CHAPTER 5

Been Cracked? Just Put PAM On It!

Pluggable Authentication Modules

Although pluggable authentication modules (PAM) cannot protect your system after it has been compromised, it can certainly help prevent the compromise to begin with. It does this through a highly configurable authentication scheme. For example, conventionally UNIX users authenticate themselves by supplying a password at the password prompt after they have typed in their username at the `login` prompt. In many circumstances, such as internal access to workstations, this simple form of authentication is considered sufficient. In other cases, more information is warranted. If a user wants to log in to an internal system from an external source, like the Internet, more or alternative information may be required—perhaps a one-time password. PAM provides this type of capability and much more. Most important, PAM modules allow you to configure your environment with the necessary level of security.

This chapter describes the use of pluggable authentication modules for Linux (Linux-PAM or just PAM[1]), as distributed with Red Hat 5.2/6.0, which provides a lot of authentication, logging, and session management flexibility. We generally describe PAM and its configuration, take a look at many of the available PAM modules,[2] and consider a number of examples.

1. Pluggable authentication modules were originally developed by Sun Microsystems, Inc.
2. Pluggable authentication modules modules (PAM modules) is brought to you by the department of redundancy department.

Most recent Linux distributions include PAM. If your version does not, check out the web site:

```
http://www.kernel.org/linux/libs/pam/
```

There you will find source code and documentation. It is well worth the effort to download, compile, and integrate PAM into your system.

PAM provides a centralized mechanism for authenticating all services. It applies to `login`, remote logins (`telnet` and `rlogin` or `rsh`), `ftp`, Point-to-Point Protocol (PPP), and `su`, among others. It allows for limits on access of applications, limits of user access to specific time periods, alternate authentication methods, additional logging, and much more. In fact, PAM may be used for any Linux application! Cool! Let's see how it works.

PAM Overview

In this section, we will describe the way in which PAM operates, generally how to configure PAM, and the keywords and options associated with the PAM configuration files. Figure 5.1 presents an overview diagram of the Linux-PAM interaction with Linux applications. This diagram depicts the major components of a PAM implementation—applications, such as `login`, `ftp`, `su`, etc.; the Linux-PAM engine (the PAM libraries, found in `/lib`), which is

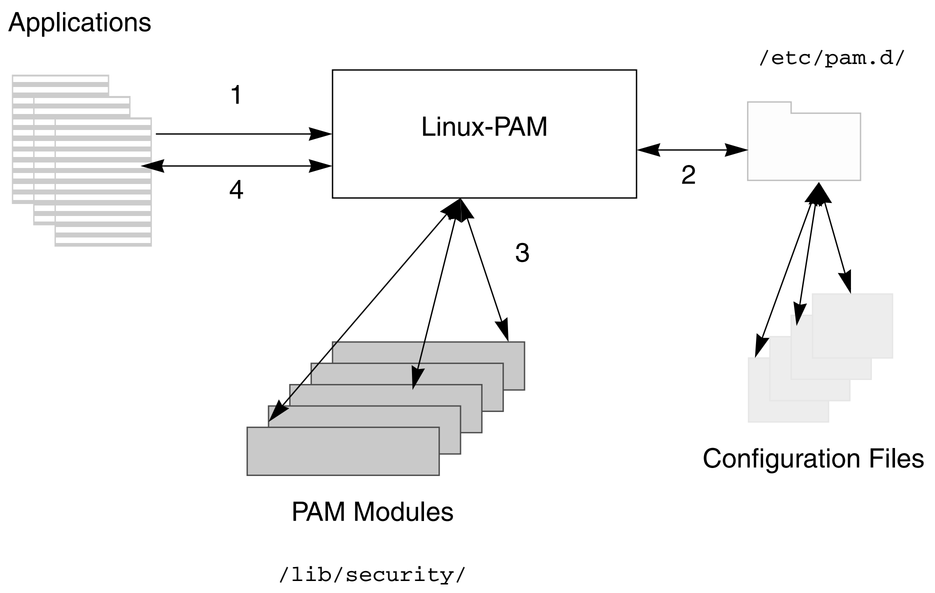

Fig. 5.1 Linux-PAM Overview

responsible for loading the necessary PAM modules based on the configuration files. The general flow of execution follows:

1. The application—for example `login`—makes an initial call to Linux-PAM.
2. Linux-PAM locates the appropriate configuration file in `/etc/pam.d` (or, alternatively, `/etc/pam.conf`) to obtain the list of modules necessary for servicing this request.
3. Linux-PAM then loads each module in the order given in the configuration file for processing. Depending upon configuration parameters, not all modules listed in the configuration file will necessarily be invoked.
4. Some, or all, of the modules may need to have a *conversation* with the user through the calling application. This conversation normally includes prompting the user for some sort of information, like a password or challenge, and receiving a response. If the user's response satisfies the particular PAM module, or if the PAM module is satisfied in some other way, control is passed back to Linux-PAM for processing of the next module (steps 3 and 4 being repeated for each module in the configuration file associated with the application in question). Ultimately, the processing completes with either success or failure. In the case of failure, it is generally true that the error message displayed to the user will not be indicative of the cause of failure. This generic error messaging approach is a security feature since it limits information that could be used in compromise efforts. Fortunately, most PAM modules offer varying levels of logging, allowing system administrators to track down problems and identify security violations.

PAM Configuration

There are two different PAM configuration compile-time options. The first causes PAM either to use a single `/etc/pam.conf` file as its configuration file or to look for a collection of configuration files in `/etc/pam.d`, but not both. The second option uses both mechanisms and entries in `/etc/pam.d` directory override those in `/etc/pam.conf`. The first option is recommended and reflects the implementation used by the Red Hat 5.2/6.0 distributions.

There is little difference between using a single `/etc/pam.conf` and a collection of files in `/etc/pam.d`. Essentially, if you are using `/etc/pam.conf`, each entry in `/etc/pam.conf` contains a leading service-type field that specifies the PAM-aware application to which this entry pertains. If you use `/etc/pam.d`, you will find a file in that directory whose name matches a PAM-aware application. Consequently, the service-type field is dropped from each of these files. We will discuss the configuration options under the assumption of the use of `/etc/pam.d`. For those of you who use `/etc/pam.conf`, just add the service type to the entries described here.

Let's begin by taking a look at the contents of `/etc/pam.d`, shown in Example 5-1. This effectively lists the PAM-aware applications that ship with the Red Hat 5.2 distribution (6.0 is similar). Each of the files listed has a PAM-aware application associated with it. In all of these configuration files, lines beginning with `#` are comment lines and are ignored by PAM.

Example 5-1 Contents of `/etc/pam.d`

```
# ls /etc/pam.d
chfn            linuxconf-pair  ppp             su
chsh            login           rexec           vlock
ftp             mcserv          rlogin          xdm
imap            other           rsh             xlock
linuxconf       passwd          samba
```

Each of the PAM configuration files contains the entry types shown in Example 5-2. The `module-type` field specifies the type of PAM module. Currently there are four module types, `auth`, `account`, `session`, and `password`. They are described in Table 5.1.

Example 5-2 PAM Configuration File Entry Fields

```
module-type    control-flag    module-path    arguments
```

The `control-flag` field specifies the action to be taken depending on the result of the PAM module. More than one PAM module may be specified for a given application (this is called *stacking*). The `control-flag` also determines the relative importance of modules in a stack. As we will see, stack order and `control-flags` are very significant. The four possible values for this field are `required`, `requisite`, `optional`, and `sufficient`. They are summarized in Table 5.2.

The `module-path` field indicates the absolute pathname location of the PAM module. Red Hat 5.2/6.0 places all PAM modules in `/lib/security`. (Table 5.15 on page 110 provides an overview of many of the available PAM modules, both from the Red Hat distribution and the public domain.)

Table 5.1 PAM Module Types

Module Type	Description
`auth`	The `auth` module instructs the application to prompt the user for identification such as a password. It may set credentials and may also grant privileges.
`account`	The `account` module checks on various aspects of the user's account such as password aging, limit access to particular time periods or from particular locations. It also may be used to limit system access based on system resources.
`session`	The `session` module type is used to provide functions before and after session establishment. This includes setting up an environment, logging, etc.
`password`	The `password` module type is normally stacked with an `auth` module. It is responsible for updating the user authentication token, often a password.

Table 5.2 PAM Control Flags

Control Flag	Description
`required`	This module must return success for the service to be granted. If this module is one in a series of stacked modules, all other modules are still executed. The application will not be informed as to which module or modules failed.
`requisite`	As above, except that failure here terminates execution of all modules and immediately returns a failure status to the application.
`optional`	As the name implies, this module is not required. If it is the only module, however, its return status to an application may cause failure.
`sufficient`	If this module succeeds, all remaining modules in the stack are ignored and success is returned to the application. In particular, if the module succeeds, this means that no subsequent modules in the stack are executed, regardless of the control flags associated with the subsequent modules. If this module fails it does not necessarily cause failure of the stack, unless it is the only module in the stack.

