



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

---

*Michael Haines, Sun™ ONE Directory Server Group  
Joep Vasseur, Solaris™ Security Technology Group  
SunBluePrints™ OnLine—October 2002*



<http://www.sun.com/blueprints>

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

Part No. 816-7670-10  
Revision A, 10/03/02  
Edition: October 2002

Copyright 2002 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95054 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 United States and other countries, exclusively licensed through X/Open Company, Ltd.

Sun, Sun Microsystems, the Sun logo, Sun BluePrints, Sun ONE, iPlanet, and Solaris are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the US 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., 4150 Network Circle, Santa Clara, Californie 95054 Etats-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 enregistrée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company Ltd.

Sun, Sun Microsystems, le logo Sun, Sun BluePrints, Sun ONE, iPlanet, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-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 Etats-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

# Extending Authentication in the Solaris™ 9 Operating Environment Using PAM: Part II

---

This article is part two of a two-part series. Part one, published in the September issue of Sun BluePrints™ Online, detailed the benefits of Pluggable Authentication Module (PAM), its components, and the PAM Lightweight Directory Access Protocol (LDAP) module. This article details the PAM application programming interface (API) and the PAM service provider interface (SPI). Also included are procedures on how to effectively write PAM modules when using the Solaris™ 9 Operating Environment (Solaris 9 OE).

By writing these PAM service modules, it is possible to extend the capability of the Solaris 9 OE authentication mechanisms in a number of different ways. 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 system without worrying about what authentication method has been chosen by the system administrator for any given client. For example, it is possible for a system administrator to prevent users from choosing new passwords that resemble their old password.

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(4)` configuration file. This is an extremely efficient mechanism that enables the selection of the most appropriate authentication mechanism for a particular environment—without sacrificing functionality or the need to depend on third-party or unsupported software.

Developing a PAM module is not as difficult as you might think. The PAM API has a rich set of features, and is fairly intuitive to learn and use. Online documentation is also available (see <http://docs.sun.com>).

This article addresses the following topics:

- Details of the PAM API
- Details of the PAM SPI
- Details on how to write a PAM service module
- Details on how to test a PAM service module

---

## The PAM API

As presented in Part I, PAM is composed of a number of components including:

- PAM API (`pam(3PAM)`) used by applications to perform authentication and authentication token (password) changes.
- PAM framework used to export the API.
- PAM SPI (`pam_sm(3PAM)`) used by the PAM framework to call on behalf of an API request.
- PAM service modules (`pam_*(5)`) used to export the PAM SPI.

The PAM framework, includes the PAM library (`libpam.so.1`), which consists of an interface library and multiple authentication service modules that are the layer implementing the PAM API.

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

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

These functions enable an application to invoke the PAM service modules and to communicate information to these modules. The functions are described in the following sections.

# PAM Framework Functions

These framework functions are PAM transaction routines for establishing and terminating a PAM session.

- `pam_start()` function 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_end()` function is called to terminate the authentication transaction identified by `pamh` and to free any storage area allocated by the authentication module. The argument, `status`, is passed to the `cleanup()` function stored within the PAM handle, and is used to determine what module-specific state must be purged.
- `pam_get_item()` and `pam_set_item()` functions are PAM 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, remote user name, and so on, from the PAM handle.

TABLE 1 details the items that may be manipulated by these functions.

**TABLE 1** Setting PAM Items

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_OLDAUTHOK	Old authentication token
PAM_REPOSITORY	Specifies which repository is to be updated.
PAM_USER_PROMPT	Prompt the module should use if asking for a username.

- `pam_getenv()`, `pam_getenvlist()`, and `pam_putenv()` functions enable applications and PAM modules to set and retrieve environment variables that are to be used for the user session.
- `pam_strerror()` function returns a textual representation of a PAM error, much like the `strerror(3c)` error.

## PAM Authentication Functions

These authentication functions are used to authenticate the user and the current process.

- `pam_authenticate()` function called to verify the identity of the current user.
- `pam_setcred()` function called to set the credentials of the current process associated with the authentication handle supplied.

Typically, this process is done after the user has been authenticated (after the `pam_authenticate()` function succeeds).

## Account Management Function

This account management function is used to validate the users account information. It typically includes checking for password and account expiration, valid login times, and etc.

- `pam_acct_mgmt()`

## Session Management Functions

These session management functions are called on the initiation and termination of a login session.

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

The `pam_unix_session` module implements these calls to update the `/var/adm/lastlog` information. These functions can also support session auditing.

