auth documentation.

A sample `access.conf` configuration file is included with the distribution.

6.2 Chroot

6.2.1 Synopsis

Module Name:

pam_chroot

Author:

Bruce Campbell <brucec@humbug.org.au>

Maintainer:

Author; proposed on 20/11/96 - email for status

Management groups provided:

account; session; authentication

Cryptographically sensitive:

Security rating:

Clean code base:

Unwritten.

System dependencies:

Network aware:

Expects localhost.

6.2.2 Overview of module

This module is intended to provide a transparent wrapper around the average user, one that puts them in a fake file-system (eg, their '/' is really `/some/where/else`).

Useful if you have several classes of users, and are slightly paranoid about security. Can be used to limit who else users can see on the system, and to limit the selection of programs they can run.

6.2.3 Account component:

Need more info here.

6.2.4 Authentication component:

Need more info here.

6.2.5 Session component:

Need more info here.

Recognized arguments:

Arguments and logging levels for the PAM version are being worked on.

Description:

Examples/suggested usage:

Do provide a reasonable list of programs - just tossing 'cat', 'ls', 'rm', 'cp' and 'ed' in there is a bit...

Don't take it to extremes (eg, you can set up a separate environment for each user, but its a big waste of your disk space.)

6.3 Cracklib pluggable password strength-checker

6.3.1 Synopsis

Module Name:

pam_cracklib

Author:

Cristian Gafton <gafton@redhat.com>

Maintainer:

Author.

Management groups provided:

password

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Requires the system library `libcrack` and a system dictionary: `/var/cache/cracklib/cracklib_dict`.

Network aware:

6.3.2 Overview of module

This module can be plugged into the `password` stack of a given application to provide some plug-in strength-checking for passwords.

This module works in the following manner: it first calls the *Cracklib* routine to check the strength of the password; if crack likes the password, the module does an additional set of strength checks. These checks are:

- **Palindrome** -

  Is the new password a palindrome of the old one?

- **Case Change Only** -

  Is the new password the the old one with only a change of case?

- **Similar** -

  Is the new password too much like the old one? This is controlled by one argument, `difok` which is a number of characters that if different between the old and new are enough to accept the new password, this defaults to 10 or 1/2 the size of the new password whichever is smaller.

- **Simple** -

  Is the new password too small? This is controlled by 5 arguments `minlen`, `dcredit`, `ucredit`, `lcredit`, and `ocredit`. See the section on the arguments for the details of how these work and there defaults.

- **Rotated** -

  Is the new password a rotated version of the old password?

- **Already used** -

  Was the password used in the past? Previously used passwords are to be found in /etc/security/opasswd.

This module with no arguments will work well for standard unix password encryption. With md5 encryption, passwords can be longer than 8 characters and the default settings for this module can make it hard for the user to choose a satisfactory new password. Notably, the requirement that the new password contain no more than 1/2 of the characters in the old password becomes a non-trivial constraint. For example, an old password of the form "the quick brown fox jumped over the lazy dogs" would be difficult to change... In addition, the default action is to allow passwords as small as 5 characters in length. For a md5 systems it can be a good idea to increase the required minimum size of a password. One can then allow more credit for different kinds of characters but accept that the new password may share most of these characters with the old password.

6.3.3 Password component

Recognized arguments:

`debug`; `type=XXX`; `retry=N`; `difok=N`; `minlen=N`; `dcredit=N`; `ucredit=N`; `lcredit=N`; `ocredit=N`;

Description:

The action of this module is to prompt the user for a password and check its strength against a system dictionary and a set of rules for identifying poor choices.

The default action is to prompt for a single password, check its strength and then, if it is considered strong, prompt for the password a second time (to verify that it was typed correctly on the first occasion). All being well, the password is passed on to subsequent modules to be installed as the new authentication token.

The default action may be modified in a number of ways using the arguments recognized by the module:

- `debug` -

  this option makes the module write information to syslog(3) indicating the behavior of the module (this option does **not** write password information to the log file).

- `type=XXX` -

  the default action is for the module to use the following prompts when requesting passwords: "New UNIX password: " and "Retype UNIX password: ". Using this option you can replace the word UNIX with `XXX`.

- `retry=N` -

  the default number of times this module will request a new password (for strength-checking) from the user is 1. Using this argument this can be increased to `N`.

- `difok=N` -

  This argument will change the default of 10 for the number of characters in the new password that must not be present in the old password. In addition, if 1/2 of the characters in the new password are different then the new password will be accepted anyway.

- `minlen=N` -

  The minimum acceptable size for the new password plus one. In addition to the number of characters in the new password, credit (of +1 in length) is given for each different kind of character (*other, upper, lower* and *digit*). The default for this parameter is 9 which is good for a old style UNIX password all of the same type of character but may be too low to exploit the added security of a md5 system. Note that there is a pair of length limits in *Cracklib* itself, a "way too short" limit of 4 which is hard coded in and a defined limit (6) that will be checked without reference to `minlen`. If you want to allow passwords as short as 5 characters you should either not use this module or recompile the crack library and then recompile this module.

- `dcredit=N` -

  This is the maximum credit for having digits in the new password. If you have less than or `N` digits, each digit will count +1 towards meeting the current `minlen` value. The default for `dcredit` is 1 which is the recommended value for `minlen` less than 10.

- `ucredit=N` -

  This is the maximum credit for having upper case letters in the new password. If you have less than or `N` upper case letters each letter will count +1 towards meeting the current `minlen` value. The default for `ucredit` is 1 which is the recommended value for `minlen` less than 10.

- `lcredit=N` -

  This is the maximum credit for having lower case letters in the new password. If you have less than or `N` lower case letters, each letter will count +1 towards meeting the current `minlen` value. The default for `lcredit` is 1 which is the recommended value for `minlen` less than 10.

- `ocredit=N` -

  This is the maximum credit for having other characters in the new password. If you have less than or `N` other characters, each character will count +1 towards meeting the current `minlen` value. The default for `ocredit` is 1 which is the recommended value for `minlen` less than 10.

Examples/suggested usage:

For an example of the use of this module, we show how it may be stacked with the password component of `pam_unix`:

```
#
# These lines stack two password type modules. In this example the
# user is given 3 opportunities to enter a strong password. The
# "use_authtok" argument ensures that the pam_unix module does not
# prompt for a password, but instead uses the one provided by
# pam_cracklib.
#
passwd  password required       pam_cracklib.so retry=3
passwd  password required       pam_unix.so use_authtok
```

Another example (in the `/etc/pam.d/passwd` format) is for the case that you want to use md5 password encryption:

```
#%PAM-1.0
#
# These lines allow a md5 systems to support passwords of at least 14
# bytes with extra credit of 2 for digits and 2 for others the new
# password must have at least three bytes that are not present in the
# old password
#
password  required pam_cracklib.so \
               difok=3 minlen=15 dcredit= 2 ocredit=2
password  required pam_unix.so use_authtok nullok md5
```

6.4 The locking-out module

6.4.1 Synopsis

Module Name:

pam_deny

Author:

Andrew G. Morgan <morgan@parc.power.net>

Maintainer:

current **Linux-PAM** maintainer

Management groups provided:

account; authentication; password; session

Cryptographically sensitive:

Security rating:

Clean code base:

clean.

System dependencies:

Network aware:

6.4.2 Overview of module

This module can be used to deny access. It always indicates a failure to the application through the PAM framework. As is commented in the overview section 3 (above), this module might be suitable for using for default (the `OTHER`) entries.