The `arguments` field is used for specifying flags or options that are passed to the module. Specifying arguments is optional. There are certain general arguments available for most modules which are listed in Table 5.3. Other arguments are available on a per-module basis and will be discussed appropriately with each module.

In summary, each file in `/etc/pam.d` is associated with the service or application after which the file is named and contains a list of records, each of which contains a module type, control flag, module name and location, and optional arguments. If the modules are of the same type, they are considered to be stacked and will be executed in the order in which they appear, unless control flags terminate execution earlier. The entire stack, not just one module,

Table 5.3 PAM Standard Arguments

Standard Arguments	Description
`debug`	Generates additional output to the `syslog`* utility. Most PAM modules support this argument. Its exact definition depends on the module to which this argument is supplied.
`no_warn`	Do not pass warning messages to the application.
`use_first_pass`	This module will use the password from the previous module. If it fails, no attempt is made to obtain another entry from the user. This argument is intended for `auth` and `password` modules only.
`try_first_pass`	As above, except that, if the password fails, it will prompt the user for another entry. This argument is intended for `auth` and `password` modules only.

*The `syslog` utility is discussed in detail in Chapter 8.

controls behavior for the given service and module type. Arguments are optionally used to further control the behavior of the module.

Now let's see how this mechanism is actually implemented.

PAM Administration

One of the joys of working with freely available software is that it is often poorly or incorrectly documented or it doesn't work quite right (or at all!). There are various news groups to which you may post queries (see Appendix A), and some vendors (such as Red Hat and S.u.S.E.) maintain their own mailing lists which are helpful from time to time. Often, however, you will find yourself having to figure it all out on your own. This section will go beyond the currently available documentation and hopefully will give you a good start on using PAM.

PAM and Passwords

We begin by taking a look at how PAM may be used to control password choices and password aging. Example 5-3 shows the `/etc/pam.d/passwd` configuration file. Notice that there are two entries with the `password` module type. This is an example of stacked entries. Let's go through these entries in detail. It gets a little complicated, but once we get through it, the rest of this chapter should be easier to understand.

Example 5-3 The `/etc/pam.d/passwd` Configuration File

```
auth       required     /lib/security/pam_pwdb.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so retry=3
password   required     /lib/security/pam_pwdb.so use_authtok
```

Note

Throughout this chapter, we will often refer to the PAM modules without the trailing `.so`. For example, we will refer to `/lib/security/pam_pwdb.so` as simply `pam_pwdb` in the text but `/lib/security/pam_pwdb.so` will be used in all examples, as required.

The Password Database Library The `/lib/security/pam_pwdb` module interacts with and requires the password database library (`pwdb` library, `libpwdb`, found in `/lib`). The purpose of the `pwdb` library is to provide a centralized database for lookups of information associated with users and groups. Specifically, it provides the source of passwords for `pam_pwdb`.

The `pwdb` library requires an `/etc/pwdb.conf` configuration file. Example 5-4 shows a sample file. The file contains two distinct sections—the first, preceded by the `user:` keyword, pertains to information associated with users.

Example 5-4 The `/etc/pwdb.conf` File

```
# This is the configuration file for the pwdb library
    #
user:
 unix+shadow
 nis+unix+shadow
group:
 unix+shadow
 nis+unix+shadow
```

The second, preceded by the `group:` keyword, pertains to information associated with groups. After the section header, you see keywords concatenated with `+` symbols, called *lists*. Each list represents the collection of databases that are merged to form the records for each user or group. For example, the `unix+shadow` list under the `user:` section is a list consisting of the contents of the `/etc/passwd` and `/etc/shadow` files. The `nis+unix+shadow` entry specifies the list containing NIS[3] (formerly yp) records as well as the contents of the `/etc/passwd` and `/etc/shadow` files. The entries for groups are entirely similar.

When the `pam_pwdb` module is invoked, it in turn invokes the `pwdb` library. The `pwdb` library will find the first entry that matches the user or group passed to it by `pam_pwdb`, based on the entries in `/etc/pwdb.conf`. Thus order is important in that file. The lists that appear first are searched first and `pwdb` stops at the first match.

The `pam_pwdb` Module The `pam_pwdb` module is capable of operating in support of all four module types.

Module type `auth`. When the `auth` type is specified, it functions to authenticate the user by prompting the user for a password and querying `pwdb` with the username/password pair. It can take the following arguments: `debug`, `use_first_pass`, `try_first_pass`, `nullok`, and `nodelay`. All other arguments supported by `pam_pwdb` but not by the `auth` module type are silently ignored. Any other arguments will be logged as errors through `syslog`, but will not affect the function of the module. The first three arguments are described in Table 5.3 on page 85.

The `nullok` argument *allows accounts with no passwords*. Of course, you would never specify this argument, right? The default behavior, therefore, is that this module treats accounts with no passwords as if they were locked accounts. This is good!

The `nodelay` argument causes this module to return immediately on failure. Normally this module will delay prior to reporting an authentication failure, making it slower for an attacker to try to guess passwords.

So what is the purpose of the line:

```
auth       required     /lib/security/pam_pwdb.so
```

3. See Chapter 3 for further details about NIS. NIS was formerly known as Yellow Pages (YP).

Table 5.4 Arguments for `pam_pwdb` Module Type `password`

Argument	Description
`nullok`	Allows for the changing of a null (nonexistent) password. For the reasons outlined earlier, use of this argument is not recommended.
`not_set_pass`	Causes this module to ignore passwords from previous modules and disallows this module from passing new passwords to subsequent modules.
`use_authtok`	This argument forces the module to set the new password to the one received from the previously stacked module.
`md5`, `bigcrypt`	Instead of using the conventional UNIX password hashing algorithm (invoked through the `crypt` function call), you may choose one or the other of these.
`shadow`, `radius`, `unix`	Allows for the transfer of passwords from one database to another through the `pwdb` library.

in Example 5-3 on page 86? It causes users to be prompted for their old password prior to being prompted (by `pam_cracklib`) for their new password! Cool, huh? The root user is excepted from this requirement.

Module type `password`. When the `pam_pwdb` module is used as module type `password`, it performs the task of updating the password. This means that, when a user invokes the `passwd` command, upon successfully entering a new password (as determined by `pam_cracklib`), `pam_pwdb` will update the new password with the `pwdb` library. The acceptable argument types are `debug`; `nullok`; `not_set_pass`; `use_authtok`; `try_first_pass`; `use_first_pass`; `md5`;[4] `bigcrypt`; `shadow`; `radius`; `unix`. Those arguments not already discussed are summarized in Table 5.4.

Notice in Example 5-3 on page 86, the `use_authtok` argument is specified. This means that `pam_pwdb` will use the new password it receives from `pam_cracklib`. Essentially, `pam_cracklib` controls the choice of the new password, but `pam_pwdb` actually does the updating.

NOTE

The use of `md5` (MD5 is discussed in Chapter 3) or `bigcrypt` (a modified `crypt(3)` that allows for up to a 16-character password) arguments instead of the default, traditional UNIX `crypt(3)` for hashing is highly recommended. It allows for longer passwords that may be harder to guess by programs such as Crack (discussed in Chapter 12). In Red Hat 6.0, choosing MD5 is an install time option.

4. RSA Data Security, Inc., MD5 Message-Digest Algorithm.

***Module type* `account`.** When using `pam_pwdb` as module type `account`, its purpose is to verify account information of the user. This includes validating that the user has an account, what password aging parameters, if any, are associated with the user, and whether or not the user needs to be warned about a pending password expiration or offered advice on the choice of a new password. As this module type, `pam_pwdb` recognizes only the `debug` argument.

***Module type* `session`.** When using `pam_pwdb` as module type `session`, its sole purpose is to log the username and service type to `syslog`, once at login and then subsequently at logout. It recognizes no arguments.

The `pam_cracklib` Module The `pam_cracklib` module is intended to work only with the `password` module type. It's purpose is to check a password for strength and for length, both elements being configurable with arguments described below. This module functions only in a stack, since it has no updating capabilities. It requires the `libcrack` library and the `cracklib_dict` Crack dictionary, both of which are found in `/usr/lib` of the Red Hat 5.2/6.0 distribution. As you can see, this module depends heavily on elements of the Crack utility, which is discussed in Chapter 12.

The flexibility of PAM is evidenced by the fact that this is not the only password strength checking PAM module. Another is `pam_passwd+`, which is available at

```
http://www.us.kernel.org/pub/linux/libs/pam/modules.html
```

The arguments available to `pam_cracklib` are described in Table 5.5.

