

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

A Technical White Paper



Table of Contents

Executive Summary	1
Introduction	3
Traditional Solaris Authentication	5
How UNIX® Passwords Work	6
Benefits and Drawbacks of crypt(3c)	6
PAM Components	8
Solaris 9 OE PAM Framework	8
PAM Module Types	10
Stacking	11
How PAM Works	12
PAM Modules	12
PAM Configuration File Update	13
Configuration File Syntax	13
Control Flags	14
Generic pam.conf File	15
PAM Password Management Extensions	17
How to Add a PAM Module	19
Verification	19
How to Prevent Unauthorized Access with PAM from Remote Systems	19
How to Initiate PAM Error Reporting	20
Example: Initiating PAM Error Reporting	20
The PAM LDAP Module	21
try_first_pass	22
How PAM and LDAP Work	23
pam_unix Authentication	23
pam_ldap Authentication	23
Overview of the PAM API	
and PAM-SPI	25
PAM Framework Functions	26
PAM Authentication Functions	27
Account Management Functions	27
Session Management Functions	27

Password Management Functions	27
PAM-SPI Functions	27
Authentication Module Functions	27
Account Management Module Functions	28
Session Management Module Functions	28
Password Management Module Functions	28
Writing the First PAM Module	29
Source Code	30
Makefile	30
pam_compare Source File	31
Conclusion	37

Chapter 1

Executive Summary

This white paper discusses the pluggable authentication module (PAM), which is an integral part of the authentication mechanism for the Solaris™ 9 Operating Environment (OE). By using PAM, users who need access to certain network resources, or who log in from outside the firewall, may utilize stronger authentication mechanisms. Administrators are able to select one or more authentication technologies, without modifying applications or utilities. PAM also insulates application developers from evolutionary improvements to authentication technologies, while at the same time allowing deployed applications to use those improvements. By using the PAM layer, applications can perform authentication without worrying about what authentication method is defined by the system administrator for the given client. Administrators can deploy the appropriate authentication mechanism for each service throughout the network.

PAM offers a number of benefits, including:

- *Flexible configuration policy:* PAM enables 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 PAM mechanism to require multiple passwords, protection can be enhanced on high-security systems.
- *Ease of use for end users:* Password usage can be simplified using PAM. Configured and implemented properly, PAM provides the ability to use a single password for multiple authentication methods with the password mapping feature — even if the passwords associated with each authentication method are different. And if users have the same password for different mechanisms, they do not need to retype the password. For those systems with multiple authentication methods, PAM offers a way to prompt the user for passwords without requiring the user to enter multiple commands.

PAM can enhance security and provide ease of use in an extensible way. By increasing overall security, users enjoy greater service levels and lower cost of ownership.

This technical overview covers how the Solaris 9 implementation of PAM works, and demonstrates the straightforward way in which it can be configured to accommodate site-specific security policy requirements. The PAM architecture, as well as the components that make up PAM are discussed, including the PAM API, the PAM framework, and the PAM service provider interface (SPI). To assist the reader in using PAM, annotated examples of how to write pluggable authentication modules are also provided.

Chapter 2

Introduction

Originally developed by Sun and adopted by the Open Software Foundation (OSF) for inclusion in CDE/Motif, pluggable authentication modules (PAM) provide a mechanism for dynamic system authentication, as well as related services such as password, account, and session management. PAM was introduced in the Solaris 2.6 OE to address and overcome the issue of recoding system entry services — such as `login`, `passwd`, `dtlogin`, `telnet`, `rlogin`, and so on — when a new authentication mechanism was developed and introduced.

The security mechanisms accessible through PAM are implemented as dynamically loadable, shared software modules that can be installed by system administrators transparently to applications. PAM enables the administrator to configure the user authentication mechanism on a per-application basis. For example, a site may require certificate-based password authentication for telnet access, while allowing console login sessions with just a UNIX® password. PAM can also be used to configure multiple authentication mechanisms for each application. For example, an administrator may want users to be authenticated by both Kerberos and CRAM-MD5.

Using PAM, whenever a user requests an operation that involves changing or setting the user's identity (such as logging in to the system, invoking an 'r' command such as `rccp` or `rsh`, using `ftp` or `su`), a set of configurable modules are used to provide authentication, account management, credentials management, and session management. PAM employs runtime pluggable modules to provide authentication for system entry services.

These modules are broken into four types based on function:

- Authentication
- Account management
- Session management
- Password management

Each of the four service modules may be implemented as a shared-library object which can be referenced in the `pam.conf` configuration file.

A stacking feature is provided that enables user authentication through multiple services, as well as a password-mapping feature so users are not required to remember multiple passwords. PAM can be used to integrate login services with several authentication technologies, such as RSA, DCE, Kerberos, S/Key, and smart card-based systems.

Chapter 3

Traditional Solaris Authentication

Traditional Solaris 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 naming service, from which it is retrieved during the user login process. The traditional UNIX method of Solaris authentication, using `crypt(3c)`, is very popular and has been enhanced to use an LDAP directory as its data store.

Before proceeding with a detailed discussion on authentication, an understanding of `crypt(3c)` is useful. Some confusion arose due to a naming conflict with an application named `crypt`; the latter is a standard tool that ships with the Solaris OE 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)`,” as available to C programmers in the `libcrypt.a` library.

A more sophisticated authentication method based on public key technology was introduced with the NIS+ naming service. This 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 RPC mechanism, a network password is required.

