UNIX Authentication and Pluggable-authentication Modules (PAMs)

Russell Bateman

Description of UNIX authentication, PAM authentication and configuration, how to make an application "PAM aware," how to write a PAM (sample code), comprehensive notes and bibliography.

UNIX Authentication

- */etc/passwd*: the good old days
  - Need to change? simply edit

```
root:x:0:0:root:/root:/bin/bash
russ:x:1002:100:Rusell Bateman:/home/russ:/bin/bash
```

- MD5 and shadow passwords become popular
  - applications had to code to different schemes
  - to change schemes, recompile

Enter PAM!

PAM eliminates mess by enabling programs to authenticate transparently, regardless of scheme employed

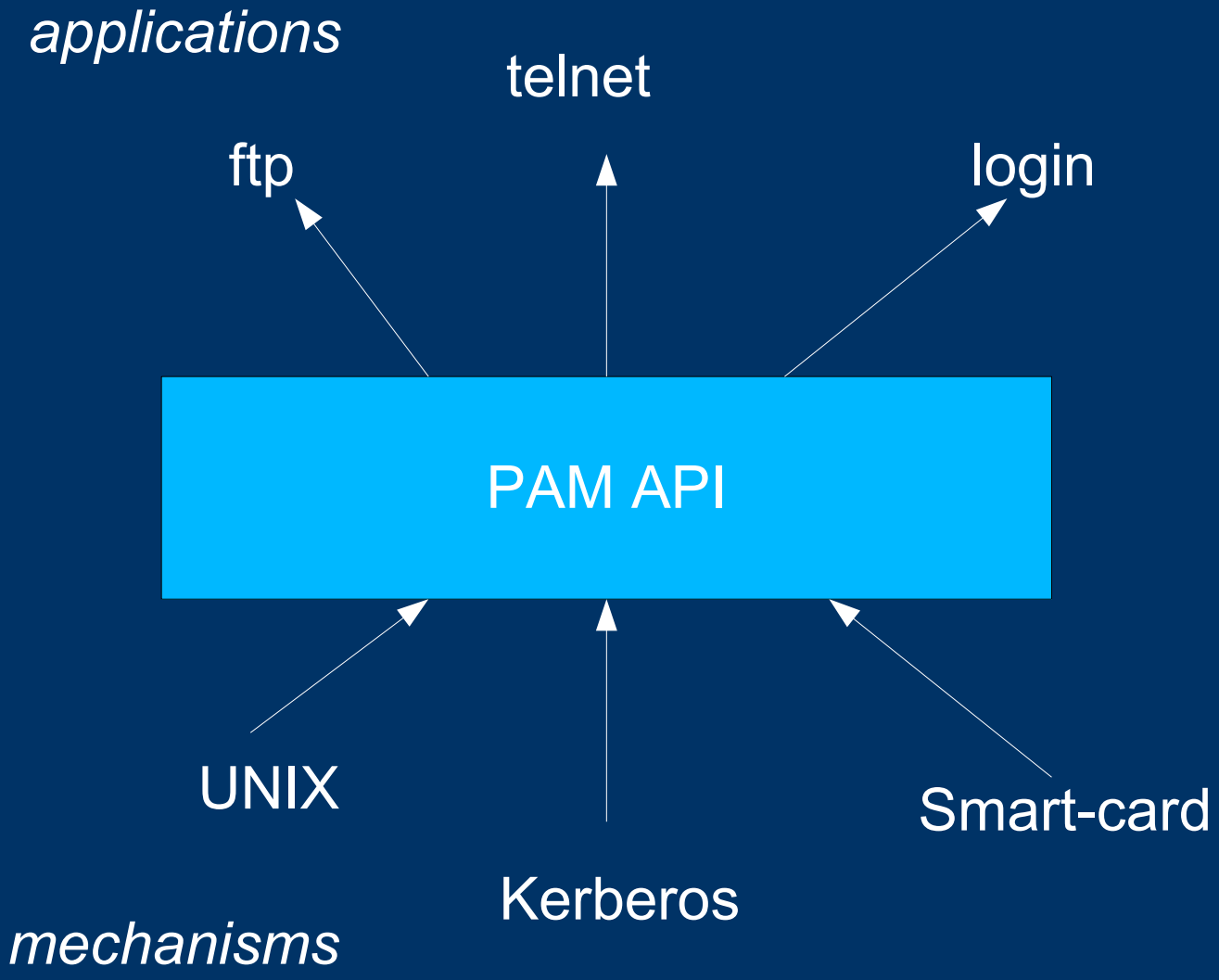
PAM Authentication

- Sun's pluggable-authentication module scheme
- Similar but not always identical between UNIX and Linux, or even Linux and Linux
- Simply about “security”: no longer an application's concern