## Password Management Function

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

- `pam_chauthtok()`

---

# The PAM SPI

The PAM service modules are a set of dynamically loadable objects invoked by the PAM SPI to provide a particular type of user authentication. The functions comprising the PAM SPI are provided by the modules called by the PAM infrastructure, and are grouped, in the following sections, on the basis of the module type.

## Authentication Module Functions

These authentication module functions are used to authenticate the user and the current process.

- `pam_sm_authenticate()` module function is called to verify the identity of the current user, as specified by the `PAM_USER` item.
- `pam_sm_setcred()` module function is called to set the credentials of the current process associated with the authentication handle supplied. Typically, this process is done after the user has been authenticated.

---

**Note** – A service module that is specified as `auth must` implement both interfaces. If the module has no credentials to set, the `pam_sm_setcred` function should return the `PAM_IGNORE` value.

---

## Account Management Module Function

This account management module function is used to validate the account of the user when signing on. It is meant to check for password and account expiration, valid login times, and etc.

- `pam_sm_acct_mgmt()`

## Session Management Module Functions

These session management module functions are called on the initiation and termination of a login session.

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

## Password Management Module Function

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

- `pam_sm_chauthtok()`

---

**Note** – For an understanding of the relationship between the different APIs, please refer to the PAM Framework Architecture documentation available at <http://docs.sun.com>.

---

---

## Writing a PAM Service Module

The easiest way to gain experience writing PAM service modules is to code one. The following example, `pam_compare.so.1`, is a stand-alone module for the PAM password stack. It enables a system administrator to prevent users from choosing a new password string that resembles their old password string.

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

The configuration of the `pam_compare` module is accomplished by adding the following entries to the `/etc/pam.conf(4)` file. These entries need to be placed before the `pam_authtok_store.so.1` definition.

Here is an example of such 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
```

The `pam_compare` module accepts two flags:

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

If the `maxequal` flag is not specified, old and new passwords will not be allowed to contain any shared characters.

## Source Code

The source code (available on <http://www.sun.com/solutions/blueprints/tools/index.html>) consists of a number of files. These files are several makefiles, the C-source `pam_compare.c`, and the associated man page. Prerequisites are an ANSI C compiler and the `make` utility found in the `/usr/ccs/bin/make` utility. All work is performed in the Solaris 9 OE.

---

**Note** – The supplied makefiles create both 32-bit and 64-bit versions of the module. Deployable modules should be compiled for both 32- and 64-bit operation because you cannot make assumptions about whether the application using these modules is a 32-bit or 64-bit application.

---

## Makefiles

A combination of makefiles that have been tested on the Solaris 9 OE are provided. You should not need to modify these files. Included in the makefiles are comments that explain what options are required to compile the PAM module. The following code box is the top-level makefile.

```
SUBDIRS=      sparc sparcv9

all          :=      TARGET= all
clean       :=      TARGET=clean

all clean:    $(SUBDIRS)

$(SUBDIRS):  FRC
              @cd $@; pwd; $(MAKE) $(TARGET)

FRC:
```

It contains the various implemented make-targets and a list of subdirectories (conveniently named after the architectures that the modules are being built for). This makefile descends into the architecture subdirectories in order to build the given target.

Each of the architecture subdirectories contain a makefile with declarations that are specific to the target architecture together with an `include` statement that includes the makefile with declarations common for all architectures, named `Makefile.common`.

The contents of the `sparc/Makefile` file are:

```
include ../Makefile.common
```

The contents of the `sparcv9/Makefile` file are:

```
include ../Makefile.common

CFLAGS += -xarch=v9
LDFLAGS += -xarch=v9
```

These extra flags instruct the compiler to generate a 64-bit code.

The makefile containing the architecture-independent declarations (Makefile.common) contains the actual target definitions.

```
# -D_REENTRANT is used when writing multi-threaded code. It enables
# multi-thread-safe declarations from the system header files.
# -Kpic causes the compiler to generate code that is suitable for
# dynamic loading into an applications address space.

