PAM Administration

The Pluggable Authentication Module (PAM) framework allows for new authentication technologies to be “plugged-in” without changing commands such as `login`, `ftp`, `telnet`, and so on. It can be used to integrate UNIX login with other security mechanisms like DCE or Kerberos. Mechanisms for account, session and password management can also be “plugged-in” using this framework.

Introduction to PAM

PAM allows the system administrator to choose any combination of services to provide authentication. The list below includes some of the advantages of PAM to the system administrator.

- Flexible configuration policy
  - Per application authentication policy
  - Can choose a default authentication mechanism for non-specified applications
  - Multiple passwords on high-security systems
- Ease of use for the end-user
  - No retyping of passwords if they are the same
  - Use a single password, even if the password associated with separate authentication methods are different, through password mapping
- Can pass optional parameters to the services

PAM Terminology

PAM employs run-time pluggable modules to provide authentication related services. These modules are broken into four different types based on their function: authentication, account management, session management and password management.

The authentication modules provide authentication for the users and allows for credentials to be set, refreshed or destroyed. These modules allow for the user to be identified.

The account modules checks for password aging, account expiration and access hour restrictions. Once the user is identified using the authentication modules, the account modules will determine if the user can be given access.

The session modules primarily manage the opening and closing of an authentication session. They can log activity or can provide for clean-up after the session is over.

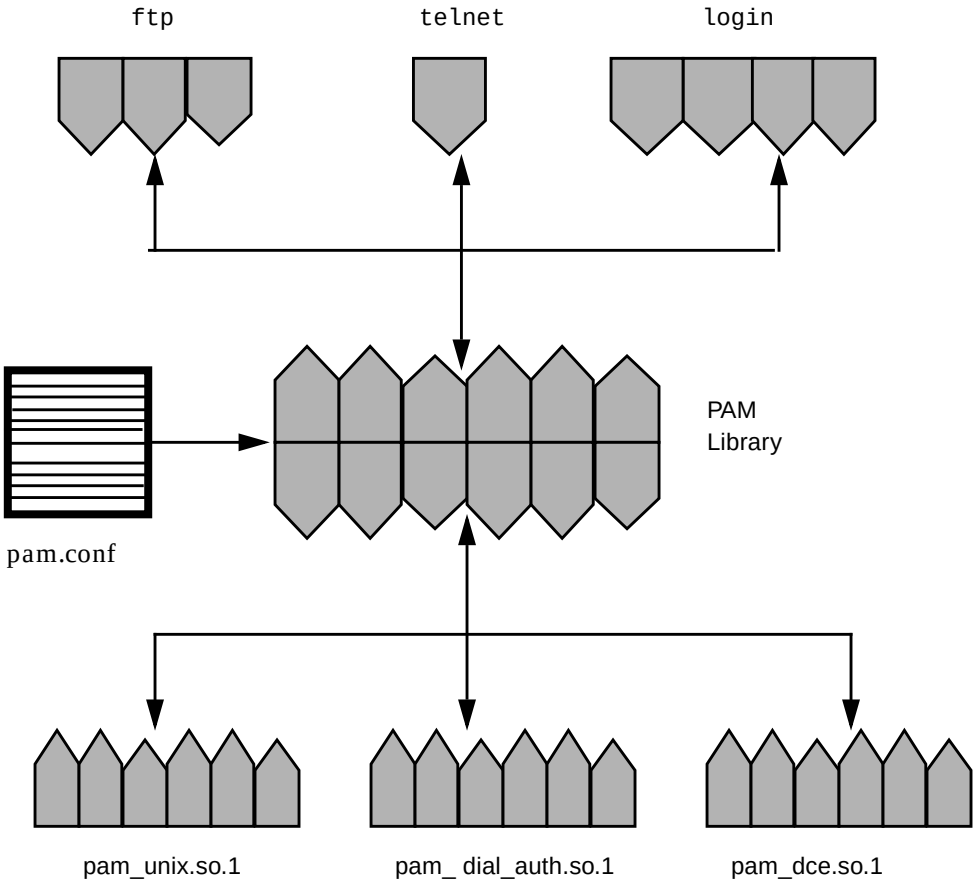
The password modules allow for changes to the password and the password-related attributes.