6.4.3 Account component

Recognized arguments:

Description:

This component does nothing other than return a failure. The failure type is `PAM_ACCT_EXPIRED`.

Examples/suggested usage:

Stacking this module with type `account` will prevent the user from gaining access to the system via applications that refer to **Linux-PAM**'s account management function `pam_acct_mgmt()`.

The following example would make it impossible to login:

```
#
# add this line to your other login entries to disable all accounts
#
login   account  required       pam_deny.so
```

6.4.4 Authentication component

Recognized arguments:

Description:

This component does nothing other than return a failure. The failure type is `PAM_AUTH_ERR` in the case that `pam_authenticate()` is called (when the application tries to authenticate the user), and is `PAM_CRED_UNAVAIL` when the application calls `pam_setcred()` (to establish and set the credentials of the user – it is unlikely that this function will ever be called in practice).

Examples/suggested usage:

To deny access to default applications with this component of the `pam_deny` module, you might include the following line in your **Linux-PAM** configuration file:

```
#
# add this line to your existing OTHER entries to prevent
# authentication succeeding with default applications.
#
OTHER   auth     required       pam_deny.so
```

6.4.5 Password component

Recognized arguments:

Description:

This component of the module denies the user the opportunity to change their password. It always responds with `PAM_AUTHTOK_ERR` when invoked.

Examples/suggested usage:

This module should be used to prevent an application from updating the applicant user's password. For example, to prevent `login` from automatically prompting for a new password when the old one has expired you should include the following line in your configuration file:

```
#
# add this line to your other login entries to prevent the login
# application from being able to change the user's password.
#
login   password required       pam_deny.so
```

6.4.6 Session component

Recognized arguments:

Description:

This aspect of the module prevents an application from starting a session on the host computer.

Examples/suggested usage:

Together with another session module, that displays a message of the day perhaps (XXX - such a module needs to be written), this module can be used to block a user from starting a shell. Given the presence of a `pam_motd` module, we might use the following entries in the configuration file to inform the user it is system time:

```
#
# An example to see how to configure login to refuse the user a
# session (politely)
#
login   session  required       pam_motd.so \
                         file=/etc/system_time
login   session  required       pam_deny.so
```

6.5 Set/unset environment variables

6.5.1 Synopsis

Module Name:

`pam_env`

Author:

Dave Kinchlea <kinch@kinch.ark.com>

Maintainer:

Author

Management groups provided:

Authentication (setcred)

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

`/etc/security/pam_env.conf`

Network aware:

6.5.2 Overview of module

This module allows the (un)setting of environment variables. Supported is the use of previously set environment variables as well as *PAM_ITEM*s such as `PAM_RHOST`.

6.5.3 Authentication component

Recognized arguments:

`debug`; `conffile=`*configuration-file-name*; `envfile=`/*env-file-name*; `readenv=`/*0*|*1*

Description:

This module allows you to (un)set arbitrary environment variables using fixed strings, the value of previously set environment variables and/or *PAM_ITEM*s.

All is controlled via a configuration file (by default, `/etc/security/pam_env.conf` but can be overriden with `connfile` argument). Each line starts with the variable name, there are then two possible options for each variable **DEFAULT** and **OVERRIDE**. **DEFAULT** allows and administrator to set the value of the variable to some default value, if none is supplied then the empty string is assumed. The **OVERRIDE** option tells pam_env that it should enter in its value (overriding the default value) if there is one to use. **OVERRIDE** is not used, `""` is assumed and no override will be done.

```
VARIABLE   [DEFAULT=[value]]  [OVERRIDE=[value]]
```

(Possibly non-existent) environment variables may be used in values using the `${string}` syntax and (possibly non-existent) *PAM_ITEM*s may be used in values using the `@{string}` syntax. Both the `$` and `@` characters can be backslash-escaped to be used as literal values (as in `\$`. Double quotes may be used in values (but not environment variable names) when white space is needed **the full value must be delimited by the quotes and embedded or escaped quotes are not supported**.

This module can also parse a file with simple KEY=VAL pairs on seperate lines (/etc/environment by default). You can change the default file to parse, with the *envfile* flag and turn it on or off by setting the *readenv* flag to 1 or 0 respectively.

The behavior of this module can be modified with one of the following flags:

- `debug` - write more information to `syslog(3)`.

- `conffile=`*filename* - by default the file `/etc/security/pam_env.conf` is used as the configuration file. This option overrides the default. You must supply a complete path + file name.
- `envfile=`*filename* - by default the file `/etc/environment` is used to load KEY=VAL pairs directly into the env. This option overrides the default. You must supply a complete path + file name.
- `readenv=`*0|1* - turns on or off the reading of the file specified by envfile (0 is off, 1 is on). By default this option is on.

Examples/suggested usage:

See sample `pam_env.conf` for more information and examples.

6.6 The filter module

6.6.1 Synopsis

Module Name:

pam_filter

Author:

Andrew G. Morgan <morgan@parc.power.net>

Maintainer:

Author.

Management groups provided:

account; authentication; password; session

Cryptographically sensitive:

Not yet.

Security rating:

Clean code base:

This module compiles cleanly on Linux based systems.

System dependencies:

To function it requires *filters* to be installed on the system.

Network aware:

6.6.2 Overview of module

This module was written to offer a plug-in alternative to programs like ttysnoop (XXX - need a reference). Since writing a filter that performs this function has not occurred, it is currently only a toy. The single filter provided with the module simply transposes upper and lower case letters in the input and output streams. (This can be very annoying and is not kind to termcap based editors).

6.6.3 Account+Authentication+Password+Session components

Recognized arguments:

`debug`; `new_term`; `non_term`; `runX`

Description:

Each component of the module has the potential to invoke the desired filter. The filter is always `execv(2)`d with the privilege of the calling application and **not** that of the user. For this reason it cannot usually be killed by the user without closing their session.

The behavior of the module can be significantly altered by the arguments passed to it in the **Linux-PAM** configuration file:

- `debug` -

  this option increases the amount of information logged to `syslog(3)` as the module is executed.

- `new_term` -

  the default action of the filter is to set the `PAM_TTY` item to indicate the terminal that the user is using to connect to the application. This argument indicates that the filter should set `PAM_TTY` to the filtered pseudo-terminal.

- `non_term` - don't try to set the `PAM_TTY` item.

- `runX` -

  in order that the module can invoke a filter it should know when to invoke it. This argument is required to tell the filter when to do this. The arguments that follow this one are respectively the full pathname of the filter to be run and any command line arguments that the filter might expect.

  Permitted values for `X` are `1` and `2`. These indicate the precise time the that filter is to be run. To explain this concept it will be useful to have read the Linux-PAM Module developer's guide. Basically, for each management group there are up to two ways of calling the module's functions.

  In the case of the *authentication* and *session* components there are actually two separate functions. For the case of authentication, these functions are `_authenticate` and `_setcred` – here `run1` means run the filter from the `_authenticate` function and `run2` means run the filter from `_setcred`. In the case of the session modules, `run1` implies that the filter is invoked at the `_open_session` stage, and `run2` for `_close_session`.

  For the case of the account component. Either `run1` or `run2` may be used.

  For the case of the password component, `run1` is used to indicate that the filter is run on the first occasion `_chauthtok` is run (the `PAM_PRELIM_CHECK` phase) and `run2` is used to indicate that the filter is run on the second occasion (the `PAM_UPDATE_AUTHTOK` phase).