CFLAGS=          -D_REENTRANT -Kpic

#
# The -G option tells the linker to generate a shared object.
#
# The -z defs option forces a fatal error if any undefined
# symbols remain at the end of the link-phase (see ld(1))

LDFLAGS=        -G -z defs
LDLIBS=         -lpam -lc

VPATH=  ..
SRC=    pam_compare.c
OBJ=    $(SRC:.c=.o)

all: pam_compare.so.1

pam_compare.so.1: $(OBJ)
    $(CC) $(LDFLAGS) -o $@ $(OBJ) $(LDLIBS)

clean:
    $(RM) $(OBJ) pam_compare.so.1
```

There are four Makefiles:

```
./Makefile
./Makefile.common
./sparc/Makefile
./sparcv9/Makefile
```

The actual `compile` commands are executed in the architecture-specific directory, so remember to instruct `make` to look for sources in the parent directory (`VPATH`). The object- and resulting the library command files are placed in the architecture-specific directory.

## pam\_compare Source File

This section details the `pam_compare.c` source file, including information on how this specific PAM module is built. Examples present the best way to explain how to write your own PAM service modules, such as the following:

```
1 /*
2  * Copyright 2002 Sun Microsystems, Inc. All rights reserved.
3  * Use is subject to license terms.
4  */
5
6 #pragma ident    "@(#)pam_compare.c    1.1    02/09/03 SMI"
7
8 #include <stdarg.h>
9 #include <syslog.h>
10 #include <stdio.h>
11 #include <stdlib.h>
12 #include <string.h>
13 #include <security/pam_appl.h>
14 #include <security/pam_modules.h>
15
```

In order to write a PAM module, you need to include the `security/pam_appl.h` file, which contains PAM error codes and structures used by the PAM API, and the `security/pam_modules.h` file, which contains the PAM SPI prototypes.

Because PAM service modules can't assume that the interaction with the user is based on a simple terminal-like interface (for example, compare the `telnet` interface and `dtlogin` interface)—the command `printf` or the function `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 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, the `dtlogin` interface draws an alert box to display messages to the user, while the `telnetd` interface might simply perform a `write` command on a socket.

The following example module introduces a simple routine (`pam_display()`) that is used to display a one-line message to the user, using the conversation function present in the PAM handle.

```
16 /*
17  * Display a one-line message to the user
18  */
19 static void
20 pam_display(pam_handle_t *pamh, int style, char *fmt, ...)
21 {
22     struct pam_conv *pam_convp;
23     char buf[512];
24     va_list ap;
25     struct pam_message *msg;
26     struct pam_response *response = NULL;
27
28     va_start(ap, fmt);
29     (void)vsnprintf(buf, sizeof (buf), fmt, ap);
30     va_end(ap);
31
32     if (pam_get_item(pamh, PAM_CONV, (void **)&pam_convp)
33         != PAM_SUCCESS) {
34         syslog(LOG_ERR, "pam_compare: Can't get PAM_CONV
35             item");
36         return;
37     }
38 }
39
40 (continued on the next page)
```

```

(continued from the previous page)
37     if ((pam_conv == NULL) || (pam_conv->conv == NULL)) {
38         syslog(LOG_ERR, "pam_compare: no conversation
           function defined");
39         return;
40     }
41
42     msg = (struct pam_message *)calloc(1, size of (struct
           pam_message));
43     if (msg == NULL) {
44         syslog(LOG_ERR, "pam_compare: out of memory");
45         return;
46 }
47
48     msg->msg_style = style;
49     msg->msg = buf;
50
51 (pam_conv->conv)(1, &msg, &response, pam_conv->appdata_ptr);
52
53     if (response)
54         free(response);
55
56     free(msg);
57 }

```

In Line 25, the `pam_message` is the structure used to pass a prompt, an error message, or an informational message from the PAM service modules to the application. The application displays the message to the user. Note that it is the responsibility of the PAM service modules to localize the messages. The memory used by the `pam_message` must be allocated and freed by the PAM service modules.

In Line 26, the `pam_response` is a structure used to receive information back from the user. The storage used by the `pam_response` is allocated by the application, and should be freed by the PAM service modules.

Line 32 obtains the conversation structure, `pam_conv`, to retrieve the address of the function needed to call and display the information to the user. The `pam_conv` structure contains two members as illustrated in the following:

```
struct pam_conv {
    int (*conv)(int, struct pam_message **, struct pam_response **,
               void *);
    void *appdata_ptr;
};
```

In Line 48, PAM defines several message styles: `PAM_PROMPT_ECHO_ON`, `PAM_PROMPT_ECHO_OFF`, `PAM_ERROR_MSG`, and `PAM_TEXT_INFO`. Even though this basic routine only deals with displaying messages (it doesn't deal with input from the user), the `style` attribute is parameterized as an example.

In Line 49, the application conversation function receives an array of `pam_message` structures. The function pointer `conv` contains the address of the input/output function that the service module uses to interact with the user. The extra `appdata_ptr` element is private to the application. The service module is supposed to supply this pointer when calling the conversation function. Because only a single message is displayed, only one element initializes and passes its address to the conversation function.

Line 51 transfers control to the application conversation function. An array of messages containing just one element are passed. Although the application should not collect any responses (this function only displays information), you should still check for responses and free any memory allocated by the application's conversation function (see line 53—54).

Line 56 frees the space allocated to pass the message to the conversation function.

