Name

pam.conf, pam.d − PAM configuration files

DESCRIPTION

When a *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. The presence of this directory will cause Linux−PAM to ignore `/etc/pam.conf`.

These files list the *PAM*s that will do the authentication tasks required by this service, and the appropriate behavior of the PAM−API in the event that individual *PAM*s 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 *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
: this module type 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.

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 or other privileges through its credential granting properties.

password
: this 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) type.

session
: this module type 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.

If the *type* value from the list above is prepended with a − character the PAM library will not log to the system log if it is not possible to load the module because it is missing in the system. This can be useful especially for modules which are not always installed on the system and are not required for correct authentication and authorization of the login session.

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:

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.

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
: 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*). A failure of this module is not deemed as fatal to satisfying the application that this type has succeeded. If the module succeeds the PAM framework returns success to the application immediately without trying any other modules.

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*.

include
: include all lines of given type from the configuration file specified as an argument to this control.

substack
: include all lines of given type from the configuration file specified as an argument to this control. This differs from *include* in that evaluation of the *done* and *die* actions in a substack does not cause skipping the rest of the complete module stack, but only of the substack. Jumps in a substack also can not make evaluation jump out of it, and the whole substack is counted as one module when the jump is done in a parent stack. The *reset* action will reset the state of a module stack to the state it was in as of beginning of the substack evaluation.

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

```
      [value1=action1 value2=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*, *conv_again*, *incomplete*, 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, *n*, signifying an action of ´jump over the next *n* 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.

Each of the four keywords: required; requisite; sufficient; and optional, have an equivalent expression in terms of the [...] syntax. They are as follows:

required
: [success=ok new_authtok_reqd=ok ignore=ignore default=bad]

requisite
: [success=ok new_authtok_reqd=ok ignore=ignore default=die]

sufficient
: [success=done new_authtok_reqd=done default=ignore]

optional
: [success=ok new_authtok_reqd=ok default=ignore]

module−path 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/` or `/lib64/security/`, depending on the architecture.

module−arguments 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. Note, if you wish to include spaces in an argument, you should surround that argument with square brackets.

```
squid auth required pam_mysql.so user=passwd_query passwd=mada \
      db=eminence [query=select user_name from internet_service \
      where user_name=´%u´ and password=PASSWORD(´%p´) and \
    service=´web_proxy´]
```

When using this convention, you can include ‘[´ characters inside the string, and if you wish to include a ‘]´ character inside the string that will survive the argument parsing, you should use ‘\]´. In other words:

```
[..[..\]..]    −−>   ..[..]..
```

Any line in (one of) the configuration 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).

More flexible than the single configuration file is it 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.

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:

```
type  control  module-path  module-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.

SEE ALSO

pam(3), **PAM**(8), **pam_start**(3)