Examples/suggested usage:

At the time of writing there is little real use to be made of this module. For fun you might try adding the following line to your login's configuration entries

```
#
# An example to see how to configure login to transpose upper and
# lower case letters once the user has logged in(!)
#
login   session  required       pam_filter.so \
                          run1 /usr/sbin/pam_filter/upperLOWER
```

6.7 Anonymous access module

6.7.1 Synopsis

Module Name:

pam_ftp.so

Author:

Andrew G. Morgan <morgan@linux.kernel.org>

Maintainer:

Author.

Management groups provided:

authentication

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Network aware:

prompts for email address of user; easily spoofed (XXX - needs work)

6.7.2 Overview of module

The purpose of this module is to provide a pluggable anonymous ftp mode of access.

6.7.3 Authentication component

Recognized arguments:

`debug`; `users=XXX,YYY,...`; `ignore`

Description:

This module intercepts the user's name and password. If the name is "`ftp`" or "`anonymous`", the user's password is broken up at the '`@`' delimiter into a `PAM_RUSER` and a `PAM_RHOST` part; these pam-items being set accordingly. The username is set to "`ftp`". In this case the module succeeds. Alternatively, the module sets the `PAM_AUTHTOK` item with the entered password and fails.

The behavior of the module can be modified with the following flags:

- `debug` - log more information to with `syslog(3)`.
- `users=XXX,YYY,...` - instead of "`ftp`" or "`anonymous`", provide anonymous login to the comma separated list of users; "`XXX,YYY,...`". Should the applicant enter one of these usernames the returned username is set to the first in the list; "`XXX`".
- `ignore` - pay no attention to the email address of the user (if supplied).

Examples/suggested usage:

An example of the use of this module is provided in the configuration file section 4 (above). With care, this module could be used to provide new/temporary account anonymous login.

6.8 The group access module

6.8.1 Synopsis

Module Name:

`pam_group`

Author:

Andrew G. Morgan <morgan@parc.power.net>

Maintainer:

Author.

Management groups provided:

authentication

Cryptographically sensitive:

Security rating:

Sensitive to *setgid* status of file-systems accessible to users.

Clean code base:

System dependencies:

Requires an `/etc/security/group.conf` file. Can be compiled with or without `libpwdb`.

Network aware:

Only through correctly set `PAM_TTY` item.

6.8.2 Overview of module

This module provides group-settings based on the user's name and the terminal they are requesting a given service from. It takes note of the time of day.

6.8.3 Authentication component

Recognized arguments:

Description:

This module does not authenticate the user, but instead it grants group memberships (in the credential setting phase of the authentication module) to the user. Such memberships are based on the service they are applying for. The group memberships are listed in text form in the `/etc/security/group.conf` file.

Examples/suggested usage:

For this module to function correctly there must be a correctly formatted `/etc/security/groups.conf` file present. The format of this file is as follows. Group memberships are given based on the service application satisfying any combination of lines in the configuration file. Each line (barring comments which are preceded by '`#`' marks) has the following syntax:

```
services  ;  ttys  ;  users  ;  times  ;  groups
```

Here the first four fields share the syntax of the `pam_time` configuration file; `/etc/security/pam_time.conf`, and the last field, the `groups` field, is a comma (or space) separated list of the text-names of a selection of groups. If the users application for service satisfies the first four fields, the user is granted membership of the listed groups.

As stated in above this module's usefulness relies on the file-systems accessible to the user. The point being that once granted the membership of a group, the user may attempt to create a *setgid* binary with a restricted group ownership. Later, when the user is not given membership to this group, they can recover group membership with the precompiled binary. The reason that the file-systems that the user has access to are so significant, is the fact that when a system is mounted *nosuid* the user is unable to create or execute such a binary file. For this module to provide any level of security, all file-systems that the user has write access to should be mounted *nosuid*.

The `pam_group` module fuctions in parallel with the `/etc/group` file. If the user is granted any groups based on the behavior of this module, they are granted *in addition* to those entries `/etc/group` (or equivalent).

6.9 Add issue file to user prompt

6.9.1 Synopsis

Module Name:

`pam_issue`

Author:

Ben Collins <bcollins@debian.org>

Maintainer:

Author

Management groups provided:

Authentication (pam_sm_authenticate)

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Network aware:

6.9.2 Overview of module

This module prepends the issue file (*/etc/issue* by default) when prompting for a username.

6.9.3 Authentication component

Recognized arguments:

`issue=issue-file-name`; `noesc`;

Description:

This module allows you to prepend an issue file to the username prompt. It also by default parses escape codes in the issue file similar to some common getty's (using \x format).

Recognized escapes:

- `d` - current date
- `s` - operating system name
- `l` - name of this tty
- `m` - architecture of this system (i686, sparc, powerpc, ...)
- `n` - hostname of this system
- `o` - domainname of this system
- `r` - release number of the operation system (eg. 2.2.12)
- `t` - current time
- `u` - number of users currently logged in
- `U` - same as `u`, except it is suffixed with "user" or "users" (eg. "1 user" or "10 users"
- `v` - version/build-date of the operating system (eg. "#3 Mon Aug 23 14:38:16 EDT 1999" on Linux).

The behavior of this module can be modified with one of the following flags:

- `issue` - the file to output if not using the default
- `noesc` - turns off escape code parsing

Examples/suggested usage:

login auth pam_issue.so issue=/etc/issue

6.10 The Kerberos 4 module.

6.10.1 Synopsis

Module Name:

`pam_krb4`

Author:

Derrick J. Brashear <shadow@dementia.org>

Maintainer:

Author.

Management groups provided:

authentication; password; session

Cryptographically sensitive:

uses API

Security rating:

Clean code base:

System dependencies:

libraries - `libkrb`, `libdes`, `libcom_err`, `libkadm`; and a set of Kerberos include files.

Network aware:

Gets Kerberos ticket granting ticket via a Kerberos key distribution center reached via the network.

6.10.2 Overview of module

This module provides an interface for doing Kerberos verification of a user's password, getting the user a Kerberos ticket granting ticket for use with the Kerberos ticket granting service, destroying the user's tickets at logout time, and changing a Kerberos password.

6.10.3 Session component

Recognized arguments:

Description:

This component of the module currently sets the user's `KRBTKFILE` environment variable (although there is currently no way to export this), as well as deleting the user's ticket file upon logout (until `PAM_CRED_DELETE` is supported by *login*).

Examples/suggested usage:

This part of the module won't be terribly useful until we can change the environment from within a `Linux-PAM` module.

6.10.4 Password component

Recognized arguments:

`use_first_pass`; `try_first_pass`

Description:

This component of the module changes a user's Kerberos password by first getting and using the user's old password to get a session key for the password changing service, then sending a new password to that service.

Examples/suggested usage:

This should only be used with a real Kerberos v4 `kadmind`. It cannot be used with an AFS kaserver unless special provisions are made. Contact the module author for more information.

6.10.5 Authentication component

Recognized arguments:

`use_first_pass`; `try_first_pass`

Description:

This component of the module verifies a user's Kerberos password by requesting a ticket granting ticket from the Kerberos server and optionally using it to attempt to retrieve the local computer's host key and verifying using the key file on the local machine if one exists.

It also writes out a ticket file for the user to use later, and deletes the ticket file upon logout (not until `PAM_CRED_DELETE` is called from *login*).