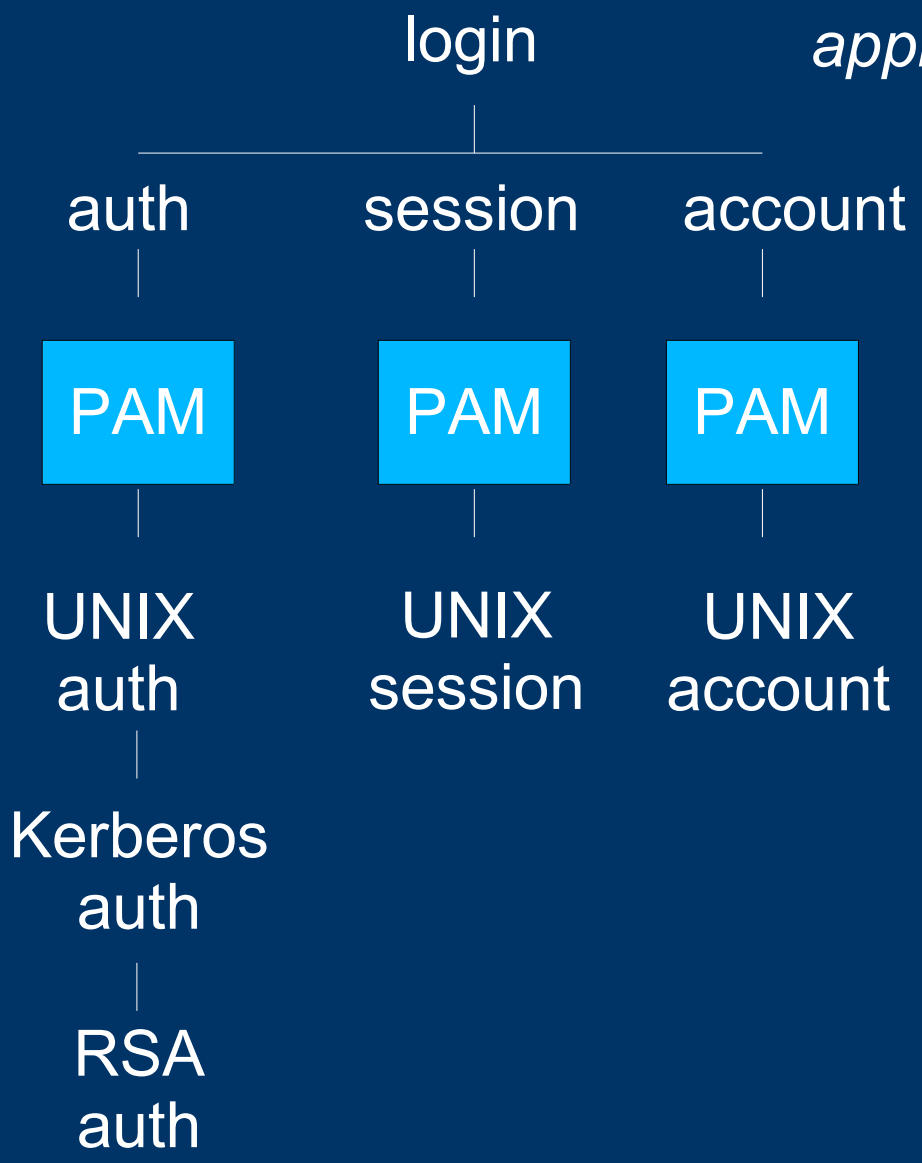
PAM Authentication (continued)