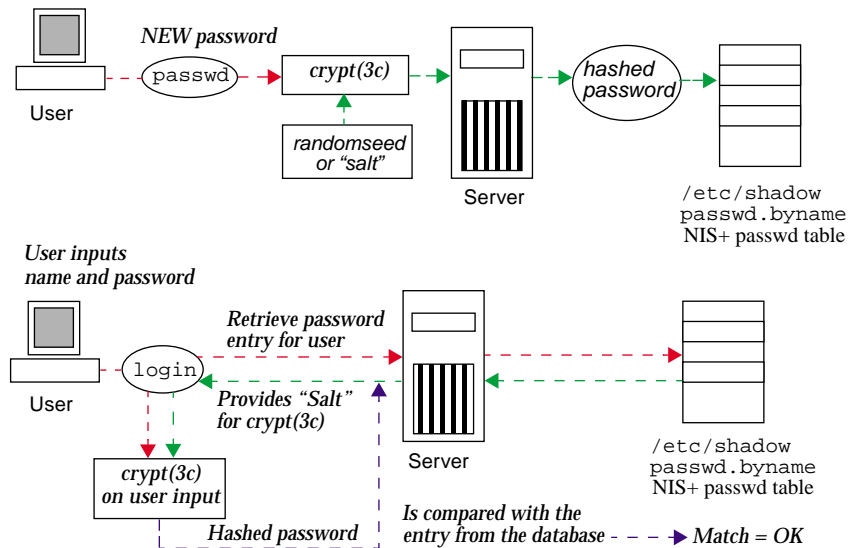
Realizing that new authentication models will continue to be developed over time, Sun created the PAM architecture, which allows additional methods to be added without disturbing existing ones. The PAM architecture and alternatives to traditional Solaris authentication are presented in the next section, “The Solaris 9 OE PAM Framework.”

How UNIX® Passwords Work

Passwords are created with the Solaris OE `passwd` command. This command prompts for a (new) password, which the user enters as a text string. In previous versions of 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 as a *salt string*, so that the result will be different each time the `passwd` command is run, even if the same text string is used.

When a user logs in, the Solaris `login` program challenges that 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 stored in the password database, the user is considered to have been authenticated.

Figure 3-1: How the UNIX password process works. 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 server that the user logs into, so an authentication server is not required. Since clear-text passwords are never stored or sent over the network, there is no reason to be concerned about eavesdroppers intercepting the password.

Since `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` format to another format that may be required by a different authentication method.

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, can provide significantly stronger authentication.

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

The PAM interface in the Solaris 9 OE can make it easier for IT administrators to deploy a variety of authentication technologies without modifying administrative commands such as login, telnet, and others. Administrators are able to select one or several 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 which can increase overall system security.

Chapter 4

PAM Components

This section details the components that make up PAM:

- Solaris 9 OE PAM framework
- PAM module types
- PAM configuration file update
- PAM service module update
- PAM password management extensions

Solaris 9 OE PAM Framework

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

The framework consists of four specific components: the PAM API which is presented to the application programs, the PAM framework which is responsible for implementing the API, the PAM service provider interface (SPI) that implements back-end functionality for the PAM API, and finally a configuration file `pam.conf` that 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:

- Flexible configuration policy
 - Per-application authentication policy
 - Choice of a default authentication mechanism for nonspecified applications
 - Multiple passwords on high-security systems
- Ease of use for the end user
 - No retyping of passwords if they are the same
 - Password mapping, whereby a single password can be used even if the password associated with separate authentication methods is different
 - Optional parameters passed to the services

Chapter 5

PAM Module Types

The PAM framework currently defines 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 functions they perform:

- *Authentication*: Provide user authentication and enables credentials to be set, refreshed, or destroyed.
- *Account management*: Check for password aging, account expiration, and access hour restrictions. Once a user is identified by the authentication modules, the account management modules determine whether the user can be given access.
- *Session management*: Primarily manage opening and closing a session. The modules can log activity, or clean up after the session is over. For example, the `unix_session` module implements and updates the `lastlog` file.
- *Password management*: Contain functionality that enables users to change their authentication tokens (usually passwords).

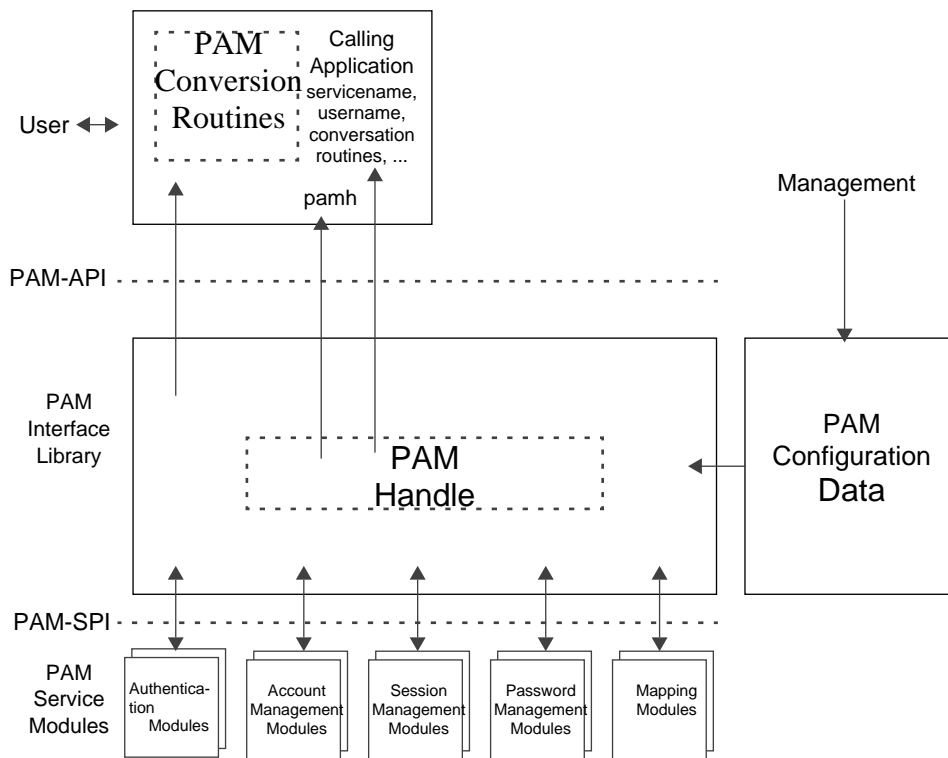
Stacking

PAM enables authentication by multiple methods through stacking. When a user is authenticated through PAM, multiple methods may be selected to fully identify the user. Depending on the configuration, the user can be prompted for passwords for each authentication method, so the user need not execute another command to be fully authenticated. The order in which the methods are selected is determined through the configuration file, `/etc/pam.conf`.

Note – Stacking may increase security risk, because the security of each mechanism is limited by the least-secure password method used in the stack.

The next section presents an architectural overview of the PAM framework.

