

Understanding PAM

PAM is the Pluggable Authentication Module, invented by Sun. It's a beautiful concept, but it can be confusing and even intimidating at first. We're going to look at it on a RedHat system, but other Linuxes will be similar - some details may vary, but the basic ideas will be the same.

The first thing to understand is that PAM is NOT something like tcpd (tcp wrappers) or xinetd that encloses and restricts access to some service. An application needs to be "PAM aware"; it needs to have been written and compiled specifically to use PAM. There are tremendous advantages in doing so, and most applications with any interest in security will be PAM aware.

PAM is about security - checking to see that a service should be used or not. Most of us first learned about PAM when we were told that login was using it, but PAM can do much more than just validate passwords. A lot of applications now use PAM - even things like SAMBA can call on PAM for authentication.

The big advantage here is that security is no longer the application's concern: if PAM says its OK, its OK. That makes things easier for the application, and it makes things easier for the system administrator. PAM consults text configuration files to see what security actions to take for an application, and the administrator can add and subtract new rules at any time. PAM is also extensible: should someone invent a device that can read your brain waves and determine ill intent, all we need is a PAM module that can use that device. Change a few files, and login now reads your mind and grants or denies access appropriately. We're a bit away from that feature, but there are a tremendous number of available PAM modules that administrators can use.

Configuration Files

On modern RedHat systems, the configuration files are found in /etc/pam.d, one file for each PAM aware application (plus a special "other" file we'll get to later). One word of warning: changes to these files take effect instantly. You aren't going to get logged out if you make a mistake here. but if you DO screw up and blithely log out, you may not be able to log back in. So test changes before you exit.

We're going to use a very simple example to get started here. In a number of articles here, we've talked about SSH Security. Most of those articles have been about changes to ssh's configuration files, but here we'll use PAM to add some additional restriction: the time of day you are allowed to use ssh. To do this, we need a PAM module called pam_time.so - it's probably in your /lib/security/ directory already. It uses a configuration file "/etc/security/time.conf". That file is pretty well commented, so I'm not going to go into detail about it and will just say that I added the line

```
sshd;*:*;!A12200-0400
```

which says that sshd cannot be used between 10:00 PM and 4:00 AM. I'm usually rather soundly asleep between those times, so why let ssh be used? I could still login at the console if I woke up with an urgent need to see an ls of my /tmp directory, but I couldn't ssh in, period. Configuring the time.conf file by itself doesn't affect ssh; we need to add the pam module to /etc/pam.d/sshd. My file ends up looking like this:

```
##PAM-1.0
account      required      pam_time.so
authrequired pam_stack.so service=system-auth
authrequired pam_nologin.so
account      required      pam_stack.so service=system-auth
password     required      pam_stack.so service=system-auth
session      required      pam_stack.so service=system-auth
session      required      pam_limits.so
session      optional     pam_console.so
```

Understanding PAM

I put the time.so module first so that it is the very first thing that is checked. If that module doesn't give sshd a green light, that's the end of it: no access. That's the meaning of "required": the module HAS to say that it is happy. The "account" type is specified here. That's a bit of a confusing thing: we have "account", "auth", "password" and "session". The man page isn't all that helpful:

account - provide account verification types of service: has the user's password expired?; is this user permitted access to the requested service?

Authentication: establish the user is who they claim to be. Typically this is via some challenge-response request that the user must satisfy: if you are who you claim to be please enter your password. Not all authentications are of this type, there exist hardware based authentication schemes (such as the use of smart-cards and biometric devices), with suitable modules, these may be substituted seamlessly for more standard approaches to authentication - such is the flexibility of Linux-PAM.

Password: this group's responsibility is the task of updating authentication mechanisms. Typically, such services are strongly coupled to those of the auth group. Some authentication mechanisms lend themselves well to being updated with such a function. Standard UNIX password-based access is the obvious example: please enter a replacement password.