- Makes life easier for application developer and also for the system administrator
- Based on configuration files under system administrator control
- Extensible to thumb readers, retina scanners, devices that can measure evil intent via brainwaves 

Pictorial of PAM Framework



Stacking PAMs



PAM Configuration

- Configuration done in files off */etc/pam.d*
- One file per PAM-aware application
- (Some implementations use */etc/pam.conf*)

PAM Configuration--Example

- Prohibiting SSH (***s***ecure ***sh***ell) log-in
  - PAM module, *pam_time.so* (ships with RedHat or can be written); this module reads...
  - ...file */etc/security/time.conf* (used by *pam_time.so*). This statement happens to direct the behavior for SSH, but syntax is specific to and arbitrary in *pam_time.so*:

```
sshd;*;*;!Al2200-0400   // services;ttys;users;times
```

PAM Configuration--Example *(continued)*

- file */etc/pam.d/sshd*:

```
#%PAM-1.0
                                        *
account    required pam_time.so
auth       required pam_stack.so  service=system-auth
auth       required 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
```

* if *pam_time.so* doesn't give sshd a green light, there will be no SSH access by any account.

PAM Configuration (continued)

- the preceding example only applies to SSH (for example, via PuTTY); it does not prohibit console log-in, for example
- if it were useful to apply restrictions to the console, say, lock it each day from 2200 until 0400, the same change could be made to */etc/pam.d/login*

PAM Stacking

- For more than one authentication restriction or set of restrictions, PAMs may be “stacked” in some implementations
- Stacked merely means that a given instance may course through more than one PAM implementing different aspects of the total security solution on the host
- (see *pam_stack.so* in sample above)

PAM Defaults: the “other” File

- If a PAM-aware application has no corresponding file on the path */etc/pam.d*, the “other” file (here in defaults) comes into play

```
#%PAM-1.0
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
```

- This is frightful though, because if the application's file goes somehow missing, what's in “other” takes over and, by default, the application stops working completely!

Potential Uses of PAM

- Black-list hosts whose number of bad log-ins exceeds a threshold
- Prohibit access by *vi* to certain files
- Prohibiting removal of certain files with *rm*
- Licensing (simultaneous consumption, etc.)

Programming

Writing PAM-aware applications and PAMs

Making an Application "PAM-aware"

- Initialization
  - **`pam_start`**—initializes interface, reads configuration file and yields a handle
- Termination
  - **`pam_end`**—shuts down authentication stack, causes module to call its clean-up, etc.
- Getting and setting items
  - as for modules

PAM-aware Applications

- Authentication
  - **`pam_authenticate`**
- Setting user credentials
  - **`pam_setcred`**—called after authentication; consists of a cookie like a Kerberos ticket or other unique thing; onus of correctness and security on application's shoulders
- Updating authentication tokens
  - **`pam_chauthtok`**

PAM-aware Applications

- Miscellaneous
  - `pam_acct_mgmt`
  - `pam_open_session`
  - `pam_close_session`
  - `pam_getenv`
  - `pam_getenvlist`
  - `pam_putenv`

PAM-aware Applications

- What is expected of a PAM-aware application
  - **`struct pam_conv`**—provide conversation structure to module already initialized
  - when module calls **`conv()`**, **`appdate_ptr`** is set to second element of structure
  - etc.

Sample Application Code

```
/*
** pam_demo.c
** Loop checking user/password pairs until "q"...
**
** Note: An application consuming pluggable-authentication module(s)
** (PAMs) links libpam.a and libdl.a and, in this case, at least, a
** helper library, libpam_misc.a.
*/
#include <stdio.h>
#include <string.h>
#include <security/pam_misc.h>
#include <security/pam_appl.h>

#define TRUE    1
#define FALSE   0

struct pam_conv gConv =
{
   misc_conv,                   // convenient helper from pam_misc.h
   (void *) NULL                // 'appdata_ptr'
};
```

Sample Application Code

```
int main( void )
{
    int          err;
    pam_handle_t *pamh;

    while (TRUE)                    // loop until "q" typed...
    {
        char   user[128];

        printf("Enter log-in name: ");
        gets(user);

        if (stricmp(user, "q") == 0)
            break;

        if (err = pam_start("check_user", user, &gConv, &pamh)
                != PAM_SUCCESS)
        {
            printf("Authentication service failed to initialize...\n");
            exit(err);
        }

        ...
```