Figure 5-2: The PAM Framework Architecture

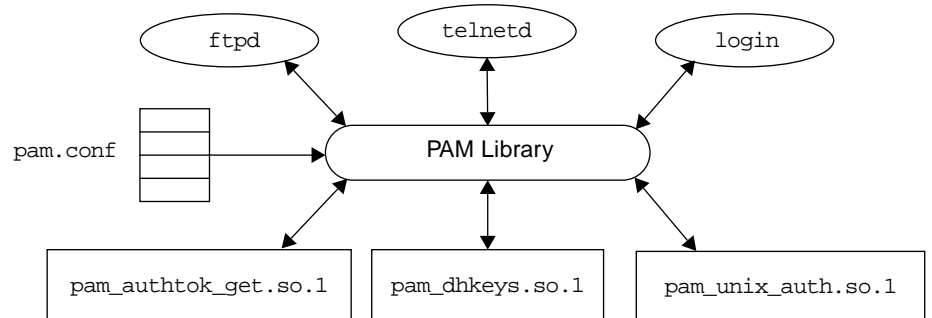


How PAM Works

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

FIGURE 3 illustrates how the applications, library, and modules relate to each other. 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.

Figure 5-3: The relationship between the PAM applications, library, and modules



PAM Modules

Each module implements 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 three modules are:

- **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 tasks, and not worry about obtaining information from the user.
- **pam_dhkeys** — Also supports authentication and password management. This module specifically deals with the establishment and modification of Diffie-Hellman keys which are used, for example, for Secure RPC calls (NIS+ and Secure NFS).
- **pam_auth_unix** — Can only be used for authentication. This module verifies a password entered by the user against any password repository specified in the `nsswitch.conf` using normal UNIX crypt(3c)-style password encryption.

For security, these files are required to be owned by `root` and 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.

Chapter 6

PAM Configuration File Update

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

Configuration File Syntax

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

```
service_name module_type control_flag module_path module_options
```

where:

`service_name` is the name of the service (for example, `ftp`, `login`, `telnet`)

`module_type` is the module type for the service

`module_type` is the module type for the service (`auth`, `account`, `session`, `password`)

`control_flag` determines the continuation or failure semantics for the module (see below)

`module_options` are the specific options passed to the service modules

Comments can be 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 1 Summarizes 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
rexed	/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
ssh	/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, 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:

required — This module must return `success` in order to achieve a successful result. If all 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 as `required`, all modules in the stack are still tried, but failure is returned. If none of the modules are flagged as `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 as `requisite`, an error is immediately returned to the application and no additional authentication is attempted. If the stack does not include prior modules labeled as `required` that failed, then the error from this module is returned. If an earlier module labeled as `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 users need to have permission associated with a specific mechanism to get their work done, do not label it as `optional`.

sufficient — If this module is successful, skip the remaining modules in the stack, even if they are labeled as `required`. The `sufficient` flag indicates that one successful authentication will be 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.

Generic pam.conf File

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

```
# PAM configuration
# Authentication management
#
login    auth required    pam_authtok_get.so.1
login    auth required    pam_unix_auth.so.1
login    auth required    pam_dial_auth.so.1
#
rlogin   auth sufficient  pam_rhosts_auth.so.1
rlogin   auth required    pam_authtok_get.so.1
rlogin   auth required    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
#
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     am_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 required     pam_authtok_store.so.1
```

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

1. When running `login`, authentication must succeed for the `pam_authtok_get`, the `pam_unix_auth`, and the `pam_dial_auth` modules.
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 the `rlogin 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 that require authentication utilize 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 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/$ISA` is prepended to the file name, where `$ISA` is expanded by the framework to contain the instruction set architecture of the executing machine (see `isainfo(1)`).

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)`).

The `use_first_pass` and `try_first_pass` options, which are supported by (for example) the `pam_ldap` module, allow reuse of the same password for authentication without retyping it.

If `login` specifies authentication through both `pam_unix_auth` and `pam_ldap`, then the user is prompted to enter a password for each module. In situations where the passwords are the same, the `use_first_pass` module option prompts for only one password and uses it to authenticate the user for both modules. If the passwords are different, the authentication fails.

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

If the `try_first_pass` module option is used instead, the `pam_ldap` module prompts for a second password if the passwords do not match, or an error is made.

Chapter 5

PAM Password Management Extensions

Before proceeding with a discussion of the new framework associated with PAM in the Solaris 9 OE, it is helpful to review what was available in the Solaris 8 OE PAM, and why it was important to change.

Customer feedback requested enhancements made to the PAM functionality in the Solaris OE. Changes included improving the mechanism that can be used to validate password structures, as well as adding the ability to change such things as number of characters, total password length, and so on.

In previous versions of `pam_unix`, this functionality was tightly coupled into a single, autonomous module, and local extensions could not be incorporated. It simply was not possible to extend part of the operations performed by this module.

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 which can be easily extended or replaced by modifying the `pam.conf` file.

For example, these modules are provided for password management:

other	password	required	pam_authtok_get.so.1
other	password	requisite	pam_authtok_check.so.1
other	password	required	pam_authtok_store.so.1

It has become quite trivial to extend or replace the standard password-strength checks by amending or replacing the `pam_authtok_check.so.1` module. Another possible capability would be to store the user's new password in repositories not currently supported by Sun by adding another module after the `pam_authtok_store.so.1` module, which updates the repositories as specified by the `nsswitch.conf` file. It would be a good idea to mark the `pam_authtok_store` module as requisite in that case, so that none of the repositories are updated if the first store module fails.

Chapter 6

How to Add a PAM Module

1. Become superuser.
2. Determine the control flags and other options that should be used.
3. Copy the new module to `/usr/lib/security`.
4. Set permissions so that the module file is owned by `root` and permissions are `555`.
5. Edit the PAM configuration file, `/etc/pam.conf`, and add this module to the appropriate services.

Verification

It is essential to do some testing before logging out, in case the configuration file is misconfigured. Test the modified service or the `other` configuration; run `rlogin`, `su`, and `telnet` if these have been changed. 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 verifying that the module has been added. However, it might be possible to restart the daemon using the appropriate `/etc/init.d/` script.

How to Prevent Unauthorized Access with PAM from Remote Systems

Remove the `rlogin auth rhosts_auth.so.1` entry from the PAM configuration file. This prevents reading the `~/rhosts` files during an `rlogin` session, therefore preventing 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 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`. Changing the PAM configuration file does not prevent the service from being started.