Table 5.5 Arguments for `pam_cracklib`

Argument	Description
`debug`	This argument writes additional module behavior information to `syslog`, but **does not** log passwords.
`type=STRING`	This argument replaces the string `UNIX` with `STRING` in the messages it generates, such as `New UNIX password:`.
`retry=`*n*	This is the number of retries this module allows a user when changing a password. The default is 1.
`difok=`*n*	This represents the number of characters in the new password that must be different from the old password. The default is 10. Regardless of this limit, however, any new password that has at least half the characters different from the old will be accepted.
`minlen=`*n*	This argument specifies the minimum password length + 1. By default it is set to 9 which means the minimum password length is actually 10. To further confuse the issue, this minimum length may actually be reduced depending upon the values specified for the `*credit` parameters listed below.

Table 5.5 Arguments for `pam_cracklib` (Continued)

Argument	Description
`lcredit=`*n*	The value specified here is the number of characters by which the `minlen` value is *reduced* by virtue of having at least one lowercase character in the new password. The default is 1. Can be set to 0 to eliminate the credit.
`ucredit=`*n*	The value specified here is the number of characters by which the `minlen` value is *reduced* by virtue of having at least one uppercase character in the new password. The default is 1. Can be set to 0 to eliminate the credit.
`dcredit=`*n*	The value specified here is the number of characters by which the `minlen` value is *reduced* by virtue of having at least one numeric character in the new password. The default is 1. Can be set to 0 to eliminate the credit.
`ocredit=`*n*	The value specified here is the number of characters by which the `minlen` value is *reduced* by virtue of having at least one nonalphanumeric character in the new password. The default is 1. Can be set to 0 to eliminate the credit.

In addition to the configurable options in Table 5.5, the `pam_cracklib` checks the new password for strength by

1. Verifying that the new password is not the reverse of the old password.
2. Verifying that the new password is not a simple case change of some characters of the old password.
3. Checking if the new password is in the `cracklib_dict`. If it is, it warns the user but does not force another password choice.

The Effect of Stacking `pam_pwdb` and `pam_cracklib` For Module Type `password` In this section, we will take a look at how `pam_cracklib` interacts with `pam_pwdb` in the stack shown in Example 5-3 on page 86.

Recall that in Example 5-3 the two stacked entries appeared in `/etc/pam.d/passwd`.

```
password   required     /lib/security/pam_cracklib.so retry=3
password   required     /lib/security/pam_pwdb.so use_authtok
```

The first entry invokes `pam_cracklib` and prompts the user for his or her new password (remember that the `auth` module type `pam_pwdb` entry is responsible for prompting the user for his or her *old* password, for authentication). After the user has supplied the new password, `pam_cracklib` requests that it be repeated for verification. Once completed, `pam_cracklib` performs its checks to see if the password is acceptable. If so, it passes the new password to `pam_pwdb` which has the `use_authtok` argument meaning it will accept this new password and request the `pwdb` library to update the appropriate database.

Let's take a look at the power and flexibility of these modules by considering an example. Suppose that we would like to use `md5` instead of the standard UNIX `crypt(3)` mechanism for hashing purposes. This is a good idea, because popular password-guessing tools like Crack require significantly more CPU resources to guess passwords (see *The White Hat Use of Crack* on page 337). The major benefit of using `md5` is that you can require longer passwords—20, 30, or even more characters. Let's look at an example requiring 20-character passwords. We'll also set the `type` argument to see if our users are paying attention. Example 5-5 shows what the stack might look like if we impose these changes in `/etc/pam.d/passwd`.

Example 5-5 Using `md5` and `minlen` in `/etc/pam.d/passwd`

```
password   required     /lib/security/pam_cracklib.so minlen=20\
         retry=3 type=SECRET
password   required     /lib/security/pam_pwdb.so md5 use_authtok
```

WARNING

If you make changes similar to what is shown in Example 5-5, you must also change all equivalent instances of `pam_pwdb` and `pam_cracklib` using module type password. In Red Hat 5.2, this would minimally include the files `chfn`, `chsh`, `login`, `rlogin`, su, and xdm in `/etc/pam.d`.

Now that we have made these changes, let's see what happens to the user, `mary`, when she tries to change her password in Example 5-6. She is offered three opportunities to select a password. This is due to the `retry=3` argument to `pam_cracklib` (see Example 5-5 on page 91). Actually, it appears that Mary is attempting to make good password choices. Unfortunately she doesn't know about the changes to PAM and therefore doesn't know that she needs to choose a longer password. So you, being the responsive administrator, inform her that she needs to use a 20-character password. "What?!" she replies. And you gently tell her that she can use a passphrase. Happy now, she goes about her task (Example 5-7).

Example 5-6 Unsuccessful Password Change

```
$ passwd
Changing password for mary
(current) UNIX password: j3n#Ky
New SECRET password: Rt!72g
BAD PASSWORD: is too simple
New SECRET password: 8x@$iI
BAD PASSWORD: is too simple
New SECRET password: P5-+yh
BAD PASSWORD: is too simple
New SECRET password: 8x@$iI
passwd: Authentication token manipulation error
$
```

Example 5-7 Successful Password Change

```
$ passwd
Changing password for mary
(current) UNIX password: j3n#Ky
New SECRET password: I need a #%$3+ raise
Retype new SECRET password: I need a #%$3+ raise
passwd: all authentication tokens updated successfully
$
```

Notice that the message displayed by `pam_cracklib` contains our type entry, `New SECRET password:`. This change does not appear in the message from `pam_pwdb`—`(current) UNIX password:`—because `pam_pwdb` does not support the `type` argument.

NOTE

Normally, the passwords displayed in Example 5-6 and Example 5-7 are not visible. They are shown here for clarifying the examples.

While she chose a password of 20 characters (spaces count!), she wouldn't have been required to, because the default values of `dcredit`, `ucredit`, `lcredit`, and `ocredit` (see Table 5.5 on page 89) are 1 each. Because of her password choice, she would have a credit of 4, which would have allowed her to choose a password as short as 16 characters in length.

PAM and Passwords Summary

Figure 5.2 reviews Example 5-3 on page 86 in its entirety. When Mary executes the `passwd` command, Linux-PAM is invoked. Linux-PAM reads the `/etc/pam.d/passwd` file and executes each module listed in order. First

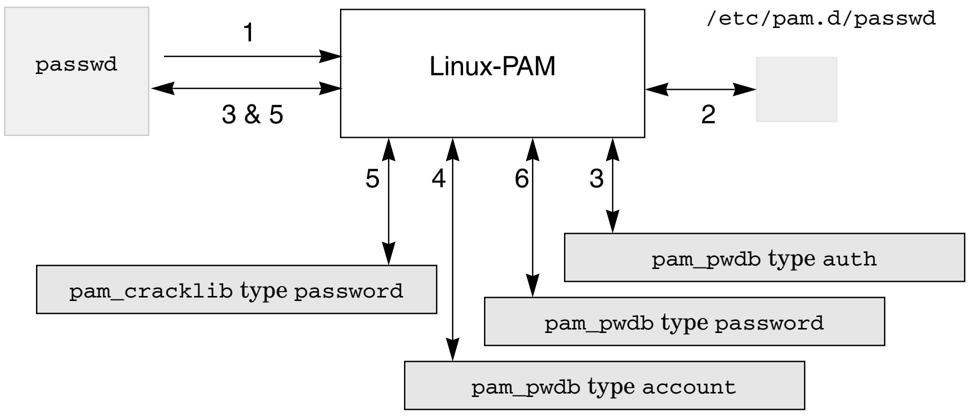

Fig. 5.2 PAM-Controlled Password Change

Mary is authenticated with her old password; this occurs due to the `pam_pwdb` entry with module type `auth`. Second, `pam_pwdb` is invoked with module type `account` to verify Mary's account (and to check, for example, if password aging permits her to change it now). Third, Mary is prompted for the new password by the `pam_cracklib` entry with module type `password`. Fourth, and finally, after Mary has successfully entered a new password, `pam_pwdb` with module type `password` updates the `pwdb` library. Now she has a new password.

NOTE

The root user is not subject to any of these constraints and may set any password for any user.

Notice that all four entries in `/etc/pam.d/passwd` use the control flag `required`, which means that all four modules must be satisfied in order for the password change to be successful.

Now that we have a fundamental understanding of PAM, let's go on and look at some of the other services it manages.

PAM and `login`

The `/etc/pam.d/login` configuration file is read by Linux-PAM whenever the `login` (`/bin/login`) program is executed. Example 5-8 is a sample of this file. From the previous section, we have a good idea of what is going on here. Let's examine the details of the action of each module type.