Sample Application Code

```
    ...
    err = pam_authenticate(pamh, 0);   // bonafide user?

    if (err == PAM_SUCCESS)
       err = pam_acct_mgmt(pamh, 0);   // ...with access?

    printf((err)
              ? "Authentication failed for %s...\n"
              : "Authentication succeeded for %s...\n", user);

    if (err = pam_end(pamh, err))
    {
       printf("Authentication service shutdown failed...\n");
       exit(err);
    }
  }

  return (err == PAM_SUCCESS) ? 0 : err;
}
```

Making a PAM

- Getting and setting instance data
  - in general, PAMs should not make use of C *static*
  - **`pam_set_data`**—initialization
  - **`pam_get_data`**—retrieving the instance data
- Getting and setting **`PAM_ITEMS`**
  - **`pam_set_item`**—initialization
  - **`pam_get_item`**—retrieving the instance data
- Conversation mechanism
  - allows the module to prompt for password consistent with the application (command-line, X Window dialog, etc.)

Making a PAM (continued)

- Getting user name
  - **`pam_get_user`**—library macro function
- Getting and setting PAM environment variables
  - **`pam_putenv`**
  - **`pam_getenv`**
  - **`pam_getenvlist`**
- Errors
  - facilitate time delays following a failed call to authenticate (hinders timed and brute-force attacks)
  - **`pam_fail_delay`**

Making a PAM (continued)

- What is expected of a PAM
  - authentication (*auth* in configuration statements)
    - **`pam_sm_authenticate`**
    - **`pam_sm_setcred`** (set credential)
  - account
    - **`pam_sm_acct_mgmt`**
  - session
    - **`pam_sm_open_session`**
    - **`pam_sm_close_session`**
  - password
    - **`pam_sm_chauthtok`** (change authorization token)

Sample Code

```
/* pam_checkuser.c
**
** A pluggable-authentication module (PAM) is a single executable
** binary file that can be loaded by the PAM interface library.
** This library is configured locally using a system file, either
** /etc/pam.conf or files off /etc/pam.d. The binary is stored on
** the path /usr/lib/security as a “special object” module (.so).
**
** Except for interacting with the user (entering a password, etc.),
** the PAM should not call the application directly. Instead, the
** documented "conversation mechanism" should be used.
*/
#include <stdio.h>
#include <security/pam_modules.h>

int pam_sm_authenticate
(
   pam_handle_t *pamh,
   int          flags,
   int          argc,
   const char   **argv
)
{
#pragma unused(flags,argc,argv)
   int          err;
   const char   *user;
```

Sample Code

```
    // our caller doesn't tell us what this is, but PAM will...
    err = pam_get_user(pamh, &user, NULL);

    if (err != PAM_SUCCESS)
    {
        printf("pam_get_user: %s", pam_strerror(pamh, err));
        return err;
    }

    if (!user || !*user)
    {
        printf("User name unknown--will not supply a default...\n");
        return PAM_USER_UNKNOWN;
    }

    return PAM_SUCCESS;
}

...
```

Sample Code

```
...

int pam_sm_acct_mgmt
(
   pam_handle_t *pamh,
   int          flags,
   int          argc,
   const char   **argv)
{
#pragma unused(pamh,flags,argc,argv)
   /*
   ** It's not yet abundantly clear what to do here in support of
   ** pam_demo.c. We would have to call into a UNIX authentication
   ** piece, or Kerberos, eDirectory, etc. depending on what we
   ** were trying to do.
   */
   return PAM_SUCCESS;
}

...
```

Sample Code

```
...

/* ---------------------------------------------------------------
** The remainder of this code does nothing except satisfy the caution
** that all six functions be supplied so that if called, they are
** extant. They are all, therefore, mere stubs that return success.
** ---------------------------------------------------------------
*/
int pam_sm_setcred
(
    pam_handle_t *pamh,
    int          flags,
    int          argc,
    const char   **argv)
{
#pragma unused(pamh,flags,argc,argv)
    return PAM_IGNORE;
}
```

likewise...

```
int pam_sm_chauthtok()
int pam_sm_open_session()
int pam_sm_close_session()
```

