SUN BLUEPRINTS
[illegible]
APPROVED

Extending Authentication in the Solaris™ 9 Operating Environment Using Pluggable Authentication Modules (PAM): Part I

Michael Haines, Sun™ ONE Directory Server Group

Sun BluePrints™ OnLine—September 2002

`http://www.sun.com/blueprints`

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
650 960-1300

Part No. 816-7669-10
Revision 1.0, 08/29/02
Edition: September 2002

Extending Authentication in the Solaris™ 9 Operating Environment Using Pluggable Authentication Modules (PAM): Part I

Pluggable Authentication Modules (PAM) are an integral part of the authentication mechanism for the Solaris™ 9 Operating Environment (Solaris OE). PAM provides system administrators with the ability and flexibility to choose any authentication service available on a system to perform end-user authentication. Other PAM implementations are Linux-PAM and OpenPAM.

By using PAM, applications can perform authentication regardless of what authentication method is defined by the system administrator for the given client.

PAM enables system administrators to deploy the appropriate authentication mechanism for each service throughout the network. System administrators can also select one or multiple authentication technologies without modifying applications or utilities. PAM insulates application developers from evolutionary improvements to authentication technologies, while at the same time allowing deployed applications to use those improvements.

PAM employs run-time pluggable modules to provide authentication for system entry services. PAM offers a number of benefits, including:

- Flexible configuration policy by enabling each application or service to use its own authentication policy. PAM provides the ability for system administrators to choose a default authentication mechanism. By using the PAM mechanism to require multiple passwords, protection can be enhanced on high security systems. For example, a system administrator might want users to get authenticated by both Kerberos and Digest-MD5.
- Ease of use for the end user. Password usage can be made easier by using PAM. If users have the same passwords for different mechanisms, they do not need to retype the password. Configured and implemented properly, PAM offers a way

to prompt the user for passwords for multiple authentication methods without having the user enter multiple commands. For example, a site may require certificate-based password authentication for telnet access, while allowing console login sessions with just a UNIX® password.

- Enhances security and provides ease of use of the Solaris 9 OE in an extensible way. The security mechanisms accessible through PAM are implemented as dynamically loadable, shared software modules that can be installed by system administrators in a manner that is transparent to applications. By increasing overall security, users enjoy greater service levels and lower cost of ownership.

This article is part one of a two-part series that offers a technical overview of how the Solaris 9 OE implementation of PAM works, and demonstrates the straightforward way in which it can be configured to accommodate site-specific security policy requirements. This article examines the PAM architecture and the components that make up PAM.

Extending Authentication in the Solaris 9 OE Using PAM: Part II due in the October issue of Sun BluePrints^{TM} Online details the PAM application programming interface (API) and the PAM service provider interface (SPI). It also details how to write your first PAM module, including annotated examples of how to write pluggable authentication modules.

Part one of this article contains the following information:
- Traditional Solaris OE authentication
- PAM components
- How to add a PAM module
- PAM LDAP module

Traditional Solaris OE Authentication

Traditional Solaris OE authentication is based on the method developed for early UNIX implementations. This method employs an one-way encryption hashing algorithm called crypt(3c). The encrypted password is stored either in a file or in a Solaris OE naming service, from which it is retrieved during the user login process. The traditional UNIX method of the Solaris OE authentication, using crypt(3c), is very popular and has been enhanced to use an LDAP directory as its data store.

Before proceeding with the details on authentication, you must have a good understanding of what crypt(3c) is. There is some confusion because of a naming conflict with an *application* named *crypt*; the latter is a standard tool that ships with the Solaris OE and is a program for encrypting and decrypting the contents of a file. (This program can be found in `/usr/bin/crypt`.)

However, when the term "crypt" is referred to in authentication, it is normally cited as crypt(3c) and refers to the standard UNIX password hashing algorithm *crypt(3c)*, available to C programmers in the `libc.so` library.

A more sophisticated authentication method based on public key technology was introduced with the Network Information System (NIS+) naming service (now rebranded as the Sun OS^{TM} 5.0 Network Information Service). The NIS+ naming service method does not replace crypt(3c), but rather provides an additional security layer by introducing the concept of a network password. When users access network services through the secure remote procedure call (RPC) mechanism, the network password is required.

Originally developed by Sun Microsystems, Inc. and adopted by the Open Software Foundation (OSF) for inclusion in Common Desktop Environment (CDE)/Motif, pluggable authentication modules (PAM) provide a mechanism for dynamic system authentication and related services such as password, account, and session management. Realizing that new authentication models continue to be developed, Sun Microsystems, Inc. created the PAM architecture that allows additional methods to be added without disturbing existing ones. PAM was introduced in the Solaris 2.6 OE to overcome having to recode system entry services such as, `login`, `passwd`, `dtlogin`, `telnet`, and `rlogin` when a new authentication mechanism was developed and introduced.

The PAM architecture and alternatives to traditional Solaris OE authentication are presented in the Section , "Solaris 9 OE PAM Framework," on page 6.

UNIX Passwords

Passwords are created with the Solaris OE `passwd` command. This command prompts the user for a (new) password, which the user enters as a text string. In the Solaris OE, this text string is then hashed—or one-way encrypted—using the crypt(3c) algorithm. The result is stored either in `/etc/shadow`, or in the `passwd.byname` and `passwd.byuid` NIS maps. If the NIS+ naming service is used, the results are stored in the `Passwd` and `Cred` table type. The crypt(3c) algorithm is provided with a random seed, known technically as a *salt string*, so that the result is different each time the `passwd` command is run, even if the same text string is used.

When a user logs in, the Solaris OE `login` program challenges the user to provide a password. This password is hashed in the same manner as the `passwd` command. If the output from this process matches the output that is stored in the password database, the user is authenticated.

FIGURE 1 illustrates how the UNIX password process works.

