

**NAME**

Linux-PAM – Pluggable Authentication Modules for Linux

**SYNOPSIS**

`/etc/pam.conf`

**DESCRIPTION**

This manual is intended to offer a quick introduction to **Linux-PAM**. For more information the reader is directed to the **Linux-PAM system administrators' guide**.

**Linux-PAM** is a system of libraries that handle the authentication tasks of applications (services) on the system. The library provides a stable general interface (Application Programming Interface - API) that privilege granting programs (such as **login**(1) and **su**(1)) defer to perform standard authentication tasks.

The principal feature of the PAM approach is that the nature of the authentication is dynamically configurable. In other words, the system administrator is free to choose how individual service-providing applications will authenticate users. This dynamic configuration is set by the contents of the single **Linux-PAM** configuration file `/etc/pam.conf`. Alternatively, the configuration can be set by individual configuration files located in the `/etc/pam.d/` directory. *The presence of this directory will cause **Linux-PAM** to ignore `/etc/pam.conf`.*

From the point of view of the system administrator, for whom this manual is provided, it is not of primary importance to understand the internal behavior of the **Linux-PAM** library. The important point to recognize is that the configuration file(s) *define* the connection between applications (**services**) and the pluggable authentication modules (**PAMs**) that perform the actual authentication tasks.

**Linux-PAM** separates the tasks of *authentication* into four independent management groups: **account** management; **authentication** management; **password** management; and **session** management. (We highlight the abbreviations used for these groups in the configuration file.)

Simply put, these groups take care of different aspects of a typical user's request for a restricted service:

**account** - provide account verification types of service: has the user's password expired?; is this user permitted access to the requested service?

**authentication** - establish the user is who they claim to be. Typically this is via some challenge-response request that the user must satisfy: if you are who you claim to be please enter your password. Not all authentications are of this type, there exist hardware based authentication schemes (such as the use of smart-cards and biometric devices), with suitable modules, these may be substituted seamlessly for more standard approaches to authentication - such is the flexibility of **Linux-PAM**.

**password** - this group's responsibility is the task of updating authentication mechanisms. Typically, such services are strongly coupled to those of the **auth** group. Some authentication mechanisms lend themselves well to being updated with such a function. Standard UN\*X password-based access is the obvious example: please enter a replacement password.

**session** - this group of tasks cover things that should be done prior to a service being given and after it is withdrawn. Such tasks include the maintenance of audit trails and the mounting of the user's home

directory. The **session** management group is important as it provides both an opening and closing hook for modules to affect the services available to a user.

### The configuration file(s)

When a **Linux-PAM** aware privilege granting application is started, it activates its attachment to the PAM-API. This activation performs a number of tasks, the most important being the reading of the configuration file(s): **/etc/pam.conf**. Alternatively, this may be the contents of the **/etc/pam.d/** directory.

These files list the **PAMs** that will do the authentication tasks required by this service, and the appropriate behavior of the PAM-API in the event that individual **PAMs** fail.

The syntax of the **/etc/pam.conf** configuration file is as follows. The file is made up of a list of rules, each rule is typically placed on a single line, but may be extended with an escaped end of line: '`\<LF>`'. Comments are preceded with '#' marks and extend to the next end of line.

The format of each rule is a space separated collection of tokens, the first three being case-insensitive:

**service type control module-path module-arguments**

The syntax of files contained in the **/etc/pam.d/** directory, are identical except for the absence of any *service* field. In this case, the *service* is the name of the file in the **/etc/pam.d/** directory. This filename must be in lower case.

An important feature of **Linux-PAM**, is that a number of rules may be *stacked* to combine the services of a number of PAMs for a given authentication task.

The **service** is typically the familiar name of the corresponding application: **login** and **su** are good examples. The **service-name**, **other**, is reserved for giving *default* rules. Only lines that mention the current service (or in the absence of such, the **other** entries) will be associated with the given service-application.

The **type** is the management group that the rule corresponds to. It is used to specify which of the management groups the subsequent module is to be associated with. Valid entries are: **account**; **auth**; **password**; and **session**. The meaning of each of these tokens was explained above.

The third field, **control**, indicates the behavior of the PAM-API should the module fail to succeed in its authentication task. There are two types of syntax for this control field: the simple one has a single simple keyword; the more complicated one involves a square-bracketed selection of **value=action** pairs.

For the simple (historical) syntax valid **control** values are: **requisite** - failure of such a PAM results in the immediate termination of the authentication process; **required** - failure of such a PAM will ultimately lead to the PAM-API returning failure but only after the remaining *stacked* modules (for this **service** and **type**) have been invoked; **sufficient** - success of such a module is enough to satisfy the authentication requirements of the stack of modules (if a prior **required** module has failed the success of this one is *ignored*); **optional** - the success or failure of this module is only important if it is the only module in the stack associated with this **service+type**.

For the more complicated syntax valid **control** values have the following form:

```
[value1=action1value2=action2...]
```

Where **valueN** corresponds to the return code from the function invoked in the module for which the line is defined. It is selected from one of these: **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**; and **default**. The last of these, **default**, implies 'all **valueN**'s not mentioned explicitly. Note, the full list of PAM errors is available in `/usr/include/security/_pam_types.h`. The **actionN** can be: an unsigned integer, **J**, signifying an action of 'jump over the next J modules in the stack'; or take one of the following forms:

**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.

**module-path** - this is either the full filename of the PAM to be used by the application (it begins with a '/'), or a relative pathname from the default module location: `/lib/security/`.

**module-arguments** - these are a space separated list of tokens that can be used to modify the specific behavior of the given PAM. Such arguments will be documented for each individual module.

## FILES

`/etc/pam.conf` - the configuration file

`/etc/pam.d/` - the **Linux-PAM** configuration directory. Generally, if this directory is present, the `/etc/pam.conf` file is ignored.

`/lib/libpam.so.X` - the dynamic library

`/lib/security/*.so` - the PAMs

## ERRORS

Typically errors generated by the **Linux-PAM** system of libraries, will be written to `syslog(3)`.

## CONFORMING TO

DCE-RFC 86.0, October 1995.

Contains additional features, but remains backwardly compatible with this RFC.

## BUGS

None known.

**SEE ALSO**

The three **Linux-PAM** Guides, for **system administrators**, **module developers**, and **application developers**.