GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials Bootcamp Style (Security 401)"
at http://www.giac.org/registration/gsec

Cooking with PAM
GSEC Practical Assignment Version 1.4b (option 1)
Terry Hartman
August 20, 2003

Abstract

What is PAM? For the purposes of this document it is NOT the non-stick spray used to coat pans when cooking food. The PAM acronym stands for Pluggable Authentication Modules. The “cooking" we will be doing in this paper relates to how PAM is used in many *NIX systems as part of your local defense in depth security protection strategy. We will see how to add some new authentication modules to an otherwise "plain" type of configuration to increase the security of your system.

PAM allows you, the local security systems administrator to define exactly how applications will authenticate users. Using PAM, it is possible to change the mechanisms that applications use to authenticate without the need to recompile the application. Traditionally, it has been up to the application to authenticate the user based on it's own compiled-in set of requirements. As newer authentication technologies become available (such as retinal scanners, bio scanners, etc.), the job of authentication can stay current too as easily as adding a new PAM module to the PAM configuration file. So in general, PAM is a generic framework of interfaces (API's) that abstracts the process of authentication from the application. PAM enables users of these applications to supply a single password even though multiple authentication services may be in use.[1][7] These features are implemented by the PAM framework through a process called "stacking".

Therefore, in addition to your use of ACLs, TCP Wrappers, and other security or auditing mechanisms, the understanding and use of PAM modules in a *NIX system IS a valuable part of your security skillset repertoire.

Background

Where did PAM come from? PAM was invented at SUN Microsystems and later included as a component of OSF's Common Desktop Environment (CDE)/Motif[2]. It should be mentioned that PAM is configured differently on a SUN Solaris machine than on most Linux machines. SUN has their own specification and thus, there is only one configuration file in Solaris. However, in a typical Linux installation there is a PAM configuration file for each application. The location of SUN's PAM configuration file is /etc/pam.conf.[11] The newer more recently adopted location for most Linux PAM configuration files is the /etc/pam.d subdirectory. The advantage of the /etc/pam.d/ subdirectory location

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

is that third-party software can easily install policies (via it's application specific PAM configuration file) for its services without the need to edit the /etc/pam.conf file.[6] The newer Linux-PAM configuration and modules is what this paper will focus on.

To jump right in, we will start with the syntax or layout of every configuration file. The syntax is:

```
Type    Control     Module-path      Module-arguments
```

The naming convention used is typically based on the application's name and is: /etc/pam.d/<application>. A common example of this would be: /etc/pam.d/login. The syntax of the PAM configuration file will be discussed in more detail later.

Types of Modules

There are four independent types of PAM module groups, each performing a different service to permit access to the application in question. The groups are: authentication management, account management, session management, and password management. A single PAM module may service only one group or all four and should take into consideration that the independent services may be called in any order or stacked.

Authentication management type modules will provide the functionality to authenticate users and set up user credentials. Account management type modules typically determine if the current users account is still valid. This may include checking for password/account expiration or verifying access hour restrictions. Session management modules provide functionality to set up and terminate login sessions. Password management type modules will provide the functionality to change a user's authentication token or password.

All of the different PAM modules are implemented as shared libraries and are typically stored in the /lib/security directory on Linux machines. On Solaris machines the default would be /usr/lib/security.[3] Each module is responsible for one small part of authentication. After executing, a module will return a status value to the PAM facility, indicating whether it passed or failed to grant access to the user. The module might not even participate in the decision by returning a neutral value.

Stacking of Modules

Early on in the PAM design it was determined that for flexibility, to allow modules to be stacked. [10] This functionality provides for different authentication service modules to perform a specific job and have the return status considered in the

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

overall dispensation for the group. This required the use of a Control Flag to indicate exactly how the return status from each module would play into the overall pass/fail resultant for the service. There are four different Control Flags defined for this job. They are named: required, requisite, sufficient, and optional. In the PAM framework, the ordering of PAM modules is not important. It is the manner in which the Control Flag is specified that can make module order dependency important. The use of the sufficient and requisite control flags will cause order to become important.