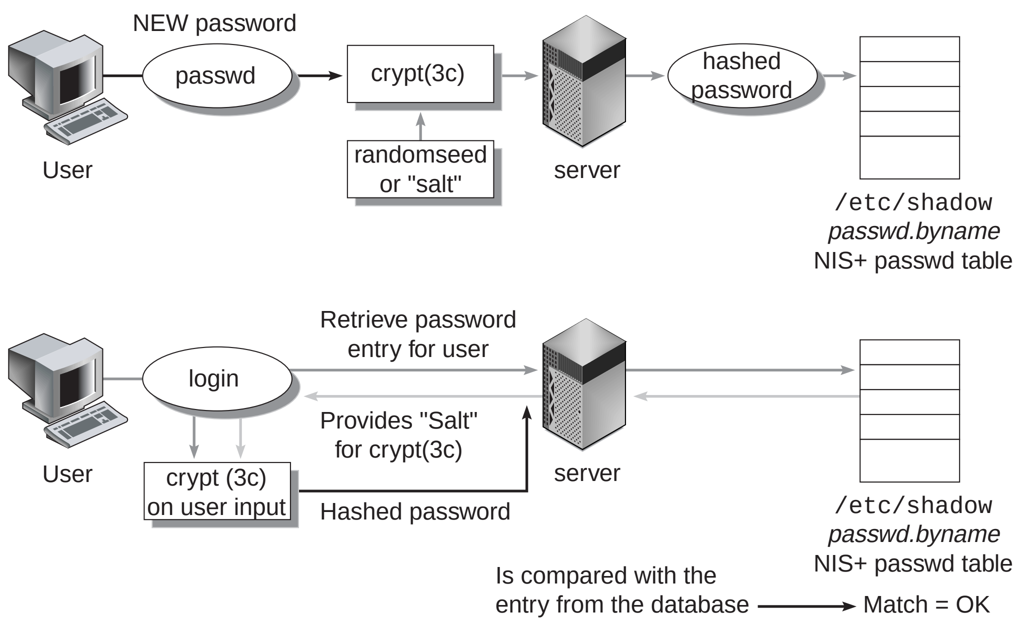

FIGURE 1 `Login` Program Text String Converting to a Hashed String

Benefits and Drawbacks of crypt(3c)

The major benefit of crypt(3c) is that it is easy to implement in a closed environment. Authentication takes place on the host that the user logs in to, so an authentication server is not required. In the case of local logins, the clear text passwords are never stored or sent over the network, so there is no reason to be concerned about eavesdroppers intercepting the password. However, when authenticating over a network using `telnet` or `rlogin`, passwords are sent in the clear text.

Because crypt(3c) uses a one-way encryption algorithm, it is difficult to decrypt passwords stored on the server. Only the user knows what the actual password is. This means that there is no way to convert passwords stored in crypt to another format required by a different authentication method.

When the crypt(3c) function is called, it takes the first eight characters and returns its computation. This computation is then injected with a randomly generated value called the *salt*. In conventional crypt, the salt is stored as the first two characters. This salt value is then added, resulting in a sequence of 13 characters. The result is that the salt is actually an important part of the password string that is stored in the specific naming service.

Note – In the future, Sun plans to update the crypt(3c) API in the Solaris 9 OE to allow different algorithms, such as MD5 and Blowfish to be used for encrypting the user's login password.

As CPUs and storage capabilities increase, the crypt(3c) algorithm becomes vulnerable to attack. The crypt(3c) mechanism shipping with the Solaris 9 OE, along with PAM authentication, is exactly the same implementation that has been in the Solaris OE for many years now, and is due to change in a subsequent update release of the Solaris 9 OE.

The Solaris OE crypt(3c) mechanisms work well for authenticating local Solaris OE clients, but they are not the only methods used by applications and services running in the Solaris OE. This can make it difficult for system developers and system administrators, who must work with multiple password systems, and for users who must remember multiple passwords.

The PAM interface in the Solaris 9 OE makes it easier for system administrators to deploy different authentication technologies without modifying administrative commands such as `login`, `telnet`, and other administrative commands. Administrators are able to select one or multiple authentication technologies, without modifying applications or utilities. PAM can also be an integral part of a single sign-on system. The PAM APIs provide a flexible mechanism that increases overall system security. The PAM APIs are detailed in *Extending Authentication in the Solaris 9 OE Using PAM: Part II* due in the October issue of Sun BluePrints Online.

PAM Components

This section details the following components that make up PAM:

- Solaris 9 OE PAM framework
- PAM module types
- PAM service module update
- PAM configuration file update
- PAM password management extensions
- `pam_ldap` password management (available in Solaris 9 12/02 OE)

Solaris 9 OE PAM Framework

The PAM framework enables new authentication technologies to be plugged in without the need to change commands such as `login`, `dtlogin`, `rsh`, `su`, `ftp`, and `telnetd`. PAM is also used to replace the UNIX login with other security mechanisms, such as Kerberos and LDAP authentication. Mechanisms for account, session, and password management can also be plugged in through this framework.

This framework consists of four specific components:

- PAM API presented to the application programs (detailed in Part II of this article)
- PAM framework responsible for implementing the API
- PAM service provider interface (SPI) implements the back end functionality for the PAM API (detailed in Part II of this article)
- Configuration file `pam.conf` specifies which service providers are used for the various programs

PAM allows the system administrator to choose any combination of services to provide authentication. These include a flexible configuration policy that enables a per application authentication policy, choice of a default authentication mechanism for non-specified applications, and multiple passwords on high security systems. Another valuable service is the ease of use for the end user that enables no retyping of user passwords if the passwords are the same, and optional parameters passed to the services.

With the introduction of the new PAM framework in the Solaris 9 OE, the LDAP service module for PAM has been extended to support the account service, which checks a user's password and account status by binding to the directory (LDAP) server. The directory server returns the password status to `pam_ldap`, which in turn maps the status to the PAM error codes. A user might be rejected when logging in with an expired password, or might see a warning message after logging in when the password is about to expire.