```

58
59 /*
60  * int compare(old, new, max)
61  *
62  * compare the strings old and new. If more than "max"
63  * characters of new
64  * also appear in old, return 1 (error). Otherwise return
65  * 0.
66  */
67 static
68 int compare(unsigned char *old, unsigned char *new, int max)
69 {
70     unsigned char in_old[256];
71     int equal = 0;
72
73     (void)memset(in_old, 0, sizeof (in_old));
74
75     while (*old)
76         in_old[*old++]++;
77
78     while (*new) {
79         if (in_old[*new])
80             equal++;
81         new++;
82     }
83
84     if (equal > max)
85         return (1);
86
87     return (0);
88 }

```

Lines 65—86 define a function used to compare the strings `old` and `new`. If more than “max” characters of the `old` string also appear in the `new` string, return 1 (error). Otherwise, return 0. Use this function in the PAM module to determine whether the new password chosen by the user is acceptable.

```
88  /*
89  * int pam_sm_chauthtok(pamh, flags, argc, argv)
90  *
91  * Make sure the old and the new password don't share too many
92  * characters.
93  */
94  int
95  pam_sm_chauthtok(pam_handle_t *pamh, int flags, int argc,
96  const char **argv)
97  {
98      int i;
99      int debug = 0;
100     int maxequal = 0;
101     int pam_err;
102     char *service;
103     char *user;
104     char *passwd; /* the newly typed password */
105     char *opasswd; /* the user's old password */
106
107     for (i = 0; i < argc; i++) {
108         if (strcmp(argv[i], "debug") == 0)
109             debug = 1;
110         else if (strncmp(argv[i], "maxequal=", 9) == 0)
111             maxequal = atoi(&argv[i][9]);
112     }
113
114     if (debug)
115         syslog(LOG_DEBUG, "pam_compare: entering
116         pam_sm_chauthtok");
```

Line 95 defines the `pam_sm_chauthtok()` function, part of the PAM SPI definition. Once the module is configured in the `/etc/pam.conf(4)` file, the PAM framework calls this routine when the application calls the `pam_chauthtok()` function. This function is the entry point of the module, but it is always through the PAM framework.

Start the module by interpreting the arguments that have been specified in the `/etc/pam.conf(4)` file. Then accept the two different arguments, or *flags* in PAM terminology, `debug` and `maxequal`. Any arguments specified in the configuration file are handed over by the PAM framework. This is just like the command-line arguments presented to an application's `main()` function receiving an argument count (`argc`), and an array of character pointers (`argv`).

Lines 113—114 log some extra messages if the `debug` flag is specified.

The PAM password management stack is different from the other PAM stacks in that it invokes each of the service modules twice. The first time a service module is invoked, the PAM framework sets the `PAM_PRELIM_CHECK` bit in the `flags`, the second time the service module is invoked, the framework sets the `PAM_UPDATE_AUTHTOK` bit in the `flags` file.

The `PAM_PRELIM_CHECK` flag indicates to the service module that the module should not start updating any passwords, but should make sure that all the module's prerequisites for password updates are met.

The module, in this example, is quite simple because it does not depend on any other services. Be aware that more complex modules might differ in this regard. For example, a module that updates a password in an LDAP server needs to check that the LDAP server is up and running before trying to update any information. This preliminary checking avoids the chance of one module updating the password in a Network Information Service (NIS) server, while the next module fails to update the password in an unavailable LDAP server. Such a scenario would leave one module with two different passwords, and complicates future logins.

Because the PAM framework invokes each of the service modules twice, when should you perform a check? You do have three options:

1. At the time you are called with the `PAM_PRELIM_CHECK` flag.
2. At the time you are called with the `PAM_UPDATE_AUTHTOK` flag.
3. Both times

Because nothing that is important to the module (old and new passwords) has changed between the first and the second time around, option 3 is not useful.

In option 1, using the `PAM_PRELIM_CHECK` flag, the password service only performs preliminary checks. No passwords should be updated. In option 2, using the `PAM_UPDATE_AUTHTOK` flag, the password service updates passwords.

---

**Note** – PAM\_PRELIM\_CHECK and PAM\_UPDATE\_AUTHTOK cannot be set at the same time.

---

Performing your checks during the preliminary round makes sure that no module updates any passwords without your consent, for example, without this module returning the PAM\_SUCCESS value.

The PAM\_IGNORE value is returned during the second round of calls (the PAM\_PRELIM\_CHECK flag is not set) because you do not actually contribute to the updating process. For example:

