

**NAME**

pam\_unix - Module for traditional password authentication

**SYNOPSIS**

**pam\_unix.so** [...]

**DESCRIPTION**

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 as well if shadow is enabled.

The account component performs the task of establishing the status of the user's account and password based on the following *shadow* elements: `expire`, `last_change`, `max_change`, `min_change`, `warn_change`. 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 [shadow\(5\)](#) manual page. Should the user's record not contain one or more of these entries, the corresponding *shadow* check is not performed.

The authentication component performs the task of checking the users credentials (password). The default action of this module is to not permit the user access to a service if their official password is blank.

A helper binary, [unix\\_chkpwd\(8\)](#), 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\(1\)](#) to work without being setuid-root. The module, by default, will temporarily turn off SIGCHLD handling for the duration of execution of the helper binary. This is generally the right thing to do, as many applications are not prepared to handle this signal from a child they didn't know was `fork()`d. The **noreap** module argument can be used to suppress this temporary shielding and may be needed for use with certain applications.

The maximum length of a password supported by the pam\_unix module via the helper binary is `PAM_MAX_RESP_SIZE` - currently 512 bytes. The rest of the password provided by the conversation function to the module will be ignored.

The password component of this module performs the task of updating the user's password. The default encryption hash is taken from the **ENCRYPT\_METHOD** variable from `/etc/login.defs`

The session component of this module logs when a user logs in or leave the system.

Remaining arguments, supported by others functions of this module, are silently ignored. Other arguments are logged as errors through [syslog\(3\)](#).

**OPTIONS****debug**

Turns on debugging via [syslog\(3\)](#).

**audit**

A little more extreme than debug.

**nullok**

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 and allows any user with a blank password to access the service.

**nullok\_secure**

The default action of this module is to not permit the user access to a service if their official password is blank. The **nullok\_secure** argument overrides this default and allows any user with a blank password to access the service as long as the value of `PAM_TTY` is set to one of the values found in `/etc/securitytty`.

**try\_first\_pass**

Before prompting the user for their password, the module first tries the previous stacked module's password in case that satisfies this module as well.

**use\_first\_pass**

The argument **use\_first\_pass** forces the module to use a previous stacked modules password and will never prompt the user - if no password is available or the password is not appropriate, the user will be denied access.

**nodelay**

This argument 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 two second.

**use\_authtok**

When password changing enforce the module to set the new password to the one provided by a previously stacked **password** module (this is used in the example of the stacking of the **pam\_cracklib** module documented below).

**not\_set\_pass**

This 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.

**nis**

NIS RPC is used for setting new passwords.

**remember=*n***

The last *n* passwords for each user 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. Instead of this option the **pam\_pwhistory** module should be used.

**shadow**

Try to maintain a shadow based system.

**md5**

When a user changes their password next, encrypt it with the MD5 algorithm.

**bigcrypt**

When a user changes their password next, encrypt it with the DEC C2 algorithm.

**sha256**

When a user changes their password next, encrypt it with the SHA256 algorithm. If the SHA256 algorithm is not known to the **crypt(3)** function, fall back to MD5.

**sha512**

When a user changes their password next, encrypt it with the SHA512 algorithm. If the SHA512 algorithm is not known to the **crypt(3)** function, fall back to MD5.

**blowfish**

When a user changes their password next, encrypt it with the blowfish algorithm. If the blowfish algorithm is not known to the **crypt(3)** function, fall back to MD5.

**rounds=*n***

Set the optional number of rounds of the SHA256, SHA512 and blowfish password hashing algorithms to *n*.

**broken\_shadow**

Ignore errors reading shadow information for users in the account management module.

**minlen=*n***

Set a minimum password length of *n* characters. The default value is 6. The maximum for DES crypt-based passwords is 8 characters.

**obscure**

Enable some extra checks on password strength. These checks are based on the "obscure" checks in the original shadow package. The behavior is similar to the **pam\_cracklib** module, but for non-dictionary-based checks. The following checks are implemented:

**Palindrome**

Verifies that the new password is not a palindrome of (i.e., the reverse of) the previous one.

**Case Change Only**

Verifies that the new password isn't the same as the old one with a change of case.

**Similar**

Verifies that the new password isn't too much like the previous one.

**Simple**

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

**Rotated**

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

Invalid arguments are logged with [syslog\(3\)](#).

**MODULE TYPES PROVIDED**

All module types (**account**, **auth**, **password** and **session**) are provided.

**RETURN VALUES**

PAM\_IGNORE

Ignore this module.

**EXAMPLES**

An example usage for /etc/pam.d/login would be:

```
# Authenticate the user
auth required pam_unix.so
# Ensure users account and password are still active
account required pam_unix.so
# Change the users password, but at first check the strength
# with pam_cracklib(8)
password required pam_cracklib.so retry=3 minlen=6 difok=3
password required pam_unix.so use_authok nullok md5
session required pam_unix.so
```

**SEE ALSO**

[login.defs\(5\)](#), [pam.conf\(5\)](#), [pam.d\(5\)](#), [pam\(7\)](#)

**AUTHOR**

pam\_unix was written by various people.