The `pam_ldap` module has also been updated to support password syntax checking, which is performed through the Sun™ Open Net Environment (Sun ONE) Directory Server 5.x (formerly known as the iPlanet™ Directory Server) password policy engine. When changing the password (using the `passwd` command), the user might see error messages such as *password too short*, *password in history*, and so forth

Pluggable Authentication Service Module Types

The PAM framework currently provides four different types of service modules, which are implemented by dynamic loadable module types to provide authentication related services. These modules are categorized based on the function they perform:

- Authentication (`auth`): Provides authentication for users and enables credentials to be set, refreshed, or destroyed.
- Account management (`account`): Checks for password aging, account expiration, and access hour restrictions. Once the user is identified by the authentication modules, the account management modules determine whether the user can be given access.
- Session management (`session`): Manages the opening and closing of a session. The modules can log activity, or clean up after the session is over. For example, the `unix_session` module updates the `lastlog` file.
- Password management (`password`): Contains functionality that enables the user to change an authentication token (usually a password).

Stacking

PAM enables 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. This means that the user need not execute another command to be fully authenticated. The order in which the methods are used is determined through the configuration file, `/etc/pam.conf`.

Note – Stacking might increase the security risk, because the security of each mechanism could be limited by the least secure password method used in the stack. For example, it may not be possible to use the strongest PAM mechanism, such as `pam_kerb5` (Kerberos V5 service module for PAM) in the Solaris OE LDAP Client implementation with the directory server, because the currently available directory server does not yet support Kerberos.

Now that some of the PAM basics have been covered, the next section presents an architectural overview of the PAM framework. FIGURE 2 illustrates the PAM framework.

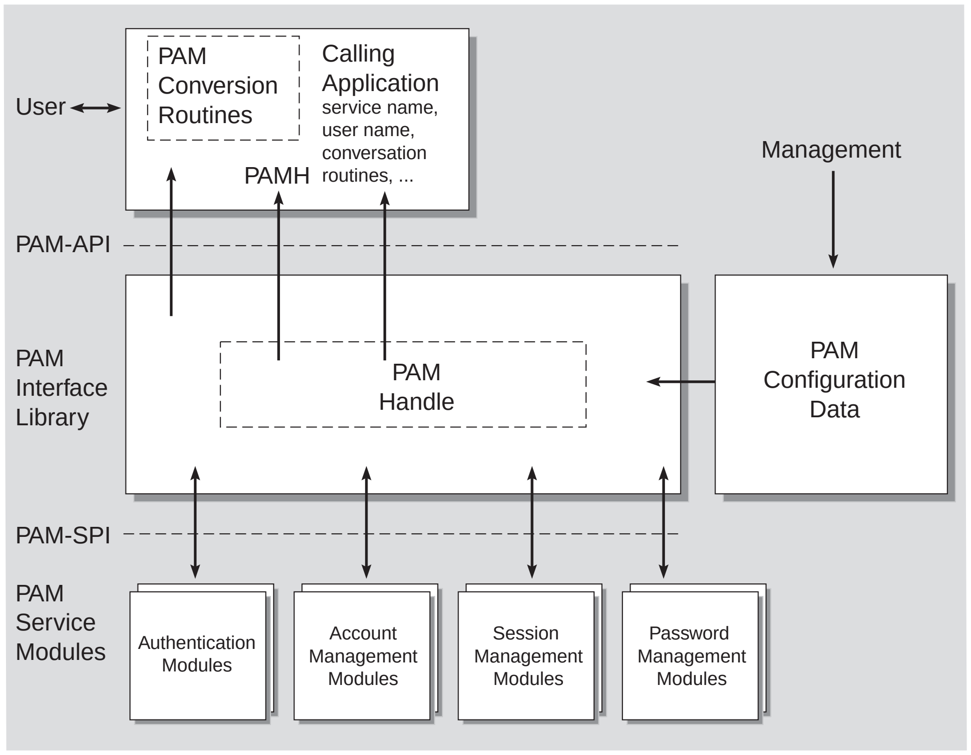

FIGURE 2 PAM Framework Architecture

PAM Operation

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

FIGURE 3 illustrates the relationship between the applications, the library, and the modules. The applications `login`, `passwd`, and `su` use the PAM library to access the appropriate module. The `pam.conf` file defines which modules are used with each application. Responses from the modules are passed back through the library to the application.

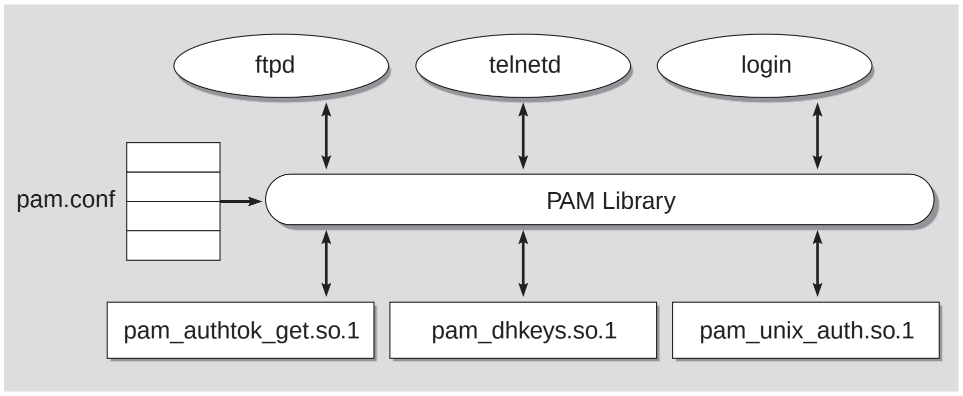

FIGURE 3 PAM and the Relationship Between Applications, Library, and Modules

Pluggable Authentication Service Modules

Each module provides the implementation of a specific mechanism. More than one module type (`auth`, `account`, `session`, or `password`) can be associated with each module, but each module needs to manage at least one module type. The following is a description of the modules that are part of the Solaris 9 OE.