PAM allows for authentication by multiple methods through stacking. When a user is authenticated through PAM, multiple methods can be selected to fully identify the user. Depending on the configuration, the user can be prompted for passwords for each authentication method. Therefore, the user will not need to remember to execute another command to get fully authenticated. The order that the methods are used is determined through the configuration file, `/etc/pam.conf`.

Stacking can require that a user would need to remember several passwords. Password-mapping, if supported by the underlying module, allows for the primary password to be used to decrypt the other passwords, so that the user will not need to remember or enter multiple passwords. The other option would be to synchronize the password across each authentication mechanism. Note that this may increase the security risk, because the security of each mechanism would be limited by the least secure password method used in the stack.

How Does PAM Work?

The figure below illustrates the relationship between the applications, the library and the modules. The applications (`ftp`, `telnet` and `login`) use the PAM library to access the appropriate module. The pam.conf file defines which modules are to be used with each application. Responses from the modules are passed back through the library to the application.



PAM Files

The PAM software consists of a library, several modules and a configuration file. The current release also includes new versions of several commands which use the PAM interfaces.

PAM Library

The PAM library, `/usr/lib/libpam`, provides the framework to load the appropriate modules and manage stacking. It provides a generic structure for all of the modules to plug into.

PAM Modules

Each module provides the implementation of a specific mechanism. More than one module type (auth, account, session or password) may be associated with each module, but each module needs to manage at least one module type. Here is a description of some of the modules.

The pam_unix module, `/usr/lib/security/pam_unix.so.1`, provides support for authentication, account management, session management and password management. Any of the four module type definitions can be used with this module (see the `pam_unix(5)` man page). It uses UNIX passwords for authentication. In Solaris, this is controlled through `/etc/nsswitch.conf`.

The dial_auth module, `/usr/lib/security/pam_dial_auth.so.1`, can only be used for authentication (see the `pam_dial_auth(5)` man page). It uses data stored in the /etc/dialups and /etc/d_passwd files for authentication. This is mainly used by `login`.

The rhosts_auth module, `/usr/lib/security/pam_rhosts_auth.so.1`, can also only be used for authentication (see the `pam_rhosts_auth(5)` man page). It uses data stored in the `~/.rhosts` and `/etc/host.equiv` files through `ruserok()`. This is mainly used by `rlogin` and `rsh`.

The pam_dce module, `/usr/lib/security/pam_dce.so.1`, provides support for authentication, account management, and password management. Any of these three module type definitions can be used with this module (see the `pam_dce(5)` man page). The module uses DCE Registry for authentication.

For security reasons, it is required that these files to be owned by `root` and for the permissions to be set such that the files are not writable through `group` or `other` permissions. If the file is not owned by `root` then PAM will not load the module.

PAM Configuration File

The PAM configuration file, `/etc/pam.conf`, can be edited to select authentication mechanisms for each system-entry application. The file consists of entries following this syntax:

service_name module_type control_flag module_path module_options

where *service_name* indicates the name of the service, *module_type* denotes the module type for the service, *control_flag* selects the continuation and failure semantics for the module, *module_path* specifies the pathname to a library object which implements the service functionality, and *module_options* are specific options that can be passed to the service modules. The only optional component is *module_options.* All other values must be defined. Comments can be added to the file by starting the line with a #. Any white-space can be used to delimit the fields.

An entry in this file is ignored if one of the following conditions exist:

- the line has less than four fields
- an invalid value is given for *module_type* or *control_flag*
- the named module is not found

Table 1 lists some of the valid service names, the module types that can be used with that service, and the daemon or command associated with the service name. There are several module types that are not appropriate for each service. For instance, the `password` module type is only specified to go with the `passwd` command. This command is not concerned with authentication so there is no `auth` module type associated with it.