Examples/suggested usage:

This module can be used with a real Kerberos server using MIT v4 Kerberos keys. The module or the system Kerberos libraries may be modified to support AFS style Kerberos keys. Currently this is not supported to avoid cryptography constraints.

6.11 The last login module

6.11.1 Synopsis

Module Name:

`pam_lastlog`

Author:

Andrew G. Morgan <morgan@parc.power.net>

Maintainer:

Author

Management groups provided:

auth

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

uses information contained in the `/var/log/lastlog` file.

Network aware:

6.11.2 Overview of module

This session module maintains the /var/log/lastlog file. Adding an open entry when called via the pam_open_seesion() function and completing it when pam_close_session() is called. This module can also display a line of information about the last login of the user. If an application already performs these tasks, it is not necessary to use this module.

6.11.3 Authentication component

Recognized arguments:

debug; nodate; noterm; nohost; silent; never

Description:

This module can be used to provide a "Last login on ..." message. when the user logs into the system from what ever application uses the PAM libraries. In addition, the module maintains the /var/log/lastlog file.

The behavior of this module can be modified with one of the following flags:

- debug - write more information to syslog(3).
- nodate - neglect to give the date of the last login when displaying information about the last login on the system.
- noterm - neglect to diplay the terminal name on which the last login was attempt.
- nohost - neglect to indicate from which host the last login was attempted.
- silent - neglect to inform the user about any previous login: just update the /var/log/lastlog file.
- never - if the /var/log/lastlog file does not contain any old entries for the user, indicate that the user has never previously logged in with a "welcome..." message.

Examples/suggested usage:

This module can be used to indicate that the user has new mail when they *login* to the system. Here is a sample entry for your /etc/pam.conf file:

```
#
# do we have any mail?
#
login   session  optional       pam_lastlog.so
```

Note, some applications may perform this function themselves. In such cases, this module is not necessary.

6.12 The resource limits module

6.12.1 Synopsis

Module Name:

pam_limits

Authors:

Cristian Gafton <gafton@redhat.com>
Thanks are also due to Elliot Lee <sopwith@redhat.com> for his comments on improving this module.

Maintainer:

Cristian Gafton - 1996/11/20

Management groups provided:

session

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

requires an `/etc/security/limits.conf` file and kernel support for resource limits. Also uses the library, `libpwdb`.

Network aware:

6.12.2 Overview of module

This module, through the **Linux-PAM** *open*-session hook, sets limits on the system resources that can be obtained in a user-session. Its actions are dictated more explicitly through the configuration file discussed below.

6.12.3 Session component

Recognized arguments:

`debug`; `conf=/path/to/file.conf`

Description:

Through the contents of the configuration file, `/etc/security/limits.conf`, resource limits are placed on users' sessions. Users of `uid=0` are not affected by this restriction.

The behavior of this module can be modified with the following arguments:

- `debug` - verbose logging to `syslog(3)`.
- `conf=/path/to/file.conf` - indicate an alternative *limits* configuration file to the default.

Examples/suggested usage:

In order to use this module the system administrator must first create a *root-only-readable* file (default is `/etc/security/limits.conf`). This file describes the resource limits the superuser wishes to impose on users and groups. No limits are imposed on `uid=0` accounts.

Each line of the configuration file describes a limit for a user in the form:

```
<domain>        <type>  <item>        <value>
```

The fields listed above should be filled as follows...
`<domain>` can be:

- a username
- a groupname, with `@group` syntax
- the wild-card `*`, for default entry

`<type>` can have the two values:

- `hard` for enforcing *hard* resource limits. These limits are set by the superuser and enforced by the Linux Kernel. The user cannot raise his requirement of system resources above such values.
- `soft` for enforcing *soft* resource limits. These limits are ones that the user can move up or down within the permitted range by any pre-exisiting *hard* limits. The values specified with this token can be thought of as *default* values, for normal system usage.

`<item>` can be one of the following:

- `core` - limits the core file size (KB)
- `data` - max data size (KB)
- `fsize` - maximum filesize (KB)
- `memlock` - max locked-in-memory address space (KB)
- `nofile` - max number of open files
- `rss` - max resident set size (KB)
- `stack` - max stack size (KB)
- `cpu` - max CPU time (MIN)
- `nproc` - max number of processes
- `as` - address space limit
- `maxlogins` - max number of logins for this user.
- `priority` - the priority to run user process with
- `chroot` - directory to chroot user to

To completely disable limits for a user (or a group), a single dash (-) will do (Example: "`bin -`", "`@admin -`"). Please remember that individual limits have priority over group limits, so if you impose no limits for `admin` group, but one of the members in this group have a limits line, the user will have its limits set according to this line.

Also, please note that all limit settings are set *per login*. They are not global, nor are they permanent; existing only for the duration of the session.

In the *limits* configuration file, the "`#`" character introduces a comment - after which the rest of the line is ignored.

The `pam_limits` module does its best to report configuration problems found in its configuration file via `syslog(3)`.

The following is an example configuration file:

```
# EXAMPLE /etc/security/limits.conf file:
# =======================================
# <domain>      <type>  <item>          <value>
*               soft    core            0
*               hard    rss             10000
@student        hard    nproc           20
@faculty        soft    nproc           20
@faculty        hard    nproc           50
ftp             hard    nproc           0
ftp             -       chroot          /ftp
@student        -       maxlogins       4
```

Note, the use of `soft` and `hard` limits for the same resource (see `@faculty`) – this establishes the *default* and permitted *extreme* level of resources that the user can can obtain in a given service-session.

For the services that need resources limits (login for example) put a the following line in `/etc/pam.conf` as the last line for that service (usually after the pam_unix session line:

```
#
# Resource limits imposed on login sessions via pam_limits
#
login   session    required     pam_limits.so
```

6.13 The list-file module

6.13.1 Synopsis

Module Name:

`pam_listfile`

Author:

Elliot Lee <`sopwith@cuc.edu`>

Maintainer:

Red Hat Software:
Michael K. Johnson <johnsonm@redhat.com> 1996/11/18
(if unavailable, contact Elliot Lee <sopwith@cuc.edu>).

Management groups provided:

authentication

Cryptographically sensitive:

Security rating:

Clean code base:

clean

System dependencies:

Network aware:

6.13.2 Overview of module

The list-file module provides a way to deny or allow services based on an arbitrary file.

6.13.3 Authentication component

Recognized arguments:

`onerr=succeed|fail`; `sense=allow|deny`; `file=`*filename*; `item=user|tty|rhost|ruser|group|shell` `apply=user|@group` `quiet`

Description:

The module gets the item of the type specified – `user` specifies the username, `PAM_USER`; tty specifies the name of the terminal over which the request has been made, `PAM_TTY`; rhost specifies the name of the remote host (if any) from which the request was made, `PAM_RHOST`; and ruser specifies the name of the remote user (if available) who made the request, `PAM_RUSER` – and looks for an instance of that item in the file *filename*. *filename* contains one line per item listed. If the item is found, then if `sense=allow`, `PAM_SUCCESS` is returned, causing the authorization request to succeed; else if `sense=deny`, `PAM_AUTH_ERR` is returned, causing the authorization request to fail.

If an error is encountered (for instance, if *filename* does not exist, or a poorly-constructed argument is encountered), then if `onerr=succeed`, `PAM_SUCCESS` is returned, otherwise if `onerr=fail`, `PAM_AUTH_ERR` or `PAM_SERVICE_ERR` (as appropriate) will be returned.