- `pam_authtok_get`: Supports authentication and password management. This module takes care of obtaining (old or new) passwords from the user so that other modules on the stack can concentrate on their task, and not worry about obtaining information from the user.
- `pam_authtok_check`: This module provides functionality to the password management stack. Specifically, it performs a number of checks on the construction of the newly entered password. See `pam_authtok_check(5)` man page for a description of the checks it performs.
- `pam_authtok_store`: Provides functionality to the PAM password management stack. When invoked with flags set to `pam_update_authtok`, this module updates the authentication token for the user specified by `pam_user`.
- `pam_dhkeys`: Supports authentication and password management. This module specifically deals with the establishment and modification of the Diffie-Hellman keys which are used, for example, for Secure RPC calls (NIS+ and Secure NFS).
- `pam_passwd_auth`: Provides authentication functionality to the password service as implemented by `passwd(1)`. It differs from the standard PAM authentication modules in its prompting behavior.
- `pam_unix_account`: Provides functionality to the PAM account management stack, as the PAM account management module for UNIX. The `pam_acct_mgmt(3PAM)` function retrieves password aging information from the repositories specified in `nsswitch.conf(4)` and verifies that the user's account and password have not expired.

- `pam_unix_auth`: Verifies the password that the user has entered against any password repository specified in the `nsswitch.conf` using normal UNIX crypt(3c) style password encryption. Can only be used for authentication.
- `pam_unix_session`: Provides functions to initiate and to terminate session as the session management PAM module for UNIX.

For security, these files are required to be owned by `root` and to have their permissions set so 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. This requirement on permissions and owner for the modules is not documented anywhere, and might change in future releases.

Note – In FIGURE 3, `pam_unix` is not layered entirely on the LDAP server. The `pam_unix` module sits on the Name Service Switch (NSS) layer and the NSS backends that could be files, NIS, NIS+, or LDAP.

PAM Configuration File Update

The PAM configuration file, `/etc/pam.conf`, determines what authentication services are used and in what order. Edit this file to select the desired authentication mechanisms for each system entry application.

Configuration File Syntax

The PAM configuration file consists of entries with the following syntax:

service_name module_type control_flag module_path module_options

TABLE 1 explains the functions of the syntax.

TABLE 1 Configuration File Syntax

Syntax	Function
service_name	Name of the service (for example, `ftp`, `login`, `telnet`)
module_type	Module type for the service (`auth`, `account`, `session`, `password`)
control_flag	Determines the continuation or failure semantics for the module (see note below)
module_options	Specific options passed to the service modules

Comments can be added to the `pam.conf` file by starting the line with a pound sign (#). Use white space to delimit the fields.

Note – An entry in the PAM configuration file is ignored if one of the following conditions exists: the line has fewer than four fields, an invalid value is given for *module_type* or *control_flag*, or the named module is not found.

TABLE 2 summarizes PAM configurations.

TABLE 2 PAM Configurations

Service Name	Daemon or Command	Module Type
`cron`	`/usr/sbin/cron`	`account`
`dtlogin`	`/usr/dt/bin/dtlogin`	`auth, account, session`
`ftp`	`/usr/sbin/in.ftpd`	`auth, account, session`
`init`	`/usr/sbin/init`	`session`
`login`	`/usr/bin/login`	`auth, account, session, password`
`passwd`	`/usr/bin/passwd`	`auth, account, password`
`ppp`	`/usr/bin/pppd`	`auth, account, session`
`rexecd`	`/usr/sbin/in.rexecd`	`auth, account`
`rexd`	`/usr/sbin/rpc.rexd`	`account, session`
`rlogin`	`/usr/sbin/in.rlogind`	`auth, account, session, password`
`rsh`	`/usr/sbin/in.rshd`	`auth, account`
`sac`	`/usr/lib/saf/sac`	`session`
`sshd`	`/usr/lib/ssh/sshd`	`auth, account, session, password`
`su`	`/usr/bin/su`	`auth, account`
`telnet`	`/usr/sbin/in.telnetd`	`auth, account, session, password`
`ttymon`	`/usr/lib/saf/ttymon`	`session`
`uucp`	`/usr/sbin/in.uucpd`	`auth, account`

Control Flags

To determine continuation or failure behavior from a module during the authentication process, you must select one of four control flags for each entry. Successful or failed attempts are indicated through control flags. Even though these flags apply to all module types, the following explanation assumes that the flags are being used for authentication modules. The control flags are as follows:

`required` — This module must return `success` in order to have an overall successful result. If all of the modules are labeled as `required`, then authentication through all modules must succeed for the user to be authenticated. If some of the modules fail, then an error value from the first failed module is reported. If a failure occurs for a module flagged `required`, all modules in the stack are still tried but failure is returned. If none of the modules are flagged `required`, then at least one of the entries for that service must succeed for the user to be authenticated.

`requisite` — This module must return `success` for additional authentication to occur. If a failure occurs for a module flagged `requisite`, an error is immediately returned to the application and no additional authentication is done. If the stack does not include prior modules labeled `required` that failed, then the error from this module is returned. If a earlier module labeled `required` has failed, the error message from the required module is returned.

`optional` — If this module fails, the overall result can be successful if another module in this stack returns `success`. The `optional` flag should be used when one success in the stack is enough for a user to be authenticated. This flag should only be used if it is not important for this particular mechanism to succeed. If your users need to have permission associated with a specific mechanism to get their work done, then you should not label it `optional`.

`sufficient` — If this module is successful, skip the remaining modules in the stack, even if they are labeled `required`. The `sufficient` flag indicates that one successful authentication is enough for the user to be granted access. More information about these flags is provided in the next section, which describes the default `/etc/pam.conf` file.