How to Initiate PAM Error Reporting

1. Edit the `/etc/syslog.conf` to add any of the following PAM error-reporting entries:

`auth.alert` – Messages about conditions that should be fixed immediately

`auth.crit` – Critical messages

`auth.err` – Error messages

`auth.info` – Informational messages

`auth.debug` – Debugging messages

2. If necessary, create the logfile as specified in the previous step.

3. Make `syslogd` reread its configuration file by sending it a `SIGHUP` signal:

```
# pkill -HUP syslogd
```

Example: Initiating PAM Error Reporting

The next example displays all alert messages on the console. Critical messages are mailed to `root`. Informational, and debug messages are added to the `/var/log/pamlog` file.

```
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 the message, and the message itself. The `pamlog` file is capable of logging a large amount of information.

Chapter 4

The PAM LDAP Module

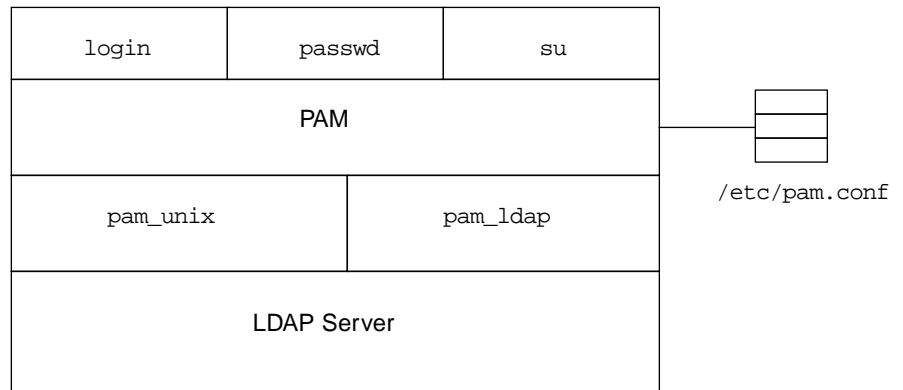
The PAM LDAP module (`pam_ldap`) was introduced in the Solaris 8 OE for use in conjunction with `pam_unix` for authentication and password management with an LDAP server. This module was written to support stronger authentication methods such as CRAM-MD5, in addition to the UNIX authentication capabilities provided by `pam_unix`. The PAM LDAP module should be employed in Solaris Native LDAP clients only. At the present time, `pam_ldap` provides support only for authentication and password management. Support for account management is expected to be added in the future.

The `pam_ldap` module should be stacked directly below the `pam_unix` module in the `/etc/pam.conf` configuration file. 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. A sample `/etc/pam.conf` 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 try_first_pass
# 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`, etc. can substitute for `login`. FIGURE 4 illustrates this structure.

Figure 4-1: 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.
- `use_first_pass`: For authentication, this option compares the password in the password database with the password entered when the user authenticated to the first auth module in the stack.

In the case of password management, the `use_first_pass` option compares the password in the password database with the user's old password entered to the first module in the stack. If these do not match (or no password was entered), then `pam_ldap` quits and does not prompt the user for the old password. It also attempts to use the new password entered to the first module in the stack as the new password for this module. If this attempt also fails, `pam_ldap` quits and does not prompt the user for a new password.

try_first_pass

Similar to the `use_first_option`, this option compares the password in the database with the password entered to the first module in the stack. But unlike the `use_first_pass` option, it prompts the user for a password if the passwords do not match.

- For password management, `try_first_pass` compares the password in the database with the old password entered to the first module in the stack; if they do not match, the user is prompted for the old password. `try_first_pass` also attempts to use the new password entered to the previous module; if that fails, `try_first_pass` prompts the user for a new password.

If the `/var/ldap/ldap_client_cache` file (which contains a list of LDAP servers, their transport addresses, and the authentication mechanisms used to access them) contains multiple authentication mechanisms specified for the `NS_LDAP_AUTH` parameter, then `pam_ldap` first attempts to authenticate with the first mechanism; if this fails, then `pam_ldap` goes to the next one and so forth, until it succeeds or runs out of mechanisms.

How PAM and LDAP Work

The authentication mechanism currently used in Solaris Native LDAP is SIMPLE 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 4.12 does not support authentication mechanisms, such as CRAM-MD5, which send only the digest over the wire. The tables below explain when the clear-text password is sent across the wire.

`pam_unix` Authentication

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 (which is why the proxy `passwd` is sent across the wire in clear text). It is important to note that the ACLs of the proxy agent allow this account to have read access to all user passwords, which is how this account can retrieve a user's password. 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 (see <Z_Xref>TABLE 2).

For updating passwords in `pam_unix`, the same comparison takes place as for authentication (since the user has to bind as the `dn`); then, the new password is sent over the wire in clear text (see <Z_XREF>TABLE 2).

TABLE 2 PAM Authentication

Authentication Mechanisms	<code>pam_unix</code>		<code>pam_ldap</code>	
SIMPLE	UP-No	PP-Yes	UP-Yes	PP-Yes
CRAM-MD5	UP-*	PP-No	UP-No	UP-No

TABLE 3 PAM update of password

Authentication Mechanisms	<code>pam_unix</code>		<code>pam_ldap</code>	
SIMPLE	UP-Yes (NP)	PP-*	UP-Yes	PP-*
CRAM MD5	UP-*	PP-*	UP-No	PP-*

`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. At present, using `pam_ldap` does not serve much purpose, since `pam_unix` is sufficient for basic LDAP authentication. The reason is twofold; first, `pam_ldap` is required for stronger authentication mechanisms such as CRAM-MD5; and second, `pam_ldap` was designed to be extended for future authentication mechanisms to be supported in future releases. For additional information, see the `pam_ldap` man page (if using `pam_ldap`) for the correct way to stack in `/etc/pam.conf`.

PAM abbreviations used in this discussion are listed in TABLE 3.

TABLE 4 PAM abbreviations

Abbreviation	Description
UP	User password
PP	Proxy agent password
NP	New password
*	Not applicable (at present)

The matrixes are easier to understand when a comparison is made 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 SHA, crypt, clear text, and more. The authentication mechanisms are SIMPLE or CRAM-MD5 (future authentication types include DIGEST and others).

Chapter 5

Overview of the PAM API and PAM-SPI

This section is a discussion of the PAM APIs, as well as examples of how to effectively write PAM modules when using the Solaris 9 OE.

By writing PAM modules, it is possible to extend the capability of the Solaris 9 OE authentication mechanisms in a number of ways. For example, a system administrator could prevent users from choosing new passwords that resemble their old passwords.

A PAM module is a shared object that is developed and written using the C programming language. The module is dynamically selected based on the contents of the `pam.conf` configuration file. This is an extremely efficient mechanism which enables the selection of the most appropriate authentication mechanism for a particular environment — without sacrificing functionality or requiring dependence on third-party or unsupported software.

Developing a PAM module is not as difficult as might be supposed. The PAM API comes with a rich set of features and is fairly intuitive to learn and use. Online documentation is available at the docs.sun.comSM site.

It is outside the scope of this paper to discuss every API and function. However, the most commonly used and well-known APIs are covered. The PAM APIs can be grouped into five functional categories:

- PAM framework
- Authentication
- Account management
- Session management
- Password management

These functions enable an application to invoke the PAM service modules and communicate information to PAM service modules.

PAM Framework Functions