Bibliography (Webography)

- *The Linux-PAM Writers' Manual* and *The Application Developers' Manual*
  - http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/
  - elucidates programming models for
    - pluggable-authentication modules
    - applications that consume pluggable authentication
- Modules/Applications available or in progress
  - http://www.kernel.org/pub/linux/libs/pam/modules.html
  - modules and applications whose source is available (and links thereto)

Bibliography (Webography)

- User Authentication How-to
  - http://www.faqs.org/docs/Linux-HOWTO/User-Authentication-HOWTO.html#AEN101
  - useful discussion on UNIX/Linux user authentication leading to PAM
- Lawrence, Tony, *Understanding PAM*
  - http://aplawrence.com/cgi-bin/printer.pl?/Basics/understandingpam.html
  - excellent conceptual treatment: PAM simplified
- ibid, SSH
  - http://aplawrence.com/cgi-bin/nsrelated.pl?ssh
  - obliquely related corral of SSH issues and answers

Bibliography (Webography)

- *Unified Login with Pluggable Authentication Modules (PAM)*
  - http://www.opengroup.org/tech/rfc/rfc86.0.html
  - RFC 86.0 (October 1995) itself

Notes on PAM

Notes recorded in presentation for use in writing a how-to or introductory document on PAM use

PAM Types

- ***account***: provide account verification types of service: “Has the user's password expired?” “Is this user permitted access to the requested service?”
  account modules check to ensure that authentication is allowed (account valid, user authorized at current time, etc.).
- ***auth***entication: establish the user is who he claims to be typically via challenge-response, but also via smart-card, biometric device, etc.
  auth modules provide the actual authentication and set credentials such as group membership or Kerberos tickets.
- ***password***: has the task of updating authentication mechanisms including setting the password.

PAM Types (continued)

- ***session***: covers things to be done prior to giving a service and after withdrawing it including
  - maintaining audit trails
  - mounting account's home directory
  - furnishing opening and closing hook by which module affects the available services
  - other tasks limited only by imagination.

PAM Control

- ***requisite***: failure to authenticate via this module results in denial of authentication to host.
- ***required***: failure results in denial of authentication only if subsequent modules also deny it.
- ***sufficient***: if module successful, PAM grants authentication even if a previous ***required*** module failed.
- ***optional***: failure of this module is significant only if it is the only of its type for this service.

Module Path

- The module path tells PAM which module to use for a given type and where to find it
- If only module name (no path), look for module in PAM module directory
  - */etc/pam.d*
  - or */lib/security*
- Some implementations put everything into one file, */etc/pam.conf* in which case, syntax is slightly different with service prepended thus:

```
login    auth     required    pam_unix.so    nullok
```

Module Path *(continued)*

- Services that authenticate, but don't have a PAM module or whose module isn't specified or is missing, have the “other” configuration file imposed: */etc/pam.d/other*

```
#%PAM-1.0
auth        required    pam_warn.so
auth        required    pam_deny.so
account     required    pam_warn.so
account     required    pam_deny.so
password    required    pam_warn.so
password    required    pam_deny.so
session     required    pam_warn.so
session     required    pam_deny.so
```

Module Arguments

- Arguments to be passed to the module
- Arbitrary; belong to the module implementation
- E.g.: *pam_unix.so*:

```
auth required    pam_unix.so    nullok
```

  - “nullok” indicates that a null password is acceptable

PAM Implementation Differences

- Redhat Linux uses *pam_pwdb*
- SuSE uses *pam_unix*
- FreeBSD does not support *session* directives

MD5...

...takes a message of arbitrary length and produces a 128-bit "fingerprint" or "message digest" of the input. The conjectured is that it is computationally infeasible to produce two messages with the same message digest, or to produce any message having a given prespecified target message digest. Intended for digital signature applications where a large file is "compressed" in a secure manner before being encrypted with a private (secret) key under a public-key cryptosystem such as RSA.

Kerberos...

...is a network authentication protocol for client/server applications using secret-key cryptography. A free implementation is available from MIT.