Note – In Solaris 9 12/02 OE, a NEW control flag has been added to the PAM framework. The control flag *binding* has a meaning of *terminate processing upon success, and report the failure if unsuccessful*. This option effectively provides a local account overriding remote (ldap) account functionality.

Generic `pam.conf` File

The following is an example of a generic `pam.conf` file:

```
# PAM configuration
# Authentication management
#
login   auth requisite pam_authtok_get.so.1
login   auth sufficient pam_unix_auth.so.1
login   auth required pam_ldap.so.1
#
rlogin  auth sufficient pam_rhosts_auth.so.1
rlogin  auth required   pam_authtok_get.so.1
rlogin  auth sufficient pam_unix_auth.so.1
#
dtlogin auth required   pam_authtok_get.so.1
dtlogin auth required   pam_unix_auth.so.1
#
rsh     auth sufficient pam_rhosts_auth.so.1
rsh     auth required   pam_unix_auth.so.1
#
dtsession auth required pam_authtok_get.so.1
dtsession auth required pam_unix_auth.so.1
#
other   auth required   pam_authtok_get.so.1
other   auth required   pam_unix_auth.so.1
#
# Account management
#
login   account requisite       pam_roles.so.1
login   account required        pam_projects.so.1
login   account required        pam_unix_account.so.1
#
```

(continued on next page)

(continued from previous page)
```
dtlogin account requisite       pam_roles.so.1
dtlogin account required        pam_projects.so.1
dtlogin account required        pam_unix_account.so.1
#
cron    account required        pam_projects.so.1
#
cron    account required        pam_unix_account.so.1
#
other   account requisite       pam_roles.so.1
other   account required        pam_projects.so.1
other   account required        pam_unix_account.so.1
# Session management
#
other   session required        pam_unix_session.so.1
#
# Password management
#
other   password requisite      pam_authtok_get.so.1
other   password requisite      pam_authtok_check.so.1
other   password sufficient     pam_authtok_store.so.1
other   password required       pam_ldap.so.1
```

This generic `pam.conf` file specifies the following behavior:

1. When running `login`, authentication must succeed for the `pam_authtok_get` module and for either the `pam_unix_auth` or the `pam_ldap` module.
2. For `rlogin`, authentication through the `pam_authtok_get` and `pam_unix_auth` modules must succeed if authentication through `pam_rhost_auth` fails.
3. The `sufficient` control flag for `rlogin`'s `pam_rhost_auth` module indicates that if the authentication performed by the `pam_rhost_auth` module is successful, the remainder of the stack is not executed, and a success value is returned.

4. Most of the other commands requiring authentication require successful authentication through the pam_unix_auth module.

The other service name allows a default to be set for any other commands requiring authentication that are not included in the file. The other option makes it easier to administer the file, since many commands that use the same module can be covered by only one entry. Also, the other service name, when used as a catchall, can ensure 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 rest of the entries are in the file control account management, session management, and password management.

Normally, the entry for the module_path is root-relative. If the file name entered for module_path does not begin with a slash (/), the path /usr/lib/security/ $ISA is added to the file name, where $ISA is expanded by the framework to contain the instruction set architecture of the executing machine (see the isainfo(1) man page for additional information).

A full path name must be used for modules located in directories other than the default. The values for the module_options can be found in the man pages for the module (for example, pam_unix_auth(5)).

If login specifies authentication through both pam_unix_auth and pam_ldap, then the user is prompted to enter a password for each module, for example:

```
# Authentication management
#
login auth required pam_authtok_get.so.1
login auth sufficient pam_unix_auth.so.1
login auth required pam_ldap.so.1
```

PAM and LDAP Password Management Extensions

It is important to provide a quick overview to clarify the difference between PAM Password Management Extensions and the new pam_ldap password management.

PAM Password Management Extensions provide the same functionality as the existing pam_unix module. The only difference is *how* the module is packaged. What used to be a single module is now split up into multiple components, known as service modules, each performing a separate function. This modular construction makes implementing custom password management policies easier.

The new pam_ldap password management facility includes two new account management features: password aging and account expiration. Because the directory server provides its own mechanism for account management, a conflict can occur if you wish pam_ldap to implement a different password policy than what the directory-wide policy is set for. For example, the directory may force all users to change passwords after 60 days but you might want some special user accounts to be able to keep their current password for a longer period of time.

To support this flexibility, the PAM framework has been enhanced by the addition of a new control flag called *binding* which instructs pam_ldap to terminate further process once the password policy criteria has been met and report a failure if it is not. Effectively, this control flag allows you to override the password policy that the directory server enforces.

A *server_policy* option has been added to instruct pam_unix to allow users that only have LDAP accounts to be processed by the password policy set on the directory server. This option can be used to instruct the pam_unix_account, pam_unix_auth, and pam_passwd_auth service modules to ignore the user being authenticated and let the pam_ldap module stacked below them process the user according to the password policy established in the directory server. This effectively allows you to override the local pam_unix password policy.

Note – The pam_authtok_store module handles this option differently.

The *server_policy* option was introduced to solve a problem found when stacking the pam_unix_account and pam_ldap modules together. When used, it tells the module to rely on the policy specified on the LDAP server and not to apply a local policy.

Because pam_unix_account receives incomplete information from the LDAP server, it might inadvertently decide that an active account has expired, or that an expired account is still active. Specifying *server_policy* in /etc/pam.conf tells pam_unix_account not to guess an account's status but to leave the decision to the LDAP server. The LDAP server keeps accurate current status of each account and can draw the correct conclusion about its expiration status.

Because this feature enables the pam_ldap module to fully support the account management, it is reasonable to use the following PAM configuration for account management.

```
other  account  requisite  pam_roles.so.1
other  account  required   pam_projects.so.1
other  account  binding pam_unix_account.so.1 server_policy
other  account  required   pam_ldap.so.1
```

Note – In this configuration, please note the binding control flag for `pam_unix_account.so.1`.

