

PAM (Pluggable Authentication Modules)

Kurt Seifried

PAM (Pluggable Authentication Modules) provides the backbone of most authentication in modern Linux systems (and can be implemented in others, such as Solaris), yet it is typically ignored and woefully under-utilized. Anytime you log into a modern Linux system, whether via **telnet**, **ssh**, **pop**, **ftp**, and so on, you are using PAM to process the authentication request. Anytime you need to authenticate to change your password (**passwd**), or login shell (**chsh**), you are talking to PAM.

In the "olden" days of Linux (and UNIX, and most computing systems come to think of it) the usernames and passwords were stored in a text file. This file, of course, had to be readable to everyone, which resulted in many security breaches. The first improvement to this system was the introduction of encrypted passwords. Unfortunately, the early algorithms that used crypt for password encryption were good 10 years ago, but as modern computers got faster, it became feasible for an attacker to copy the password files and attempt to brute-force guess all the passwords on a fast system. The attacker could then use the passwords to gain access to the target system. This type of attack bypassed any mechanisms to prevent brute-force guessing the password (e.g., after three bad logins, you have to wait a minute to try again). This problem was partially fixed by the move to "shadow" passwords.

With shadow passwords, instead of keeping the user data and encrypted password in the same file, the password was kept in a separate file that only the system could access. Unfortunately, any program that needed to authenticate users had to be recompiled with shadow password support, and this process could take quite a while because every network daemon, and numerous local utilities, had to be updated. If you later decided to use smart cards to authenticate users, you would have to recompile everything. Additionally, the system was rather inflexible -- if you wanted only certain users to access a service (such as **ftp**), it would need some internal mechanism to support this, and many network daemons do not have these mechanisms.

Then came PAM. PAM provides a middle layer to authentication. This means the application is not "aware" of the actual authentication method, which results in a more flexible system. As long as the application supports PAM, you can basically use any (or several) criteria for authenticating users, as well as other modules allowing you to do extended logging, filtering, manipulation of the users environment, etc.

Unfortunately, most people don't know about PAM and many of the cool things it can do. For example, I was at a security convention in March and ended up in the elevator with a smart-card vendor, so I naturally inquired if they would be rolling out Linux support. He said he wasn't sure, it might take too much work (or something similar). I explained that all he had to do was write a PAM module and he could support every Linux distribution, and the applications wouldn't need to be changed at all to support it. His response was "what is PAM?" I've had systems administrators ask me how they can restrict user's access to services. Some organizations, for example, would like certain users to have only POP access to their email, and give other users IMAP access. How can you do this without using two mail servers or hacking (in the traditional sense) your POP and IMAP daemons? With PAM, you can set this up in about five minutes by adding one line to the configuration and making a list of users. It's really that easy.

Limiting Users

When users have shell accounts, it is generally a good idea to limit the amount of system resources they can use, because sometimes programs go berserk, and sometimes users make mistakes. Most shells have built-in limiting (**bash has ulimit**, for example). However these are harder to apply

PAM Pluggable Authentication Modules)

Kurt Seifried

selectively (i.e., they are global, or per-user unless you do some complicated scripting for the global ones). PAM, on the other hand, supports per-user, per-group, and global groups of users. Additionally, like shell limitations, you can place soft and hard limits. (Soft limits are like warnings; hard limits are the law.) Another problem with the shell limitations is that some lack certain features. (**tcsh**, for example, only has hard limits.) To enforce limits through PAM, add the following line to the configuration file for your login services, which is usually in **/etc/pam.d/ssh** (or **telnet** for example):

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

The path to the file **pam_limits.so** may vary, but simply look at the other lines in the file for the location. All PAM modules are usually in one central directory. Now all you need to do is put limits in **/etc/security/limits.conf**. Again, the location may vary but it should be in **/etc**, and called **limits.conf**. Lines are in the form:

```
<domain>    <type>    \  
    <item>    <value>
```

A domain is either a username (bob, sally, root), a group name (sales, bin, nobody), or you can use the wildcard ***** to cover everyone (meaning it is a default entry). The type is either soft or hard, and hard limits are absolute. Be careful when setting hard limits on administrative accounts! The item can be:

core -- Limits the core file size (KB); usually set to 0 for most users to prevent core dumps.

data -- Maximum data size (KB).

fsiz -- 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 (MIN).

nproc -- Maximum number of processes.

as -- Address space limit.

maxlogins -- Maximum number of logins for this user or group.

priority -- The priority to run user process with.