An additional argument, `apply=`, can be used to restrict the application of the above to a specific user (`apply=`*username*) or a given group (`apply=@`*groupname*). This added restriction is only meaningful when used with the `tty`, `rhost` and `shell` *items*.

Besides this last one, all arguments should be specified; do not count on any default behavior, as it is subject to change. The one exception is the `quiet` options, which reduces the amount of logging for non-fatal errors.

No credentials are awarded by this module.

Examples/suggested usage:

Classic "ftpusers" authentication can be implemented with this entry in `/etc/pam.conf`:

```
#
# deny ftp-access to users listed in the /etc/ftpusers file
#
ftp     auth     required       pam_listfile.so \
        onerr=succeed item=user sense=deny file=/etc/ftpusers
```

Note, users listed in `/etc/ftpusers` file are (counterintuitively) **not** allowed access to the ftp service.

To allow login access only for certain users, you can use an pam.conf entry like this:

```
#
# permit login to users listed in /etc/loginusers
#
login   auth     required       pam_listfile.so \
        onerr=fail item=user sense=allow file=/etc/loginusers
```

For this example to work, all users who are allowed to use the login service should be listed in the file `/etc/loginusers`. Unless you are explicitly trying to lock out root, make sure that when you do this, you leave a way for root to log in, either by listing root in `/etc/loginusers`, or by listing a user who is able to *su* to the root account.

6.14 The mail module

6.14.1 Synopsis

Module Name:

`pam_mail`

Author:

Andrew G. Morgan <morgan@linux.kernel.org>

Maintainer:

Author

Management groups provided:

Authentication (credential) Session (open)

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Default mail directory `/var/spool/mail/`

Network aware:

6.14.2 Overview of module

This module looks at the user's mail directory and indicates whether the user has any mail in it.

6.14.3 Session component

Recognized arguments:

`debug`; `dir=`*direcory-name*; `nopen`; `close`; `noenv`; `empty`; `hash=`*hashcount*; `standard`; `quiet`;

Description:

This module provides the "you have new mail" service to the user. It can be plugged into any application that has credential hooks. It gives a single message indicating the *newness* of any mail it finds in the user's mail folder. This module also sets the **Linux-PAM** environment variable, `MAIL`, to the user's mail directory.

The behavior of this module can be modified with one of the following flags:

- `debug` - write more information to `syslog(3)`.
- `dir=`*pathname* - look for the users' mail in an alternative directory given by *pathname*. The default location for mail is `/var/spool/mail`. Note, if the supplied *pathname* is prefixed by a '~', the directory is interpreted as indicating a file in the user's home directory.
- `nopen` - instruct the module to *not* print any mail information when the user's credentials are acquired. This flag is useful to get the `MAIL` environment variable set, but to not display any information about it.
- `close` - instruct the module to indicate if the user has any mail at the as the user's credentials are revoked.
- `noenv` - do not set the `MAIL` environment variable.
- `empty` - indicate that the user's mail directory is empty if this is found to be the case.
- `hash=`*hashcount* - mail directory hash depth. For example, a *hashcount* of 2 would make the mailfile be `/var/spool/mail/u/s/user`.
- `standard` - old style "You have..." format which doesn't show the mail spool being used. this also implies "empty"
- `quiet` - only report when there is new mail.

Examples/suggested usage:

This module can be used to indicate that the user has new mail when they *login* to the system. Here is a sample entry for your `/etc/pam.conf` file:

```
#
# do we have any mail?
#
login   session  optional       pam_mail.so
```

Note, some applications may perform this function themselves. In such cases, this module is not necessary.

6.14.4 Authentication compent

Then authentication companent works the same as the session component, expect that everything is done during the pam_setcred() phase.

6.15 Create home directories on initial login

6.15.1 Synopsis

Module Name:

`pam_mkhomedir`

Author:

Jason Gunthorpe <jgg@ualberta.ca>

Maintainer:

Ben Collins <bcollins@debian.org>

Management groups provided:

Session

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Network aware:

6.15.2 Overview of module

Creates home directories on the fly for authenticated users.

6.15.3 Session component

Recognized arguments:

`debug`; `skel=skeleton-dir`; `umask=octal-umask`;

Description:

This module is useful for distributed systems where the user account is managed in a central database (such as NIS, NIS+, or LDAP) and accessed through miltiple systems. It frees the administrator from having to create a default home directory on each of the systems by creating it upon the first succesfully authenticated login of that user. The skeleton directory (usually /etc/skel/) is used to copy default files and also set's a umask for the creation.

The behavior of this module can be modified with one of the following flags:

- `skel` - The skeleton directory for default files to copy to the new home directory.
- `umask` - An octal for of the same format as you would pass to the shells umask command.

Examples/suggested usage:

session required pam_mkhomedir.so skel=/etc/skel/ umask=0022

6.16 Output the motd file

6.16.1 Synopsis

Module Name:

`pam_motd`

Author:

Ben Collins <bcollins@debian.org>

Maintainer:

Author

Management groups provided:

Session (open)

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Network aware:

6.16.2 Overview of module

This module outputs the motd file (*/etc/motd* by default) upon succesful login.

6.16.3 Session component

Recognized arguments:

`debug`; `motd=motd-file-name`;

Description:

This module allows you to have arbitrary motd's (message of the day) output after a succesful login. By default this file is */etc/motd*, but is configurable to any file.

The behavior of this module can be modified with one of the following flags:

- `motd` - the file to output if not using the default.

Examples/suggested usage:

login session pam_motd.so motd=/etc/motd

6.17 The no-login module

6.17.1 Synopsis

Module Name:

`pam_nologin`

Author:

Written by Michael K. Johnson <johnsonm@redhat.com>
(based on code taken from a module written by Andrew G. Morgan <morgan@parc.power.net>).

Maintainer:

Michael K. Johnson <johnsonm@redhat.com>

Management groups provided:

authentication

Cryptographically sensitive:

Security rating:

Clean code base:

1 warning about dropping const

System dependencies:

Network aware:

6.17.2 Overview of module

Provides standard Unix *nologin* authentication.

6.17.3 Authentication component

Recognized arguments:

Description:

Provides standard Unix *nologin* authentication. If the file `/etc/nologin` exists, only root is allowed to log in; other users are turned away with an error message. All users (root or otherwise) are shown the contents of `/etc/nologin`.

If the file `/etc/nologin` does not exist, this module succeeds silently.

Examples/suggested usage:

In order to make this module effective, all login methods should be secured by it. It should be used as a `required` method listed before any `sufficient` methods in order to get standard Unix nologin semantics.

6.18 The promiscuous module

6.18.1 Synopsis

Module Name:

pam_permit

Author:

Andrew G. Morgan, <morgan@parc.power.net>

Maintainer:

Linux-PAM maintainer.

Management groups provided:

account; authentication; password; session

Cryptographically sensitive:

Security rating:

VERY LOW. Use with extreme caution.

Clean code base:

Clean.

System dependencies:

Network aware:

6.18.2 Overview of module

This module is very dangerous. It should be used with extreme caution. Its action is always to permit access. It does nothing else.

6.18.3 Account+Authentication+Password+Session components

Recognized arguments:

Description:

No matter what management group, the action of this module is to simply return `PAM_SUCCESS` – operation successful.

In the case of authentication, the user's name will be acquired. Many applications become confused if this name is unknown.