```
116         if ((flags & PAM_PRELIM_CHECK) == 0)
117             return (PAM_IGNORE);
```

It is important to note the difference between returning the PAM\_SUCCESS value and the PAM\_IGNORE value. In this case, if the PAM\_SUCCESS value is returned, it might cause the complete PAM stack to succeed, even if all other modules returned the PAM\_IGNORE value. You do not want the password management stack to succeed without performing any work, so you want to return the PAM\_IGNORE value.

```

119 pam_err = pam_get_item(pamh, PAM_SERVICE, (void **)&service);
120 if (pam_err != PAM_SUCCESS) {
121 syslog(LOG_ERR, "pam_compare: error getting service item");
122 return (pam_err);
123 }
124
125 pam_err = pam_get_item(pamh, PAM_USER, (void **)&user);
126 if (pam_err != PAM_SUCCESS) {
127 syslog(LOG_ERR, "pam_compare: can't get user item");
128 return (pam_err);
129 }
130
131 /*
132 * Make sure "user" and "service" are set. Otherwise it might
133 * be misconfigured and dump core when these items are for used
134 * for error reporting.
135 */
136 if (user == NULL || service == NULL) {
137 syslog(LOG_ERR, "pam_compare: %s is NULL",
138 user == NULL ? "PAM_USER" : "PAM_SERVICE");
139 return (PAM_SYSTEM_ERR);
140 }

```

Although you are only interested in the old and the new passwords (the PAM items `PAM_OLDAUTHOK` and `PAM_AUTHOK`), you must also obtain the two other PAM items: `PAM_SERVICE` (the name of the application) and `PAM_USER` (the login name of the user whose password that is about to be updated). This information is used to create error messages.

Note that there is a distinction between the `pam_get_item()` function returning a value other than `PAM_SUCCESS`, and the item not being set by the application (for example, `user == NULL`). The first error indicates a problem with the PAM stack (probably a misconfiguration), while the second condition indicates a malfunctioning application because the user's password can not be changed without setting the `PAM_USER` item.

```

142     pam_err = pam_get_item(pamh, PAM_AUTHTOK, (void
143     **) &passwd);
143     if (pam_err != PAM_SUCCESS) {
144         syslog(LOG_ERR, "pam_compare: can't get password
145         item");
145         return (pam_err);
146     }
147
148     pam_err = pam_get_item(pamh, PAM_OLDAUTHTOK, (void
149     **) &opasswd);
149     if (pam_err != PAM_SUCCESS) {
150         syslog(LOG_ERR, "pam_compare: can't get old
151         password item");
151         return (pam_err);
152     }
153
154     /*
155     * PAM_AUTHTOK should be set. If it is NULL, the check
156     * can't be performed
157     * so this module should be ignored (another module
158     * will probably fail)
159     */
158     if (passwd == NULL) {
159         if (debug)
160             syslog(LOG_DEBUG, "pam_compare:
161             PAM_AUTHTOK = NULL.");
161         return (PAM_IGNORE);
162     }
163
164     /*
165     (continued on next page)

```

```

(continued from previous page)
165      * If PAM_OLDAUTHOK is NULL (possible i.e. when root
        executes passwd)
166      * there isn't an old password to compare the new
        with. return

167      * PAM_SUCCESS since there is no reason to reject the
        new password.
168      */
169
170      if (opasswd == NULL)
171          return (PAM_SUCCESS);

```

Lines 142 and 148, respectively, retrieve the old and new password so you can compare them. Before invoking the `compare()` routine, make sure that both passwords are actually set, because there might be valid reasons for these passwords to not be set.

Line 158 checks the new password. If, for whatever reason, the password is not set, you cannot perform the check. If you cannot perform the check, the result of the module should not contribute to the overall result of the password stack, as configured in `/etc/pam.conf(4)` file, so the `PAM_IGNORE` value must be returned.

You can choose to return the `PAM_AUTHOK_ERR` value to force an error of the overall stack, but, in general, you should return a `PAM_IGNORE` value if your module somehow results in a nonoperation. This is the case for this module, if you cannot perform a check.

---

**Note** – Please remember this about return values; besides the `PAM_SUCCESS` value being returned on a successful completion, error return values as described in the  `pam(3PAM)` function may also be returned. For example, `PAM_AUTHOK_ERR` indicates an authentication token manipulation error.

---

Also check the old password to see if it is set. There is only one valid reason why the old password might be not set, and that is if the system administrator sets the password for an ordinary user. If that is the case, then the password program doesn't always ask for the old password. So if the old password is not set, accept the new password and return the `PAM_SUCCESS` value.

