PAM

(*Pluggable Authentication Modules*)

Motivation

- Users
  - Unification of authentication mechanisms for different applications
- Manufacturers
  - Authenticated access to services independently of the authentication mechanism
- Administrators
  - Simple management and matching of N authentication mechanisms for M services requiring client authentication
  - Flexibility to configure specific authentication mechanisms for each host
- Manufacturers and Administrators
  - Flexible and modular approach for integrating novel authentication mechanisms

Existing (1/2)

- Services requiring client authentication use hardcoded mechanisms
- The services that implement authentication mechanisms use hardcoded options
- It is not easy to integrate several authentication mechanisms

Existing problems (2/2)

- Different services may require different authentication mechanisms
  - rlogin can use information about trustworthy hosts
  - *Login* without repeated passwords
    - *One-time keys*
  - *Login* with biometrics
- Different approaches for graphical and non-graphical (text) interfaces

PAM: goals (1/2)

- Default mechanism per host
  - The administrator should be able to choose and configure the default authentication mechanism
    - Username/password, biometrics, smart-cards, etc.
- Application-specific mechanisms
  - Each application should be able to exploit different authentication mechanisms
    - Login with S/Key for remote sessions
    - Ordinary username/password login for local sessions
- Several interface approaches
  - Input from text consoles of graphical windows
  - Access to special devices (smart-cards, biometric readers, etc.)
- Several authentication protocols
  - Ex. Linux authentication + S/Key authentication

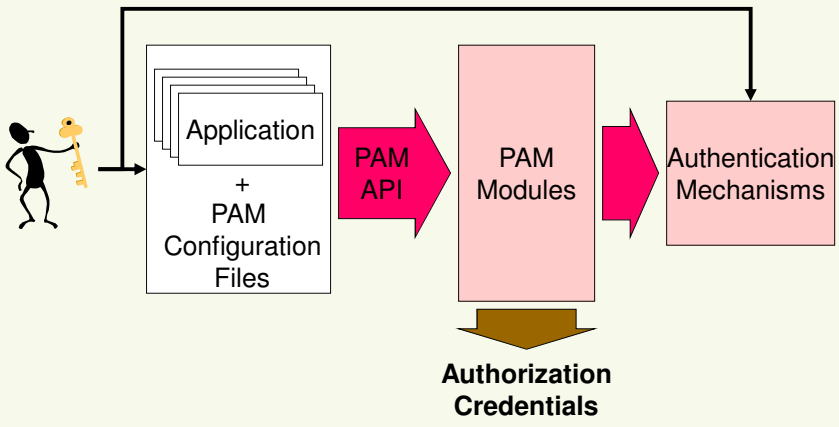
PAM: goals (2/2)

- Simplicity
  - Stacking of mechanisms
  - Minimal user perception
    - Ex. single password input request
- Increased security
  - Multi-factor authentication
  - Different keys/secrets/PINs/passcodes/passwords/passphrases
- Services don't need to be changed
  - The update of authentication mechanisms for a particular service does not imply a modification of the service code/configuration
- Modular architecture
  - Dynamic loading of required modules
  - Handling of several actions besides authentication
    - Password management,
    - Accounting management
    - Session management

Classic Unix authentication

- Requested information: *username* + password
- Validation
  - Existence of an active account
    - Entry with the username in the /etc/passwd file
  - Ciphered password
  - Comparison of the provided password with the content of the ciphered password (salted)
- Obtained credentials
  - UID + GID [+ list of secondary GIDs]
  - Allowance to create new process descriptor (*login shell*)

PAM: Architecture



PAM: Actions (Management Group)

- Authentication (auth)
  - Identity verification
- Account Management (account)
  - Enforcement of access policies based on account properties
- Password Management (password)
  - Control of the password modification process
- Session Management (session)
  - Verification of operational parameters
  - Enforcement of session parameters
    - max memory, max file descriptions...

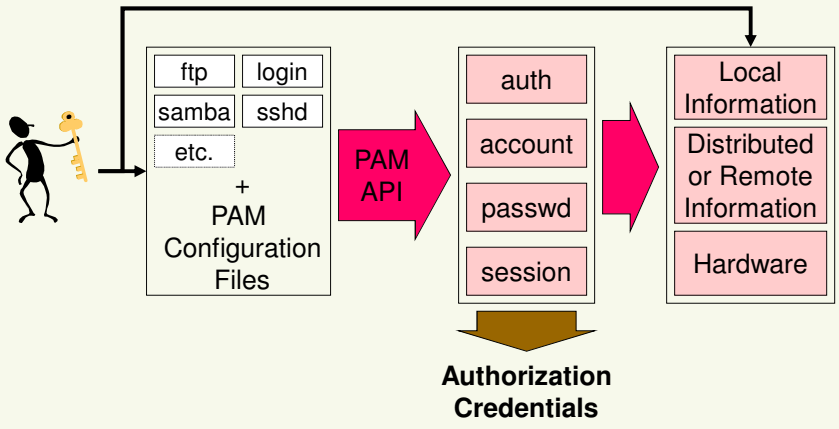
PAM: Modules

- Standard API
  - Functions provided by modules are invoked
    - Functions have well known prototypes (name, parameters, return value)
  - Decision based on the status code
    - PAM_SUCCESS, PAM_AUTH_ERR, PAM_AUTHINFO_UNAVAIL, etc…
- Dynamically loaded (*shared libraries*)
  - /lib/security/pam_*.so
- Modules can be used for one or more actions
  - According to the functions implemented

