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

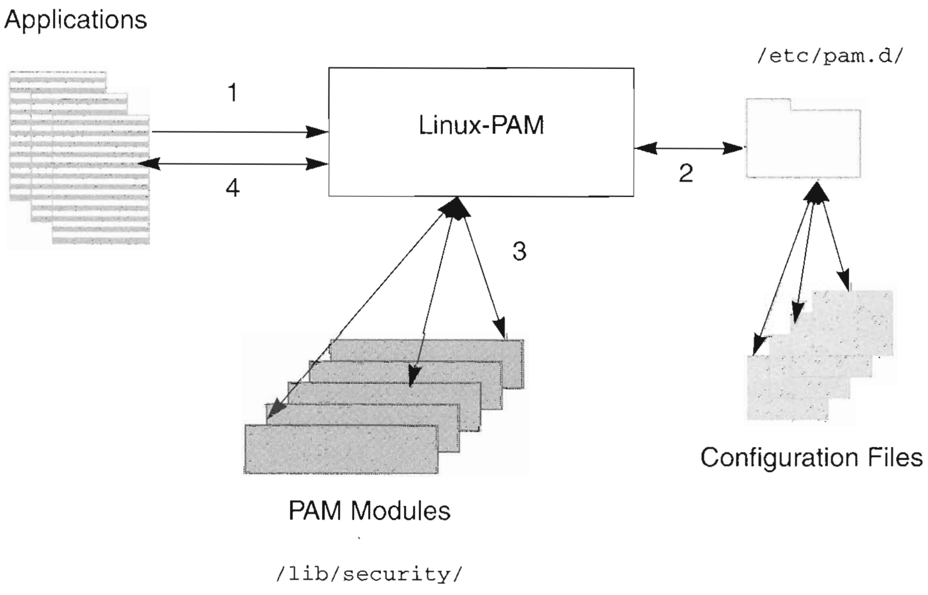

Fig. 5.1 Linux-PAM Overview

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.