If the Control Flag for a module is defined to be required, it means that that the module must return a successful value for the operation to succeed. In the case of stacking, the failure does not get reported until all modules in the stack have completed.

The requisite Control Flag indicates that the module must return successfully for authentication to succeed, but it will cause a return of control immediately to the program. No further execution of modules the stack is performed.

The sufficient Control Flag indicates that the modules report whether or not they succeed. If previous required modules have succeeded, then no more subsequent modules for this type will be executed. If it fails, the remaining modules in the stack will be applied.

The optional Control Flag indicates that the module is non-critical. So whether the module succeeds or fails makes no difference to the success of the entire module stack.

Each module referenced in the stack for a service is processed in the order that it occurs in the configuration file. If all required modules in the stack succeed, then success is returned (error return values from modules with the Control Flags optional and sufficient are ignored). If one or more required modules fail, then the error value from the first required module that failed is returned. When none of the service modules in the stack are designated as required, the PAM facility requires that there be at least one optional or sufficient module that succeeds. If all fail, then the error value from the first service module in the stack is returned. The only exception to the above is caused when the sufficient Control Flag is used. When a service module that is designated as sufficient succeeds, the PAM facility immediately returns success to the application and all subsequent services modules, even required ones, in the stack are ignored, given that all prior required modules had also succeeded. If a prior required module failed, then the error value from that module is returned. So you can see, as stated earlier, the ordering of modules in a stack can become important depending on the way that the Control Flag is specified.

I do need to mention that newer Linux-PAM versions include an even more complex Control Flag return value syntax, but I will not elaborate in great detail.

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

For an exhaustive description of this newer control syntax, one only needs to look to the Linux-PAM System Administrators' Guide[5].

The syntax for this newer syntax for the Control Flag is delimited with square brackets and consists of a series of value=action strings:

[value1=action1 value2=action2...]

The return value "valueX" will be one of roughly 31 predefined keywords. They are: success, open_err, symbol_err, service_err, system_err, buf_err, perm_denied, auth_err, cred_insufficient, authinfo_unavail, user_unknown, maxtries, new_authtok_reqd, acct_expired, session_err, cred_unavail, cred_expired, cred_err, no_module_data, conv_err, authtok_err, authtok_recover_err, authtok_lock_busy, authtok_disable_aging, try_again, ignore, abort, authtok_expired, module_unknown, bad_item, and default.

The "action" can be a positive integer or one of the following tokens: ignore, ok, done, bad, die, and reset. When the "action" is a positive integer, it is indicating that the next X number of modules of this same module-type will be skipped. This allows the system administrator a lot of flexibility to develop a fairly sophisticated stack of modules with multiple paths of execution. The actual execution path taken will be determined by the return values of each individual module in the stack.

What I will call the “traditional” Control Flag(CF) values, can also be explained in terms of this newer more complex CF syntax. I will list them here, but we won't go into further detail as it is beyond the scope of this paper.

Newer Complex CF Style	Traditional CF
[success=ok new_authtok_reqd=ok ignore=ignore default=bad]	required
[success=ok new_authtok_reqd=ok ignore=ignore default=die]	requisite
[success=done new_authtok_reqd=done default=ignore]	sufficient
[success=ok new_authtok_reqd=ok default=ignore]	optional

It should be noted that OpenPAM and Solaris do not implement this syntax so its usage might be considered by some to be non-standard. It is closely tied in with the way Linux-PAM dispatches service calls, which differs from the way that both Solaris and OpenPAM implement them.[12]

Module Arguments

The arguments, if any, to a given module are specified in the PAM configuration file. There are some typical arguments that most modules support, but there may also be module-specific arguments. The man page for the module should

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

indicate these module specific arguments. The arguments that most modules support are: debug, expose_account, no_warn, try_first_pass, use_first_pass, use_mapped_pass, and expose_account.