Examples/suggested usage:

It is seldom a good idea to use this module. However, it does have some legitimate uses. For example, if the system-administrator wishes to turn off the account management on a workstation, and at the same time continue to allow logins, then she might use the following configuration file entry for login:

```
#
# add this line to your other login entries to disable account
# management, but continue to permit users to log in...
#
login   account  required       pam_permit.so
```

6.19 The rhosts module

6.19.1 Synopsis

Module Name:

`pam_rhosts_auth`

Author:

Al Longyear <longyear@netcom.com>

Maintainer:

Management groups provided:

authentication

Cryptographically sensitive:

Security rating:

Clean code base:

Clean.

System dependencies:

Network aware:

Standard `inet_addr()`, `gethostbyname()` function calls.

6.19.2 Overview of module

This module performs the standard network authentication for services, as used by traditional implementations of *rlogin* and *rsh* etc.

6.19.3 Authentication component

Recognized arguments:

`no_hosts_equiv`; `no_rhosts`; `debug`; `no_warn`; `privategroup`; `promiscuous`; `suppress`

Description:

The authentication mechanism of this module is based on the contents of two files; `/etc/hosts.equiv` (or `_PATH_HEQUIV` in `#include <netdb.h>`) and `~/.rhosts`. Firstly, hosts listed in the former file are treated as equivalent to the localhost. Secondly, entries in the user's own copy of the latter file is used to map "`remote-host remote-user`" pairs to that user's account on the current host. Access is granted to the user if their host is present in `/etc/hosts.equiv` and their remote account is identical to their local one, or if their remote account has an entry in their personal configuration file.

Some restrictions are applied to the attributes of the user's personal configuration file: it must be a regular file (as defined by `S_ISREG(x)` of POSIX.1); it must be owned by the *superuser* or the user; it must not be writable by any user besides its owner.

The module authenticates a remote user (internally specified by the item `PAM_RUSER`) connecting from the remote host (internally specified by the item `PAM_RHOST`). Accordingly, for applications to be compatible this authentication module they must set these items prior to calling `pam_authenticate()`. The module is not capable of independently probing the network connection for such information.

In the case of `root`-access, the `/etc/host.equiv` file is *ignored* unless the `hosts_equiv_rootok` option should be used. Instead, the superuser must have a correctly configured personal configuration file.

The behavior of the module is modified by flags:

- `debug` - log more information to `syslog(3)`. (XXX - actually, this module does not do any logging currently, please volunteer to fix this!)
- `no_warn` - do not give verbal warnings to the user about failures etc. (XXX - this module currently does not issue any warnings, please volunteer to fix this!)

- `no_hosts_equiv` - ignore the contents of the `/etc/hosts.equiv` file.
- `hosts_equiv_rootok` - allow the use of `/etc/hosts.equiv` for superuser. Without this option `/etc/hosts.equiv` is not consulted for the superuser account. This option has no effect if the `no_hosts_equiv` option is used.
- `no_rhosts` - ignore the contents of all user's personal configuration file `~/.rhosts`.
- `privategroup` - normally, the `~/.rhosts` file must not be writable by anyone other than its owner. This option overlooks group write access in the case that the group owner of this file has the same name as the user being authenticated. To lessen the security problems associated with this option, the module also checks that the user is the only member of their private group.
- `promiscuous` - A host entry of '+' will lead to all hosts being granted access. Without this option, '+' entries will be ignored. Note, that the `debug` option will syslog a warning in this latter case.
- `suppress` - This will prevent the module from `syslog(3)`ing a warning message when this authentication fails. This option is mostly for keeping logs free of meaningless errors, in particular when the module is used with the `sufficient` control flag.

Examples/suggested usage:

To allow users to login from trusted remote machines, you should try adding the following line to your `/etc/pam.conf` file *before* the line that would otherwise prompt the user for a password:

```
#
# No passwords required for users from hosts listed above.
#
login  auth  sufficient  pam_rhosts_auth.so no_rhosts
```

Note, in this example, the system administrator has turned off all *personal rhosts* configuration files. Also note, that this module can be used to *only* allow remote login from hosts specified in the `/etc/host.equiv` file, by replacing `sufficient` in the above example with `required`.

6.20 The root access module

6.20.1 Synopsis

Module Name:

pam_rootok

Author:

Andrew G. Morgan <morgan@parc.power.net>

Maintainer:

Linux-PAM maintainer

Management groups provided:

authentication

Cryptographically sensitive:

Security rating:

Clean code base:

Clean.

System dependencies:

Network aware:

6.20.2 Overview of module

This module is for use in situations where the superuser wishes to gain access to a service without having to enter a password.

6.20.3 Authentication component

Recognized arguments:

`debug`

Description:

This module authenticates the user if their `uid` is `0`. Applications that are created *setuid*-root generally retain the `uid` of the user but run with the authority of an enhanced *effective*-`uid`. It is the real `uid` that is checked.

Examples/suggested usage:

In the case of the `su` application the historical usage is to permit the superuser to adopt the identity of a lesser user without the use of a password. To obtain this behavior under `Linux-PAM` the following pair of lines are needed for the corresponding entry in the configuration file:

```
#
# su authentication. Root is granted access by default.
#
su      auth     sufficient     pam_rootok.so
su      auth     required       pam_unix_auth.so
```

Note. For programs that are run by the superuser (or started when the system boots) this module should not be used to authenticate users.

6.21 The securetty module

6.21.1 Synopsis

Module Name:

`pam_securetty`

Author[s]:

Elliot Lee <sopwith@cuc.edu>

Maintainer:

Red Hat Software:
currently Michael K. Johnson <johnsonm@redhat.com>
(if unavailable, contact Elliot Lee <sopwith@cuc.edu>).

Management groups provided:

authentication

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

`/etc/securetty` file

Network aware:

Requires the application to fill in the `PAM_TTY` item correctly in order to act meaningfully.

6.21.2 Overview of module

Provides standard Unix securetty checking.

6.21.3 Authentication component

Recognized arguments:

Description:

Provides standard Unix securetty checking, which causes authentication for root to fail unless `PAM_TTY` is set to a string listed in the `/etc/securetty` file. For all other users, it succeeds.

Examples/suggested usage:

For canonical usage, should be listed as a `required` authentication method before any `sufficient` authentication methods.

6.22 Time control

6.22.1 Synopsis

Module Name:

`pam_time`

Author:

Andrew G. Morgan <`morgan@parc.power.net`>

Maintainer:

Author

Management groups provided:

account

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Requires a configuration file `/etc/security/time.conf`

Network aware:

Through the `PAM_TTY` item only

6.22.2 Overview of module

Running a well regulated system occasionally involves restricting access to certain services in a selective manner. This module offers some time control for access to services offered by a system. Its actions are determined with a configuration file. This module can be configured to deny access to (individual) users based on their name, the time of day, the day of week, the service they are applying for and their terminal from which they are making their request.

6.22.3 Account component

Recognized arguments:

Description:

This module bases its actions on the rules listed in its configuration file: `/etc/security/pam.conf`. Each rule has the following form,

services;*ttys*;*users*;*times*

In words, each rule occupies a line, terminated with a newline or the beginning of a comment; a '`#`'. It contains four fields separated with semicolons, '`;`'. The fields are as follows:

- *services* - a logic list of service names that are affected by this rule.
- *ttys* - a logic list of terminal names indicating those terminals covered by the rule.
- *user* - a logic list of usernames to which this rule applies

  By a logic list we mean a sequence of tokens (associated with the appropriate `PAM_` item), containing no more than one wildcard character; '`*`', and optionally prefixed with a negation operator; '`!`'. Such a sequence is concatenated with one of two logical operators: `&` (logical AND) and | (logical OR). Two examples are: `!morgan&!root`, indicating that this rule does not apply to the user `morgan` nor to `root`; and `tty*&!ttyp*`, which indicates that the rule applies only to console terminals but not pseudoterminals.

