Linux-PAM demystified and beyond

Dmitry Levin

Brussels, 2026

- What and Why
- Core concepts
- Configuration files and modules
- Configuration rules syntax
- Configuration rules control values
- Frozen stack
- The setuid helper issue
- Troubleshooting and Best Practices

Framework

- shared libraries: libpam, libpamc, libpam_misc
- Pluggable Authentication Modules: pam_*.so
- configuration: /etc/pam.d/*

Purpose

- Enable system administrators to choose how applications authenticate users.
- Switch between the authentication mechanisms without recompiling applications.

The "Before PAM" Problem

Hardcoded logic

Every application that needed to authenticate a user had its own code to handle /etc/passwd, later also /etc/shadow.

Rigidity

To introduce a new authentication method like Kerberos or OTP, every application had to be modified and recompiled.

Inconsistency

Different applications might implement authentication, password checking, or account lockout rules slightly differently.

Limited control

Enforcing system-wide security policies, e.g. all interactive logins must use 2FA, is difficult.

Authentication: Are you who you say you are?

- Verify the user's identity.
- Grant credentials.

Account management: Are you allowed to use this service right now?

- Check account validity and restrictions.

Session management: What needs to be set up for your session?

- Actions that need to occur before the service is granted.
- Actions that need to occur after the service termination.

Password management: How can you change your authentication token?

- Prompting for a new password.
- Enforcing password quality rules.
- Changing the password.

Are you who you say you are?

- Checking a typed password against */etc/shadow* (e.g. via *pam_unix.so*).
- Validating a U2F dongle (e.g. via *pam_u2f.so*) or a biometric scan (e.g. via *pam_fprintd.so*).
- Querying a remote server for credentials (e.g. via *pam_sss.so*).
- Prompting for a One-Time Password.

Are you **allowed** to use this service **right now**?

- Is the account enabled? Is the password not expired? (*pam_unix.so*)
- Is the account locked due to too many failed login attempts? (*pam_faillock.so*)
- Are there time-based restrictions on when this user can log in? (*pam_time.so*)
- Is the user allowed for this service? (*pam_access.so*)

What needs to be set up for your session?

- Initialize kernel session keyring (*pam_keyinit.so*).
- Set resource limits (*pam_limits.so*).
- Set the file mode creation mask (*pam_umask.so*).
- Register user session in the login manager (*pam_systemd.so*).
- Create a home directory (*pam_mkhomedir.so*).

How can you change your authentication token?

- Prompting for a new password.
- Enforcing password quality rules (*pam_passwdqc.so* or *pam_pwquality.so*).
- Making sure the user does not use the same password too frequently (*pam_pwhistory.so*).
- Changing the password (*pam_unix.so*).

Decoupling

- PAM separates the applications (like *login* or *sshd*) from the underlying authentication, account, session, and password management policies.
- Applications just talk to the PAM library.
- Administrators configure PAM.

Applications talk to the PAM library

type	**API function name**	**description**
auth	pam_authenticate	Authenticate this user
auth	pam_setcred	Manage credentials of this user
account	pam_acct_mgmt	Check account validity and restrictions for this user
password	pam_chauthtok	Change the authentication token for this user
session	pam_open_session	Set up a session for this user
session	pam_close_session	End the session for this user

Service configuration files

- PAM configuration is service-specific.
- Configuration files are stored in **/etc/pam.d/**.
- Service configuration files are named after services (like *login* or *sshd*).
- When a service, e.g. *login*, needs to authenticate a user, it tells the PAM library: I'm the login service, please handle this authentication based on my configuration.
- If a specific service file doesn't exist, or when it doesn't specify a management group, PAM falls back to a default configuration for this management group defined in /etc/pam.d/other, which usually denies access.