This configuration specifies that `pam_unix_account` should check the user's local account first. Because of the binding control flag, the stack succeeds or fails depending on the values returned by `pam_unix_account`. If only the ldap account exists for the user, `pam_unix_account` does nothing and allows `pam_ldap` determine the stack's success or failure.

Customer feedback indicated that the PAM functionality in the Solaris OE needed some enhancements. The requested changes included improving the mechanism used to validate password structures, adding the ability to change numbers of characters, total password length, and so forth.

In previous versions of the Solaris OE, this functionality was tightly coupled in a single monolithic module (`pam_unix`) and local extensions could not be incorporated in the module.

Only with a great deal of effort could you extend part of the operations performed by this module. Because of this, the `pam_unix(5)` functionality has been replaced with a new set of modular PAM service modules that are listed in this section. The functionality of `pam_unix` has been entirely replaced in the Solaris 9 OE. New PAM modules are now provided that replace a specific piece of `pam_unix`. This makes it easier to customize the PAM behavior by inserting or replacing individual modules. The Solaris 9 OE no longer uses `pam_unix` by default. During upgrades any existing instances of `pam_unix` in `pam.conf` are replaced by the new modules.

In the Solaris 9 OE, the functionality provided by the old `pam_unix` module has been split over a number of small modules, each performing a well-defined task, that can be easily extended or replaced by modifying the `pam.conf` file.

These new modules are:
- `pam_authtok_get(5)`
- `pam_authtok_check(5)`
- `pam_authtok_store(5)`
- `pam_unix_auth(5)`
- `pam_dhkeys(5)`
- `pam_unix_account(5)`
- `pam_unix_session(5)`

You no longer have to replace the `pam_authtok_check` module to extend or replace the standard password strength checks. Just list the module in the `/etc/pam.conf` file right before, after, or instead of the `pam_authtok_check` file.

▼ To Add a PAM Module

1. **Determine the control flags and other options to be used.**
2. **Become** `superuser`**.**
3. **Copy the new module to** `/usr/lib/security`**.**

Note – If you have a 64-bit version of the module, you should place that version in `/usr/lib/security/sparcv9`.

---

4. **Set the permissions so that the module file is owned by** `root` **and the permissions are 755.**
5. **Edit the PAM configuration file,** `/etc/pam.conf`**, and to add this module to the appropriate services.**

▼ To Verify The Configuration

It is essential to do some testing before logging out, in case the configuration file is misconfigured.

1. **Test the modified service or the other configuration.**
2. **Run** `rlogin`**,** `su`**, and** `telnet` **if these services have been changed.**

   If the service is a daemon spawned only once when the system is booted, it might be necessary to reboot the system before you can verify that the module has been added, however it might be possible to restart the daemon using the appropriate `/etc/init.d/` script.

▼ To Disable `.rhosts` Access With PAM From Remote Systems

A common use of the `.rhosts` file is to simplify remote logins between multiple accounts owned by the same user. For example, if you have multiple accounts on more than one system, you might need to perform specific tasks and using the `.rhosts` file is ideal.

However, using the `.rhosts` file as an authentication mechanism is a weak form of security and should be avoided.

- **Remove the** `rlogin` **and** `rsh` **(**`pam_rhosts_auth.so.1`**) entries from the PAM configuration file.**

This prevents reading the `~/.rhosts` files during an `rlogin` session and therefore prevents unauthenticated access to the local system from remote systems. All `rlogin` access requires a password, regardless of the presence or contents of any `~/.rhosts` or `/etc/hosts.equiv` files.

Note – To prevent other unauthenticated access to the `~/.rhosts` files, remember to disable the `rsh` service. The best way to disable a service is to remove the service entry from `/etc/inetd.conf`. The remote shell server, `rshd,` and the remote login server, `rlogind,` only use PAM, they do not call the `ruserok()` function themselves.

PAM Error Reporting

Diagnostic messages generated by the PAM modules or the PAM framework are output using `syslog(3c)`. They are logged to the facility that was specified at the time the application (`login`, `telnet`, `sshd`) called `openlog(3c)`, so the exact location of these messages depends upon whether the application uses PAM.

For example, `login` sends its messages to the `LOG_AUTH` facility, while `rlogind` sends its messages to the `LOG_DAEMON` facility. Other daemons might use a configurable facility (`sshd, ftpd,` and so forth) which can be set in the configuration file of the particular service.

Depending on the severity of the diagnostic message, the PAM module directs the message to one of the eight available log priorities.

Note – For additional details on the `syslog()` function and priorities, see the `syslog(3c)` and `syslog.conf(4)`man pages.

Debug messages are logged with:

```
syslog(LOG_DEBUG, "...")
```

Critical messages are logged with:

```
syslog(LOG_CRIT, "...")
```

For example, a general error message (LOG_ERR) from PAM, used by `login`, is directed to `auth.crit` and ends up in a `log`file as:

```
Jul 22 22:11:43 host login: [ID 887986 auth.error]
ACCOUNT:pam_sm_acct_mgmt: illegal option debuf
```

▼ To Initiate Diagnostics Reporting for PAM

1. **Backup the `syslog.conf` file before editing it.**
2. **Determine the `syslog` facility used by the application you want to receive diagnostic reports from.**
3. **Edit the `/etc/syslog.conf` to add a line describing where the message with the intended facility and priority will be logged, for instance:**

   ```
   auth.debug /var/adm/authlog
   ```

   Note that these message levels are part of a hierarchy:

   ```
   High -------------------------------------Low
   EMERG ALERT CRIT ERR WARNING NOTICE INFO DEBUG
   ```

   Due to this hierarchical ordering, a `syslog` channel specified to log `debug` messages also logs messages at all higher levels (for example, logs messages with priority `debug` *and up*).