Example 5-8 Sample `/etc/pam.d/login` File

```
auth       required     /lib/security/pam_securetty.so
auth       required     /lib/security/pam_pwdb.so
auth       required     /lib/security/pam_nologin.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so minlen=20\
         retry=3 type=SECRET
password   required     /lib/security/pam_pwdb.so md5 use_authtok
session    required     /lib/security/pam_pwdb.so
```

Module Type `auth` There are three modules stacked for type `auth`. The first is `pam_securetty`. This is an `auth` module type only. It accepts no arguments. Its sole purpose is to check `/etc/securetty` against the device of the login attempt, if the user logging in is root. It will fail only if someone is trying to log in as root from a device not in `/etc/securetty`. Since this module is required, a failure here would cause the login to fail. Recall, however, that the control flag, `required`, will still allow subsequent modules to be executed, and therefore the user will not be refused access until after all three modules in this stack are executed.

The second module is `pam_pwdb`. As discussed in *PAM and Passwords* on page 86, its role here is to authenticate the user.

The third module in the stack is `pam_nologin`. This is also an `auth`-only module type that accepts no arguments. Its purpose is to check for the existence of `/etc/nologin`. If `/etc/nologin` exists, then no users, except root, are allowed to log in; if `/etc/nologin` contains a message, it is displayed. For example, if the `/etc/nologin` is

```
System down for maintenance until 2/28/99 at 4pm
```

then Example 5-9 exhibits the system's behavior when the user `joe` attempts to `telnet` to the system.

Example 5-9 Attempted Login with the Presence of `/etc/nologin`

```
$ telnet livfreeordie
Trying 10.1.1.1...
Connected to livfreeordie.
Escape character is '^]'.

Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i686
login: joe
Password:
System down for maintenance until 2/28/99 at 4pm

Login incorrect

login:
```

OH, BY THE WAY...

Did you ever wonder about how to get rid of the message that appears when you connect to a system? In Example 5-9, it is the two-line message beginning with "Red Hat Linux." This message gives away information about your system, which does not reflect good security practice. On Red Hat 5.2/6.0, the message displayed reflects the contents of `/etc/issue.net` for remote connections and `/etc/issue` for local connections. At this point, you may be tempted to rush off and modify these files to suit your needs. Don't. The files are rewritten at every reboot! In order to modify these files, you must modify `/etc/rc.d/rc.local`. `/etc/rc.d/rc2.d/S99local` is a symbolic link to `/etc/rc.d/rc.local`, which is executed upon entering run level 2. Be sure to replace this message with something appropriate, maybe something like "This is a restricted system. All activities are logged." But whatever you do, don't reveal the operating system, hardware, or other information that may be used against you. And don't use words like "Welcome"! There are many other ways to replace this message, some of which will be discussed at various points in this text.

By default, Joe will actually be able to attempt to log in three more times, but of course the result will be the same. Existing login sessions are not affected by the `/etc/nologin` file.

This file is normally used for purposes of system maintenance, but it also is beneficial in case of a system breach. This topic is discussed in Chapter 3.

Normally, the `/etc/nologin` file does not exist and the user, once authenticated, will be granted access. The subsequent modules in `/etc/pam.d/login` are then invoked.

Module Types `account`, `password`, and `session` The `pam_pwdb` entry for the `account` module type, as previously described, performs the task of checking the user's account. If the user's password has expired, for example, the entries with module type `password` will be invoked and the events described in *PAM and Passwords* on page 86 will occur. The `session` entry is for the exclusive purpose of logging connection information to `syslog`.

Many other PAM configuration files besides `/etc/pam.d/login` incorporate the use of these modules and module types. Since each configuration file represents an application, it follows that any application that requires authentication by password will incorporate some or all of the modules and module types in `/etc/pam.d/login`. Included among these are `rsh`, `rlogin`, `su`, `ppp`, `chfn`, `chsh`, and `ftp`. Bear this in mind whenever you decide to make changes to one of these configuration files. For continuity, and for security, you will ordinarily make changes across the board. Think carefully if you decide not to change a particular configuration file whenever you change others.

Next let's take a look at some additional restrictions that may be imposed through PAM.

Time and Resource Limits

You may also wish to impose access time restrictions and resource limitations on your users. PAM affords the opportunity to impose such limitations through the modules `pam_time` and `pam_limits`, respectively.

Using `pam_time` This module is used with module type `account` only. Although it accepts no arguments, it does expect a configuration file, `/etc/security/time.conf` (Red Hat 5.2/6.0; other distributions may vary), to supply it with login location and time limitations. The absence of this file has the effect of not restricting access in any way. The limitations apply to *all* users, including root.

Suppose that you'd like to impose some limits on user access to a particular system. Take a look at the `account` module type entries in Example 5-10. The `pam_time` and the `pam_pwdb` entries both use the control flag `required`. This has the effect of causing the account verification step to proceed. For example, if Mary's password has expired, when she attempts to log in, she will be so informed and refused access regardless of the limitations imposed by `pam_time`. Let's assume that Mary's password has expired and that she is attempting to connect to the system `livfreeordie` outside the limits imposed by `/etc/security/time.conf` (see *The `/etc/security/time.conf` File* on page 96). Example 5-11 shows what happens if Mary attempts to log in under these assumptions and `/etc/pam.d/login` is configured as shown in Example 5-10. Even though Mary is logging in outside of her approved time, the only information she gets is that her password has expired.

Example 5-10 Partial Configuration File Using `pam_time required`

```
account    required     /lib/security/pam_pwdb.so
account    required     /lib/security/pam_time.so
```

Example 5-11 Login Attempt with Expired Password and Outside of Permitted Times 1

```
$ telnet livfreeordie
Trying 10.1.1.1...
Escape character is '^]'.
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i686
login: mary
Password:
Your account has expired; please contact your system administrator
User account has expired
Connection closed by foreign host.
$
```

If you reverse the entries and set the control flag to `requisite` for `pam_time`, the behavior is quite different. Example 5-12 shows the new configuration. Example 5-13 shows what happens with the new configuration when Mary attempts to log in.

Example 5-12 Partial Configuration File Using `pam_time.so requisite`

```
account    requisite    /lib/security/pam_time.so
account    required     /lib/security/pam_pwdb.so
```

Example 5-13 Login Attempt with Expired Password and Outside of Permitted Times 2

```
$ telnet livfreeordie
Trying 10.1.1.1...
Escape character is '^]'.
Red Hat Linux release 5.2 (Apollo)
Kernel 2.0.36 on an i686
login: mary
Password:
Permission denied
Connection closed by foreign host.
$
```

The moral of these examples is: watch your order and control flag settings. Experiment before you implement! Make sure the configuration settings provide the functionality you expect.

Next we turn our attention to the `time.conf` file.

The `/etc/security/time.conf` File. This file controls access time and login location by device when using the `pam_time` module. Each line in the file is a record, called a rule, except lines beginning with `#` which are comments. Each record has the following syntax:

```
services;ttys;users;times
```

This syntax is further detailed in Table 5.6.

The phrase *logical list* referenced in Table 5.6 means that the special characters, described in Table 5.7, are used as conditional operators.

Logical operators may be mixed. For example, `tty* & !ttyp*` means that any serial device is allowed for this rule, but all pseudo-devices are not.

The syntax used to specify days in `timed.conf` is summarized in Table 5.8. These codes are then used with time ranges, all times being specified by the 24-hour clock. For example, `Wd0800-1600` means weekends between

Table 5.6 Description of `/etc/security/time.conf` Entries

Parameter	Description
`services`	A logical list of the services associated with PAM. Multiple records with the same service are acceptable. Examples include `login`, `rsh`, and `su`.
`ttys`	A logical list of device(s). The login device is stored in `PAM_TTY`. Normally this includes such devices as `tty1` and `tty2` for the console, `ttyS0` and `ttyS1` for serial ports, and `ttyp1` and `ttyp2` for pseudo-devices normally associated with network and X Window connections.
`users`	A logical list of (valid) users. May include root.
`times`	A logical list of times at which this rule applies.

Table 5.7 Conditional Operators Used in `/etc/security/time.conf`

Operator	Function	Examples
`&`	`logical AND`	`user1 & user2`—means this rule applies to both `user1` and `user2`.
`|`	`logical OR`	`tty1 | tty2`—means this rule applies to either `tty1` or `tty2`.
`!`	`logical NOT`	`! login`—this rule does not apply to the login service.
`*`	`wildcard`	Matches any value, its meaning depending on its location in a field.