Table 1 Service Names for `/etc/pam.conf`

Service Name	Daemon or Command	Module Type
`dtlogin`	`/usr/dt/bin/dtlogin`	`auth, account, session`
`ftp`	`/usr/sbin/in.ftpd`	`auth, account, session`
`init`	`/usr/sbin/init`	`session`

Table 1 Service Names for `/etc/pam.conf`

Service Name	Daemon or Command	Module Type
`login`	`/usr/bin/login`	`auth, account, session`
`passwd`	`/usr/bin/passwd`	`password`
`rexd`	`/usr/sbin/rpc.rexd`	`auth`
`rlogin`	`/usr/sbin/in.rlogind`	`auth, account, session`
`rsh`	`/usr/sbin/in.rshd`	`auth, account, session`
`sac`	`/usr/lib/saf/sac`	`session`
`su`	`/usr/bin/su`	`auth, account, session`
`telnet`	`/usr/sbin/in.telnetd`	`auth, account, session`
`ttymon`	`/usr/lib/saf/ttymon`	`session`
`uucp`	`/usr/sbin/in.uucpd`	`auth, account, session`

One of three *control_flags* must be selected for each entry to determine continuation or failure behavior from the module. These flags determine what the ultimate result (success or failure) will be. The values are defined below:

- `required` - this module must return success in order to have the overall result be successful
- `optional` - if this module fails the overall result can be successful if another module in this stack returns success
- `sufficient` - if this module is successful, skip the remaining modules in the stack, even if they are labeled as `required`

If all of the modules are labeled as `required`, then authentication through all modules must succeed in order for the user to be authenticated. If some of the modules fail then a error value from the first failed module is reported. If a failure occurs for a `required` module, all modules in the stack are still tried but the access is denied.

If none of the modules are labeled as `required`, then at least one of the entries for that service must succeed for the user to be authenticated. The `optional` flag should be used when one success in the stack is enough. This flag should only be used if it is not important for this mechanism to succeed. For instance if your users need to have permission associated with a specific mechanism to get their work done, then it should not be labeled as `optional`.

The `sufficient` flag allows for one successful authentication to be enough for the user to get in. More information about these flags is given in the next section which presents the default `/etc/pam.conf` file.

The generic pam.conf file looks like:

```
# PAM configuration
#
# Authentication management
#
login   auth      required    /usr/lib/security/pam_unix.so.1
login   auth      required    /usr/lib/security/pam_dial_auth.so.1
rlogin  auth      sufficient  /usr/lib/security/pam_rhost_auth.so.1
rlogin  auth      required    /usr/lib/security/pam_unix.so.1
dtlogin auth      required    /usr/lib/security/pam_unix.so.1
telnet  auth      required    /usr/lib/security/pam_unix.so.1
su      auth      required    /usr/lib/security/pam_unix.so.1
ftp     auth      required    /usr/lib/security/pam_unix.so.1
uucp    auth      required    /usr/lib/security/pam_unix.so.1
rsh     auth      required    /usr/lib/security/pam_rhost_auth.so.1
OTHER   auth      required    /usr/lib/security/pam_unix.so.1
#
# Account management
#
login   account   required    /usr/lib/security/pam_unix.so.1
rlogin  account   required    /usr/lib/security/pam_unix.so.1
dtlogin account   required    /usr/lib/security/pam_unix.so.1
telnet  account   required    /usr/lib/security/pam_unix.so.1
ftp     account   required    /usr/lib/security/pam_unix.so.1
OTHER   account   required    /usr/lib/security/pam_unix.so.1
#
# Session management
#
login   session   required    /usr/lib/security/pam_unix.so.1
rlogin  session   required    /usr/lib/security/pam_unix.so.1
dtlogin session   required    /usr/lib/security/pam_unix.so.1
telnet  session   required    /usr/lib/security/pam_unix.so.1
uucp    session   required    /usr/lib/security/pam_unix.so.1
OTHER   session   required    /usr/lib/security/pam_unix.so.1
#
# Password management
#
passwd  password  required    /usr/lib/security/pam_unix.so.1
OTHER   password  required    /usr/lib/security/pam_unix.so.1
```

The file specifies that when running `login`, authentication must succeed for both the `pam_unix` and the `pam_dial_auth` modules. For `rlogin`, authentication through the pam_unix module must succeed, if authentication through `pam_rhost_auth` fails. The `sufficient` control flag indicates that for `rlogin` the successful authentication provided by the `pam_rhost_auth` module is sufficient and the next entry will be ignored.

Most of the other commands requiring authentication require successful authentication through the `pam_unix` module. Authentication for `rsh` must succeed through the `pam_rhost_auth` module.

Selecting `OTHER` for the service name allows a default to be set for any other commands that need authentication that are not included in the file. The `OTHER` option makes it easier to administer the file, since many commands that are using the same module can be covered by only one entry. Also, the `OTHER` option when used as a “catch-all” can make sure that each access is covered by one module. By convention the `OTHER` entry is included at the bottom of the section for each module type. The *service_name* field is case-insensitive; the capitalization is included to improve readability.