4. **Make sure that the logfile specified in the previous step actually exists. If it doesn't, create it now with**

   ```
   # touch /var/adm/authlog
   ```

5. **Make `syslogd` re-read the configuration file by sending it a HUP signal:**

   ```
   # pkill -HUP syslogd
   ```

▼ Initiate PAM Error Reporting

The following example displays all alert messages on the console. Critical messages are mailed to `root`. Debug messages are added to `/var/log/pamlog`.

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

Each line in the logfile contains a timestamp, the name of the system that generated the message, and the message itself. Be aware that a large amount of information may be written to the `pamlog` file.

The log format was changed in the Solaris 8 OE and subsequent releases, and now includes a hash-value of the message generating string for example—"`user %s not found`." It now contains the message facility and severity.

There is another part to diagnostics reporting the `debug` option to a module.

- **Add the `debug` flag to a PAM module to enable diagnostics reporting of that module, for example:**

```
# PAM Module Debugging
#
login   auth requisite          pam_authtok_get.so.1
login   auth required           pam_dhkeys.so.1         debug
login   auth required           pam_unix_auth.so.1      debug
login   auth required           pam_dial_auth.so.1
```

This configuration example enables debugging information from `pam_dhkeys.so.1` and `pam_unix_auth.so.1`.

What gets logged might vary quite a bit, since there is no standard describing the information that needs to be output in response to this option. It is a good practice for module developers to recognize this `debug` flag and enable some form of debugging when the flag is specified in `/etc/pam.conf`.

PAM LDAP Module

The PAM LDAP module (pam_ldap) was introduced in the Solaris 8 OE for use in conjunction with pam_unix for authentication and password management with an LDAP server. This module was written to support stronger authentication methods such as CRAM-MD5, in addition to the other UNIX authentication capabilities provided by pam_unix.

Note – The pam_ldap module must be used in conjunction with the modules supporting the UNIX authentication, password and account management, as pam_ldap is designed to be stacked directly below these modules.

Currently, pam_ldap provides support only for authentication and password management. Support for account management is expected to be provided in the Solaris 9 12/02 OE.

The pam_ldap module should be stacked directly below the pam_unix module in the configuration file /etc/pam.conf. If there are other modules that are designed to be stacked in this manner, they could be stacked *under* the pam_ldap module. This design must be followed in order for authentication and password management to work when pam_ldap is used. The following is a sample of /etc/pam.conf file with pam_ldap stacked under pam_unix:

```
# Authentication management for login service is stacked.
# If pam_unix succeeds, pam_ldap is not invoked.
login   auth sufficient /usr/lib/security/pam_unix.so.1
login   auth required /usr/lib/security/pam_ldap.so.1
# Password management
other   password sufficient /usr/lib/security/pam_unix.so.1
other   password required /usr/lib/security/pam_ldap.so.1
```

It is important to note that the control flag for pam_unix is sufficient. This flag means that if authentication through pam_unix succeeds, then pam_ldap is not invoked. Also, other service types, such as dtlogin, su, telnet, and so forth can substitute for login. See FIGURE 4.

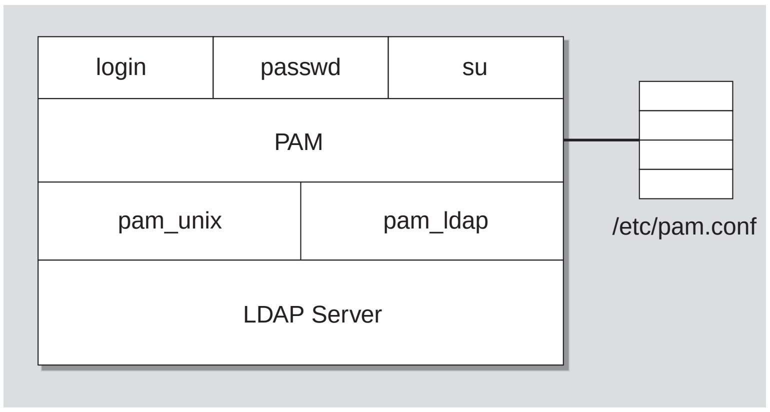

FIGURE 4 `pam_ldap` Structure

The options supported by the `pam_ldap` are:

- `debug`: If this option is used with `pam_ldap`, then debugging information is output to the `syslog(3C)` files.
- `nowarn`: This option turns off warning messages.

How PAM and LDAP Work

Before discussing the details of how PAM and LDAP work, it is important to provide a quick overview to understand and distinguish between how the password is stored and how the authentication mechanism is used to authenticate to the LDAP server. The password can be stored in a variety of formats in the Sun ONE Directory Server, such as, salted secure hash algorithm (SSHA) secure hash algorithm (SHA), CRYPT, and so forth.

The authentication mechanisms currently used and supported in the Solaris 8 OE LDAP Client, are NONE, SIMPLE, and CRAM-MD5 authentication. SIMPLE authentication requires the client to pass a distinguished name (DN) and password to the server in clear text. Currently, the Sun ONE Directory Server 5.x does not support the authentication mechanism CRAM-MD5, which sends only the digest over the wire. CRAM-MD5 is implemented as a simple authentication and security layer (SASL) mechanism and both the client and server must use it. What happens is the client request authentication is based on SASL/CRAM-MD5 and the server must support this to complete the authentication. In general, very few clients use CRAM-MD5, now that RFC 2831 mandates the use of Digest-MD5, which is intended to be an improvement over CRAM-MD5. With the introduction of the Solaris 8 OE and the LDAP Client, a usable security model is not provided.

With the introduction of the Sun ONE Directory Server 5.x; SASL External and Digest-MD5 are now supported for authentication. With Digest-MD5, a digest is created and sent across the wire to authenticate to the directory server. The directory server then compares the digest that was sent with the digest created by itself with the stored password and returns success if it matches. In this case, the password is *not* sent in clear text. To address the absence of a security model in the Solaris 8 OE LDAP Client, the Solaris 9 OE now incorporates the Sun ONE Directory Server 5.1 and Solaris 9 OE Secured LDAP Client, addressing the security issues found in the LDAP Client.