Session: this group of tasks cover things that should be done prior to a service being given and after it is withdrawn. Such tasks include the maintenance of audit trails and the mounting of the user's home directory. The session management group is important as it provides both an opening and closing hook for modules to affect the services available to a user.

I think that the distinction between account and session in that man page is a little confusing. I think it would be quite reasonable to think you should use "session" for this module. Now, sometimes you have a man page for the module that shows you what to use, but pam_time doesn't help us there. Technically, it's not up to the library: the application is the one that is checking with account or session, but keep this in mind: session happens AFTER authentication. I liked the older PAM manual better, which said:

- auth modules provide the actual authentication, perhaps asking for and checking a password, and they set "credentials" such as group membership or kerberos "tickets."
- account modules check to make sure that the authentication is allowed (the account has not expired, the user is allowed to log in at this time of day, and so on).
- password modules are used to set passwords.
- session modules are used once a user has been authenticated to allow them to use their account, perhaps mounting the user's home directory or making their mailbox available.

For me, that was more clear.

Stacking

In this case, I only wanted to apply this restriction to ssh. If I'm physically at the box, I want no time restrictions. If I DID want these same restrictions, I'd make the same change to /etc/pam.d/login. But what if there are a whole bunch of things I want to apply the same rules to? RedHat has a special module "pam_stack". It functions much like an "include" statement in any programming language. We saw it in my /etc/pamd/sshd file:

```
authrequired      pam_stack.so service=system-auth
```

Understanding PAM

That says to look in /etc/pam.d/system-auth for other modules to use. Both login and sshd have this line (as does just about every other file in /etc/pam.d/), so we can look in system-auth to see what gets called by them:

```
##PAM-1.0
# This file is auto-generated.
# User changes will be destroyed the next time authconfig is run.
auth required      /lib/security/$ISA/pam_env.so
auth sufficient    /lib/security/$ISA/pam_unix.so likeauth nullok
auth required      /lib/security/$ISA/pam_deny.so
auth required /lib/security/$ISA/pam_tally.so no_magic_root onerr=fail
account required  /lib/security/$ISA/pam_unix.so
account required  /lib/security/$ISA/pam_tally.so onerr=fail
file=/var/log/faillog deny=1 no_magic_root even_deny_root_account

password required /lib/security/$ISA/pam_cracklib.so retry=3 type=
password sufficient /lib/security/$ISA/pam_unix.so nullok use_authtok md5
shadow
password required /lib/security/$ISA/pam_deny.so

session required /lib/security/$ISA/pam_limits.so
session required /lib/security/$ISA/pam_unix.so
```

Therefore, if we really wanted our time restrictions to apply to just about everything, we could add it to system-auth. Note the warning about authconfig though, and also consider that you will be making sudden sweeping changes to a LOT of applications and services.

Other

What if a PAM aware app doesn't have a file in /etc/pam.d? In that case, it uses the "other" file, which looks like this by default:

```
##PAM-1.0
auth required/lib/security/$ISA/pam_deny.so
account required/lib/security/$ISA/pam_deny.so
password required/lib/security/$ISA/pam_deny.so
session required/lib/security/$ISA/pam_deny.so
```

That "deny" module is a flat-out no access, red light, stop you dead right here module that is always going to say no. That's excellent from a security point of view, but can be a bit harsh should you accidentally delete something like "login". Login would now use the "other" file, and you couldn't login. That could be unpleasant.

There are many, many useful and clever PAM modules. While our brain wave interpreter doesn't exist yet, many other possibilities are available to you. There are modules to automatically black list hosts that have many failed logins, and much more. See

<http://www.kernel.org/pub/linux/libs/pam/modules.html>.

Wouldn't it be nice if EVERY application were PAM aware? Imagine limiting vi access to certain files, or not allowing rm to remove certain files. Sure, you can do that other ways, but doing it with PAM could give you fine grained control. Maybe someday more apps will be.