The debug module tells the module to log extra output to the syslog() utility. Nowarn instructs the module to not output any warning messages. The use_first_pass argument tells the module to compare the password in the password database with the user's initial password (entered when the user authenticated to the first authentication module in the stack). When the passwords don't match, or when no password is entered, the module quits without prompting the user for a password. This option should only be used if the authentication service is designated as "optional" in the configuration file. Like use_first_pass, the try_first_pass indicates to the module that it should compare the password in the password database with the user's initial password, but if the passwords don't match, or when no password is entered, it will prompt the user for a password. The use_mapped_pass is not implemented in the Linux-PAM distribution due to potential problems with U.S. encryption exporting restrictions. The expose_account argument would have the module be more verbose about account information it is processing. The use of this argument is typically not good security practice. It is included for the system administrator to use as dictated by the local security policy.

The argument that you will likely use most often will be the use_first_pass argument. This argument tells the module use the password from the previous “auth” module in the stack and not to prompt for a password. The debug argument can be very useful when you are first setting up your PAM configuration file because it tells the module to send debugging output statements to syslog so you can track it.

The PAM configuration file

As stated previously, the syntax of each PAM configuration file is:

```
Type    Control     Module-path      Module-arguments
```

A typical example for the login process would be something like:

```
account  required      pam_unix.so
auth     requisite     pam_nologin.so
auth     required      pam_unix.so
session  required      pam_unix.so
```

The location and name of this file would be /etc/pam.d/login. Notice that if the Module-path is not specified, it is considered relative to /lib/security on Linux machines and /usr/lib/security on Solaris machines. The above configuration file

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

tells PAM to use pam_unix.so to perform Unix authentication, password management, and user account setup. It uses system calls to retrieve and set password and account information. The pam_unix.so module relies on the /etc/shadow and /etc/passwd files. The pam_nologin.so module provides standard *NIX nologin authentication. In other words, if the file /etc/nologin exists, only root is allowed access. All other users would see the contents of the /etc/nologin file. However, if /etc/nologin is not present, the module returns success. When you reference the man page for pam_nologin.so, you'll see it has no arguments, and only an auth component. Therefore, it should be listed in the configurations for all login methods as a required module and ordered before any sufficient modules.

Depending on the specifics of your site policy, another good security practice would be to disallow all root logins from insecure terminals. This is very easy to implement with a PAM module. All you would need to do is to include the pam_securetty.so module in your /etc/pam.d/login and /etc/pam.d/rlogin PAM configuration files as shown below:

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

What the pam_securetty.so module will do is to prevent root logins from happening on terminals that are not considered secure. This basically disallows all root rlogin attempts for security reasons. Root access would have to occur on a secure terminal such as the system console. The tty's that are considered "secure" would be listed in the /etc/securetty file.

At this time, I should mention the need for a strong "other" PAM configuration file. The default policy that the PAM framework will use is found in the /etc/pam.d/other configuration file. PAM uses the "other" configuration file when it cannot match any configuration file to an application. Therefore since we want to be as security conscious as possible, placing the following lines in the /etc/pam.d/other file would be appropriate.

```
auth        required      pam_deny.so
auth        required      pam_warn.so
account     required      pam_deny.so
password    required      pam_deny.so
password    required      pam_warn.so
session     required      pam_deny.so
```

The pam_deny.so module will always return a failure status to deny access. The pam_warn.so module will log the attempt to the syslog utility.

Another "gotcha" that I will mention is that if you mistakenly happen to delete or severely misconfigure an applications' configuration file, you could possibly lock yourself out of your system. Therefore, when you are experimenting with an

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

applications' PAM configuration file, it is best to have an "open" or simple version as a backup that allows all access. That way, when something bad happens all you will have to do is boot to "single user mode" and copy the "open" backup configuration file over to the original. Then you should be good to go.

Adding a new module

Lets say, for instance, that your boss informs you that a new security policy has been written and needs to be implemented. It is now required to have a strong password with uppercase and lowercase letters and numbers as well as a special character. The policy has also been written such that users are only allowed two chances to enter a strong password.