```
pam_start()
pam_end()
```

The `pam_start()` and `pam_end()` functions are PAM transaction routines for establishing and terminating a PAM session. `pam_start()` takes as arguments the name of the application calling PAM, the name of the user to be authenticated, and the address of the callback conversation structure provided by the caller. It returns a handle for use with subsequent calls to the PAM library.

```
pam_get_item()
pam_set_item()
```

The `pam_get_item()` and `pam_set_item()` functions are routines that enable both applications and the PAM service modules to access and update common PAM information such as service name, user name, remote host name, and remote user name from the PAM handle. Table 4 lists some of these values.

TABLE 5 The values that may be manipulated by PAM functions

Item Name	Description
PAM_SERVICE	Service (application) name
PAM_USER	User name
PAM_RUSER	Remote user name
PAM_TTY	tty name
PAM_RHOST	Remote host name
PAM_CONV	<code>pam_conv</code> structure
PAM_AUTHTOK	Authenticated token
PAM_OLDAUTHTOK	Old authentication token

```
pam_getenv()
pam_getenvlist()
pam_putenv()
```

The `pam_getenv()`, `pam_getenvlist()`, and `pam_putenv()` functions enable calling application and PAM modules to set and retrieve environment variables for the user session that will be established with the `pam_open_session()` function.

```
pam_strerror()
```

The `pam_strerror()` function returns error status information.

PAM Authentication Functions

```
pam_authenticate()  
pam_setcred()
```

The `pam_authenticate()` function is called to verify the identity of the current user. The `pam_setcred()` function is called to set the credentials of the current process associated with the authentication handle supplied. Typically, this is done after the user has been authenticated.

Account Management Functions

```
pam_acct_mgmt()
```

This function is used to verify the authorization of the user to sign on. It will typically include checking for password and account expiration, valid login times, and so on.

Session Management Functions

```
pam_open_session()  
pam_close_session()
```

These functions are called at the initiation and termination of a PAM session. They can also support session auditing.

Password Management Functions

```
pam_chauthtok ()
```

This function is called to change the authentication token (password) associated with the user.

PAM-SPI Functions

The functions comprising the PAM-SPI are provided by the modules called by the PAM infrastructure and are grouped below on the basis of module type.

Authentication Module Functions

```
pam_sm_authenticate()  
pam_sm_setcred()
```

The `pam_sm_authenticate()` function is called to verify the identity of the current user as specified by the `PAM_USER` item.

The `pam_sm_setcred()` function is called to set the credentials of the current process associated with the authentication handle supplied. Typically, this is done after the user has been authenticated.

Note – A service module that is specified as `auth` must implement both interfaces.

Account Management Module Functions

```
pam_sm_acct_mgmt()
```

This function is used to verify the authorization of the user when signing on. It will typically include checking for password and account expiration, valid login times, and so on.

Session Management Module Functions

```
pam_sm_open_session()  
pam_sm_close_session()
```

These functions are called on the initiation and termination of a PAM session. They also support session auditing.

Password Management Module Functions

```
pam_sm_chauthtok()
```

This function is called to change the authentication token (password) associated with the user.

Note – For an understanding of the relationship between the various APIs, please refer to the PAM Framework Architecture documentation available at the docs.sun.com site.

Chapter 6

Writing the First PAM Module

The easiest way to get experience with writing PAM modules is to code one. The following example, `pam_compare.so.1`, is a standalone module for the PAM password stack. It enables a system administrator to prevent users from choosing a new password string that resembles an old password string.

The concept for this module is as follows: Each character in the user's new password is checked against a particular character that may have been present in the old password string. The module is based on the logic that there can be only a configurable number of matching characters between the old and new passwords. If the limit is exceeded, the new password will be rejected.

The configuration of the `pam_compare` module is accomplished by adding the following entries to the `/etc/pam.conf` file. These entries need to be placed before the `pam_authtok_store.so.1` definition.

Example of an entry:

```
other password required pam_authtok_get.so.1
other password requisite pam_authtok_check.so.1
other password requisite pam_compare.so.1 maxequal=4
other password required pam_authtok_store.so.1
```

Note – If the `maxequal` flag is not specified, then passwords will not be allowed to contain any shared characters.

Finally, the `pam_compare` module accepts two flags:

- `debug`: Turns on debugging messages through the `syslog` level `LOG_DEBUG`
- `maxequal=n`: Configures the maximum number of allowable shared characters

Source Code

The source code consists of three files: Makefile, C file, `pam_compare.c`, plus associated man pages. Other prerequisites are an ANSI C compiler and a make utility. All work is expected to be performed in the Solaris 9 OE.

Note – The supplied `makefile` creates only a 32-bit version of the module. Deployable modules should be compiled for both 32- and 64-bit operations.

Makefile

LISTING 1 shows the Makefile, which has been tested on Solaris 9. It is not necessary to modify this Makefile. Included are comments to explain what options are required to compile the PAM module successfully.

```

#ident "@(#)Makefile    1.3 12/11/01 SMI"
#
# Copyright (c) 001 by Sun Microsystems, Inc.
# All rights reserved.
#

## PAM Module: pam_compare.so.1

TOP=../
include $(TOP)common/Makefile.master

#####
#
LDLIBS=        -lpam -lc
#####
# The -G option passes the option to the link editor to produce
# a shared object rather than a dynamically linked executable.
#
# The -Kpic option produces Position independant Code
#
# The -z defs option makes sure all symbols are resolvable. If
# not, you will get a failure at the time the modules is mapped
# into memory. This is a very hard problem to debug, and using
# -zdefs makes sure it won't occur.