Table 5.8 Day Codes in `/etc/security/timed.conf`

Day Code	Description
`Mo, Tu, We, Th, Fr, Sa, Su`	Each code individually applies to the day of the week it indicates. These codes may be concatenated; for example, `MoTuWe` means Monday, Tuesday, and Wednesday.
`Wk` and `Wd`	`Wk` means weekdays while `Wd` means weekends (Saturday and Sunday). Note that, for example, `MoWk` means all weekdays except Monday.
`Al`	All seven days. Note that, for example, `AlSu` means all days except Sunday.

the hours of 8 A.M. and 4 P.M.. Notice that there are no spaces between the day code and the time.

Example 5-14 shows a series of entries in a sample /etc/security/time.conf file. In this example, the root user has access to all services all of the time so long as root logs in from tty1. The users—joe, bill, and jane—have access every day of the week between 8 A.M. and 6 P.M. from any device so long as they connect via login or rsh. The user guest may log in from anywhere, Monday through Friday between the hours of 9 A.M. and 4 P.M., except between 12 noon and 1 P.M. Finally, any user may access the system using ftp from any source Monday through Friday between the hours of 9 A.M. and 4 P.M. All other users are unrestricted.

Example 5-14 Sample /etc/security/time.conf File

```
*;tty1;root;Al0000-2400
login & rsh;*;joe|bill|jane;Al0800-1800
login;*;guest;Wk0900-1600&!Wk1200-1300
ftp;*;*;Wk0900-1600
```

This file is not order dependent. If there are entries that overlap, the least permissive (actually, the intersection of all entries) is used. Remember, you must place a pam_time entry in each file in /etc/pam.d for those services for which you want to restrict access. If you are going to limit access using pam_time, consider placing entries in at least the following files (and hence the associated services will be limited) in /etc/pam.d: ftp, login, ppp, rexec, rlogin, rsh, su, and xdm.

WARNING!

If you use an entry of the form

```
*;*;*;!AL0000-24000
```

in the /etc/security/time.conf file, you will lock out all users, including root, from the system! Should you find yourself locked out of the system, see the section *Recovering a Corrupt System* on page 24 in Chapter 3 for recovery procedures.

Using `pam_limits` You may additionally restrict user accounts by imposing limits on the system resources available to each user login session. This may be useful in limiting system-based DoS attacks. The pam_limits module operates as a session module type only. It supports two arguments—debug and conf=/path/to/config_file (the default configuration file is /etc/security/limits.conf). It does not impose any limits on the root account.

The /etc/security/limits.conf file is used to impose limits on a per-user or per-group basis. All limits specified apply to a single session. Each line in the file is a record, except for those beginning with #. The syntax of a record is

```
username|@groupname type resource limit
```

where `username|@groupname` specifies that either a username or a groupname preceded with @ may be used. The wildcard character `*` is acceptable and represents all users (hence all groups). The `type` field is either the `hard` or `soft` parameter; `hard` imposes a fixed limit and `soft` specifies a default limit. The `resource` parameter is one of the items described in Table 5.9. The `limit` parameter is the limit itself on the associated `resource`.

It is important to note that the limits imposed are on a per-session basis. The total limitation may be controlled with the `maxlogins` parameter. Limits may be completely disabled for particular users with the special character, `-`, an instance of which is shown in Example 5-15 . In this example, all users are limited to a resident set size of approximately 10 megabytes (MB) per session. All users may also have a maximum of only four simultaneous logins. It is this value that sets the overall maximum per user. The user, `bin`, has all limits disabled, including the previous entries in the file. The remaining limitations in the file are additional limitations for the users and groups indicated. The user `ftp` is allowed only 10 logins (this is an excellent limit to impose on anonymous `ftp` accounts since it limits the number of simultaneous logins). All users in the group `managers` have a process limitation of 40 and all users in the group `developers` have a `memlock` limit of approximately 64 MB.

Example 5-15 Sample `/etc/security/limits.conf` File

```
*               hard    rss             10000
*               hard    maxlogins       4
bin             -
ftp             hard    maxlogins       10
@managers       hard    nproc           40
@developers     hard    memlock         64000
```

Table 5.9 Limitable Resources in `/etc/security/limits.conf`

Resource	Description
`core`	Limits the size of a core file (KB*)
`data`	Maximum data size (KB)
`fsize`	Maximum file size (KB)
`memlock`	Maximum locked-in memory address space (KB)
`nofile`	Maximum number of open files
`rss`	Maximum resident set size (KB)
`stack`	Maximum stack size (KB)
`cpu`	Maximum CPU time in minutes
`nproc`	Maximum number of processes
`as`	Address space limit
`maxlogins`	Maximum number of logins allowed for this user

*kilobytes

Of course, you must determine the limits necessary for each system at your site. Make sure you place the entry

```
session   required   /lib/security/pam_limits.so
```

in each appropriate file in `/etc/pam.d`.

Access Control with `pam_listfile`

Any PAM-aware application may be given an access control list with `pam_listfile`. This is an authentication-only module that takes a number of arguments, as displayed in Table 5.10. In order to clarify Table 5.10, we'll look at two examples.

Suppose that we have a guest account on our system and we would like to disable `chsh` for guest. The `chsh` command allows a user to change his or her default shell in `/etc/passwd` to any shell listed in `/etc/shells`. Since `chsh` is a PAM-aware application, we can use `pam_listfile` to implement this restriction (no problem!). Add the `pam_listfile` entry to the existing `/etc/pam.d/chsh` configuration file as shown in Example 5-16. By now, everything in this file should be familiar except the `pam_rootok` entry (and of course the `pam_listfile` entry, which we haven't finished talking about). Actually, the `pam_rootok` entry is quite simple. Notice that it uses the control flag `sufficient` meaning that, if this module is satisfied, none of the other `auth` module types needs to be executed. The `pam_rootok` module does what you'd expect. If it's root, it's OK! So, in this case, if root wants to change any user's shell, root will not be authenticated (not prompted for a password).

Example 5-16 The `chsh` Configuration File with a `pam_listfile` Entry

```
auth       sufficient   /lib/security/pam_rootok.so
auth       required     /lib/security/pam_listfile.so onerr=fail\
             item=user  sense=deny   file=/etc/security/nochsh
auth       required     /lib/security/pam_pwdb.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so minlen=20 retry=3
password   required     /lib/security/pam_pwdb.so md5 use_authtok
session    required     /lib/security/pam_pwdb.so
```

Now back to `pam_listfile`. In Example 5-16, the argument `onerr=fail` is set. This means that, if there are *any* error conditions generated by the execution of this module, the module will fail. Since it is a required module, this further implies that authentication will fail and the user will not be allowed to change his or her shell. Errors will be logged to `syslog`, so you may view them in `/var/log/messages`. Unless you are debugging in a safe environment (i.e., not connected to a production environment), this is the recommended setting for this argument.

The remaining arguments deal with the access control file, which in this case is `/etc/security/nochsh`. The `item=user` argument tells `pam_listfile`

Table 5.10 Arguments to `pam_listfile`

Argument	Description
`onerr`	Takes either `succeed` or `fail`. If an error occurs, such as an unreadable configuration file, should this module return success or failure?
`sense`	Takes either `allow` or `deny`. This tells the module whether the list is an allow or deny list.
`file`	Requires the absolute pathname to the configuration file.
`item`	One of `user`, `tty`, `rhost`, `ruser`, `group`, or `shell`. It tells the module what to look for in the configuration file.
`apply`	Takes a username or a groupname preceded by `@`. It is only meaningful if `item` is set to `tty`, `rhost`, or `shell`.

that it should expect to find usernames—one per line—in `/etc/security/nochsh`. The `sense=deny` argument tells `pam_listfile` that `/etc/security/nochsh` is a deny list; that is, any user listed in that file will cause `pam_listfile` to fail and therefore (because of the `required` control flag) cause authentication to fail and disallow the user from changing the shell.

All that remains is to create `/etc/security/nochsh` and list the users to whom we wish to deny `chsh` capability. Here is an example file:

```
guest
joe
```

The two users, `guest` and `joe` (we don't trust him anymore), will not be able to successfully execute `chsh`, as the user `guest` demonstrates in Example 5-17. Hopefully, the flow of events is becoming clear. When guest executes `chsh`, a PAM-aware application, Linux-PAM is invoked and the `auth` stack in `/etc/pam.d/chsh` is executed. Referring back to Example 5-16 on page 100, the first `auth` module invoked is `pam_rootok`. Since `guest` is not `root`, that module fails and `pam_pwdb` is invoked and causes the password prompt. The user `guest` successfully enters the correct password (you'll have to trust me here) and execution is passed to `pam_listfile`, which checks its deny list and finds `guest` in it, causing authentication to fail (hence the generic `Password error` message).