As the security systems administrator, it is your job to implement this policy. What do you do to enforce it? The easy answer is that you add the cracklib.so PAM module. The cracklib.so module performs these password strength-checking functions for you by calling the Cracklib routine. If it passes that, additional checks are performed. A short synopses of the cracklib.so PAM module capabilities are; palindrome checks, case change only, similar (is the password too similar to the old), too simple, rotated, or already used.[8]

The line that you would add to your /etc/pam.d/passwd file would look something like this:

```
password  required  pam_cracklib.so retry=2 dcredit=-1 ucredit=-1 \
                                    ocredit=-1 lcredit=-1 minlen=6
password  required  pam_unix.so     use_authtok
```

These lines would force the users to select a password with a minimum length of six and with at least one digit number, one upper case letter, and one "other" or special character to succeed as well as only giving them two chances to enter a strong password. The use_authtok argument to pam_unix.so is used to force the module to set the new password to the one provided by the stacked pam_cracklib.so module.

What can we do if we need to have better password encryption than what is provided by the crypt() system call? The easy answer is to use md5 encryption. This would just take one additional argument to the pam_unix.so module. Using our example above we would have:

```
password  required  pam_cracklib.so retry=2 dcredit=-1 ucredit=-1 \
                                    ocredit=-1 lcredit=-1 minlen=6
password  required  pam_unix.so     use_authtok md5
```

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

The MD5 argument indicates that the MD5 function should be used to encrypt passwords.

What else is available?

It would seem that the current “hot” trend these days is to use LDAP for authentication as well as other services. This is fairly easy for the systems administrator to implement because there’s a pam_ldap.so PAM module readily available. The LDAP installation is typically set up such that a central authentication server is used so that users have a single login that works across all application accesses. These could include POP servers, IMAP servers, Samba servers, or even Windows machines. Using a central LDAP sever simplifies the system administrator’s job because all login scenarios would rely on the same user ID and password combination. This also alleviates the user from having to remember a multitude of different (strong) passwords.

As you can see it is easy to add modules to implement your sites' required security policy, but what else might be available that you could take advantage of? If your security policy requires voice authentication, a good place to start is a website by Lucas Correia Villa Real located at: (http://cscience.org/~lucasvr/projects/voiceauth.php)

A number of other PAM modules are also readily available or are in work. There are modules for SecurID, Kerberos, Samba, SSH, Radius, and a multitude of other modules.[9] Therefore, I would recommend that you frequent the Linux-PAM modules web page shown below for the latest news on PAM module development. (http://www.kernel.org/pub/linux/libs/pam/modules.html)

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Bibliography

1-Andrew G. Morgan, The Linux-PAM System Administrators' Guide, 1. Introduction, (http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-1.html)

2-Open Software Foundation, OSF Announces Integrated Login Solution Security technology unites disparate authentication mechanisms into common infrastructure, (http://wwws.sun.com/software/solaris/pam/osf-pr.html)

3-Andrew G. Morgan, The Linux-PAM System Administrators' Guide, 2. Some comments on the text, (http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-2.html)

4-Aeleen Frisch, GURU GUIDANCE, Linux User Authentication, (http://www.linux-mag.com/2000-06/guru_04.html)

5-Andrew G. Morgan, The Linux-PAM System Administrators' Guide, 4.1 Configuration file syntax, (http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-4.html#ss4.1)

6-Jennifer Vesperman, Introduction to PAM, Configuring PAM, (http://linux.oreillynet.com/pub/a/linux/2001/09/27/pamintro.html)

7-Vipin Samar, Charlie Lai Sun Microsystems, Inc.,Making Login Services Independent of Authentication Technologies, (http://java.sun.com/security/jaas/doc/pam.html)

8-Cristian Gafton, Cracklib pluggable password strength-checker (http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam-6.html#ss6.3)

9- Andrew G. Morgan, Modules/Applications available or in progress, (http://www.kernel.org/pub/linux/libs/pam/modules.html)