For example, you can deny core files to everyone with the simple addition of:

```
* hard core 0
```

You could also create a group called "sales", add all the sales people to it, and then restrict their maximum number of logins to 2:

```
@sales hard maxlogins 2
```

Additionally, any login events that violate the limits are logged. For example, the user **test1** is limited to 1 maxlogin:

```
May 31 20:54:09 server sshd[8436]: \  
    Accepted password for test1 from \  
    10.3.0.100 port 1497  
May 31 20:54:09 server PAM_pwd[8436]: \  
    (sshd) session opened for user test1 \  
    by (uid=0)  
May 31 20:54:09 server PAM_pwd[8437]: \  
    (login) session opened for user test1 \  
    by (uid=0)
```

PAM Pluggable Authentication Modules)

Kurt Seifried

```
by test1(uid=0)
May 31 20:54:13 server sshd[8458]: \
Accepted password for test1 from \
10.3.0.100 port 1498
May 31 20:54:14 server PAM_pwdb[8458]: \
(sshd) session opened for user test1 \
by (uid=0)
May 31 20:54:14 server pam_limits[8458]: \
Too many logins (max 1) for test1
May 31 20:54:16 server sshd[8458]: fatal: \
PAM session setup failed: Permission \
denied
May 31 20:54:16 server PAM_pwdb[8458]: \
(sshd) session closed for user test1
Events, such as violating the maximum number of processes allowed, are not logged.
However, the user will see something like:
bash$ ls
bash: fork: Resource temporarily \
unavailable
```

This can lead to problems if the user has exhausted his available number of processes. Until a process exits, the user cannot do anything -- he can't even run kill. Again, be very careful when placing limits on administrative accounts. Additionally, you can give the location of the config file, so you can specify different limits for a user if he logs in via **telnet**, as opposed to logging in via **ssh**. Another thing to remember is that user limits have priority over group limits. You can use "-" to disable the limit, so you might want to limit "admins" to only 50 processes, but remove all limits for **root**:

```
@admins hard nproc 50
root -
```

Controlling User Passwords and Aging

One area where Linux sometimes receives criticism is in its apparent inability to enforce password policies (i.e., passwords must be seven letters long and changed every 30 days). Through PAM's "**cracklib**" module, you can easily enforce strong passwords with minimum length, types of characters used, and so on. The system used is interesting -- you set a **minlen** (minimum length) for the password. For example, **minlen=8** means the password needs to be eight characters long, and you can assign "bonus" values to uppercase, lowercase digits, and other characters. The default is usually nine points for the password and each special character is worth one, which means the password will minimally be eight characters if it is just lowercase letters. For each number, uppercase, or other character added, then the length is one less. For example, the password **mnagrwuc** is equal to **najdhw5** in points. There is a defaulted minimum of six, and anything more is just icing on the cake. If you wish to enforce stronger passwords, set the **minlen** higher. To encourage the use of non-alphanumeric passwords, set the "bonus" on other character to two or three.

It is also a good idea to make users aware of your intentions, rather than forcing them to try changing their passwords several dozen times before they hit a combination that is accepted. Password expiry is not handled through PAM -- PAM handles authentication only, it doesn't clean under the couch. Instead, the program "**chage**" is used:

```
[root@server html]# chage -l test
Minimum: 0
Maximum: 99999
Warning: 7
```

PAM Pluggable Authentication Modules)

Kurt Seifried

```
Inactive: -1
Last Change: Jun 02, 2000
Password Expires: Never
Password Inactive: Never
Account Expires: Never
```

You can then set the minimum, maximum, inactivity timeout, account expiry date, and warning time.