LDFLAGS=        -G -Kpic -z defs
#
#####
# The -D_REENTRANT is used when you are writing multithreaded
# code. This option must be used in the compile stage.
#
# The pam_modules *MUST*BE* multi-thread safe.
CFLAGS=        -D_REENTRANT

SRCS=    pam_compare.c

OUTDIR=lib-$(PLAT)

all: output_dir pam_compare.so.1

pam_compare.so.1: pam_compare.o
    $(CC) $(LDFLAGS) -o pam_compare.so.1 pam_compare.o $(LDLIBS)
    mv pam_compare.so.1 pam_compare.o $(OUTDIR)

```

```

lint: $(SRCS)
      lint -y $(CFLAGS) $(SRCS) $(LDLIBS)

```

pam_compare Source File

This section includes a discussion of the `pam_compare` source file, including information on how this PAM module is built. Understanding PAM modules is certainly best explained by an example, such as the following:

```

1 #include <security/pam_appl.h>
2 #include <security/pam_modules.h>
3 #include <stdarg.h>
4 #include <syslog.h>
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <string.h>

```

The interesting include files are `<security/pam_appl.h>` which contains the PAM error codes and structures used by the PAM API, and the `<security/pam_modules.h>`, which contains the PAM SPI definitions.

Since PAM modules simply cannot assume that the interaction with the user will be based on a simple terminal-like interface (for example, compare the `telnet` interface and `dtlogin` interface) `-printf` or `puts` cannot be used to communicate with the user. Therefore, the PAM framework introduces the concept of a conversation function.

This function is supplied to the module by the application. For any input/output interaction with the user, the application's conversation function is called. It is up to the application to make sure that the correct interface is used to communicate with the user. For example, `dtlogin` will draw an alert box to display messages to the user, while `telnetd` might simply perform a write on a socket.

```

9  /*
10   * Display a one-line message to the user
11   */
12  static void
13  pam_display(pam_handle_t *pamh, int style, char *fmt, ...)
14  {
15      struct pam_conv *pam_convp;
16      char buf[512];
17      va_list ap;
18      struct pam_message *msg;
19      struct pam_response *response = NULL;
20
21      va_start(ap, fmt);
22      vsnprintf(buf, sizeof (buf), fmt, ap);
23      va_end(ap);
24
25      if (pam_get_item(pamh, PAM_CONV, (void **)&pam_convp)
!= PAM_SUCCESS) {
26          syslog(LOG_ERR, "pam_compare: Can't get
PAM_CONV item");
27          return;
28      }
29
30      if ((pam_convp == NULL) || (pam_convp->conv == NULL)) {
31          syslog(LOG_ERR, "pam_compare: no conversation
function defined");
32          return;

```

```

33     }
34
35     msg = (struct pam_message *)calloc(1, sizeof (struct
pam_message));
36     if (msg == NULL) {
37         syslog(LOG_ERR, "pam_compare: out of memory");
38         return;
39     }
40
41     msg->msg_style = style;
42     msg->msg = buf;
43
44     (pam_conv->conv)(1, &msg, &response, pam_conv
->appdata_ptr);
45
46     if (response)
47         free(response);
48
49     free(msg);
50 }

```

The example module introduces a simple routine that can be used to display a one-line message to the user, utilizing the conversation function in the PAM handle.

Line 13 is the PAM handle `pam_handle_t`, `*pamh`, which is used to store the address of the handle.

Line 15, the `pam_conv` structure, `pam_conv`, contains the address of the conversation function provided by the application. The `pam_conv` structure is used by the authentication application for passing call-back function pointers and application data pointers. The underlying PAM service module invokes this function to output information to and retrieve input from the user.

Line 18, `pam_message`, is a message structure that is used to pass, prompt, error, or message any text information from the PAM services to the application or user. Note that it is the responsibility of the PAM service modules to localize the messages. The memory used by `pam_message` must be allocated and freed by the PAM modules.

Line 19, `pam_response`, is a message structure that is used to get the response back from the application or user. The storage used by `pam_response` must be allocated by the application and freed by the PAM modules.

```

25 if (pam_get_item(pamh, PAM_CONV, (void **)&pam_conv) !=
PAM_SUCCESS) {
26     syslog(LOG_ERR, "pam_compare: Can't get PAM_CONV item");
27     return;
28 }
29
30 if ((pam_conv == NULL) || (pam_conv->conv == NULL)) {
31     syslog(LOG_ERR, "pam_compare: no conversation function
defined");
32     return;
33 }
34
35 msg = (struct pam_message *)calloc(1, sizeof (struct
pam_message));
36 if (msg == NULL) {
37     syslog(LOG_ERR, "pam_compare: out of memory");
38     return;
39 }
40
41 msg->msg_style = style;

```

```

42 msg->msg = buf;
43
44 (pam_conv->conv)(1, &msg, &response, pam_conv->appdata_ptr);
45
46 if (response)
47     free(response);
48
49 free(msg);
50 }

```

Line 25 is the `pam_get_item()` function, which returns to the caller the PAM information for the `item_type` supplied. `item` is assigned the address of the requested item. The data within the item is valid until it is modified by a subsequent call to the `pam_set_item()` function.

```

58 static
59 int compare(char *old, char *new, int max)
60 {
61     unsigned char in_old[256];
62     int equal = 0;
63
64     memset(in_old, 0, sizeof (in_old));
65
66     while (*old)
67         in_old[*old++]++;
68
69     while (*new) {
70         if (in_old[*new])
71             equal++;
72         new++;
73     }
74
75     if (equal > max)
76         return (1);
77
78     return (0);
79 }

```

Lines 58-79 comprise a function that is used to compare the strings `old` and `new`. If more than “max” characters of `new` also appear in `old`, the function returns 1 (error). Otherwise it returns 0. This function interface can be characterized in the following manner:

- It is good programming practice to declare functions whose return value is `int`.
- The format of this function is as follows:
- `<data type> <function name>(parameter-type-list);`
- The function call and its argument list are expressions in the calling program. The arguments passed in this function call are the same data type as that which the called function was defined to accept.
- Control returns to the calling function when the closing brace “}” in the called function, or a `return`, is encountered.