Example 5-17 Failed `chsh` Attempt Due to `pam_listfile`

```
$ telnet livfreeordie
Trying 10.1.1.1...
Escape character is '^]'.

This is a restricted system. All activity is logged.
login: guest
Password:

livfreeordie$ chsh -s /bin/bash
Changing shell for guest.
Password:
Password error.
livfreeordie$
```

Consider another example. Suppose that we want to limit the users to which others may su—we want to restrict su use generally (not just su to root) to a specific set of users. We add a pam_listfile entry to /etc/pam.d/su as displayed in Example 5-18. This time we are using an allow list. Just place each allowed username, one per line, in the /etc/security/suok file. For example, if our /etc/security/suok contains the users:

```
root
mary
bill
jane
efram
```

then these are the only users that will be accepted as a user argument to su. Anyone may execute su, but only to become one of the users in this list. Example 5-19 shows what happens when paul tries to su to guest and then to root. The su attempt to guest fails because guest is not in the /etc/security/suok file. The su to root, however, succeeds because root is in the /etc/security/suok file and Paul knows the root password.

Example 5-18 The su Configuration File with pam_listfile

```
auth       required     /lib/security/pam_listfile.so onerr=fail \
       item=user  sense=allow  file=/etc/security/suok
auth       required     /lib/security/pam_pwdb.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so minlen=20 retry=3
password   required     /lib/security/pam_pwdb.so md5 use_authtok
session    required     /lib/security/pam_pwdb.so
```

Example 5-19 Failed su Attempts Due to pam_listfile

```
$ whoami
joe
$ su - guest
Password:
su: incorrect password
$ su -
Password:
#
```

Notice that the error message is not indicative of the actual failure. If you review the previous failure messages from other PAM modules, you'll see that this is a feature of PAM. The idea is to not reveal any information to the user through error messages. As an administrator with root access, you may always check out the log files. By default, you will find PAM-generated log messages in /var/log/messages (these files and syslog, in general, will be discussed in Chapter 8).

Note that when sense=allow changes to sense=deny in Example 5-18, the /etc/security/suok file becomes a deny list, meaning that a user would not be able to su to any of the users in the list. This is particularly useful if you wish to implement sudo (sudo is discussed in Chapter 9) and completely disallow su to root.

PAM and su

Unlike the last pam_listfile example, which restricts the users that may be switched to with su, pam_wheel is used to specifically restrict the successful execution of su to the root user. It does so by utilizing a special group called wheel with the GID of 0. Its default behavior is to allow only members of the wheel group to su to root when this module is in force. This is another authentication-only module. Its arguments are described in Table 5.11. This module has some arguments that you really don't want to use. The use_id argument causes the pam_wheel to use the effective UID of the user. In this way a non-wheel group member could su to a wheel group member and then su to root. This is probably not the behavior you seek.

The trust argument could cause wheel members to be able to su to root without a password, depending on the way in which modules are stacked. Avoid these two arguments, unless you are debugging or are otherwise prepared for their consequences.

On many releases of Linux, there is a GID 0, the root group. You may wish, therefore, to create a wheel group with a different GID—for example GID=10 (Red Hat 5.2/6.0 does this for you)—then use the group argument to pam_wheel. Example 5-20 displays a representative /etc/pam.d/su file. Make sure that you have a group called wheel in /etc/group. Any member of that group will be allowed to su to root. All other users will get a Password incorrect error message, even if they know the correct password.

Example 5-20 The /etc/pam.d/su File with pam_wheel

```
auth       required     /lib/security/pam_wheel.so group=wheel
auth       required     /lib/security/pam_pwdb.so
account    required     /lib/security/pam_pwdb.so
password   required     /lib/security/pam_cracklib.so minlen=20 retry=3
password   required     /lib/security/pam_pwdb.so md5 use_authtok
session    required     /lib/security/pam_pwdb.so
```

Table 5.11 Arguments of pam_wheel

Argument	Description
debug	Generates additional output to syslog.
use_id	Uses the current process UID and not that returned by getlogin. This may result in the use of an effective UID and is not recommended for production use.
trust	Causes this module to succeed if the user is a member of the wheel group. This option may cause members of wheel to become root without a password. Be *very* careful when using this argument.
deny	Reverse the logic of this module.
group=*groupname*	Instead of allowing users in the group wheel, allow the users in *groupname*.

Using `pam_access`

The `pam_access` module is another access control module. It is similar to `pam_listfile` in that it is a generic access control mechanism. It differs from `pam_listfile`, however, in two ways. First, it supports only module type `account`. Essentially, this difference means that we have similar access control functionality available to use for module type `auth` (`pam_listfile`) and module type `account` (`pam_access`). This allows us to control applications that do not support one or the other module type. An example of such a situation is given in the section *Further Restricting Access with PAM* on page 304 in Chapter 11.

Second, it requires the configuration file, `/etc/security/access.conf`. Entries in this file are of the form

```
permission : users : origins
```

Each of the fields in `/etc/security/access.conf` are described in Table 5.12. When the `pam_access` module is invoked, the `/etc/security/access.conf` file is searched for the first entry that matches the `username` and `tty` or `hostname` pair. If no match is found, then access is granted.

For example, suppose that you wish to restrict login access to certain users from certain hosts on a particular system; let's call the local host `pyramid`. Example 5-21 illustrates a sample `/etc/security/access.conf` file that provides access restrictions on `pyramid`. The line numbers in Example 5-21 are provided for clarity and are not part of the file. In this case, line 2 disallows all access from the domains, `evil.com` and `spam.org`. Line 3 disallows all access at the console except by root. Line 4 grants access to all users except root if the connection is arriving from the 172.17.0.0 network. Line 5 grants access to all members of the `wheel` group and to the user `paul` from the host `leghorn`. Line 6 denies all other access.

Table 5.12 Fields in `/etc/security/access.conf`

Field	Description
`permission`	Either + indicating access is allowed or – indicating access is denied.
`users`	A space-separated list of usernames, groupnames, or netgroups. All netgroup names must be preceded by `@`. The special wildcard `ALL` may also be used to always match in this field. You may also use the special keyword `EXCEPT` to conditionalize a list.
`origins`	A space-separated list of ttynames, hostnames, domainnames (any name beginning with a "`.`"), or network addresses (the network portion of the IP address ending in a "`.`"). The wildcards `ALL` (which always matches) and `LOCAL` (which matches any name not ending with a "`.`") may also be used. You may also use the special keyword `EXCEPT` to conditionalize a list.

Example 5-21 Sample `/etc/security/access.conf` File

```
# access.conf file
-:ALL:.evil.com .spam.org
-:ALL EXCEPT root: tty1
+:ALL EXCEPT root:172.17.
+:wheel paul:leghorn
-:ALL:ALL
```

Now, simply add the line

```
account    required     /lib/security/pam_access.so
```

as desired to any of the configuration files in the `/etc/pam.d` directory. Example 5-22 shows this entry in **`bold`** in the `/etc/pam.d/login` file.

Example 5-22 Adding `pam_access` to the `/etc/pam.d/login` File

```
auth       required     /lib/security/pam_securetty.so
auth       required     /lib/security/pam_pwdb.so
auth       required     /lib/security/pam_nologin.so
account    required     /lib/security/pam_pwdb.so
account    required     /lib/security/pam_access.so
password   required     /lib/security/pam_cracklib.so minlen=20\
         retry=3 type=SECRET
password   required     /lib/security/pam_pwdb.so md5 use_authtok
session    required     /lib/security/pam_pwdb.so
```

Any attempted access from a denied location will result in a `Permission denied` error message, as shown in Example 5-23, where Paul attempts to log in at the console.

Example 5-23 Failed Login Attempt Due to `pam_access`

```
pyramid login: paul
Password:

Permission denied
pyramid login:
```

All failed attempts due to `pam_access` are logged in `/var/log/messages` by default. See Chapter 8 for further information about log files.

Using `pam_lastlog`

This module provides the capability of displaying the last time-logged-in message and the `You have new mail` message as module type `auth` and `session`, respectively. The former is of greater concern than the latter, as it gives away information about the computing environment. Fortunately, `pam_lastlog` gives you control over what is displayed.

The `pam_lastlog` module may operate in either module type `auth` or `session`. As an `auth` module it serves the purpose of controlling `lastlog` (see *One Other Command* on page 153 of Chapter 7 for more details) displays after a

Table 5.13 Arguments of `pam_lastlog` as Module Type `auth`

Argument	Description
`debug`	Provides verbose output to `syslog`.
`nodate`	Suppresses the display of the date of last login by this user.
`noterm`	Suppresses the display of the terminal name used in the last login of this user.
`nohost`	Suppresses the display of the host from which this user last logged in. By utilizing this argument, hostnames in your environment are not disclosed.
`silent`	Suppresses the entire `lastlog` message.
`never`	If the user has never logged in before, this will cause a welcome message to be displayed.

user login. In this mode it takes the arguments listed in Table 5.13. To use this module, simply put a record similar to the following line in all appropriate `/etc/pam.d` configuration files (e.g., `login`, `rlogin`, `rsh`).

```
auth    optional    /lib/security/pam_lastlog.so nohost
```

Notice the `optional` control flag. If you use `required` instead, no one will be able to log in! You also probably want to put this line last in your `auth` stack. In this example, the `lastlog` message will be displayed, but no previous host information will be shown.