- *times* - a logic list of times at which this rule applies. The format of each element is a day/time-range. The days are specified by a sequence of two character entries. For example, `MoTuSa`, indicates Monday Tuesday and Saturday. Note that repeated days are *unset*; `MoTuMo` indicates Tuesday, and `MoWk` means all weekdays bar Monday. The two character combinations accepted are,

  ```
  Mo Tu We Th Fr Sa Su Wk Wd Al
  ```

  The last two of these being *weekend* days and *all 7 days* of the week respectively.

  The time range part is a pair of 24-hour times, *HHMM*, separated by a hyphen – indicating the start and finish time for the rule. If the finsish time is smaller than the start time, it is assumed to apply on the following day. For an example, `Mo1800-0300` indicates that the permitted times are Monday night from 6pm to 3am the following morning.

Note, that the given time restriction is only applied when the first three fields are satisfied by a user's application for service.

For convenience and readability a rule can be extended beyond a single line with a '*newline*'.

Examples/suggested usage:

The use of this module is initiated with an entry in the **Linux-PAM** configuration file of the following type:

```
#
# apply pam_time accounting to login requests
#
login   account  required       pam_time.so
```

where, here we are applying the module to the *login* application.

Some examples of rules that can be placed in the `/etc/security/time.conf` configuration file are the following:

`login ; tty* & ; !ttyp* ; !root ; !Al0000-2400`

all users except for `root` are denied access to console-login at all times.

`games ; * ; !waster ; Wd0000-2400 | Wk1800-0800`

games (configured to use Linux-PAM) are only to be accessed out of working hours. This rule does not apply to the user `waster`.

Note, currently there is no daemon enforcing the end of a session. This needs to be remedied.

Poorly formatted rules are logged as errors using `syslog(3)`.

6.23 The Unix Password module

6.23.1 Synopsis

Module Name:

pam_unix

Author:

Maintainer:

Authors.

Management groups provided:

account; authentication; password; session

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Network aware:

6.23.2 Overview of module

This is the standard Unix authentication module. It uses standard calls from the system's libraries to retrieve and set account information as well as authentication. Usually this is obtained from the /etc/passwd and the /etc/shadow file aswell if shadow is enabled.

6.23.3 Account component

Recognized arguments:

`debug`; `audit`

Description:

The `debug` argument makes the accounting functions of this module `syslog(3)` more information on its actions. (Remaining arguments supported by the other functions of this module are silently ignored, but others are logged as errors through `syslog(3)`). The `audit` argument causes even more logging.

Based on the following `shadow` elements: `expire`; `last_change`; `max_change`; `min_change`; `warn_change`, this module performs the task of establishing the status of the user's account and password. In the case of the latter, it may offer advice to the user on changing their password or, through the `PAM_AUTHTOKEN_REQD` return, delay giving service to the user until they have established a new password. The entries listed above are documented in the *GNU Libc* info documents. Should the user's record not contain one or more of these entries, the corresponding *shadow* check is not performed.

Examples/suggested usage:

In its accounting mode, this module can be inserted as follows:

```
#
# Ensure users account and password are still active
#
login   account  required       pam_unix.so
```

6.23.4 Authentication component

Recognized arguments:

debug; audit; use_first_pass; try_first_pass; nullok; nodelay

Description:

The debug argument makes the authentication functions of this module syslog(3) more information on its actions. The audit causes even more information to be logged.

The default action of this module is to not permit the user access to a service if their *official* password is blank. The nullok argument overrides this default.

When given the argument try_first_pass, before prompting the user for their password, the module first tries the previous stacked auth-module's password in case that satisfies this module as well. The argument use_first_pass forces the module to use such a recalled password and will never prompt the user - if no password is available or the password is not appropriate, the user will be denied access.

The argument, nodelay, can be used to discourage the authentication component from requesting a delay should the authentication as a whole fail. The default action is for the module to request a delay-on-failure of the order of one second.

Remaining arguments, supported by the other functions of this module, are silently ignored. Other arguments are logged as errors through syslog(3).

A helper binary, unix_chkpwd, is provided to check the user's password when it is stored in a read protected database. This binary is very simple and will only check the password of the user invoking it. It is called transparently on behalf of the user by the authenticating component of this module. In this way it is possible for applications like *xlock* to work without being setuid-root.

Examples/suggested usage:

The correct functionality of this module is dictated by having an appropriate /etc/nsswitch.conf file, the user databases specified there dictate the source of the authenticated user's record.

In its authentication mode, this module can be inserted as follows:

```
#
# Authenticate the user
#
login   auth  required       pam_unix.so
```

6.23.5 Password component

Recognized arguments:

debug; audit; nullok; not_set_pass; use_authtok; try_first_pass; use_first_pass; md5; bigcrypt; shadow; nis; min; max; obscure; remember

Description:

This part of the pam_unix module performs the task of updating the user's password.

In the case of conventional unix databases (which store the password encrypted) the md5 argument is used to do the encryption with the MD5 function as opposed to the *conventional* crypt(3) call. As an alternative to this, the bigcrypt argument can be used to encrypt more than the first 8 characters of a password with DEC's (Digital Equipment Cooperation) 'C2' extension to the standard UNIX `crypt()` algorithm.

The `nullok` argument is used to permit the changing of a password *from* an empty one. Without this argument, empty passwords are treated as account-locking ones.

The argument `use_first_pass` is used to lock the choice of old and new passwords to that dictated by the previously stacked `password` module. The `try_first_pass` argument is used to avoid the user having to re-enter an old password when `pam_unix` follows a module that possibly shared the user's old password - if this old password is not correct the user will be prompted for the correct one. The argument `use_authtok` is used to *force* this module to set the new password to the one provided by the previously stacked `password` module (this is used in an example of the stacking of the *Cracklib* module documented above).

The `not_set_pass` argument is used to inform the module that it is not to pay attention to/make available the old or new passwords from/to other (stacked) password modules.

The `debug` argument makes the password functions of this module `syslog(3)` more information on its actions. Other arguments may be logged as erroneous to `syslog(3)`. The `audit` argument causes even more information to be logged.

With the `nis` argument, `pam_unix` will attempt to use NIS RPC for setting new passwords.

The `remember` argument takes one value. This is the number of most recent passwords to save for each user. These are saved in `/etc/security/opasswd` in order to force password change history and keep the user from alternating between the same password too frequently.

The `min` and `max` options allow control over the length of the password. These have a hard coded default of 1 and 8. The values are inclusive.

The `obscure` option enables some extra checks on the password. These is taken after the same obscure checks enabled in the original shadow package. This works very similar to the pam_cracklib module and implements these checks (it does not implement dictionary checks):

- **Palindrome** -

  Is the new password a palindrome of the old one? A palindrome is where the words read the same backwards and forwards (eg. madam and radar).

- **Case Change Only** -

  Is the new password the the old one with only a change of case?

- **Similar** -

  Is the new password too much like the old one?

- **Simple** -

  Is the new password too small? This is based on the length of the password and the number of different types of characters used (ie.alpha, numeric...).