```

87 int
88 pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc, const
char **argv)
89 {
90     int i;
91     int debug = 0;
92     int maxequal = 0;
93     int pam_err;

```

```

94     char *service;
95     char *user;
96     char *passwd; /* the newly typed password */
97     char *opasswd; /* the user's old password */
98
99     for (i = 0; i < argc; i++) {
100         if (strcmp(argv[i], "debug") == 0)
101             debug = 1;
102         else if (strncmp(argv[i], "maxequal=", 9) == 0)
103             maxequal = atoi(&argv[i][9]);
104     }

```

Line 88, `pam_sm_chauthtok ()`, is a function within the PAM-SPI module called by the PAM infrastructure – in particular, the module type Password Management Module Functions. This function is called in response to a call to the `pam_chauthtok ()` function, which in turn calls the PAM framework function `pam_chauthtok ()` modules which are listed in the PAM configuration.

Upon successful completion of the call, the authentication token of the user will be ready for change, or will be changed in accordance with the authentication scheme.

It is the responsibility of `pam_chauthtok ()` to determine if the new password meets certain strength requirements.

```

109         if ((flags & PAM_PRELIM_CHECK) == 0)
110             return (PAM_SUCCESS);
111
112         pam_err = pam_get_item(pamh, PAM_SERVICE, (void
113 **)&service);
114         if (pam_err != PAM_SUCCESS) {
115             syslog(LOG_ERR, "pam_compare: error getting
116 service item");
117             return (pam_err);
118         }
119         pam_err = pam_get_item(pamh, PAM_USER, (void **)&user);
120         if (pam_err != PAM_SUCCESS) {
121             syslog(LOG_ERR, "pam_compare: can't get user
122 item");
123             return (pam_err);
124         }

```

In line 109, the PAM framework invokes the password services twice. The first time the modules are invoked with the flag, `PAM_PRELIM_CHECK`. During this stage, the password modules should perform only preliminary checks (ping remote name services to see if they are ready for updates, for example).

Line 112 is the `pam_get_item` routine that allows both applications and PAM service modules to access and update common PAM information such as service name, user name, remote host name, and remote user name from the PAM handle.

```

129         if (user == NULL || service == NULL) {
130             syslog(LOG_ERR, "pam_compare: %s is NULL",
131                 user == NULL ? "PAM_USER" : "PAM_SERVICE");
132             return (PAM_SYSTEM_ERR);
133         }
134
135         pam_err = pam_get_item(pamh, PAM_AUTHTOK, (void
136 **)&passwd);
137         if (pam_err != PAM_SUCCESS) {
138             syslog(LOG_ERR, "pam_compare: can't get password
139 item");

```

```

138             return (pam_err);
139         }
140
141         pam_err = pam_get_item(pamh, PAM_OLDAUTHTOK, (void
**)&opasswd);
142         if (pam_err != PAM_SUCCESS) {
143             syslog(LOG_ERR, "pam_compare: can't get old
password item");
144             return (pam_err);
145         }

```

Line 131, PAM_USER and PAM_SERVICE, are the user name and program service name respectively.

Line 135 is the item type PAM_AUTHTOK, which is available only to the PAM service modules for security reasons. What actually happens here is that the pam_chauthtok () function clears the PAM_AUTHTOK and PAM_OLDAUTHTOK items in the PAM handle prior to returning to the calling application. The authentication module, account module, and session management module should treat PAM_AUTHTOK as the current authentication token.

Line 141 is the item type PAM_OLDAUTHTOK. It is available only to the PAM service modules for security reasons. The password management module treats PAM_OLDAUTHTOK as the current authentication token, and PAM_AUTHTOK as the new authentication token.

```

151         if (passwd == NULL) {
152             if (debug)
153                 syslog(LOG_DEBUG, "pam_compare: "
154                     "NULL password; cleared by another
module?");
155             return (PAM_AUTHTOK_ERR);
156         }

```

Lines 151-156 check the PAM_AUTHTOK and verify that is not a NULL value. If a NULL value is encountered, it can be assumed that a previous module has probably cleared the AUTHTOK because of a potential error.

```

164         if (opasswd == NULL)
165             return (PAM_SUCCESS);
166
167         if (compare(opasswd, passwd, maxequal)) {
168             pam_display(pamh, PAM_ERROR_MSG,
169                 "%s: Your old and new password can't share
more than %d "
170                 "characters.", service, maxequal);
171             syslog(LOG_AUTH|LOG_WARNING, "%s: pam_compare: "
172                 "rejected new password for %s", service,
user);
173
174             /*
175              * Clear the PAM_AUTHTOK item to prevent other
modules from
176              * continuing
177              */

```