As a `session` module, `pam_lastlog` informs the user about electronic mail. It takes no arguments and once again must use the `optional` control flag. Here is a sample entry:

```
session    optional    /lib/security/pam_lastlog.so
```

Once again, make sure that this entry appears in each appropriate `/etc/pam.d` configuration file.

NOTE

Another application may display `lastlog` or e-mail information after the PAM authentication steps are complete. Any such applications would obviate the configuration of `pam_lastlog`.

Using `pam_rhosts_auth`

The evils of the Berkeley r-commands were extolled in Chapter 1, where we looked at some of the vulnerabilities associated with these utilities. In Chapter 11, we will look at completely replacing the Berkeley r-commands.

Table 5.14 Arguments to `pam_rhosts_auth`

Arguments	Description
`debug`	Generates more output to `syslog`.
`no_hosts_equiv`	Disables `/etc/hosts.equiv` functionality.
`no_rhosts`	Ignores `$HOME/.rhosts` files.
`no_warn`	Suppresses warning messages to the user. As of this writing, the module generates no warning messages anyway!
`privategroup`	Normally, if a `$HOME/.rhosts` file is writable by other than the owner, this module will assume the file has been compromised and return a failure. This argument will allow the `$HOME/.rhosts` file to be writable by the owner and the owner's group if the owner's group is a UPG.[*] This will be true if the UID and GID are identical and greater than 500.
`promiscuous`	The default behavior of this module is to ignore + wildcards in `$HOME/.rhosts` and `/etc/hosts.equiv` files. This argument will allow such wildcards. Use of this argument cannot be advised. See Chapter 3.
`suppress`	Suppresses error messaging to `syslog`.

[*]See *Group Account Management* on page 69 of Chapter 4 for a discussion of UPGs.

Here we will look at using `pam_rhosts_auth` to limit or altogether eliminate trusted hosts.

This module is an `auth`-type module only. It accepts the arguments listed in Table 5.14. Configuration of this module is highly system specific and site dependent. Allowing this functionality internally may be acceptable and, in the eyes of your users, necessary. On certain systems, such as restricted servers, systems within perimeter networks,[5] and firewalls, `$HOME/.rhosts` and `/etc/hosts.equiv` files should be strictly regulated if not completely forbidden.

You will want to place entries for this module in `/etc/pam.d/rsh` and `/etc/pam.d/rlogin` and any other PAM-aware service that uses these files. To affect the traditional implementation (allow trusted hosts), use an entry such as

```
auth    sufficient    /lib/security/pam_rhosts_auth.so
```

at the top of the `auth` stack (minimally, this module must appear before the module in the stack that prompts the user for his or her password) in `/etc/`

5. Perimeter networks (or DMZs) typically consist of systems that are available to external access. See the references listed in *General Firewall References* on page 488 of Chapter 16 for further information.

pam.d/rsh and /etc/pam.d/rlogin. A more restrictive entry could be placed in these same files on sensitive systems. For example, an entry of the form

```
auth   required   /lib/security/pam_rhosts_auth.so no_rhosts
```

at the top of the auth stack (or, again, before a module generating a password prompt) would completely disable the use of $HOME/.rhosts files, but would preserve the use of /etc/hosts.equiv (which does not apply to root).

One-Time Password Support

Both pam_opie and pam_skey support OTP. These are discussed in Chapter 6.

PAM and the `other` Configuration File

So far we have discussed discrete PAM-aware applications and how to grant or limit access through PAM by those applications. It turns out that, if a PAM-aware application has no configuration file of its own, Linux-PAM will supply a special set of modules from the /etc/pam.d/other file. This file is normally used to reject all other requests and, by default, will likely appear as in Example 5-24. For example, if you download a PAM-aware application, such as ssh 1.2.27 (see Chapter 11), and fail to provide an ssh configuration file in /etc/pam.d, then the other file is used, and, if it is as in Example 5-24, then ssh connections will always fail. Furthermore, pam_deny generates no messages whatsoever! Unfortunately this means that you may end up spending a lot of time trying to figure out why something doesn't work. All pam_deny does, as its name suggests, is deny all requests for any available module type.

Example 5-24 A Common /etc/pam.d/other File

```
auth     required      /lib/security/pam_deny.so
account  required      /lib/security/pam_deny.so
password required      /lib/security/pam_deny.so
session  required      /lib/security/pam_deny.so
```

But there is good news! There is another module, the pam_warn module, that logs to syslog informational messages, including the service requested, the terminal name, the username, the remote username, and the remote hostname. The pam_warn module operates only for module type auth and password. So you may want to modify your /etc/pam.d/other file as in Example 5-25. In this way, all other services that make auth or password requests will have log entries generated. The pam_warn module is not limited to use with pam_deny. It may be used in any auth or password stack to generate additional log messages.

Example 5-25 Administrator Friendly /etc/pam.d/other File

```
auth     required      /lib/security/pam_warn.so
auth     required      /lib/security/pam_deny.so
```

Example 5-25 Administrator Friendly `/etc/pam.d/other` File (Continued)

```
account  required       /lib/security/pam_deny.so
password required       /lib/security/pam_warn.so
password required       /lib/security/pam_deny.so
session  required       /lib/security/pam_deny.so
```

This additional logging capability of `pam_warn` has obvious debugging advantages, but it also has advantages from the security perspective. For example, if you have identified some suspicious activity, you may want to add `pam_warn` in the `auth` stacks surrounding the services in question. You may also want to use the `debug` argument on those modules that support it for more detailed auditing information. Pay close attention when you do this because your log files will get large quickly. And, if the bad-guys are already in, they're reading the log files, too!

There is one other module in this category. It is the antithesis of `pam_deny`. It is called `pam_permit` and—you guessed it—it categorically allows access. It operates for all module types and should be used with great caution, if ever.

Additional PAM Options

There are many other PAM modules, some of which are discussed throughout this book. In *Available PAM Modules* on page 109, many of the modules, both currently available and under development, are listed; in *PAM-Aware Applications* on page 112, many available applications are described.

PAM LOGS

PAM modules log to facility `auth` in `syslog`. What this means is that, in your log, messages will end up in the file specified by the `auth` facility entry in `syslog`. On Red Hat 5.2/6.0, the default location is `/var/log/messages`. We discuss `syslog` in Chapter 8.

AVAILABLE PAM MODULES

Table 5.15 provides a list and brief description of many available PAM modules. Some come with the Red Hat (or other) distributions, while others require downloading. Those that come with Red Hat 5.2/6.0 are so noted (and may be found at `http://www.redhat.com/`); for all others, a web site is specified and an author, if known, is provided. If your system already supports these modules, they will be found in either `/lib/security` or `/usr/lib/security`. If you download and add one, make sure that you put it in the correct directory.

Table 5.15 Overview of PAM Modules

Module	Availability	Description
`pam_access`	Red Hat 5.2/6.0	Reads the file `/etc/security/access.conf` to determine whether the user/tty or user/host pair is to be granted or denied access.
`pam_console`	Red Hat 6.0 or publicly available	Sets up permissions and device ownership when logging in at a physical console device. Expects the `/etc/security/console.perms` file for permission and ownership parameters; expects the `/etc/security/console.apps/` directory for services. Supports `auth required` and `session optional` module type/control flag pairs.
`pam_cracklib`	Red Hat 5.2/6.0	Supports only `password` module type. Used for checking password choices against the `cracklib` and disallows any choices found there.
`pam_deny`	Red Hat 5.2/6.0	Supports all module types. Always returns a failure.
`pam_env`	Red Hat 5.2/6.0	Supports `auth` module type only. Uses the `/etc/security/pam_env.conf` file to set shell environment variables.
`pam_filter`	Red Hat 5.2/6.0	Supports all module types. This module offers the capability of capturing as much as every keystroke of a session. Requires a filter program, not included.
`pam_ftp`	Red Hat 5.2/6.0	Supports module type `auth` only. Implements anonymous `ftp`.
`pam_group`	Red Hat 5.2/6.0	Supports module type `auth` only. Sets GID based upon `/etc/security/group.conf` file (syntax nearly identical to `/etc/security/time.conf`, which is discussed in *The /etc/security/time.conf File* on page 96).
`pam_if`	Publicly available	Supports all module types. A simple conditional used to manage stack execution behavior. Available from `http://www.dcit.cz/~kan/pam/`. This module is discussed in *OPIE and PAM* on page 143.
`pam_lastlog`	Red Hat 5.2/6.0	Supports module type `auth` only. Used to control the display of last login information.
`pam_limits`	Red Hat 5.2/6.0	Supports module type `session` only. Uses the `/etc/security/limits.conf` file to determine whether or not users may log in based on available system resources.
`pam_listfile`	Red Hat 5.2/6.0	Supports module type `auth` only. Allows for the use of access control lists based on users, ttys, remote hosts, groups, and shells.
`pam_mail`	Red Hat 5.2/6.0	Supports module type `auth` only. Provides the `You have new mail` service.