- **Rotated** -

  Is the new password a rotated version of the old password (eg. "billy" and "illyb")?

Examples/suggested usage:

Standard usage:

```
#
# Change the users password
#
passwd   password   required   pam_unix.so
```

An example of the stacking of this module with respect to the pluggable password checking module, pam_cracklib:

```
#
# Change the users password
#
passwd   password   required   pam_cracklib.so retry=3 minlen=6 difok=3
passwd   password   required   pam_unix.so use_authtok nullok md5
```

6.23.6 Session component

Recognized arguments:

Description:

No arguments are recognized by this module component. Its action is simply to log the username and the service-type to syslog(3). Messages are logged at the beginning and end of the user's session.

Examples/suggested usage:

The use of the session modules is straightforward:

```
#
# session opening and closing
#
login   session  required       pam_unix.so
```

6.24 The userdb module

6.24.1 Synopsis

Module Name:

pam_userdb

Author:

Cristian Gafton <gafton@redhat.com>

Maintainer:

Author.

Management groups provided:

authentication

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Requires Berkeley DB.

Network aware:

6.24.2 Overview of module

Look up users in a .db database and verify their password against what is contained in that database.

6.24.3 Authentication component

Recognized arguments:

`debug`; `icase`; `dump`; `db=XXXX`;

Description:

This module is used to verify a username/password pair against values stored in a Berkeley DB database. The database is indexed by the username, and the data fields corresponding to the username keys are the passwords, in unencrypted form, so caution must be exercised over the access rights to the DB database itself..

The module will read the password from the user using the conversation mechanism. If you are using this module on top of another authetication module (like `pam_unix`;) then you should tell that module to read the entered password from the PAM_AUTHTOK field, which is set by this module.

The action of the module may be modified from this default by one or more of the following flags in the `/etc/pam.d/<service>` file.

- `debug` - Supply more debugging information to `syslog(3)`.
- `icase` - Perform the password comparisons case insensitive.
- `dump` - dump all the entries in the database to the log (eek, don't do this by default!)
- `db=XXXX` - use the database found on pathname XXXX. Note that Berkeley DB usually adds the needed filename extension for you, so you should use something like `/etc/foodata` instead of `/etc/foodata.db`.

Examples/suggested usage:

This is a normal ftp configuration file (usually placed as `/etc/pam.d/ftp` on most systems) that will accept for login users whose username/password pairs are provided in the `/tmp/dbtest.db` file:

```
#%PAM-1.0
auth       required     pam_listfile.so item=user sense=deny file=/etc/ftpusers onerr=succeed
auth       sufficient   pam_userdb.so icase db=/tmp/dbtest
auth       required     pam_unix.so shadow nullok try_first_pass
auth       required     pam_shells.so
account    required     pam_unix.so
session    required     pam_unix.so
```

6.25 Warning logger module

6.25.1 Synopsis

Module Name:

pam_warn

Author:

Andrew G. Morgan <morgan@parc.power.net>

Maintainer:

Author.

Management groups provided:

authentication; password

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Network aware:

logs information about the remote user and host (if pam-items are known)

6.25.2 Overview of module

This module is principally for logging information about a proposed authentication or application to update a password.

6.25.3 Authentication+Password component

Recognized arguments:

Description:

Log the service, terminal, user, remote user and remote host to `syslog(3)`. The items are not probed for, but instead obtained from the standard pam-items.

Examples/suggested usage:

an example is provided in the configuration file section 4 (above).

6.26 The wheel module

6.26.1 Synopsis

Module Name:

`pam_wheel`

Author:

Cristian Gafton <gafton@redhat.com>

Maintainer:

Author.

Management groups provided:

authentication

Cryptographically sensitive:

Security rating:

Clean code base:

System dependencies:

Network aware:

6.26.2 Overview of module

Only permit root access to members of the wheel (`gid=0`) group.

6.26.3 Authentication component

Recognized arguments:

`debug`; `trust`; `deny`; `group=XXXX`

Description:

This module is used to enforce the so-called *wheel* group. By default, it permits root access to the system if the applicant user is a member of the `wheel` group (first, the module checks for the existence of a '`wheel`' group. Otherwise the module defines the group with group-id `0` to be the *wheel* group).

The action of the module may be modified from this default by one or more of the following flags in the `/etc/pam.conf` file.

- `debug` - Supply more debugging information to `syslog(3)`.
- `trust` - This option instructs the module to return `PAM_SUCCESS` should it find the user applying for root privilege is a member of the wheel group. The default action is to return `PAM_IGNORE` in this situation. By using the `trust` option it is possible to arrange for `wheel`-group members to become root without typing a password. **USE WITH CARE**.

- `deny` - This is used to reverse the logic of the module's behavior. If the user is trying to get `uid=0` access and is a member of the wheel group, deny access (for the wheel group, this is perhaps nonsense!): it is intended for use in conjunction with the `group=` argument...
- `group=XXXX` - Instead of checking the `gid=0` group, use the user's `XXXX` group membership for the authentication. Here, `XXXX` is the name of the group and **not** its numeric identifier.

Examples/suggested usage:

To restrict access to superuser status to the members of the `wheel` group, use the following entries in your configuration file:

```
#
# root gains access by default (rootok), only wheel members can
# become root (wheel) but Unix authenticate non-root applicants.
#
su      auth     sufficient     pam_rootok.so
su      auth     required       pam_wheel.so
su      auth     required       pam_unix_auth.so
```

NOTE: In this context, being a member of a group means that the user is listed as a member in the /etc/groups file. The pam-wheel module does not check the users primary group (the one listed in their /etc/passwd entry).

7 Files

`/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.

8 See also

- The **Linux-PAM** Application Writers' 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. See this url:

  http://www.pilgrim.umass.edu/pub/osf_dce/RFC/rfc86.0.txt

9 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! :)

Are we going to be able to support the `use_mapped_pass` module argument? Anyone know a cheap (free) good lawyer?!

- This issue may go away, as Sun have investigated adding a new management group for mappings. In this way, libpam would have mapping modules that could securely store passwords using strong cryptography and in such a way that they need not be distributed with Linux-PAM.

10 Author/acknowledgments

This document was written by Andrew G. Morgan (morgan@linux.kernel.org) with many contributions from Chris Adams, Peter Allgeyer, Tim Baverstock, Tim Berger, Craig S. Bell, Derrick J. Brashear, Ben Buxton, Seth Chaiklin, Oliver Crow, Chris Dent, Marc Ewing, Cristian Gafton, Emmanuel Galanos, Brad M. Garcia, Eric Hester, Roger Hu, Eric Jacksch, Michael K. Johnson, David Kinchlea, Olaf Kirch, Marcin Korzonek, Stephen Langasek, Nicolai Langfeldt, Elliot Lee, Luke Kenneth Casson Leighton, Al Longyear, Ingo Luetkebohle, Marek Michalkiewicz, Robert Milkowski, Aleph One, Martin Pool, Sean Reifschneider, Jan Rekorajski, Erik Troan, Theodore Ts'o, Jeff Uphoff, Myles Uyema, Savochkin Andrey Vladimirovich, Ronald Wahl, David Wood, John Wilmes, 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**.

11 Bugs/omissions

More PAM modules are being developed all the time. It is unlikely that this document will ever be truely up to date!

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

12 Copyright information for this document

`$Id: pam_source.sgml,v 1.9 1999/11/08 05:09:17 morgan Exp $`