For example:
[root@server html]# chage -m 1 -M 30 test

sets the minimum time between changing passwords to one day, and to a maximum of 30. The line in **/etc/shadow** looks like:

```
test:$1$H5pOaf5t$KKFnYHcmzrgbC95/ \
rCyyY0:11110:1:30:7:::134540308
```

The defaults for minimum password change is 0 (users can change it as often as they want), and the maximum is 99999 days (273 years not counting leap years). Using tools like **awk** and **sed** allows you to easily create groups of users and then assign standard password aging schemes to them.

Limiting User Access to Services

You can trivially limit which users (or groups) have access to network services (assuming, of course, that they require a network login). The simplest example of this is the mail server running POP and IMAP. IMAP leaves mail on the server, so if each user used it, the mail server would get overloaded. However for users that travel, IMAP is a lifesaver, especially on those slow links. PAM has an authentication module, called **pam_listfile.so**, that checks a file for a list of user names and on finding the specified username (or group), it can either deny or grant access to the service. You need to place an entry in the program's PAM configuration file, usually in **/etc/pam.d/servicename**. For example, if you wanted to allow only certain users access with IMAP:

```
auth required /lib/security/pam_listfile.so \
  item=user sense=allow \
  file=/etc/imapusers-allow onerr=fail
```

use the auth module **pam_listfile.so** and the item user (or you could use group). If the check succeeds, the user is listed in the file then granted access. Use the file **/etc/imapusers-allow**, and if the name is not found, generate an error (and access is denied).

If, on the other hand, you want to deny access to certain users, use:

```
auth required /lib/security/pam_listfile.so \
  item=user sense=deny \
  file=/etc/imapusers-deny onerr=succeed
```

This is the same as before, except that the default is to allow **onerr=succeed**, and it will only deny a user if he or she is listed in **/etc/imapusers-deny**. As always, a policy of default deny is safer than one that defaults to allowing access. This allows you to specify that if the users (or group) are not found, they are granted or denied access, or if they are not found, they are allowed or denied access. Make sure you test your configuration. Sometimes daemons don't behave or may not have compiled in PAM support (e.g., contributed packages), so make sure it blocks/grants access as expected.

PAM Pluggable Authentication Modules)

Kurt Seifried

Other Tricks

If I covered every cool thing PAM could do, this article would be the size of the entire magazine. You can restrict access to login by time using the **pam_time** module, applying the rule to physical consoles or remote logins (via **telnet**, **ssh**, and the like). One minor glitch with this module is that you cannot enforce the logout time. For example, if you restrict logins from 9 a.m. to 5 p.m., a user can log in at 4:59 p.m. and stay logged in.

Some programs (like games in Red Hat) are configured to use PAM, so you can restrict access to them during business hours. This might be useful if you have a sensitive application that you only want accessed during business hours. Using **pam_env**, you can set user environmental variables. The advantage of using this instead of per-shell configuration files is that it is global, so if a user changes his shell, he is still covered. There are also PAM modules to do authentication via kerberos, RADIUS, SMB, smartcards, and tokens. If you do purchase a smartcard or token solution for Linux, make sure the vendor has PAM modules, because this will simplify implementation.

Summary

PAM provides a very flexible framework for handling authentication and other related services. One of the only problems with PAM is the added overhead, which on an extremely busy system (such as a POP server handling 10,000 users) can cause problems. However, it won't pose a significant problem for most people. Additionally, by using PAM, you can concentrate most of your configuration files into several easily managed files that affect users as individuals, groups, or globally, which reduces the amount of scripting needed to accomplish this through other means.

Resources

Linux PAM documentation -- <http://www.kernel.org/pub/linux/libs/pam/>

Sun PAM documentation (old) --
<http://www.sun.com/software/solaris/pam/>

Authenticate users to SMB via PAM -- http://www.csn.ul.ie/~airlied/pam_smb/

Packages for authenticating users to SMB via PAM -- http://rpmfind.net/linux/RPM/pam_smb.html

PAM Smartcard module --
<http://www.linuxnet.com/applications/applications.html>

PAM Cryptocard module -- <http://www.jdimedia.nl/igmar/pam/>

Linux Security Knowledge Base (more PAM articles) -- <http://www.securityportal.com/lskb/articles/>