```
178             pam_set_item(pamh, PAM_AUTHTOK, NULL);
179
180             return (PAM_AUTHTOK_ERR);
181         }
182
183         return (PAM_SUCCESS);
184     }
```

In lines 164-177, if `PAM_OLDAUTHOK` is `NULL`, `PAM_SUCCESS` is returned. This is possible when `root` executes `passwd` and there is no old password to compare to the new password.

Line 178, the `pam_set_item()`, is used to set the value of `PAM_USER`. This function enables the caller to update PAM information as needed.

Chapter 7

Conclusion

The PAM interface in the Solaris 9 OE provides a set of APIs that can be used by third-party applications to extend authentication capabilities. By using the PAM layer, applications authenticate to the server without needing to determine what authentication method has been chosen by the system administrator for any given client. In the Solaris 9 OE, PAM modules can now be constructed to support site-specific authentication requirements.

By plugging multiple, low-level authentication mechanisms into applications at runtime, PAM integrates them with a single, high-level API. These authentication mechanisms, which may be standalone operating system or network mechanisms, are encapsulated as dynamically loadable, shared-software modules. These modules may be installed independent of applications or executed by applications, depending on the system configuration.

In an environment where there is an LDAP directory server, either `pam_unix` or `pam_ldap` can be used to authenticate users. Because of its increased flexibility and support of stronger authentication methods, the use of `pam_ldap` is recommended. For organizations using the Solaris 9 OE, which offers Native LDAP for naming and directory services, `pam_ldap` offers an ideal way to extend the authentication capabilities.

Using the PAM interface in the Solaris OE offers administrators greater flexibility and maintainability in deploying authentication mechanisms throughout the network. Administrators can choose a default authentication method, and require more secure mechanisms as necessary.

In addition, users may find that authentication is easier when using a PAM-enabled network. PAM can be used to enable single sign-on, overcoming the need to remember multiple passwords when logging into multiple network applications and services.

It is the combination of flexibility and ease of use that makes the PAM interface in the Solaris 9 OE an effective way to raise the overall security in a network environment. PAM is part of an overall security solution from Sun. Increasing security can help businesses protect their IT assets while improving the availability of their network infrastructure.

For more information on Sun security products, please see www.sun.com/security.

Copyright 2002 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303 U.S.A. All rights reserved.

This product or document is protected by copyright and distributed under licenses restricting its use, copying, distribution, and decompilation. No part of this product or document may be reproduced in any form by any means without prior written authorization of Sun and its licensors, if any. Third-party software, including font technology, is copyrighted and licensed from Sun suppliers.

Parts of the product may be derived from Berkeley BSD systems, licensed from the University of California. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Solaris, and docs.sun.com are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The OPEN LOOK and Sun™ Graphical User Interface was developed by Sun Microsystems, Inc. for its users and licensees. Sun acknowledges the pioneering efforts of Xerox in researching and developing the concept of visual or graphical user interfaces for the computer industry. Sun holds a non-exclusive license from Xerox to the Xerox Graphical User Interface, which license also covers Sun's licensees who implement OPEN LOOK GUIs and otherwise comply with Sun's written license agreements.

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227-14(g)(2)(6/87) and FAR 52.227-19(6/87), or DFAR 252.227-7015(b)(6/95) and DFAR 227.7202-3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2002 Sun Microsystems, Inc., 901 San Antonio Road, Palo Alto, California 94303 États-Unis. Tous droits réservés.

Ce produit ou document est protégé par un copyright et distribué avec des licences qui en restreignent l'utilisation, la copie, la distribution, et la décompilation. Aucune partie de ce produit ou document ne peut être reproduite sous aucune forme, par quelque moyen que ce soit, sans l'autorisation préalable et écrite de Sun et de ses bailleurs de licence, s'il y en a. Le logiciel détenu par des tiers, et qui comprend la technologie relative aux polices de caractères, est protégé par un copyright et licencié par des fournisseurs de Sun.

Des parties de ce produit pourront être dérivées des systèmes Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux États-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, et Solaris sont des marques de fabrique ou des marques déposées de Sun Microsystems, Inc. aux États-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux États-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

L'interface d'utilisation graphique OPEN LOOK et Sun™ a été développée par Sun Microsystems, Inc. pour ses utilisateurs et licenciés. Sun reconnaît les efforts de pionniers de Xerox pour la recherche et le développement du concept des interfaces d'utilisation visuelle ou graphique pour l'industrie de l'informatique. Sun détient une licence non exclusive de Xerox sur l'interface d'utilisation graphique Xerox, cette licence couvrant également les licenciés de Sun qui mettent en place l'interface d'utilisation graphique OPEN LOOK et qui en outre se conforment aux licences écrites de Sun.

CETTE PUBLICATION EST FOURNIE "EN L'ETAT" ET AUCUNE GARANTIE, EXPRESSE OU IMPLICITE, N'EST ACCORDEE, Y COMPRIS DES GARANTIES CONCERNANT LA VALEUR MARCHANDE, L'APTITUDE DE LA PUBLICATION A REpondre A UNE UTILISATION PARTICULIERE, OU LE FAIT QU'ELLE NE SOIT PAS CONTREFAISANTE DE PRODUIT DE TIERS. CE DENI DE GARANTIE NE S'APPLIQUERAIT PAS, DANS LA MESURE OU IL SERAIT TENU JURIDIQUEMENT NUL ET NON AVENU.



Please
Recycle



Adobe PostScript

Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, CA 94303-4900 USA Phone 800 786-7638 or +1 512 434-1577 Web sun.com



We make the net work.

Sun Worldwide Sales Offices: Africa (North, West and Central) +33-13-067-4680, Argentina +5411-4317-5600, Australia +61-2-9844-5000, Austria +43-1-60563-0, Belgium +32-2-704-8000, Brazil +55-11-5187-2100, Canada +905-477-6745, Chile +56-2-3724500, Colombia +571-629-2323, Commonwealth of Independent States +7-502-935-8411, Czech Republic +420-2-3300-9311, Denmark +45 4556 5000, Egypt +202-570-9442, Estonia +372-6-308-900, Finland +358-9-525-561, France +33-134-03-00-00, Germany +49-89-46008-0, Greece +30-1-618-8111, Hungary +36-1-489-8900, Iceland +354-563-3010, India-Bangalore +91-80-2298989/2295454; New Delhi +91-11-6106000; Mumbai +91-22-697-8111, Ireland +353-1-8055-666, Israel +972-9-9710500, Italy +39-02-641511, Japan +81-3-5717-5000, Kazakhstan +7-3272-466774, Korea +822-2193-5114, Latvia +371-750-3700, Lithuania +370-729-8468, Luxembourg +352-49 11 33 1, Malaysia +603-21161888, Mexico +52-5-258-6100, The Netherlands +00-31-33-45-15-000, New Zealand-Auckland +64-9-976-6800; Wellington +64-4-462-0780, Norway +47 23 36 96 00, People's Republic of China-Beijing +86-10-6803-5588; Chengdu +86-28-619-9333; Guangzhou +86-20-8755-5900; Shanghai +86-21-6466-1228; Hong Kong +852-2202-6688, Poland +48-22-8747800, Portugal +351-21-4134000, Russia +7-502-935-8411, Singapore +65-6438-1888, Slovak Republic +421-2-4342-94-85, South Africa +27 11 256-6300, Spain +34-91-596-9900, Sweden +46-8-631-10-00, Switzerland-German 41-1-908-90-00; French 41-22-999-0444, Taiwan +886-2-8732-9933, Thailand +662-344-6888, Turkey +90-212-335-22-00, United Arab Emirates +9714-3366333, United Kingdom +44 0 1252 420000, United States +1-800-555-9SUN or +1-650-960-1300, Venezuela +58-2-905-3800