Common configuration files

- conventionally stored in **/etc/pam.d/**
- included by service-specific configuration files and other common configuration files
- used to implement system-wide policies

Modules: the workhorses (*pam_*.so*)

- Modules are shared objects loaded dynamically by the PAM library according to the service configuration.
- Typically located in /lib/security/ or /lib64/security/.
- Each module is designed to perform a specific task.
- There are many modules available:
- standard modules packaged along with the PAM library
- other modules provided by other packages

  pam_deny.so always returns failure

  pam_permit.so always returns access, useful as a placeholder

PAM rule: **type control module-path [module-arguments]**

- **type**: the management group that the rule corresponds to
- **control**: determines how the return value of this module affects the overall outcome for the management group
- **module-path**: the filename of the PAM module to be used
- **module-arguments**: optional arguments passed to the module

Example: /etc/pam.d/login (simplified)

```
auth        required     pam_unix.so nullok
account     required     pam_nologin.so
account     required     pam_unix.so
password    requisite    pam_passwdqc.so config=/etc/passwdqc.conf
password    required     pam_unix.so use_authtok shadow nullok
session     required     pam_loginuid.so
session     optional     pam_keyinit.so force revoke
session     required     pam_limits.so
-session    optional     pam_systemd.so
session     required     pam_unix.so
```

required

- Failure will lead to the PAM framework returning failure but only after the remaining stacked modules for this management group have been invoked.

Example: /etc/pam.d/login (simplified)

```
auth        required    pam_unix.so nullok
account     required    pam_nologin.so
account     required    pam_unix.so
password    requisite   pam_passwdqc.so config=/etc/passwdqc.conf
password    requisite   pam_pwhistory.so use_authtok
password    required    pam_unix.so use_authtok shadow nullok
session     required    pam_loginuid.so
session     optional    pam_keyinit.so force revoke
session     required    pam_limits.so
-session    optional    pam_systemd.so
session     required    pam_unix.so
```

requisite

- Like **required**, however, in the case that this module returns a failure, control is directly returned to the application or to the superior PAM stack.

Example: /etc/pam.d/login (simplified)

```
auth        required     pam_unix.so nullok
account     required     pam_nologin.so
account     required     pam_unix.so
password    requisite    pam_passwdqc.so config=/etc/passwdqc.conf
password    requisite    pam_pwhistory.so use_authtok
password    required     pam_unix.so use_authtok shadow nullok
session     required     pam_loginuid.so
session     optional     pam_keyinit.so force revoke
session     required     pam_limits.so
-session    optional     pam_systemd.so
session     required     pam_unix.so
```

sufficient

- If the module succeeds and no prior **required** module has failed, the PAM stack succeeds immediately without calling any further modules in the stack.
- Otherwise, the return value of the module is ignored and processing of the PAM module stack continues unaffected.

Example: /etc/pam.d/su (simplified)

```
auth        sufficient     pam_rootok.so
auth        required       pam_unix.so nullok
account     sufficient     pam_succeed_if.so uid = 0 use_uid quiet
account     required       pam_unix.so
password    requisite      pam_passwdqc.so config=/etc/passwdqc.conf
password    required       pam_unix.so use_authtok shadow nullok
...
```

optional

- The success or failure of this module is only important if it is the only module in the stack associated with this management group.

Example: /etc/pam.d/su (simplified)

```
auth        sufficient    pam_rootok.so
auth        required      pam_unix.so nullok
account     sufficient    pam_succeed_if.so uid = 0 use_uid quiet
account     required      pam_unix.so
password    requisite     pam_passwdqc.so config=/etc/passwdqc.conf
password    required      pam_unix.so use_authtok shadow nullok
session     optional      pam_keyinit.so revoke
session     required      pam_limits.so
-session    optional      pam_systemd.so
session     required      pam_unix.so
session     optional      pam_xauth.so
```

include

- Include all lines of the same type from the configuration file specified as an argument to this control.

Example: /etc/pam.d/su

```
auth        sufficient    pam_rootok.so
auth        required      pam_wheel.so use_uid
auth        substack      system-auth
auth        include       postlogin
account     sufficient    pam_succeed_if.so uid = 0 use_uid quiet
account     include       system-auth
password    include       system-auth
session     include       system-auth
session     include       postlogin
session     optional      pam_xauth.so
```

substack

- Include all lines of the same type from the configuration file specified as an argument to this control.
- **requisite** and **sufficient** 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 jump out of it.
- The whole substack is counted as one module when the jump is done in a parent stack.

Example: /etc/pam.d/su (excerpt)

```
auth            sufficient      pam_rootok.so
auth            substack        system-auth
auth            include         postlogin
...
```

The syntax: [*value1=action1 value2=action2 . . . valueN=actionN*]

valueN corresponds to the return value returned by the module

actionN specifies the action

valueN

- One of predefined PAM 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.
- **default**: all PAM return values not mentioned explicitly.