There are two LDAP Client libraries available. Currently, the Solaris 8 OE LDAP Client library version provides no support for Secure Socket Layer/Transport Layer Security 1.0 (SSL/TLS 1.0). To address this, a new LDAP library (`libldap (v5)`) for the Solaris 9 OE has been developed and it supports both SSL and TLSv1.

Note – In the current release of the Solaris 9 OE, the extended startTLS operation is not supported.

The LDAP Client library continues to provide support for the simple page control and the SASL CRAM-MD5 authentication mechanism.

Authentication With `pam_unix`

In authentication with `pam_unix`, the client retrieves the password that is stored in the server by making a call to the `getspnam` function. This function binds to the LDAP server with the proxy agent account (the reason the proxy `passwd` is sent across the wire in clear text). The proxy agent password is stored in the `userPassword` attribute in the directory server. This proxy agent account can reside anywhere in the directory server, but must contain the `userPassword` attribute.

Note that the access control instructions (ACIs) of the proxy agent allow this account to have read access to all user passwords. ACIs are instructions that are stored in the directory server itself. Every entry can have a set of rules that define an ACI for that entry. An ACI appears as an attribute in the entry so it can be retrieved by using LDAP search, or it can be added, updated, or deleted with an LDAP modify operation.

An entry may have one ACI, many ACIs, or none. ACIs allow or deny permissions to entries. When the directory server processes an incoming request for that entry, the server uses the ACIs specific to that entry to determine whether or not the LDAP client has permission to perform the requested operation.

Note – LDAP stores data as entries. An entry has a distinguished name (DN) to uniquely identify it within the directory server

The encrypted password is sent to the client side and compared with the crypted password supplied by the user at the password prompt. If there is a match, `pam_unix` returns `success`. The following tables illustrate the authentication mechanisms currently used.

TABLE 3 lists the PAM abbreviations used in this section.

TABLE 3 PAM Abbreviations

Abbreviation	Description
UP	User password
PP	Proxy agent password
NP	New password
NO*	Not applicable (at present)

TABLE 4 illustrates if the user password and proxy password are transmitted in the clear during PAM authentications.

TABLE 4 PAM Authentication

Authentication Mechanisms	pam_unix	pam_ldap
SIMPLE	UP-No PP-Yes	UP-Yes PP-Yes
DIGEST-MD5	UP-NO* PP-No	UP-No PP-No
TLS: SIMPLE	UP-No PP-No	UP-No PP-No
TLS: DIGEST-MD5	UP-No PP-No	UP-No PP-No

Note – In Tables 4 and 5 the reason for "NO*" as the value of the Digest-MD5 UP column is because the Sun ONE Directory Server version 5.1 requires the passwords be stored in the server in clear text for Digest-MD5 to work.

For updating passwords in pam_unix, the same comparison as for authentication takes place (since the user has to bind as the dn); then, the new password is encrypted and *not* passed over the wire in clear text (see TABLE 5).

TABLE 5 PAM Update of Password

Authentication Mechanisms	pam_unix	pam_ldap
SIMPLE	UP-No PP-Yes NP-No	UP-Yes PP-Yes NP-Yes
Digest-MD5	UP-NO* PP-No NP-NO*	UP-No PP-No NP-Yes
TLS: SIMPLE	UP-No PP-No NP-No	UP-No PP-No NP-No
TLS: Digest-MD5	UP-No PP-No NP-No	UP-No PP-No NP-No

The matrices are easier to understand when you distinguish between how the password is stored and how the authentication mechanism is used to authenticate to the LDAP server. The password can be stored in a variety of formats, such as SSHA, SHA, crypt, clear text, and so forth. The authentication mechanisms that are currently supported are NONE, SIMPLE, SASL/CRAM-MD5, SASL/Digest-MD5, TLS:NONE, TLS:SIMPLE, TLS:SASL/CRAM-MD5, and TLS:SASL/Digest-MD5.

pam_ldap Authentication

In authentication that uses pam_ldap, the user password is passed to the server in an auth structure in clear text, since authentication is being attempted with the user dn and password. If SIMPLE authentication is used and the password matches, then success is returned. Using pam_ldap in the Solaris 9 Secure LDAP client now provides SASL/Digest-MD5 authentication, privacy, and data integrity with SSL/TLS. If you require stronger authentication mechanisms such as Digest-MD5; then you must use pam_ldap. In addition, pam_ldap is designed to be extended for future authentication mechanisms that will be supported in future Solaris OE releases. One of the benefits of using pam_ldap, is it does not require passwords to be stored in any specific format, so you can store passwords using SSHA, SHA, or CRYPT formats.

For additional information, see the pam_ldap man page for the correct way to stack the authentication management for login service, and password management modules in the /etc/pam.conf configuration file.

Note – CRAM-MD5 is supported by the Secure LDAP client, but not by the Sun ONE Directory Server. However, Digest-MD5 is supported by both.

About The Author

Michael Haines is a member of the Sun ONE Directory Server Group, and has worked in the Solaris Operating Environment and Naming Services engineering field for over 10 years. Since joining Sun Microsystems, Inc. in 1989, he has worked in various engineering roles. Michael is also the co-author of the Sun BluePrints book *Solaris and LDAP Naming Services*.

Acknowledgments

The author would like to recognize the following individuals for their contributions to this article:

- Joep Vesseur, Technical Staff, is a member of the Solaris Security Technology Group.
- Michen Chang, Technical Staff, is a member of the Solaris Naming Services Group.

Ordering Sun Documents

The SunDocs^{SM} program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `docs.sun.com` archive or search for a specific book title or subject. The URL is `http://docs.sun.com/.`

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine Web site at: `http://www.sun.com/blueprints/online.html.`