The rest of the entries in the file control the account, session and password management. With the use of the default service name, `OTHER`, the file could be simplified to:

```
#
# PAM configuration
#
# Authentication management
#
login   auth      required    /usr/lib/security/pam_unix.so.1
login   auth      required    /usr/lib/security/pam_dial_auth.so.1
rlogin  auth      sufficient  /usr/lib/security/pam_unix.so.1
rlogin  auth      required    /usr/lib/security/pam_rhost_auth.so.1
rsh     auth      required    /usr/lib/security/pam_rhost_auth.so.1
OTHER   auth      required    /usr/lib/security/pam_unix.so.1
#
# Account management
#
OTHER   account   required    /usr/lib/security/pam_unix.so.1
#
# Session management
#
OTHER   session   required    /usr/lib/security/pam_unix.so.1
#
# Password management
#
OTHER   password  required    /usr/lib/security/pam_unix.so.1
```

Normally the entry for the *module_path* is “root-relative”. If the entry for module_path does not begin with a slash, “/”, the path /usr/lib/security/ is prepended to the filename. Paths to modules located in other directories must start from root.

The values for the *module_options* can be found in the man pages for the module (for example, `pam_unix(5)` and `pam_dce(5)`). The `use_first_pass` and `try_first_pass` options, which are supported by the pam_unix and pam_dce modules, allow for reuse of the same password for authentication without retyping it.

If `login` specifies authentication through both pam_dce and pam_unix, then the user would be prompted to type in a password for each module. In situations where the passwords are the same, the `use_first_pass` module option would prompt for only one password and would use that password to

authenticate the user for both modules. If the passwords are different, the authentication would fail and the user would not be able to login. In general, this option should be used with an `optional` control flag, as shown below, to make sure that the user can still get in.

```
# Authentication management
#
login  auth   required    /usr/lib/security/pam_unix.so.1
login  auth   optional    /usr/lib/security/pam_dce.so.1  use_first_pass
```

If `try_first_pass` module option was used instead, the DCE module will prompt for a second password if the passwords do not match or if an error is made. If both methods of authentication are necessary for a user to get access to all the tools they need, using this option could cause some confusion with the user since the user could get access with only one type of authentication.

Configuring PAM

The section below discusses some of the tasks that may be required to allow PAM to be fully functional. In particular, you should be aware of some of the security issues associated with the configuration file.

Planning for PAM

When deciding how best to employ PAM in your environment, start by focusing on these issues:

- Determine what your needs are, especially which modules you should select
- Identify the services that need special attention, use OTHER if appropriate
- Decide on the order in which the modules should be run
- Select the control flag for that module
- Choose the options if necessary for the module

Here are some suggestions to consider before changing the configuration file:

- Use the OTHER entry for each module type, so that each application does not have to be included
- Make sure to consider the security implications of the `sufficient` and `optional` control flags
- Review the man pages associated with the modules to understand how they will function and what options are available
- Review the man pages to study the interactions between stacked modules

▼ How to disable `~/.rhosts` entries

♦ **Remove the `rlogin auth rhosts_auth.so.1` entry from the configuration file.**

This will prevent reading the `~/.rhosts` files during an rlogin session. It will prevent non-authenticated access to the local system from remote systems. All rlogin access will require a password, regardless of the presence or contents of any `~/.rhosts` or `/etc/hosts.equiv` files.

Note – To prevent other non-authenticated access using the `~/.rhosts` files remember to disable the `rsh` service. The best way to disable a service is toremove the service entry from `/etc/inetd.conf`. Making changes to the PAM configuration file will not prevent the service from being started.