The syntax: [*value1=action1 value2=action2 . . . valueN=actionN*]

valueN corresponds to the return value returned by the module

actionN specifies the action

actionN

- **ignore**: return value ignored, stack processing continues
- **bad**: module fails, stack processing continues
- **die**: module fails, stack processing terminates
- **ok**: module succeeds, stack processing continues
- **done**: module succeeds; stack processing terminates if no prior **required** module has failed
- **reset**: the stack resets, stack processing continues
- **N (an unsigned integer)**: jump over the next N modules in the stack

The syntax: [*value1=action1 value2=action2 . . . valueN=actionN*]

If a return value is not specifically listed via a *valueN* token, and the **default** value is not specified, the implicit default action for it is **bad**.

Equivalents of traditional 4 control keywords in the advanced syntax

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**]

Why use this?

- Complex logic that traditional controls cannot express.
- Conditional branching.

Example: /etc/pam.d/system-auth (excerpt)

```
...
password  requisite   pam_pwquality.so
password  [success=ok default=1 ignore=ignore] \
                      pam_localuser.so
password  requisite   pam_pwhistory.so use_authtok
password  sufficient  pam_unix.so shadow nullok use_authtok
password  required    pam_deny.so

session   optional    pam_keyinit.so revoke
session   required    pam_limits.so
session   optional    pam_systemd.so
session   [success=1 default=ignore] \
                      pam_succeed_if.so service in crond quiet use_uid
session   required    pam_unix.so
...
```

Example: https://github.com/linux-pam/linux-pam/issues/680

My custom PAM file:

```
...
auth [ignore=1 default=ignore] pam_env.so envfile=/etc/test_env
auth required pam_echo.so "111"
auth required pam_echo.so "222"
...
```

My /etc/test_env:

```
TEST_VAR=foo
```

Looks like:

- the env variable TEST_VAR is not set at all
- pam_env.so always return PAM_IGNORE as I didn't see "111" in the logs

Applications talk to the PAM library

type	API function name	description
auth	pam_authenticate	Authenticate this user
auth	pam_setcred	Manage credentials of this user
account	pam_acct_mgmt	Check account validity and restrictions for this user
password	pam_chauthtok	Change the authentication token for this user
session	pam_open_session	Set up a session for this user
session	pam_close_session	End the session for this user

Frozen stack

The PAM library determines and fixes the list and order of modules for a specific management group (like **auth** or **session**) during the first API call to the stack, and then reusing that exact same (frozen) sequence of modules for subsequent API calls to this stack.

Example: https://github.com/linux-pam/linux-pam/issues/680

My custom PAM file:

```
...
auth [ignore=1 default=ignore] pam_env.so envfile=/etc/test_env
auth required pam_echo.so "111"
auth required pam_echo.so "222"
...
```

- When invoked by pam_authenticate, pam_env.so does nothing and always returns PAM_IGNORE.
- After pam_authenticate the PAM auth stack is already frozen, so during pam_setcred the modules are being called in the same order as they were called during pam_authenticate.

pam_unix: traditional approach

- a helper is needed to access /etc/shadow
- unix_chkpwd helper is setuid-root

pam_tcb approach (since 2001)

- /etc/shadow $\longrightarrow$ /etc/tcb/user/shadow
- tcb_chkpwd helper is setgid-shadow

pam_unix: client-server approach (since 2025)

- unix_chkpwd helper is unprivileged
- communicates with pwaccessd via a Unix domain socket

Common issues

- Getting locked out
- Incorrect module order or control directives
- Syntax errors or typos in config files
- Missing module arguments

Best Practices

- Know what you are doing
- Backup before making changes
- Always test the changes
- Make incremental changes
- Always have an emergency root shell open when testing on a non-disposable system
- Use distribution tools, study distribution defaults

System logs is the primary tool

- journalctl -u *service-name*

Increase PAM verbosity

- PAM is usually quite verbose already.
- Many modules become more verbose with **debug** argument.

RTFM

- Manual pages: PAM(8), pam.conf(5), pam_*(8).
- The Linux-PAM System Administrators' Guide.
- The Linux-PAM Module Writers' Guide.
- The Linux-PAM Application Developers' Guide.

strace

- -p $PID
- -f/–follow-forks
- -b execve
- -r/–relative-timestamps

Questions?