10-V. Samar & R. Schemers (SunSoft), Design Goals, Request For Comments:86.0, (http://www.kernel.org/pub/linux/libs/pam/pre/doc/rfc86.0.txt.gz)

11-SUN Microsystems, PAM Configuration File, (http://wwws.sun.com/software/solaris/pam/pam.admin.pdf)

12-FreeBSD, 4.2 Breakdown of a configuration line, (http://www.freebsd.org/doc/en_US.ISO8859-1/articles/pam/pam-config.html)

© SANS Institute 2003, As part of GIAC practical repository. Author retains full rights.

Last Updated: July 24th, 2017

Upcoming Training

SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Salt Lake City 2017	Salt Lake City, UT	Aug 14, 2017 - Aug 19, 2017	Live Event
Community SANS Omaha SEC401*	Omaha, NE	Aug 14, 2017 - Aug 19, 2017	Community SANS
SANS New York City 2017	New York City, NY	Aug 14, 2017 - Aug 19, 2017	Live Event
SANS Chicago 2017	Chicago, IL	Aug 21, 2017 - Aug 26, 2017	Live Event
SANS Virginia Beach 2017	Virginia Beach, VA	Aug 21, 2017 - Sep 01, 2017	Live Event
SANS Adelaide 2017	Adelaide, Australia	Aug 21, 2017 - Aug 26, 2017	Live Event
Community SANS Trenton SEC401	Trenton, NJ	Aug 21, 2017 - Aug 26, 2017	Community SANS
Virginia Beach 2017 - SEC401: Security Essentials Bootcamp Style	Virginia Beach, VA	Aug 21, 2017 - Aug 26, 2017	vLive
Community SANS Pasadena SEC401 @ NASA	Pasadena, CA	Aug 23, 2017 - Aug 30, 2017	Community SANS
Mentor Session - SEC401	Minneapolis, MN	Aug 29, 2017 - Oct 10, 2017	Mentor
SANS San Francisco Fall 2017	San Francisco, CA	Sep 05, 2017 - Sep 10, 2017	Live Event
SANS Tampa - Clearwater 2017	Clearwater, FL	Sep 05, 2017 - Sep 10, 2017	Live Event
Mentor Session - SEC401	Edmonton, AB	Sep 06, 2017 - Oct 18, 2017	Mentor
SANS Network Security 2017	Las Vegas, NV	Sep 10, 2017 - Sep 17, 2017	Live Event
Community SANS Albany SEC401	Albany, NY	Sep 11, 2017 - Sep 16, 2017	Community SANS
Mentor Session - SEC401	Ventura, CA	Sep 11, 2017 - Oct 12, 2017	Mentor
Community SANS Columbia SEC401	Columbia, MD	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Dallas SEC401	Dallas, TX	Sep 18, 2017 - Sep 23, 2017	Community SANS
Community SANS Boise SEC401	Boise, ID	Sep 25, 2017 - Sep 30, 2017	Community SANS
Baltimore Fall 2017 - SEC401: Security Essentials Bootcamp Style	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	vLive
Community SANS New York SEC401	New York, NY	Sep 25, 2017 - Sep 30, 2017	Community SANS
Rocky Mountain Fall 2017	Denver, CO	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS London September 2017	London, United Kingdom	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Baltimore Fall 2017	Baltimore, MD	Sep 25, 2017 - Sep 30, 2017	Live Event
SANS Copenhagen 2017	Copenhagen, Denmark	Sep 25, 2017 - Sep 30, 2017	Live Event
Community SANS Sacramento SEC401	Sacramento, CA	Oct 02, 2017 - Oct 07, 2017	Community SANS
SANS DFIR Prague 2017	Prague, Czech Republic	Oct 02, 2017 - Oct 08, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Oct 02, 2017 - Oct 07, 2017	Community SANS
Mentor Session - SEC401	Arlington, VA	Oct 04, 2017 - Nov 15, 2017	Mentor