▼ How to add the DCE PAM module

♦ **Edit the `/etc/pam.conf` file to look like the following:**

```
#
# PAM configuration
#
# Authentication management
#
login   auth     sufficient  /usr/lib/security/pam_dce.so.1
login   auth     required    /usr/lib/security/pam_unix.so.1
rlogin  auth     required    /usr/lib/security/pam_unix.so.1
rsh     auth     required    /usr/lib/security/pam_rhost_auth.so.1
OTHER   auth     required    /usr/lib/security/pam_unix.so.1
#
# Account management
#
login   account   required    /usr/lib/security/pam_dce.so.1
login   account   required    /usr/lib/security/pam_unix.so.1
OTHER   account   required    /usr/lib/security/pam_unix.so.1
#
# Session management
#
OTHER   session   required    /usr/lib/security/pam_unix.so.1
#
# Password management
#
passwd  password  required    /usr/lib/security/pam_dce.so.1
passwd  password  required    /usr/lib/security/pam_unix.so.1
```

The `sufficient` flag on the login entry indicates that if a user can be authenticated through the pam_dce module, it is enough. They do not have to also be authenticated through the pam_unix module. The only time that the pam_unix module will be checked is if the authentication through the pam_dce module fails.

The two entries for `login` authentication will allow the root user to be able to get access to the local system. This extra line is necessary because DCE does not allow for `root` access. If the normal users do not have UNIX passwords then they would still not be able to get in, but a locally defined `root` account would be able to.

Note that in this example, the DCE module is only used for `login`. But if you wanted to, the DCE module could be added for other services as well. If the pam_dce module is added as an `auth` module for login, it should also be added as an `account` module as well. The DCE entry for the `passwd` service ensures that the DCE password is also changed when the user runs the `password` command.

▼ How to make changes to `/etc/pam.conf`

If the PAM configuration file is misconfigured or gets corrupted, it is possible that even the root user would not be able to login. Since `sulogin` does not use PAM, the root user would then be required to boot the machine into single user and fix it.

After making changes to the file and sanity check it as much as possible while still logged in as root. Test all of the commands that might have been affected by your changes. For example, if you added a new module to the `telnet` service, then use the `telnet` command to the system and verify that the changes you made act as expected.

▼ How to add a module

1. **Study the documentation on the module and determine which control flags and other options should be used.**
2. **Copy the new module to /usr/lib/security.**
3. **Set the permissions so that the module file is owned by root and permissions are 555.**
4. **Edit the PAM configuration file, `/etc/pam.conf`, and add this module to the appropriate services.**
5. **Test the changes.**
   If the service is a daemon that is spawned only once when the system is booted, it may be necessary to reboot the system before full testing can be done. It is very important to do some testing before the system is rebooted in case the configuration file is misconfigured. At least try `rlogin`, `su` and `telnet` before rebooting the system to do the final testing.

▼ How to initiate error reporting

♦ **Add entries to** `/etc/syslog.conf`
The syslog daemon must be restarted or a SIGHUP signal sent to it for any changes to take effect. These selections can be added to the file to gather information about pam:

- auth.alert - messages about conditions that should be fixed now
- auth.crit - critical messages
- auth.err - error messages
- auth.info - informational messages
- auth.debug - debugging messages

The entry below will print all of alert messages on the console, critical messages will be mailed to root, and informational and debug messages will be added to `/var/log/pamlog`.

```
auth.alert                  /dev/console
auth.crit                   'root'
auth.info;auth.debug        /var/log/pamlog
```

Each line in the log contains a time stamp, the name of the system that generated it, and a message. The `pamlog` file can log a large amount of information.

Sun Microsystems Computer Company
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043 USA
415 960-1300
FAX 415 969-9131

Sales Offices

Australia: (02) 844-5000
Belgium: +32 2 716 79 11
Brazil: 55-11-887-9011
Canada: 905 477-6745
C.I.S.: 7-502-256-5470
Finland: +358-0-525561
France: (1) 30 67 50 00
Germany: (0) 89-46 00 8-0
Hong Kong: 852 802 4188
Hungary: 36-1-202-4415
Ireland: +353-1-6684377
Italy: 039 60551
Japan: (03) 5717-5000
Korea: 822-563-8700
Latin America: +1 415 688-9464
Mexico: 525-580-5229
The Netherlands: 033 501234
New Zealand: (04) 499 2344
Nordic Countries: +46 (0) 8 623 90 00
People's Republic of China: 861-8492828
Poland: 48-2-658-4535
Singapore: 224 3388
South Africa: (2711) 805-4305
Spain: (91) 5551648
Switzerland: (1) 825 71 11
Taiwan: 2-514-0567
U.A.E.: (9714) 366-333
United Kingdom: 0276 20444
United States: 800 821-4643
Venezuela: 582-285-6640

Worldwide Headquarters:
+1 415 960-1300