```
173         if (compare((unsigned char *)opasswd, (unsigned char
174                     *)passwd,
175                     maxequal)) {
176             pam_display(pamh, PAM_ERROR_MSG,
177                         "%s: Your old and new password can't share
178                         more than %d "
179                         "characters.", service, maxequal);
180             syslog(LOG_WARNING, "%s: pam_compare: "
181                 "rejected new password for %s", service,
182                 user);
183
184             return (PAM_AUTHOK_ERR);
185         }
186
187         return (PAM_SUCCESS);
188     }
```

Now that both passwords have been retrieved, and you have made sure they are both set, call the `compare()` function to see if the new password should be accepted. If the `compare()` function reports success (returns 0), return the `PAM_SUCCESS` value to the PAM framework (see line 184).

If however, the `compare` routine reports failure (the old and new password share more than `maxequal` characters), inform the user of the problem (using the simple one-line-message-display-routine, `pam_display()`), and log a system message to record this failure (see lines 178—179). Exit the PAM module by returning the `PAM_AUTHOK_ERROR` value which causes the PAM password stack to fail.

Remember that all these checks are done in the PRELIMINARY phase of the password stack traversal. Because any failure detected in this phase prevents the actual update of the password (which should happen in the next traversal), no password is changed if the `PAM_AUTHOK_ERR` value is returned.

This concludes the code walkthrough of the example module, `pam_compare`.

---

## Testing the PAM Module

In order to test the `pam_compare.so.1` module, update the `/etc/pam.conf(4)` file as detailed on page 6, and run the `passwd` command. With the `maxequal` variable set to 4, this is what you see in the code box:

```
$ passwd
passwd: Changing password for testuser
Enter existing login password: s3cr3t!
New Password: s3cur3!
passwd: Your old and new password can't share more than 4
characters.

Please try again
New Password: a^_g34.Q
Re-enter new Password: a^_g34.Q
passwd: password successfully changed for testuser
```

PAM provides its services to all applications that perform Password Management, and all these applications benefit from the new module. If you created the local account `testuser`, you can force a password change when `testuser` logs in the next time, with the following command:

```
$ passwd -f testuser
```

Here is the example of `testuser` logging in, (please note that the boldface type is user input):

```
$ rlogin -l testuser localhost
Passwd: a^_g34.Q
Choose a new password.
New Password: 55Q.ga_^
rlogin: Your old and new password can't share more than 4
characters.
Try again

Choose a new password.
New Password:
```

As illustrated, the `rlogins` password management service benefits immediately from the newly installed module.

By plugging multiple, low-level authentication mechanisms into applications at runtime, PAM integrates them with a single high-level API. These authentication mechanisms, are encapsulated as dynamically loadable, shared software modules. These software modules may be installed independent of applications.

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

---

**Note** – In the Solaris 9 OE, the `pam_unix` function does not exist in the same form that it does in the Solaris 8 OE. In order to accommodate proper stacking of the `pam_unix` function it has been broken up into single service modules. When used together these single service modules provide the same functionality as the existing `pam_unix` function. For example, some of the service modules are: `pam_unix_auth(5)`, `pam_authtok_*(5)`, and `pam_passwd_auth(5)`.

---

---

## Conclusion

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

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 of the Solaris OE. PAM is part of an overall security solution from Sun Microsystems, Inc. Increasing overall security can help your enterprise protect IT assets and improve overall availability.

For more information on Sun Microsystems, Inc. security products, see [www.sun.com/security](http://www.sun.com/security).

---

## About The Authors

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 coauthor of the Sun BluePrints book *Solaris and LDAP Naming Services*.

Joep Vesseur is a member of the technical staff of the Solaris Security Technology Group.

---

## Ordering Sun Documents

The SunDocs<sup>SM</sup> 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<sup>SM</sup> 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>.