Table 5.15 Overview of PAM Modules (Continued)

Module	Availability	Description
`pam_nologin`	Red Hat 5.2/6.0	Supports module type `auth` only. Provides the check for the existence of the `/etc/nologin` file, which, if it exists, will display the contents of the file and fail `auth`.
`pam_opie`	Publicly available	Supports module type `auth` only. Presents an OPIE challenge and requires an OPIE one-time password. Available from `http://www.tho.org/~andy/pam-opie.html`. This module is discussed in *OPIE and PAM* on page 143.
`pam_permit`	Red Hat 5.2/6.0	Supports all module types. Always returns success.
`pam_pwdb`	Red Hat 5.2/6.0	Supports all module types. Replaces the `pam_unix_*` modules. Colocates authentication databases depending upon the `/etc/pwdb.conf` file.
`pam_pwdfile`	Publicly available	This module was announced as this book was in its final stages. It is an authentication-only module that allows for the specification of alternate password files. In this way you can configure separate passwords for various services. For example, you could have one set of usernames and passwords for IMAP and an entirely different set for everything else. You will find this module at `http://espresso.ee.sun.ac.za/~cabotha/pam_pwdfile.html`.
`pam_radius`	Red Hat 5.2/6.0	Supports module type `session` only. Provides the session service for users authenticated through RADIUS.
`pam_rhosts_auth`	Red Hat 5.2/6.0	Supports module type `auth` only. Provides for authentication through `$HOME/.rhosts` files. May be configured to allow or deny such authentication.
`pam_rootok`	Red Hat 5.2/6.0	Supports module type `auth` only. Allows the root user access without requiring a password. Makes sense only when used with the `sufficient` control flag.
`pam_securetty`	Red Hat 5.2/6.0	Supports module type `auth` only. Applies only to root. Checks to see if root is logging in from one of the devices listed in `/etc/securetty`. If so, it returns success; otherwise it fails.
`pam_shells`	Red Hat 5.2/6.0	Supports module type `auth` only. Authenticates users if their default shell is listed in `/etc/shells`.
`pam_stress`	Red Hat 5.2/6.0	This module is used for debugging and stress testing PAM-aware applications.
`pam_tally`	Red Hat 5.2/6.0	Supports module type `auth` only. Keeps track of the number of login attempts made and can deny access based upon a specified number of failed attempts.

Table 5.15 Overview of PAM Modules (Continued)

Module	Availability	Description
pam_time	Red Hat 5.2/6.0	Supports module type account only. Restricts access based on user, tty, service, and time as specified in /etc/security/time.conf.
pam_tcpd	Publicly available	Supports module type auth only. Implements TCP_wrappers-style access control, logging, and functionality through /etc/hosts.allow and /etc/hosts.deny. TCP_wrappers is discussed in Chapter 10. The module is available from http://web.tis.calinet.it/macchese/pam/pam_tcpd.html.
pam_unix_acct pam_unix_auth pam_unix_passwd pam_unix_session	Red Hat 5.2/6.0	These modules provide similar functionality to pam_pwdb except that the authentication database is either /etc/passwd or NIS.
pam_unix-new	Publicly available	Incorporates the above four modules into one and implements many of the features of pam_pwdb. Available at ftp://hunter.mimuw.edu.pl/pub/users/baggins/PAM/.
pam_warn	Red Hat 5.2/6.0	Supports module types auth and password only. This module generates a log message including the remote user and remote host (if available) through the syslog utility.
pam_wheel	Red Hat 5.2/6.0	Supports module type auth only. Provides a way to restrict access to root to those users who are members of the wheel group.
pam_xauth	Red Hat 6.0 or publicly available	Supports module type session with control flag optional only. This module automatically passes X Window System magic cookies to other users (for example, through su), thus allowing effective UIDs to open X applications without requiring the use of the xhost command.

PAM-Aware Applications

Table 5.16 is a list of many of the applications that are PAM-aware, meaning that the application has the necessary calls to invoke PAM. As in Table 5.15, those available in the Red Hat 5.2/6.0 distribution are so noted.

Important Notes about Configuring PAM

This chapter provides an introductory look at PAM. Many examples are described and some usage tips are provided. Doubtless, however, many of you will have configuration ideas of your own. This section provides some simple, but important, notes about configuring PAM for your environment.

Table 5.16 Overview of PAM-Aware Applications

Application	Availability
`chfn`	Red Hat 5.2/6.0
`chsh`	Red Hat 5.2/6.0
`ftp`	Red Hat 5.2/6.0
`imap`	Red Hat 5.2/6.0
`linuxconf`	Red Hat 5.2/6.0
`linuxconf-pair`	Red Hat 5.2/6.0
`login`	Red Hat 5.2/6.0
`mcserv`	Red Hat 5.2/6.0
`other`	Red Hat 5.2/6.0
`passwd`	Red Hat 5.2/6.0
`ppp`	Red Hat 5.2/6.0
`rexec`	Red Hat 5.2/6.0
`rlogin`	Red Hat 5.2/6.0
`rsh`	Red Hat 5.2/6.0
`samba`	Red Hat 5.2/6.0
`su`	Red Hat 5.2/6.0
`sudo`*	Publicly Available
`vlock`	Red Hat 5.2/6.0
`xdm`	Red Hat 5.2/6.0
`xlock`	Red Hat 5.2/6.0

*Discussed in Chapter 9.

First, and foremost, *always* copy your existing, functioning `/etc/pam.d` configuration files before making any changes. It is entirely possible to lock out all users, including root, through PAM misconfiguration. By retaining working copies, you will always be able to boot into single-user mode (see *A Note about LILO* on page 22 in Chapter 3 for information about booting into single-user mode), correct the configuration, and bring the system back up.

Second, configure your `/etc/pam.d` directory with the permissions read/write/execute by root only, and configure its contents read/write by root only. No one else needs to read the contents of this directory. You may accomplish this with

```
# chmod u=rwx /etc/pam.d
# cd /etc/pam.d
# chmod u=rw *
```

Linux05 Page 114 Monday, February 7, 2000 10:06 AM

Third, test your configuration ideas in a safe, preferably nonproduction environment. Try as many possible variations as you can think of before going live. Remember, the order of PAM modules in a stack is significant. Different orders will produce different behavior. Normally you will want `pam_pwdb` last in the `auth` stack. Don't forget the control flag settings either. The use of different control flags will cause radically different behavior in many cases. The same holds true for any arguments associated with the different modules. Remember, different module types for the same PAM module will support different arguments.

Fourth, and last, Linux is publicly available software. So are the PAM modules. There aren't any exacting quality assurance programs before release and distribution. In fact, quite frequently, you are the quality assurance mechanism! In short, your success with PAM will vary depending upon the release you obtained, the version of Linux you are running, your hardware platform, and perhaps other factors. Use the resources in Appendix A and any other support mechanisms available to you.

THE FUTURE OF PAM

From the contents of this chapter, it is hopefully clear that a lot of work has been done with PAM. The future is actually quite bright. The Open Software Foundation (OSF) released RFC 86.0 (RFCs are defined in *Request for Comment* on page 40 of Chapter 3) in October 1995 specifically for PAM. Additional work is being done to enhance the control flag option to allow system administrators to specify actions based on return codes from each module. Many of the modules noted in Table 5.15 on page 110 are under development, as is true of many of the applications listed in Table 5.16 on page 113. Given its inherent flexibility, there is no doubt that PAM will be with us for a while.

SUMMARY

This chapter described pluggable authentication modules. We looked at a number of examples for configuring and using PAM. We also reviewed many of the available PAM modules and applications. We noted the flexibility and security features provided by PAM.

FOR FURTHER READING

The best documentation available for PAM is the Linux-PAM System Administrator's Guide, which may be found at

```
http://www.kernel.org/pub/linux/libs/pam/
```

or one of its mirrors. The guide is available in numerous formats including postscript and html.

On-Line Documentation

```
/usr/doc/pwdb-0.55/pwdb.txt
/usr/doc/pam-0.64/ps/pam.ps
/usr/doc/pam-0.64/rfc86.0.txt
```

Page 116

To Be Blank