PAM: Configuration files

- Typically, one per PAM client application
  - E.g: /etc/pam.d/ftp or /etc/pam.d/ssh
  - Can have shared files: /etc/pam.d/common-auth
- Specify how the actions should be applied
  - Which mechanisms to use
    - Which dynamic library (module) to load
  - Which parameters to use
  - When is the action completed
- Each module uses a particular set of resources
  - Information in local files
    - /etc/passwd, /etc/shadow, /etc/groups, etc.
  - Distributed information or located in remote servers
    - NIS, Kerberos, LDAP, etc.

PAM: Detailed Architecture



PAM APIs: PAM lib (1/2)

- Start/End of the PAM lib
  pam_start(service, user name, callback, &pam_handle)
  pam_end(pam_handle, status)
- Execution of PAM actions
  - Defined a stack of modules per action
    - All modules in stack are executed from top to bottom
    - Each module has its own parameters and calling semantic
      - **Required, requisite, sufficient, optional**
  - Execution proceeds until the end, or failure
    - To better hide the source of a failure, module execution can either abort immediately or force a failure after the stack is executed.
  - Applications can recover from failures

PAM APIs: PAM lib (2/2)

- "auth" Action
  pam_authenticate(pam_handle, flags)
  pam_setcred(pam_handle, flags)
- "account" Action
  pam_acct_mgmt(pam_handle, flags)
- "passwd" Action
  pam_chauthtok(pam_handle, flags)
- "session" Action
  pam_open_session(pam_handle, flags)
  pam_close_session(pam_handle, flags)
- Module specific data
  pam_get_data(), pam_set_data()
  pam_get_item(), pam_set_item()

PAM APIs: PAM modules

- "auth" Action
  pam_sm_authenticate(pam_handle, flags)
  pam_sm_setcred(pam_handle, flags)
- "account" Action
  pam_sm_acct_mgmt(pam_handle, flags)
- "passwd" Action
  pam_sm_chauthtok(pam_handle, flags)
- "session" Action
  pam_sm_open_session(pam_handle, flags)
  pam_sm_close_session(pam_handle, flags)

PAM: Success Control

- Syntax: action control module [parameters]
- Control is specified for each action and module

requisite
- If the module fails, the result is returned immediately

required
- If the module fails, the result is set but following modules are called

sufficient
- If module is successful
  - Returns success if all previous "required" modules also were successful
- If module fails the result is ignored

optional
- Result is ignored
- EXCEPT: if this is the only module in the action

Configuration files: /etc/pam.d/ftp

```
Standard behaviour for ftpd(8).
auth required pam_listfile.so item=user sense=deny file=/etc/ftpusers onerr=succeed

# This line is required by ftpd(8).
auth    sufficient  pam_ftp.so

# Uncomment this to achieve what used to be ftpd -A.
#auth required pam_listfile.so item=user sense=allow file=/etc/ftpchroot onerr=fail

# Standard blurb.
@include common-auth
@include common-account
@include common-session
```


Configuration files: /etc/pam.d/ssh

```
auth       required     pam_env.so # [1]
auth       required     pam_env.so envfile=/etc/default/locale

@include common-auth

account    required     pam_nologin.so
@include common-account

@include common-session
session    optional     pam_motd.so # [1]
session    optional     pam_mail.so standard noenv # [1]
session    required     pam_limits.so

# Standard Un*x password updating.
@include common-password
```

Configuration files:

/etc/pam.d/login (inc. /etc/pam.d/common-*)

```
#%PAM-1.0
auth       requisite  pam_securetty.so
auth       requisite  pam_nologin.so
auth       requisite  pam_unix.so nullok_secure
auth       optional   pam_smbpass.so migrate missingok
auth       optional   pam_group.so

account    required   pam_unix.so

password   requisite  pam_unix.so nullok obscure md5
password   optional   pam_smbpass.so nullok use_authok use_first_pass missingok

session    required   pam_selinux.so close
session    required   pam_env.so readenv=1
session    required   pam_env.so readenv=1 envfile=/etc/default/locale
session    required   pam_limits.so
session    optional   pam_lastlog.so
session    optional   pam_motd.so
session    optional   pam_mail.so standard
session    required   pam_linux.so
session    required   pam_selinux.so open
```